# TESIS DOCTORAL

# Path planning and redundancy resolution of structure-climbing redundant robots

Marc Fabregat Jaén

2025

DIRECTOR:
Óscar Reinoso García

CODIRECTOR:
Adrián Peidró Vidal

**UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE**

Programa de Doctorado en
TECNOLOGÍAS INDUSTRIALES Y DE
TELECOMUNICACIÓN

La presente Tesis Doctoral, titulada “Path planning and redundancy resolution of structure-climbing redundant robots”, se presenta bajo la modalidad de **tesis por compendio** de las siguientes **publicaciones**:

- Fabregat-Jaén, M., Peidró, A., Colombo, M., Rocco, P., & Reinoso, Ó. (2025). Topological and spatial analysis of self-motion manifolds for global redundancy resolution in kinematically redundant robots. Mechanism and Machine Theory, 210, 106020.

- Fabregat-Jaén, M., Peidró, A., Mollá-Santamaría, P., Soler, F. J., & Reinoso, O. (2024). Planificación jerárquica de movimientos de un robot trepador bípedo en estructuras tridimensionales reticulares. Revista Iberoamericana de Automática e Informática Industrial, 21(3), 262-273.

El Dr. D. Óscar Reinoso García, director, y el Dr. D. Adrián Peidró Vidal, codirector de la tesis doctoral titulada **“Path planning and redundancy resolution of structure-climbing redundant robots”**

**INFORMAN:**

Que D. Marc Fabregat Jaén ha realizado bajo nuestra supervisión el trabajo titulado **“Path planning and redundancy resolution of structure-climbing redundant robots”** conforme a los términos y condiciones definidos en su Plan de Investigación y de acuerdo al Código de Buenas Prácticas de la Universidad Miguel Hernández de Elche, cumpliendo los objetivos previstos de forma satisfactoria para su defensa pública como tesis doctoral.

Lo que firmamos para los efectos oportunos, en a de de 2025



| Director | Codirector/a de la tesis |
| --- | --- |
| Dr. D. Óscar Reinoso García | Dr. D. Adrián Peidró Vidal |

El Dr. D. Germán Torregrosa Penalva, Coordinador del Programa de Doctorado en Tecnologías Industriales y de Telecomunicación de la Universidad Miguel Hernández de Elche

**INFORMA**:

Que D. Marc Fabregat Jaén ha realizado bajo la supervisión de nuestro Programa de Doctorado el trabajo titulado **"Path planning and redundancy resolution of structure-climbing redundant robots"** conforme a los términos y condiciones definidos en su Plan de Investigación y de acuerdo al Código de Buenas Prácticas de la Universidad Miguel Hernández de Elche, cumpliendo los objetivos previstos de forma satisfactoria para su defensa pública como tesis doctoral.

Lo que firmo para los efectos oportunos, en a de de 2025



Prof. Dr. D. Germán Torregrosa Penalva

Coordinador del Programa de Doctorado en Tecnologías Industriales y de Telecomunicación de la Universidad Miguel Hernández de Elche

# ABSTRACT

This thesis addresses critical challenges in the planning and control of biped structure-climbing redundant robots that navigate three-dimensional truss structures such as bridges, building skeletons, and power transmission towers. Maintenance tasks in these environments, traditionally performed by human workers, present significant safety risks that can be mitigated through robotic solutions.

Two fundamental challenges are addressed in the present thesis: path planning for structure navigation and redundancy resolution for optimal joint trajectory generation.

First, we develop a hierarchical path planning algorithm that decomposes complex 3D exploration into manageable subproblems: a first planner determines the global path in the complete structure, while a second planner derives the local path in each visited face of the structure. This approach is generalizable, and efficiently handles transitions between structural faces while avoiding collisions.

Second, we present three complementary approaches to redundancy resolution. The first method is based on Feasibility Maps (FMs), which provide a representation of the redundant-time space, on which we deploy an RRT-based algorithm to generate feasible joint trajectories while optimizing performance criteria. Building upon this, we extend the concept to Augmented Feasibility Maps (AFMs), which incorporates the task dimension into the redundancy resolution process, effectively tackling both the task-trajectory generation and redundancy resolution problems simultaneously.

For the third approach, we developed a global redundancy resolution framework based on Self-Motion Manifolds (SMMs), which characterize the complete set of inverse kinematics solutions. We perform a spatial and topological analysis on the transformations of the SMMs throughout the task trajectory, and derive optimal joint trajectories from this analysis. All of these methods are able to incorporate task constraints, such as joint limits and obstacle avoidance, into the redundancy resolution process.

Finally, we introduce a novel homotopy-based method for computing SMMs that addresses limitations of existing approaches. This method effectively identifies all disjoint manifold components without requiring closed-form kinematic solutions, demonstrating superior performance compared to traditional continuation methods.

The contributions are validated through extensive simulation tests, showing significant improvements over existing methods in terms of computational efficiency, path quality, and practical applicability. This research advances the state

of the art in structure-climbing robotics and redundancy resolution, bringing these climbing robots closer to practical deployment in industrial inspection and maintenance applications.

# RESUMEN

Esta tesis aborda desafíos en la planificación y control de robots trepadores bípedos redundantes que navegan por estructuras reticulares tridimensionales como puentes, esqueletos de edificios y torres de transmisión eléctrica. Las tareas de mantenimiento en estos escenarios, tradicionalmente realizadas por operarios humanos, presentan riesgos significativos de seguridad que pueden mitigarse mediante soluciones robóticas.

En la presente tesis se abordan dos desafíos fundamentales: la planificación de trayectorias para la navegación en estructuras y la resolución de redundancia para la generación óptima de trayectorias articulares.

En primer lugar, desarrollamos un algoritmo jerárquico de planificación de trayectorias que descompone la compleja exploración tridimensional en subproblemas manejables: un primer planificador determina la trayectoria global en la estructura completa, mientras que un segundo planificador deriva la trayectoria local en cada cara visitada de la estructura. Este enfoque es generalizable y maneja eficientemente las transiciones entre caras estructurales mientras se evitan colisiones.

En segundo lugar, presentamos tres enfoques complementarios para la resolución de redundancia. El primer método se basa en Mapas de Factibilidad (FMs), que proporcionan una representación del espacio redundante-tiempo, sobre el cual implementamos un algoritmo basado en RRT para generar trayectorias articulares factibles mientras optimizamos criterios de rendimiento. Sobre esta base, extendemos el concepto a Mapas de Factibilidad Aumentados (AFMs), que incorporan la dimensión de la tarea en el proceso de resolución de redundancia, abordando simultáneamente tanto los problemas de generación de trayectorias de la tarea como de resolución de redundancia.

Para el tercer enfoque, desarrollamos un marco global de resolución de redundancia basado en Variedades de Automovimiento (SMMs), que caracterizan el infinito conjunto soluciones de la cinemática inversa. Realizamos un análisis espacial y topológico sobre las transformaciones de las SMMs a lo largo de la trayectoria de la tarea, y derivamos trayectorias articulares óptimas a partir de este análisis. Todos estos métodos son capaces de incorporar restricciones adicionales, como límites articulares y evasión de obstáculos, durante el proceso de resolución de redundancia.

Finalmente, introducimos un nuevo método basado en homotopía para calcular SMMs que solventa algunas de las limitaciones de los enfoques existentes. Este método identifica eficazmente todos los componentes disjuntos de la va-

riedad sin requerir soluciones cinemáticas de forma cerrada, demostrando un rendimiento superior en comparación con los métodos tradicionales de continuación.

Las contribuciones son validadas mediante extensas pruebas de simulación, mostrando mejoras significativas sobre los métodos existentes en términos de eficiencia computacional, calidad de trayectoria y aplicabilidad práctica. Esta investigación avanza el estado del arte en robótica trepadora de estructuras y resolución de redundancia, acercando estos robots trepadores a su despliegue práctico en aplicaciones industriales de inspección y mantenimiento.

# ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my thesis supervisors, Adrián Peidró and Óscar Reinoso, for their guidance, support, and insightful feedback throughout this journey.

Especially, I would like to thank Adrián for his expertise and encouragement, which have been fundamental in shaping both this work and my development as a researcher.

I would also like to thank my lab colleagues, Álvaro, Antonio, Enrique, Esther, Fran Martínez, Fran Soler, Juanjo, Judith, Marcos, María, Míriam, Paula and Sergio, for their support. You have made this long journey much more enjoyable.

My appreciation extends to the rest of the professors of the ARVC research group: Arturo, David, José María, Luis, Luis Miguel, and Mónica; most of whom I have had the pleasure of collaborating with and learning from during my research and teaching activities throughout these years.

Outside of the academic world, and most importantly, I would like to thank my loved ones for their love and encouragement during the toughest moments. I am especially grateful to my girlfriend, Alicia, for her unwavering support and understanding, and to my parents, who have always believed in me and encouraged me to pursue my aspirations. Your love and support have been invaluable to me.

# FUNDING

This thesis has been developed under the Research, Development and Innovation project entitled *Robots híbridos y reconstrucción multisensorial para aplicaciones en estructuras reticulares (HyReBot)* (Hybrid robots and multisensory reconstruction for applications in truss structures), with reference PID2020-116418RB-I00, funded by the Spanish Ministry of Science, Innovation and Universities (MCIN) the Spanish State Research Agency (AEI), with reference MCIN/AEI/10.13039/501100011033.

Associated with this project, this thesis was financially supported by the *Ayuda para contratos predoctorales para la formación de doctores (FPI)* grant, with reference PRE2021-099226, awarded by the Spanish Ministry of Science, Innovation and Universities (MCIN) and the Spanish State Research Agency (AEI), with reference MCIN/AEI/10.13039/501100011033, and the European Social Fund Plus (ESF+).

This grant also supported a four-month research stay (from September to December 2023) at the *Mechatronics and Robotics Laboratory for Innovation (MeRLIn)*, hosted by the *Department of Electronics Information and Bioengineering (DEIB)* of the *Politecnico di Milano* (Italy).

# CONTENTS

# LIST OF TABLES

## LIST OF ALGORITHMS



## ACRONYMS

| | |
|---|---|
| DoF | Degrees of Freedom |
| FK | Forward Kinematics |
| IK | Inverse Kinematics |
| IKP | Inverse Kinematics Problem |
| RRT | Rapidly-exploring Random Tree |
| TFP | Transition Footholds Planner |
| FFP | Face Footholds Planner |
| DFFP | Discrite Face Footholds Planner |
| CFFP | Continuous Face Footholds Planner |

FM Feasibility Map

AFM Augmented Feasibility Map

SMM Self-Motion Manifold

SMD Self-Motion Domain

QP Quadratic Programming

# 1 INTRODUCTION

## 1.1 MOTIVATION

This thesis is situated in the field of robotics, specifically focusing on structure-climbing redundant robots designed to navigate three-dimensional truss structures such as bridges, skeletons of buildings, and power distribution towers. These structures require regular inspection and maintenance tasks to guarantee their safety and proper functioning, which are typically performed by human operators. However, these tasks present significant dangers to human workers, with falls from considerable heights being the most evident and critical risk.

To mitigate these risks to human life, researchers worldwide have been investigating for more than three decades the potential of employing climbing robots to perform these hazardous tasks. Despite numerous climbing robot designs being proposed during this period, the vast majority of dangerous industrial inspection tasks continue to be performed by human workers in practice. While industrial manipulators have become ubiquitous in manufacturing environments, climbing robots for industrial inspection rarely progress beyond research laboratories into practical applications.

Structure-climbing robots generally follow an inchworm-like design, composed of a multi-DoF manipulator with grippers at its ends that adhere to the structure for climbing. This architecture effectively transforms these robots into mobile manipulators, where the base continuously changes as the robot explores the structure. This design provides the high maneuverability necessary for navigating complex three-dimensional structures with obstacles.

However, this manipulator-like architecture also introduces significant challenges in deployment, as these robots tend to be slow and considerably more difficult to control and plan movements for, when compared to simpler wheeled climbing robots. Indeed, the inherent complexity of inchworm-like structure-climbing robots appears to be a primary factor preventing their widespread adoption in industrial inspection applications.

The HyReCRo (Hybrid Redundant Climbing Robot) robot, which serves as the primary research platform in this thesis, represents a biped design approach to structure-climbing. Each leg of this robot features a hybrid serial-parallel architecture, consisting of a serial combination of two identical 2RPR-PR parallel mechanisms. With its ten degrees of freedom, the HyReCRo robot is kinemati-

cally redundant, offering additional flexibility in motion planning and obstacle avoidance.

The principal advantage of the HyReCRo robot lies in its ability to perform the essential postures required for exploring three-dimensional structures: convex transitions between different faces of a beam (or different beams) and concave transitions between different beams. These capabilities make it particularly well-suited for navigating complex industrial structures that feature both types of transitions.

In this thesis, we address several fundamental challenges related to the operation of structure-climbing biped redundant robots like the HyReCRo. First, we develop a path planning algorithm that enables the robot to navigate three-dimensional structures efficiently. Second, we tackle the redundancy resolution problem, proposing novel approaches based on both Feasibility Maps (FMs) and Self-Motion Manifolds (SMMs). Finally, we introduce a homotopy-based method for computing SMMs that overcomes limitations of existing techniques, especially for complex kinematic chains where analytical solutions are difficult or impossible to obtain.

## 1.2 OBJECTIVES

The primary goal of this thesis is to develop efficient methods for path planning and redundancy resolution in structure-climbing robots, with specific application to the HyReCRo robot. These methods aim to overcome the key challenges that have limited the practical deployment of structure-climbing robots in industrial applications. To achieve this goal, the following specific objectives were established:

1. **Develop a path planning algorithm for step-by-step biped robots on three-dimensional structures**: We found that the best approach consisted in a hierarchical path planning algorithm. This involves creating a two-level planning strategy that divides the complex three-dimensional planning problem into more manageable subproblems. The algorithm must be capable of finding optimal sequences of footholds that allow the robot to navigate complex environments with both convex and concave transitions while avoiding collisions and respecting kinematic constraints.

2. **Establish comprehensive methods for redundancy resolution in redundant robots**: This major objective addresses the fundamental challenge of kinematic redundancy through multiple approaches:

    a) **Create a framework for redundancy resolution based on Feasibility Maps**: This framework should generate feasible joint-space trajectories for redundant robots following prescribed task trajectories. The

approach must consider the kinematic constraints of the robot, handle obstacle avoidance, and optimize criteria such as energy consumption or manipulability throughout the trajectory.

b) **Formulate the concept of Augmented Feasibility Maps**: Extending the FM concept to simultaneously solve the redundancy resolution and task trajectory generation problems. This integration allows for generating task trajectories that are feasible by construction, eliminating the need for predefined paths that may not be kinematically achievable.

c) **Develop a comprehensive framework for global redundancy resolution**: Based on the SMMs and Self-Motion Domain (SMD) of the given task trajectory, this framework aims to overcome the limitations of local approaches by providing a global description of the solution space. It should identify all available joint configurations that satisfy the task constraints and determine optimal transitions between them throughout the entire task trajectory.

d) **Introduce an efficient homotopy-based method for computing Self-motion Manifolds**: This novel computational approach should bridge the gap between sampling-based methods and continuation methods for SMM computation. It must not require closed-form solutions of robot kinematic equations, be able to identify all disjoint manifold components, and demonstrate better computational efficiency than traditional continuation methods.

3. **Validate the proposed algorithms and frameworks through extensive simulations and experimental tests**: Applying the developed methods to the HyReCRo robot and other relevant robotic platforms to demonstrate their effectiveness, robustness, and practical applicability in real-world simulated scenarios.

These objectives collectively aim to advance the state of the art in planning and control strategies for structure-climbing robots, making them more practical for real-world industrial applications. By addressing both the path planning and redundancy resolution challenges in a cohesive framework, this thesis seeks to contribute to the broader goal of increasing the adoption of climbing robots for dangerous inspection and maintenance tasks.

## 1.3 FRAMEWORK OF THIS THESIS

The different grants, research projects, collaborations and publications that have contributed to the development of this thesis are summarized in this section.

### 1.3.1 *Research Projects and Grants*

This thesis has been developed under the Research, Development and Innovation project entitled *Robots híbridos y reconstrucción multisensorial para aplicaciones en estructuras reticulares (HyReBot)* (Hybrid robots and multisensory reconstruction for applications in truss structures), with reference PID2020-116418RB-I00, funded by the Spanish Ministry of Science, Innovation and Universities (MCIN) the Spanish State Research Agency (AEI), with reference MCIN/AEI/10.13039/501100011033.

Associated with this project, this thesis was financially supported by the *Ayuda para contratos predoctorales para la formación de doctores (FPI)* grant, with reference PRE2021-099226, awarded by the Spanish Ministry of Science, Innovation and Universities (MCIN) and the Spanish State Research Agency (AEI), with reference MCIN/AEI/10.13039/501100011033, and the European Social Fund Plus (ESF+).

This grant also supported a research stay at a foreign university, as the next subsection describes.

### 1.3.2 *Research Stays and Collaborations*

From September to December 2023, the author of this thesis conducted a four-month research stay at the *Mechatronics and Robotics Laboratory for Innovation (MeRLIn)*, hosted by the *Department of Electronics Information and Bioengineering (DEIB)* of the *Politecnico di Milano* (Italy). This stay was supervised by Prof. *Paolo Rocco*, and focused on the research and development of a novel method for global redundancy resolution based on SMMs and SMDs, which is presented in Chapter 5 of this thesis. The research stay was funded by the *Ayuda para contratos predoctorales para la formación de doctores (FPI)* grant, with reference PRE2021-099226, awarded by the Spanish Ministry of Science, Innovation and Universities (MCIN) and the Spanish State Research Agency (AEI), with reference MCIN/AEI/10.13039/501100011033, and the European Social Fund Plus (ESF+).

## 1.4 PUBLICATIONS

The research presented in this thesis has led to two JCR-ranked journal publications, four international conference papers, and two national conference papers. The following subsections list the publications that have resulted from this work, including those that are in preparation.

### 1.4.1 *Journal Publications Supporting this Thesis*

Two of the main contributions of this thesis are supported by two publications in JCR-ranked journals belonging to the first and fourth quartiles of their respective categories:

- Marc Fabregat-Jaén, Adrián Peidró, Matteo Colombo, Paolo Rocco, and Óscar Reinoso (2025). "Topological and spatial analysis of self-motion manifolds for global redundancy resolution in kinematically redundant robots." In: *Mechanism and Machine Theory* 210, p. 106020. ISSN: 0094-114X. DOI: 10.1016/j.mechmachtheory.2025.106020
  **SCI-JCR Impact Factor: 5.3**, Quartile **Q1**

- Marc Fabregat-Jaén, Adrián Peidró, Paula Mollá-Santamaría, Francisco José Soler, and Oscar Reinoso (2024). "Planificación jerárquica de movimientos de un robot trepador bípedo en estructuras tridimensionales reticulares." In: *Revista Iberoamericana de Automática e Informática Industrial* 21.3, pp. 262–273. DOI: 10.4995/riai.2024.20779
  **SCI-JCR Impact Factor: 1.2**, Quartile **Q4**

The first publication, (Fabregat-Jaén et al., 2025), presents the framework for global redundancy resolution based on SMMs and SMDs, which is detailed in Chapter 5 of this thesis. The method successfully overcomes the limitations of local approaches by providing a global description of the solution space, enabling the identification of optimal joint trajectories that satisfy task constraints throughout the entire trajectory.

The second publication, (Fabregat-Jaén et al., 2024), introduces the hierarchical path planning algorithm for step-by-step biped robots on three-dimensional truss structures, which is presented in Chapter 3 of this thesis. This algorithm effectively divides the complex three-dimensional planning problem into manageable subproblems, allowing the robot to navigate complex environments with both convex and concave transitions while avoiding collisions and respecting kinematic constraints.

### 1.4.2 *Pending Publications*

The research presented in this thesis has also led to the preparation of two additional publications, which are currently in preparation:

- A paper with the final refined version of the path planning algorithm for step-by-step biped robots on three-dimensional truss structures, related to Chapter 3, with real experimental results with a prototype of the HyReCRo robot.

- A paper that presents the homotopy-based method for computing SMMs detailed in Chapter 6, with comparisons with state-of-the-art methods and experimental results.

## 1.5 STRUCTURE OF THE THESIS

This thesis is organized into seven chapters. Following this introduction, the subsequent chapters are structured as follows:

- **Chapter 2: Literature Review** presents a comprehensive review of the scientific literature related to climbing robots, path planning techniques for three-dimensional robots, and redundancy resolution methods. This chapter establishes the context for the research and identifies gaps in existing approaches that this thesis aims to address. It explores the characteristics of climbing robot designs, focusing particularly on structure-climbing robots and their path-planning approaches. It also examines the spectrum of redundancy resolution approaches, from local velocity-based methods to global position-based techniques, highlighting their respective strengths and limitations.
- **Chapter 3: Path Planning on Three-dimensional Structures** introduces a hierarchical path planning algorithm for step-by-step biped robots. The algorithm is divided into two levels: the Transition Footholds Planner (TFP) that finds a global path by establishing a sequence of transition footholds, and the Face Footholds Planner (FFP) that solves the simpler two-dimensional problems of navigating each face in the structure. The latter level is once again divided into two subproblems: the Continuous Face Footholds Planner (CFFP) that finds a continuous path on the face, and the Discrete Face Footholds Planner (DFFP) that finds the final discrete footholds on the face. This chapter details the mathematical foundation of the approach, including structure representation, foothold definition, and transition types. It also presents simulation results demonstrating the algorithm's effectiveness in complex truss environments.
- **Chapter 4: Feasibility Maps for Redundancy Resolution and Task Trajectory Generation** presents the concept of Feasibility Maps (FMs) and how they can be used to solve the redundancy resolution problem. It develops an RRT-based algorithm for efficiently exploring these maps to generate near-optimal joint-space trajectories. The chapter then introduces Augmented Feasibility Maps (AFMs) as an extension of FMs to simultaneously solve the redundancy resolution and task trajectory generation problems. This novel approach eliminates the need for predefined task trajectories

whose feasibility may not be guaranteed, instead generating trajectories that are feasible by construction.

- **Chapter 5: Self-motion Manifolds for Global Redundancy Resolution** introduces the concepts of Self-motion Manifolds (SMMs) and Self-motion Domains (SMDs), and proposes a framework for global redundancy resolution based on the analysis of the SMMs that compose the SMD. This approach overcomes limitations of local methods by providing a global description of the solution space. The chapter details algorithms for computing SMMs and SMDs, methods for generating and analyzing the transformation graph between SMMs, and techniques for efficiently generating and optimizing joint trajectories based on this framework.
- **Chapter 6: Homotopy-based Computation of Self-motion Manifolds** develops a novel method for computing SMMs based on homotopy theory. This approach bridges the gap between sampling-based methods (which require analytical inverse kinematics solutions) and continuation methods (which struggle with computational efficiency and finding disconnected manifolds). The chapter first recaps the sampling-based computation of SMMs, then introduces the concept of homotopy and its application to the inverse kinematics problem. It presents the developed homotopy-based method, which combines homotopy sampling with efficient Newton-based propagation, and provides experimental validation across multiple examples.
- **Chapter 7: Conclusions** summarizes the main contributions of the thesis, discusses limitations of the proposed approaches, and outlines directions for future research.

Each chapter builds upon the concepts introduced in previous chapters, forming a cohesive narrative that addresses the challenge of making structure-climbing robots more practical and viable for real-world applications.

## 1.6 Summary of Materials, Methods and Discussion of Results

This thesis combines theoretical developments with practical implementations and experimental validations.

### 1.6.1 *Materials*

- **Simulation Environments:** The algorithms are validated through extensive simulations in various environments, particularly focusing on truss-like structures that resemble real-world applications for path planning of

climbing robots, and complex environments with obstacles for redundancy resolution approaches. These simulations test the algorithms under different conditions, including various structural complexities, obstacle configurations, and kinematic constraints. The simulations have been mainly performed in Python, using the following libraries:

- **NumPy:** A fundamental package for scientific computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. This library has been the foundation for implementing the algorithms and performing numerical computations throughout the thesis, including the vectorization of the algorithms.
- **SciPy:** A library that builds on NumPy and provides additional functionality for optimization, integration, interpolation, eigenvalue problems, and other advanced mathematical computations. This library has been used for nearest-neighbor searches, interopolation, and optimization tasks in the algorithms, among other tasks.
- **Flexible Collision Library (FCL):** A library for collision detection and proximity queries in robotics. It has been used for collision detection tasks in every algorithm. We started using the Python bindings of FCL, but later switched to the extension of this library called **HPP-FCL**, which provides a more efficient implementation of the collision detection algorithms. More recently, HPP-FCL has been renamed to **Coal**.
- **Matplotlib:** A plotting library for Python that enables the creation of static, animated, and interactive visualizations in Python. It has been used for visualizing and representing the results of the algorithms, including 2D and 3D plots of the robot's trajectories, footholds, and environment representations.
- **PyTorch:** A deep learning framework that provides a flexible platform for building and training neural networks. It has been used for implementing the neural network-based approaches in the algorithms, particularly for the reachability test in the path planning algorithm.
- **SciKit-learn:** A machine learning library for Python that provides simple and efficient tools for data analysis and modeling. It has been used for implementing machine learning algorithms, particularly for the clustering tasks in the algorithms.
- **NetworkX:** A library for the creation, manipulation, and study of complex networks. It has been used for manipulating graphs in the different algorithms.

  - **QPSolvers:** A library for solving quadratic programming problems in Python. It has been used for optimization tasks in the algorithms, particularly in the redundancy resolution approaches. It provides a unified interface to various QP solvers, allowing for easy integration and experimentation with different optimization methods.
  - **Shapely:** A library for 2D and 3D geometric objects and operations. It has been used for performing geometric operations, such as intersection, union, and difference, on the shapes representing the robot and the environment. This library has been particularly useful for handling geometric representations of the different faces of the truss structures.

- **HyReCRo Robot Platform:** The HyReCRo robot serves as the primary test for the developed methods, providing a realistic platform with hybrid serial-parallel architecture and kinematic redundancy. The robot's ten degrees of freedom and ability to perform both convex and concave transitions make it an ideal platform for validating the proposed algorithms.

### 1.6.2 *Methods*

- **Theoretical Frameworks:** This thesis develops mathematical frameworks for path planning in three-dimensional spaces, redundancy resolution using Feasibility Maps, and computation of Self-motion Manifolds using homotopy methods. These frameworks incorporate concepts from robotics, differential geometry, numerical methods, and optimization theory, among other fields. Some of the key theoretical concepts include:
  - **Continuation Methods:** A numerical technique for tracing manifolds starting from a known solution. This method is used to compute SMMs in the context of redundancy resolution.
  - **Sampling Methods:** A SMM computation method that samples the redundant configuration space and solves the inverse kinematics problem to identify the manifold components.
  - **Homotopy Theory:** A branch of topology that studies the properties of spaces that are preserved under continuous transformations. This theory is used to develop a novel method for computing SMMs without requiring closed-form solutions.
  - k**-d Trees:** A data structure used for organizing points in a k-dimensional space. This structure is used for nearest-neighbor searches and efficient exploration of the configuration space in several algorithms of the present thesis.

- **DBSCAN:** A density-based clustering algorithm used for identifying clusters in spatial data. This algorithm is used for clustering the sampled points in the configuration space to identify disjoint SMMs.
- **Quadratic Programming:** A mathematical optimization problem where the objective function is quadratic and the constraints are linear. This method is used for solving optimization problems in redundancy resolution, particularly in the context of optimizing joint trajectories while satisfying kinematic constraints.
- **Neural Networks:** A computational model inspired by the human brain, used for regression and classification tasks. In this thesis, neural networks are used for approximating the reachability test in the path planning algorithm, allowing for efficient evaluation of the robot's ability to reach specific footholds.
- **Pathfinding Algorithms:** Algorithms used for finding paths in graphs or grids. In this thesis, pathfinding algorithms such as A* and RRT are used for exploring the configuration space and finding optimal paths in the context of path planning and redundancy resolution.
- **Newton's Method:** A numerical method for finding successively better approximations to the roots (or zeroes) of a real-valued function. This method is used in several of the proposed algorithms for solving nonlinear equations.

- **Algorithmic Developments:** Several algorithms are developed and implemented throughout this work. The main algorithms, which encapsulate other sub-algorithms within them, are:
  - A hierarchical path planning algorithm that separates the complex problem into manageable subproblems: transition footholds planning and face footholds planning
  - RRT-based methods for exploring Feasibility Maps and Augmented Feasibility Maps
  - A framework for global redundancy resolution through Self-motion Manifold analysis
  - Novel homotopy-based approaches for computing Self-motion Manifolds that overcome limitations of existing methods
- **Parallelization and Vectorization:** We have implemented parallelization and vectorization techniques to enhance the computational efficiency of the algorithms. This includes using Python's multiprocessing library for

parallel operations on different CPU cores, and NumPy's vectorized operations for efficient array manipulations. These techniques significantly reduce the computational time required for the algorithms, as demonstrated in the results.

- **Experimental Validation:** The algorithms are validated through extensive simulations and experimental tests on the HyReCRo robot. The results demonstrate the effectiveness and robustness of the proposed methods in real-world scenarios, showcasing their potential for practical applications in industrial inspection tasks.
- **Comparative Analysis:** The performance of the proposed algorithms is compared with existing methods in the literature, when applicable. This includes comparisons of computational efficiency, path quality, and feasibility of generated trajectories. The results highlight the advantages of the proposed methods in terms of both performance and applicability to real-world scenarios.

### 1.6.3 *Results and Discussion*

The main results and contributions of this thesis include:

- **Efficient Path Planning on Three-dimensional Structures:** The hierarchical approach to path planning demonstrates significant computational efficiency and path quality. By dividing the problem into global transition planning and local face navigation, the algorithm reduces the computational complexity while maintaining the ability to find optimal paths through complex structures.
- **Feasibility Maps for Redundancy Resolution:** The proposed approach based on Feasibility Maps provide an effective framework for solving the redundancy resolution problem, enabling the generation of feasible joint trajectories for redundant robots following prescribed task trajectories. The extension to Augmented Feasibility Maps further enhances this approach by simultaneously addressing task trajectory generation, ensuring that generated paths are inherently feasible.
- **Global Redundancy Resolution Framework:** The framework based on Self-motion Manifolds and Self-motion Domains provides global solutions that overcome limitations of local methods, ensuring feasibility throughout the entire task trajectory, and the global optimality of the solution. This approach offers a global exploration of the solution space, enabling the identification of optimal transitions between different configurations while maintaining task constraints.

- **Novel Homotopy-based Method:** The developed homotopy-based method for computing Self-motion Manifolds addresses a critical gap in the literature, offering a general approach that does not require closed-form kinematic solutions while efficiently identifying all disjoint manifold components. This method demonstrates significantly better performance than traditional continuation methods, particularly as the number of degrees of freedom increases.

Through these contributions, this thesis advances the state of the art in planning and control of structure-climbing robots, bringing them closer to practical deployment in industrial inspection and maintenance tasks. The developed methods not only solve specific problems related to the HyReCRo robot but also provide general frameworks that can be applied to a wide range of redundant robotic systems, extending their impact beyond the specific application domain of structure-climbing robots.

# 2 LITERATURE REVIEW

This chapter presents a comprehensive review of the scientific literature relative to the research in this thesis. The review establishes both the broader context of climbing robots, and then delves into the specific topics of the thesis, from path planning, to redundancy resolution.

We begin by examining the field of climbing robots, their applications, and classification schemes, with particular focus on structure-climbing robots: the category to which our subject robot, HyReCRo, belongs.

Following this contextual foundation, we explore the current state of path planning techniques for three-dimensional climbing robots, highlighting the unique requirements of step-by-step climbing robots compared to their continuous-motion counterparts, to introduce the first main contribution of this thesis: the development of a path planning algorithm specifically designed for step-by-step climbing robots.

The chapter then delves into the theoretical core of our second group of contributions: redundancy resolution and the computation of self-motion manifolds. We present a spectrum of approaches ranging from local, velocity-based methods to global, position-based techniques, analysing their respective strengths and limitations. This review provides the basis for our development of new methods presented in later chapters, particularly our homotopy-based approach to computing self-motion manifolds and our framework for global redundancy resolution.

## 2.1 CLIMBING ROBOTS

The field of climbing robots has developed significantly over the past three decades through extensive worldwide research. These robots primarily serve to perform hazardous tasks at height that would otherwise endanger human workers through risks of falling, electrocution, or operation in difficult-to-access locations.

In the literature, climbing robots have been proposed for a variety of applications, including:

- Inspection of ship hulls (Huang et al., 2017)
- Inspection of metal bridges (Nguyen and La, 2021)
- Calibration of metal tanks (Chen et al., 2019)

- Crack evaluation (Jang et al., 2020)
- Structural health monitoring (Li et al., 2017)
- Decommissioning of nuclear reactors (Hirai et al., 2013)

According to Schmidt and Berns (2013), climbing robot design typically addresses two fundamental aspects:

- The *adhesion system* that secures the robot to surfaces, using technologies such as magnetic attachment, pneumatic suction, mechanical grippers, electrostatic adhesion, or chemical adhesives
- The *locomotion mechanism* that enables movement across the environment, including arms/legs, wheels/tracks, sliding frames, cables, or rail systems

While Schmidt and Berns (2013)'s classification focuses on adhesion and locomotion methods, we adopt an alternative classification based on the environmental geometry to better contextualize the robot examined in this thesis. Under this framework, climbing robots generally fall into four categories:

- Wall-climbing robots
- Structure-climbing robots
- Cable-climbing robots
- Rough-terrain traversal robots

This thesis focuses specifically on structure-climbing robots, the category to which our research subject belongs.

### 2.1.1 *Structure-climbing Robots*

Structure-climbing robots operate within three-dimensional frameworks of interconnected beams and structural nodes. A defining characteristic is that the robot's dimensions are comparable to the width of the beams it traverses: when beam width substantially exceeds robot size, the environment effectively becomes a wall-climbing scenario, falling outside the scope of this thesis.

These robots serve numerous inspection and maintenance functions in built environments:

- Identifying structural defects like corrosion or coating damage in building frames and bridges (Balaguer et al., 2000; Pagano et al., 2012)
- Performing non-destructive testing on industrial pipeline networks (Tavakoli et al., 2011)

- Maintaining infrastructure like street lights and navigating complex junctions (Guan et al., 2011; Tavakoli et al., 2005)
- Participating directly in construction processes (Nigl et al., 2013)

Unlike the highly diverse architectures of wall-climbing robots, structure climbers typically follow more standardized designs, falling into two principal categories (Tavakoli et al., 2011):

- **Continuous-motion robots:** These wheel-based systems move smoothly along structures, offering simplicity and speed but limited obstacle-negotiation capabilities. Some examples can be found in (Chen et al., 2019; Huang et al., 2017; Li et al., 2017; Nguyen and La, 2021).
- **Step-by-step robots:** These inchworm-like robots use two grippers connected by a multi-DOF manipulator. While one gripper anchors to the structure, the other (functioning as an end effector) positions itself at a new location. The grippers then exchange roles to advance. Though slower and more complex than continuous-motion systems, they offer superior manoeuvrability and obstacle-handling capability. Examples include ROMA 2 (Gimenez et al., 2002), Climbot (Guan et al., 2013), 3DCLIMBER (Tavakoli et al., 2011), TREMO (Rochat et al., 2011), CROC (Yang et al., 2016) and HyReCRo (Peidró et al., 2019).

Hybrid designs incorporating both wheeled locomotion and articulated limbs also exist, combining continuous movement efficiency with enhanced manoeuvrability (Chung and Xu, 2011; Viegas and Tavakoli, 2014).

The design of step-by-step robots centres on their grippers and connecting manipulator, which may employ serial, parallel, or hybrid architectures. Serial examples include ROMA 2 (Gimenez et al., 2002), 3DCLIMBER (Tavakoli et al., 2011), and Climbot (Guan et al., 2013), while parallel designs often utilize Stewart platforms (Saltaren et al., 2005). Hybrid architectures (Figliolini et al., 2010; Tavakoli et al., 2005) are particularly well-suited for structure climbing as they combine the extensive workspace of serial manipulators (essential for obstacle negotiation) with the favourable payload-to-weight ratio of parallel mechanisms (crucial for self-supporting climbing robots).

For adhesion, structure-climbing robots predominantly use either magnetic systems (for ferromagnetic structures) (Pagano et al., 2012; Rochat et al., 2011; Shvalb et al., 2013; Ward, 2016) or mechanical grippers that grasp the relatively narrow beam dimensions (Balaguer et al., 2000; Saltaren et al., 2005; Tavakoli et al., 2011, 2005). Suction cups appear occasionally in certain applications (Gimenez et al., 2002).

Typical obstacles in structural environments include beam junctions, multibeam nodes, section changes, valves, and transitions between planes.

Continuous-motion robots generally struggle with these features unless equipped with articulated arms, while step-by-step climbers navigate them more effectively through their inherent manipulator manoeuvrability.

### 2.1.2 *The HyReCRo Robot*

The HyReCRo robot, which many chapters of this thesis validate their proposed method on, belongs to the step-by-step structure-climbing category. Originally proposed by Úbeda et al. (2012), HyReCRo is designed to explore three-dimensional steel frameworks, such as those found in buildings and bridges.

It features a bipedal design, with each *leg* consisting of two parallel 2RPR-PR mechanisms in series. Figure 2.1 shows the robot's architecture. Each parallel mechanism is actuated by two linear actuators $l_{ij}$ and $r_{ij}$, which connect the *platform* to the *base* of each mechanism. Subscript $i$ refers to the mechanism index $i = \{1, 2\}$, while subscript $j$ refers to the leg identifier $j = \{A, B\}$. The two mechanisms that comprise leg $j$ are connected to a common base, which we denote as *core link* of leg $j$. Platform of mechanism $1j$ is connected to the *gripper* of leg $j$, while platform of mechanism $2j$ is connected to the *hip* of the robot by means of a revolute joint $\theta_j$. This configuration yields a total of ten DoFs for the robot, which renders it kinematically redundant.

In the original work (Úbeda et al., 2012) the robot was proposed with binary actuation, where the actuators operate only at their extreme positions (e.g., fully retracted or extended). This design choice simplifies control and motion planning, addressing a significant challenge in deploying step-by-step climbers beyond laboratory environments. However, purely binary actuation limits the robot to discrete postures, hindering essential movements in 3D structures. Specific transitions, such as navigating convex corners between faces or concave junctions between beams, require precise gripper positioning that may not coincide with the limited set of binary-actuated configurations.

To overcome this limitation, the HyReCRo robot was later modified to incorporate continuous actuation in its legs (Peidró et al., 2016a), allowing the actuators to operate at any position within their limits. From then on, multiple studies have been conducted to explore the robot's capabilities, including:

- Peidró et al. (2015) analysed the inverse kinematics of the robot, proposing a method to compute the inverse kinematics of the robot using a closed-form solution, which will be helpful for the redundancy resolution techniques presented in Chapters 4 and 5.
- Peidró et al. (2016b) analysed and simulated the planar and symmetric postures of the robot. (Peidró et al., 2016a) proposed a Monte Carlo method to compute the robot's workspace, which is the set of all reachable configura-

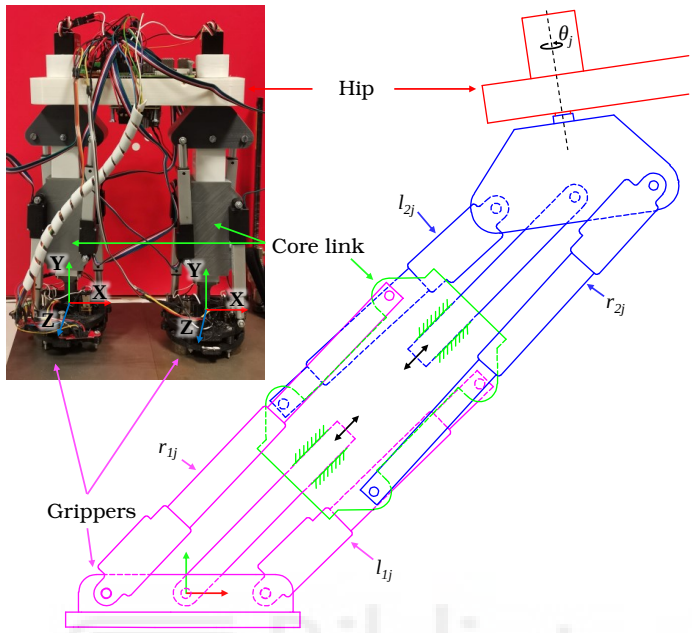


Figure 2.1: HyReCRo robot architecture. Representation of leg j.

tions of the robot. Both of these works will be useful for the path planning algorithm presented in Chapter 3.

- Peidró et al. (2016c) computed the internal barriers and boundaries of the HyReCRo robot's workspace. This would later inspire the sampling-based approach to self-motion manifold computation developed in (Peidro et al., 2018). The method presented in Chapter 5 of this thesis builds upon the latter paper while introducing significant improvements in computational efficiency.

In (Peidró et al., 2019), the authors presented the design of compact switchable magnetic grippers for the HyReCRo robot. The grippers are based on switchable permanent magnets, which can be turned on and off by means of a circuit of DC motors that rotate the magnets. An analysis of the adhesion force of the grippers was performed, and additional accesories that increase the friction coefficient of the grippers were presented.

Later, Fabregat-Jaén et al. (2022) developed an automated gripper attachment algorithm for the HyReCRo robot that, starting from an approximate alignment with the target surface, employs distance sensor feedback to kinematically control the approaching to the surface of adherence, perform fine position adjust-

ments, and securely activate the magnetic grippers by ensuring that the gripper and surface are parallel and in contact, significantly enhancing the robot's autonomous climbing capabilities in real-world environments.

## 2.2 PATH PLANNING FOR THREE-DIMENSIONAL CLIMBING ROBOTS

In order to perform tasks in a three-dimensional structure, climbing robots must be able to autonomously navigate it and reach the desired position. The *path planning* problem consists in finding a valid sequence $\mathcal{P}$ of configurations that moves the robot from a starting point $\mathbf{T}_0$ to a target point $\mathbf{T}_f$. Not only must the path reach the target, but it must also consider obstacles, transitions between planes, and the robot's kinematic constraints.

Conventional path planning solutions, such as those suggested for cars, robotic vacuum cleaners or rovers, commonly involve a search space of two dimensions. Nevertheless, climbing robots require a 3D search space (even more dimensions when taking into account orientation or the state of their grippers), which remarkably increases the computation and complexity of the problem.

Whereas compelling solutions have been proposed in the literature for path planning of continuous-motion climbing robots (Breitenmoser and Siegwart, 2012; Fang et al., 2019; Stumm et al., 2012; Yue et al., 2010), path planning of step-by-step climbing robots is still an open subject. They demand a sequence of discrete attachment points rather than a continuous path, resulting in the previous methods being unsuitable.

Few authors have proposed solutions to the path planning problem of step-by-step climbing robots. In (Yang et al., 2016), a path is obtained using the Levenberg-Marquardt algorithm to solve a non-linear system comprised of objective sigmoid functions that accomplish collision avoidance, joint limits checking and control of the position and orientation of the end effector.

Zhu et al. (2021) present a multi-staged algorithm. First, the transition points between planes of the structure are obtained optimizing a non-linear system similarly to (Yang et al., 2016). Then, the path on each plane is calculated by determining the best next foothold based on its distance to the goal and a pre-calculated continuous path using a version of the A* algorithm. Finally, the path planning of the robot as a manipulator is solved for each step of the obtained path.

Chen et al. (2016) propose a method that models the environment using voxels and employ a multi-layered algorithm based on the A* to solve the problem. In the multi-layered algorithm, up to four sequential layers are used in case that the previous layer fails or falls into a local optimum.

The A* algorithm is also applied in (Quin et al., 2016). The proposed solution builds a graph of valid attachment points and determines a heuristic function

that evaluates a point in terms of the information of the environment that the robot can observe through its sensors.

In a similar way, Pagano and Liu (2016) obtain a solution by constructing a tree based on the line of sight of the robot (observable environment). Nevertheless, tuning is needed to obtain an optimal solution and close obstacles make it hard to find a path between them.

## 2.3 REDUNDANCY RESOLUTION

Kinematic redundancy occurs when a robot possesses more DoFs than those required to perform a given task. This leads to an infinite number of possible solutions to the Inverse Kinematics Problem (IKP) for a specific task position. *Redundancy resolution* is the process of selecting a solution from this infinite set.

When controlling a redundant robot, like the HyReCRo robot, to execute a predefined task trajectory $\mathbf{x}(t)$, it becomes necessary to determine a continuous joint-space trajectory $\mathbf{q}(t)$ that not only satisfies the primary task constraints (like following the desired end-effector trajectory) but also utilizes the redundant DoFs to fulfil additional objectives such as satisfying joint limits (Siciliano et al., 2009) or avoiding collisions (Khatib, 1986).

The literature contains numerous approaches to redundancy resolution, which can be categorized on a spectrum ranging from local to global methods. Local methods consider only the immediate vicinity of the current configuration, while global methods characterize the entire solution space. Figure 2.2 illustrates this spectrum, with local methods on the left-hand side of the axis and global methods on the right.

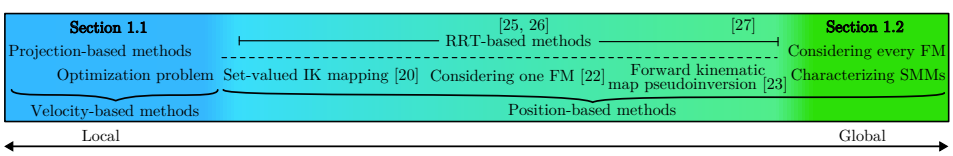


Figure 2.2: Spectrum of redundancy resolution methods, from local to global.

### 2.3.1 *Local Redundancy Resolution*

*Local redundancy resolution* techniques typically approach the problem using the velocity differential equation $\dot{\mathbf{x}} = \mathbf{J}\,\dot{\mathbf{q}}$, where $\mathbf{J}(\mathbf{q})$ is the configuration-dependent Jacobian matrix mapping joint velocities $\dot{\mathbf{q}}$ to task velocities $\dot{\mathbf{x}}$. These methods are considered local because the Jacobian only provides information about the system's behavior in the vicinity of the current configuration.

These methods can be categorized into two main approaches: *projection-based* and *optimization-based* methods.

#### 2.3.1.1 *Projection-based Methods*

Note that the Jacobian matrix is non-square for redundant robots, as it has more columns than rows, leading to an infinite number of solutions for the joint velocities $\dot{\mathbf{q}}$ that satisfy the task velocity $\dot{\mathbf{x}}$. Projection-based methods resolve redundancy by utilizing the pseudoinverse of the Jacobian matrix: $\dot{\mathbf{q}} = \mathbf{J}^{\dagger}\,\dot{\mathbf{x}}$ (Whitney, 1969), where $(\cdot)^{\dagger}$ denotes the Moore-Penrose pseudoinverse. An additional null-space projection term is often included to accomplish secondary objectives without affecting the primary task:

$$\dot{\mathbf{q}} = \mathbf{J}^{\dagger}\,\dot{\mathbf{x}} + \mathbf{N}(\mathbf{q})\,\dot{\mathbf{q}}_n \tag{2.1}$$

where $\mathbf{N}(\mathbf{q})$ is the null-space projector and $\dot{\mathbf{q}}_n$ are free joint velocities projected onto the null-space by $\mathbf{N}(\mathbf{q})$.

Adding the additional null-space projection was first introduced by Liegeois et al. (1977). Since then, many works have built upon this idea. Slotine and Siciliano (1991) proposed a framework that incorporates multiple hierarchically ordered secondary tasks, using the null-space projection to prioritize the primary task while satisfying secondary tasks in a sequential manner. In (Flacco et al., 2015), the authors extended this approach to simultaneously resolve multiple tasks by progressively employing the residual DoFs that remain after each task is resolved in what they named the SNS method.

#### 2.3.1.2 *Optimization-based Methods*

Optimization-based methods formulate redundancy resolution as an optimization problem with an objective function quantifying solution quality. Rather than pseudoinverting the Jacobian, these methods use the velocity differential equation $\dot{\mathbf{x}} = \mathbf{J}\,\dot{\mathbf{q}}$ as an equality constraint in the optimization problem, avoiding pseudoinversion-related issues such as singularities.

Optimization problems can be defined and solved in various ways, such as sequental quadratic programming (SQP) (Zanchettin and Rocco, 2017), model predictive control (MPC) (Faroni et al., 2018), or nonlinear optimization variants (Mavrommati et al., 2021), among others.

Some researchers have proposed adding random components to the optimization process in order to address the limitations of local methods. For instance, Ratliff et al. (2009) and Kalakrishnan et al. (2011) introduced probabilistic methods with random components to improve the exploration of the solution space and avoid local minima. However, some of the issues we will discuss in the next section remain.

#### 2.3.1.3 *Limitations of Local Methods*

Despite their computational efficiency and flexibility, local redundancy resolution techniques face several challenges. Albu-Schäffer and Sachtler (2023) highlighted these limitations for projection-based methods, some of which could be extended to every local redundancy resolution method:

- They typically produce non-cyclic solutions for periodic tasks, which is problematic in industrial settings requiring repeatability,
- They may fail to find solutions even when they exist, due to the limited scope of the Jacobian, which may lead the algorithm to empty solution sets near kinematic constraints or singularities, which materialize as external or internal barriers of the workspace (Peidro et al., 2018).
- They may converge to locally-optimal solutions rather than globally optimal ones, as they cannot explore the entire solution space.

### 2.3.2 *Global Redundancy Resolution*

Global redundancy resolution methods are position-based (not operating at velocity-level) approaches that address the limitations of local techniques by characterizing the entire solution space. We have considered the global scope of a method is determined by its ability to identify all possible solutions to the IKP.

These methods typically operate by calculating the SMMs, which are subsets of the joint space where the robot can move without affecting the task value (Burdick, 1989). SMMs include all possible solutions to the IKP for a given task. A comprehensive literature review on computation of SMMs is provided in Section 2.4.

The applications of SMMs to redundancy resolution are numerous and diverse. For instance, Lück (1997) proposed a method that constructs a graph of nodes representing sets of SMMs, and edges representing the transitions between these sets. These sets of SMMs are actually an important concept in the context of our work presented in Chapter 5, and will be thoroughly discussed in said chapter. The information of the graph is then discretized, both in the joint space and in the task space, resulting in a second connectivity graph. This second graph is finally searched using an A* algorithm to find the optimal joint-space trajectory that satisfies the task trajectory. However, the algorithm yields a discrete trajectory where each point corresponds to the centroid of a discretized cell in the joint space. While smoothness can be achieved by interpolating between these points, it will result in a non-natural trajectory with high accelerations peaks.

Zhou et al. (2023) introduced a Rapidly-exploring Random Tree (RRT)-based path planning method that simultaneously solves the path planning and redundancy resolution problems. The method constructs an RRT by sampling points in the task space and utilizing SMMs to evaluate the continuity between sampled task-space points. Due to this uncertain continuity, the construction of the RRT may need an significantly small step size in order to not miss the connection between two sampled points, which can lead to a very large number of samples and a long computation time. After the tree is built, the algorithm connects the sampled SMMs to derive the joint trajectory based on an exhaustive search that avaluates a given metric (e.g., reciprocal condition number) to select the best point in every SMM.

Another work that solves the redundancy by means of SMMs is the one by Albu-Schäffer and Sachtler (2023), who proposed a formal strategy for redundancy resolution at position level, by using foliations orthogonal to SMMs to establish a global alternative to null-space projection. They propose a coordinate-growing approach to compute the foliations. However, this approach is limited to one-dimensional tasks, and is stated as rather expensive in terms of computation time. For higher-dimensional tasks, the authors suggest training a neural network to approximate the foliations, which may lead to a loss of mathematical properties, such as the loss of strict orthogonality, and the need for extensive training data and time.

In addition, global redundancy resolution may be approached by other tools, such as FM (Wenger, 1992). FMs are trajectory-wise representations of the infinite set of solutions to the IKP for a given task trajectory, rather than point-wise (for a given task value) representations like SMMs. FMs are defined as the set of points $(t, \mathbf{q}_r)$, that satisfy the robot's kinematic equations and imposed constraints, where $t$ is the variable that parameterizes the trajectory and $\mathbf{q}_r$ is a selection of redundant coordinates. For every point $(t, \mathbf{q}_r)$ in the FM space, the joint coordinates $\mathbf{q}$ that satisfy the given task value $\mathbf{x}(t)$ are solved and checked for feasibility to determine if the point is part of the FM. Every solution $\mathbf{q}$ for a given pair $(t, \mathbf{q}_r)$ lies on an *extended aspect* (Pámanes G et al., 2002), and the number of FMs is equal to the number of extended aspects that exist for the robot, as each FM is associated with a different extended aspect. Given that all FMs are considered, these methods can be classified as global redundancy resolution tools. Under this approach, Ferrentino and Chiacchio (2020) developed a method that explores every FM using dynamic programming to find optimal joint-space trajectories $\mathbf{q}(t)$ that satisfy the task trajectory $\mathbf{x}(t)$.

These methods successfully solve the global redundancy resolution problem, but they exhibit some limitations:

- In general, they are computationally expensive, especially for high-dimensional robots. Chapter 5 will compare the computation time of some of these methods.
- Except for the method presented by Albu-Schäffer and Sachtler (2023), none of them present examples with degrees of redundancy higher than one.

In Chapter 5, we will present a framework for global redundancy resolution that addresses the limitations of existing methods by providing a computationally efficient solution for high-dimensional robots with multiple degrees of redundancy.

### 2.3.3 *In-between Approaches*

Between purely local velocity-based methods and fully global methodss, a range of position-based methods offers intermediate coverage of the solution space. These techniques have a broader scope than local methods, but do not consider the whole space of solutions like global methods.

Haug (2024) introduces a set-valued inverse kinematics mapping in which the $n$ joint variables $\mathbf{q}$ are expressed as functions of the $m$ task variables $\mathbf{x}$ and $r = n - m$ self-motion parameters $\mathbf{v}$. Around a given configuration $\bar{\mathbf{q}}$, a basis $\mathbf{V}$ of the null-space of $\mathbf{J}(\bar{\mathbf{q}})$ is extracted to define a local parameterization of nearby solutions. Although this mapping is initially valid only in a neighbourhood of $\bar{\mathbf{q}}$, it can be maintained along a trajectory by updating $\mathbf{V}$ whenever conditioning deteriorates. In many manipulators the neighbourhood of validity is surprisingly large, requiring few or no reparameterizations. This position-level approach remedies issues of velocity-based pseudoinverse methods, such as non-cyclicity, and enforces more naturally constraints defined in configuration space, like obstacle avoidance (Peidro and Haug, 2023).

Fabregat-Jaen et al. (2023) propose an RRT-based planner that samples directly on a FM associated with the task. By growing a tree within that FM (rather than in joint or task space), the method solves redundancy on the chosen extended aspect. However, restricting exploration to one FM prevents transitions to other branches of the inverse solution, and potentially better solutions may be missed.

Hauser and Emmons (2018) combine task-space and self-motion manifold sampling in a probabilistic roadmap. They concurrently sample pairs $(\mathbf{x}, \mathbf{q})$ satisfying the kinematic constraints, then apply a heuristic three-stage local search to define a unique pseudoinverse function that maps each $\mathbf{x}$ to a single $\mathbf{q}$. Because only one inverse solution per task point is retained, potentially better alternatives may be missed. Although this method appears near the global end of the spectrum (Fig. 2.2), it is not grouped with fully global techniques in Section 2.3.2

since it virtually removes redundancy by retaining only one solution per task point.

More broadly, sampling-based path planners, including numerous RRT variants, span a range of scopes depending on how the constraint manifold is explored. Berenson et al. (2009) project each sampled configuration onto the full constraint manifold $\{(\mathbf{x}, \mathbf{q}) \mid \mathbf{F}(\mathbf{x}, \mathbf{q}) = 0\}$, enabling direct incorporation of position-level constraints. Jaillet and Porta (2012) compute the manifold via an atlas that is built and coordinated alongside an RRT expansion. AtlasRRT* (Jaillet and Porta, 2013) further extends this idea with an asymptotically optimal planner that refines manifold coverage as the tree grows. Because different RRT-based algorithms employ distinct sampling and connectivity strategies, they collectively occupy a broad range of our local-global spectrum rather than a single point.

## 2.4 COMPUTATION OF SELF-MOTION MANIFOLDS

SMMs are central to global redundancy resolution, representing the set of joint configurations that produce the same task-space position. Computing these manifolds is challenging, particularly for robots and tasks with high degrees of redundancy. Various approaches have been developed to tackle this problem.

### 2.4.1 *Continuation methods*

Continuation methods trace the SMM by following paths along its surface, starting from an initial point on the manifold. These methods rely on the tangent space of the SMM at each configuration to guide the exploration.

The general procedure is illustrated in Figure 2.3, and involves:

1. Computing (if not provided) an initial configuration $\mathbf{q}_0$ that lies on the SMM
2. Finding the tangent space (a null-space basis of the Jacobian) $\mathbf{NS}_0$ at that configuration $\mathbf{q}_0$
3. Taking a step $\rho$ along the tangent space
4. Projecting back onto the SMM (often using Newton's method) to obtain a new configuration $\mathbf{q}_1$
5. Repeating the process until the entire manifold is traced

The original method was proposed by Burdick (1989). However, it was limited to one-dimensional SMMs. Henderson (2002) extended this approach to system-

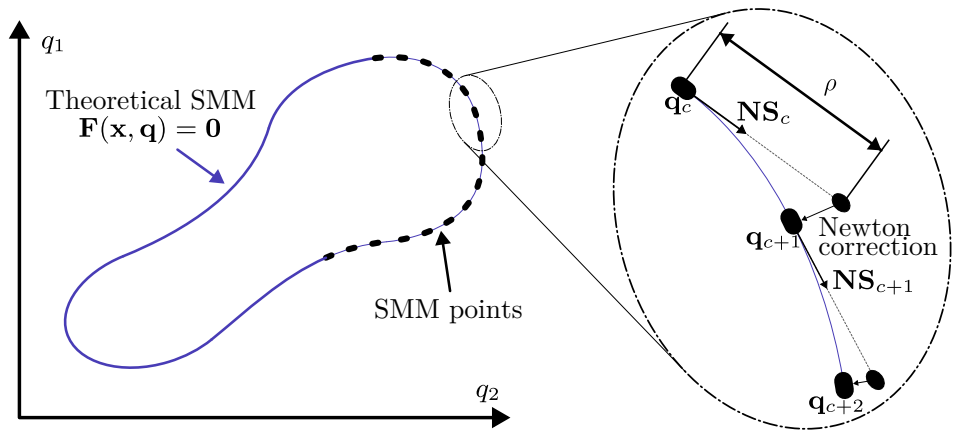


Figure 2.3: Continuation method for tracing a self-motion manifold.

atically explore higher-dimensional manifolds using an atlas of local parameterizations. The method involves creating a collection of charts that together cover the manifold, with each chart providing a local coordinate system.

While continuation methods can effectively trace SMMs, they face challenges with higher-dimensional manifolds, where the computational cost increases exponentially. Moreover, the main drawback of these methods is their reliance on an initial point on the manifold, since only the SMM to which the initial point belongs can be traced. To trace every disjoint SMM, the algorithm must be restarted from different initial points, one per manifold, which may not be feasible in practice, since the number of disjoint SMMs is not known a priori (Burdick, 1989).

### 2.4.2 *Sampling methods*

Sampling methods generate a discrete representation of SMMs by sampling points in the joint space and checking if they satisfy the inverse kinematics constraint. Later, the computed point cloud is clustered to identify disjoint SMMs. These methods are particularly useful for higher-dimensional SMMs where continuation methods become inefficient.

DeMers and Kreutz-Delgado (1994) proposed a sampling method that generates a point cloud of joint configurations by randomly sampling the entire joint space and checking if the sampled points satisfy the inverse kinematics constraint. Then, the joint configurations that are mapped (by forward kinematics) within a certain tolerance of the desired task-space position are retained. To identify disjoint SMMs, the method uses a Minimum Spanning Tree algorithm to

cluster the sampled point cloud. The method is computationally expensive, as it requires the dense sampling of the complete joint space, which is not efficient, as SMMs are typically sparse in the joint space because they are lower-dimensional manifolds embedded in a higher-dimensional joint space.

Peidro et al. (2018) proposed a more efficient sampling method. The method is based on the idea of sampling a subset of $n - m = r$ joint variables as parameters, and then solving the IKP for the remaining $m$ joint variables analytically, generally by means of algebraic elimination. In order to do this, the method must perform this sweeping process for every possible combination of $r$ joint variables. By sweeping through every combination of redundant joints, the method generates a densely sampled point cloud of the SMM. The method is computationally efficient, as every sampled point is guaranteed to be on the SMM, not wasting computational resources on points that do not belong to the manifold. For the clustering step, the method uses a recursive algorithm based on $k$-d tree (Bentley, 1975) nearest neighbor queries to identify disjoint SMMs. The main challenge of this method is the need to express the IKP in closed form, by means of algebraic elimination, which is not always possible for every robot and task. In Chapter 6, we will present a tool to eliminate the necessity for sweeping through *every* combination of $r$ redundant joints, enabling the use of this method by only sweeping through a single combination of redundant joints. Regarding advantages, the method is computationally efficient and can be applied to high-dimensional SMMs, as Chapters 5 and 6 will demonstrate. Additionally, the method is capable of identifying every disjoint SMM in the joint space.

In Chapter 5, we will present an optimized version of this method that further reduces the computation time. Chapter 6 will also introduce a redensification method that allows for using the method with a lower number of combinations (potentially just one) of redundant joints, while still ensuring a dense sampling of the SMM. In addition, we also present a homotopy-based method that, combined with the redensification process, does not require the IKP to be expressed in closed form, making it applicable to a wider range of robots and tasks.

### 2.4.3 *Other methods*

Beyond continuation and sampling approaches, several alternative methods have been developed to compute or approximate SMMs:

In (Zhou et al., 2023), the authors proposed a method to compute SMMs by means of the Artificial Bee Colony algorithm. Thanks to this approach, the method is able to compute SMMs without the need for computing the Jacobian matrix. However, similarly to the continuation methods, the algorithm requires an initial point on the SMM to start the search, rendering it unable to compute every disjoint SMM unless a point on each manifold is provided.

Wu et al. (2023) proposed a method to compute SMMs based on the concept of *cellular automata*. The authors of (Banfield and Rodríguez, 2022) proposed a SMM-computation approach based on multi-objective optimization. Finally, Yang et al. (2021) presented a method that computes SMMs using interval branch-and-bound techniques. These methods are not widely used in the literature, and their performance seems rather expensive.

Although not explicitly computing SMMs, Albu-Schäffer and Sachtler (2023) proposed a method that uses foliations orthogonal to SMMs to establish a global alternative to null-space projection. The authors also suggested approximating these foliations using neural networks, which can efficiently represent complex manifolds but may sacrifice strict mathematical properties and require extensive training data.

# 3 PATH PLANNING ON THREE-DIMENSIONAL STRUCTURES

This chapter presents a solution to the problem of path planning on three-dimensional vertical structures for a step-by-step two-legged (or biped) robot. Specifically, we have conducted the study using the HyReCRo robot (Section 2.1.2), but the proposed solution is general and can be applied to any biped climbing robot. The locomotion of these robots is achieved by alternately moving its two legs, which are equipped with grippers that can attach to and detach from the structure. Therefore, in order to reach a goal point on the structure, the robot must find a sequence of points (i.e., the *path*) on which to place its grippers in order to reach the goal, while avoiding collisions with the structure.

This chapter is organized as follows. Section 3.1 introduces the important concepts and definitions that will be used throughout the chapter, such as the representation of the structure, footholds, and face transitions. The main contribution of this chapter is the hierarchical path planning algorithm, which is presented in Section 3.2. This algorithm is divided into two levels: the Transition Footholds Planner (TFP) and the Face Footholds Planner (FFP). The TFP planner finds a global three-dimensional path by establishing a sequence of transition footholds that dictate the faces that the robot must traverse in order to reach the goal foothold. The FFP planner solves the simpler two-dimensional problems of navigating each face in the structure. In Section 3.3, we compare the proposed solution with the one presented in (Zhu et al., 2021), listing our contributions and improvements, while also presenting simulations and results that demonstrate the effectiveness of the proposed algorithm. Finally, Section 3.4 presents the simulations and results of the proposed algorithm, showing the robot's path through a truss structure, and the performance of the algorithm in terms of runtime and efficiency.

## 3.1 IMPORTANT CONCEPTS

### 3.1.1 *Structure Representation*

Climbing robots must be able to navigate through a variety of three-dimensional scenarios, such as truss structures, which can often be found in industrial environments, such as bridges, warehouses, or electrical distribution towers. These are highly-structured scenarios formed by interconnected beams. Each beam may be defined by the set of faces that compose it. In this work, we consider

that the beams are composed of planar faces, which is a common assumption in the literature (Cai et al., 2019; Quin et al., 2016; Zhu et al., 2021). Each face $F$ is defined by a polygon $\mathcal{F}$, which is described by a set of coplanar vertices in the three-dimensional space, and a normal vector $\mathbf{n}$ that points outwards from the structure (Zhu et al., 2021):

$$F = (\mathcal{F}, \mathbf{n}) \tag{3.1}$$

where

$$\mathcal{F} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n_v} \mid n_v \leqslant 3\} \tag{3.2}$$

and $\mathbf{v}_i$ is each of the $n_v$ vertices of the polygon. This representation is exemplified in Figure 3.1(a). Note that at least three of the vertices must not be collinear, otherwise the polygon would degenerate into a line.

Therefore, a three-dimensional truss structure can be represented as a set of faces:

$$\mathcal{S} = \{F_1, F_2, \ldots, F_{n_s}\} \tag{3.3}$$

where $n_s$ is the number of faces in the structure.

Note that every vector is represented in the global coordinate system, that is fixed with respect to the structure.

### 3.1.2 *Footholds*

Biped robots navigate through the structure by moving their grippers and placing them on different points of the structure. These points in the structure where the grippers can be placed are called *footholds*. A foothold is defined by a position and an orientation in the three-dimensional space, and can be described by a homogeneous transformation matrix **FH**:

$$\mathbf{FH} = \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

where $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ are the unit vectors of the foothold's orientation, and $\mathbf{p}$ is the position of the foothold in the global coordinate system.

It is important to note that, once a position on the structure and an orientation vector (either $\mathbf{x}$ or $\mathbf{z}$) are chosen, the foothold is uniquely defined. For instance, take any of the points illustrated in Figure 3.1(b), which are pairs of positions $\mathbf{p}$ and vectors $\mathbf{x}$, that result from the preprocessing of the structure that will be described in Section 3.2.1. This point $\mathbf{p}$ lies on some face of the structure, which is associated with a normal vector $\mathbf{n}$ (see Equation (3.1)). Therefore, since footholds

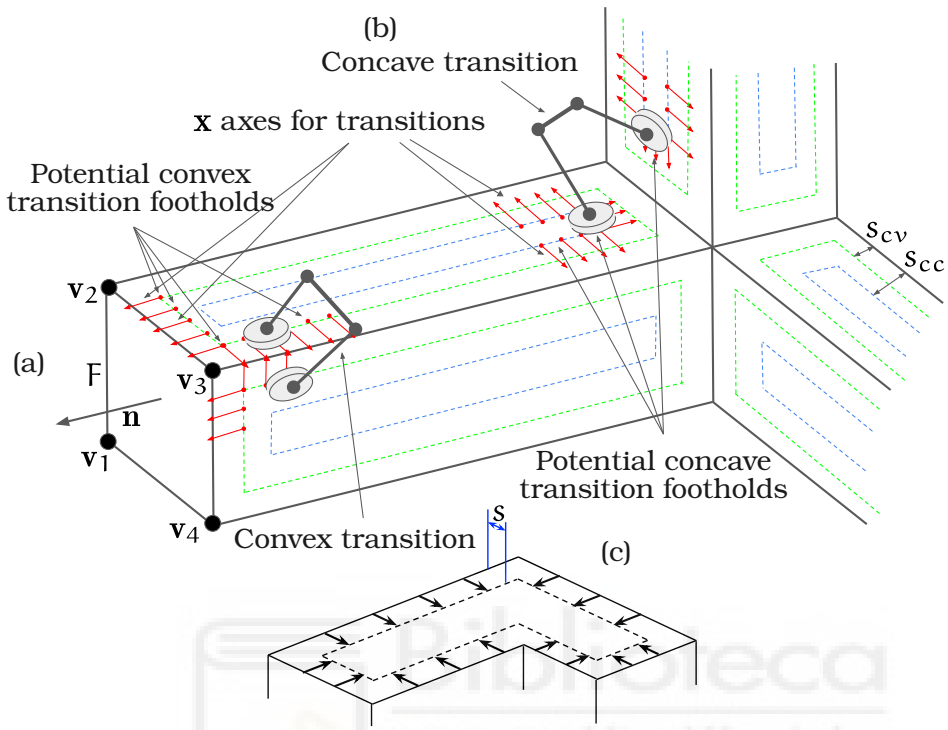


Figure 3.1: Structure representation and preprocessing. (a) Representation of a face of a truss structure. (b) Preprocessing of the structure to determine potential transition footholds. (c) Shrinking of the faces to determine potential transition footholds.

**FH** are, by definition, attached to a face, the orientation vector **y** of the foothold will be the normal vector **n** of the face. Since both **x** and **y** are known, the vector **z** is uniquely defined as the cross product of **x** and **y**. This results in a uniquely defined foothold **FH**.

Throughout this chapter, we will plot footholds by three unit arrows representing the vectors **x**, **y**, and **z** of the foothold, that will have their tails at the position **p** of the foothold. The colour of the arrows will follow the RGB convention, where **x** is red, **y** is green, and **z** is blue.

### 3.1.3 *Face Transitions*

In order to navigate the structure, it is most likely that the robot will need to move from one face to another. We will refer to this movement as a *face transition*.

Note that this movement is of paramount importance, and a study of the possible transitions is necessary in order to plan the robot's path through the structure.

Two types of face transitions must be considered: *convex* and *concave* transitions. These transitions are illustrated in Figure 3.1(b). A convex (or exterior) transition is necessary when the robot must move between adjacent faces of the same beam. Conversely, a concave (or interior) transition is necessary when the robot must move between faces of different beams.

Note that depending on the kinematics of the robot, these transitions may be very restrictive. For example, the viability of some transitions is highly sensitive to the geometric parameters of the HyReCRo robot (e.g., the lower and upper limits of the extension of its legs, the length of its hip, etc.), as studied in (Peidró et al., 2016b). Therefore, a constant-orientation workspace analysis is necessary in order to determine the feasibility of the transitions. Peidró et al. (2016a) studied the constant-orientation workspace of the HyReCRo robot, and determined the geometry of the robot that allows it to perform both types of transitions. In this work, we have employed the same methodology to determine separation distances ($s = s_{cc}$ and $s = s_{cv}$ in Figure 3.1(b)) that best allow the robot to perform these transitions. These distances are measured from the edge of the face to the foothold, and guarantee that the robot can place its grippers completely on both faces without being partially on air. These values will be used in the preprocessing of the structure, as will be described in Section 3.2.1.

## 3.2 HIERARCHICAL PATH PLANNING ALGORITHM

In this section, we propose a hierarchical path planning algorithm for step-by-step biped robots on three-dimensional truss structures. Given a start foothold $\mathbf{FH}_s$ and a goal foothold $\mathbf{FH}_g$, the algorithm finds a sequence of footholds that the robot can use to reach the goal foothold. The hierarchy of the algorithm and their different components are shown in Figure 3.2.

The upper level of the hierarchy tackles the problem by dividing it into two subproblems. First, the TFP finds a *global* three-dimensional path by establishing a sequence of *transition footholds* that dictate the faces that the robot must traverse in order to reach the goal foothold. Each traversal of a face is determined by the *arrival footholds* $\mathbf{FH}_a$ and *departure footholds* $\mathbf{FH}_d$, which are the footholds that the robot must use to enter and exit the face, respectively. Once the global transition path has been obtained, the original three-dimensional problem becomes a series of simpler two-dimensional problems, one for each traversed face, which are solved by the FFP.

The FFP planner adds a new level of hierarchy to the algorithm by, once again, dividing the problem into two subproblems. First, the Continuous Face Footholds Planner (CFFP) finds a *continuous path* on the face that connects the arrival

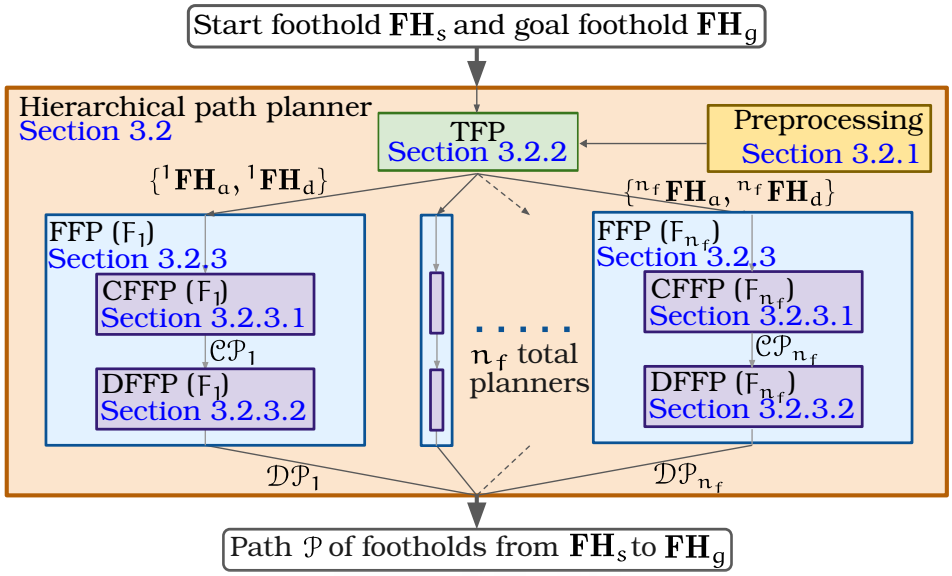


Figure 3.2: Hierarchy of the proposed path planning algorithm.

$\mathbf{FH}_a$ and departure $\mathbf{FH}_d$ footholds of the face. Then, the Discrite Face Footholds Planner (DFFP) utilizes the continuous path in order to derive a *discrete path*, which is a sequence of discrete footholds that the robot can use to reach $\mathbf{FH}_d$ from $\mathbf{FH}_a$.

The solution presented draws inspiration from the general idea of fragmenting the global three-dimensional problem into subproblems on on planar patches or faces, as outlined in (Zhu et al., 2021). However, the methods used to address the problem at each level of the planner's hierarchy differ from those employed in the referenced article. In Section 3.3, we will compare the proposed solution with the one presented in (Zhu et al., 2021), listing our contributions and improvements, while also presenting simulations and results that demonstrate the effectiveness of the proposed algorithm.

### 3.2.1 *Structure Preprocessing*

In order to effectively utilize the method we propose in this chapter, a preprocessing of the structure is necessary. Since there are infinite possible footholds from and to which the robot can perform a face transition, it is necessary to limit the number of possible footholds to a finite set. These footholds are called *potential transition footholds*.

The preprocessing of the structure is done by taking the set of faces $\mathcal{S}$, and for each face $F_i$, we shrink the polygon $\mathcal{F}_i$ by a distance $s$, as illustrated in Figure 3.1(c). This operation is performed twice, once for concave transitions with a larger distance $s = s_{cc}$ (blue polygons in Figure 3.1(b)), and once for convex transitions with a shorter distance $s = s_{cv}$ (green polygons in Figure 3.1(b)). The selection of these distances is based on the constant-orientation workspace analysis of the robot (Peidró et al., 2016a), with orientations needed to perform concave and convex transitions, and choosing distances that maximize the intersection of the workspace with the face to which the robot is transitioning.

Next, we discretize the shrunk polygons by sampling points along their perimeters with a specific resolution, where finer resolutions generate more potential footholds. These sampled points represent the positions $\mathbf{p}$ of the potential transition footholds (red points in Figure 3.1(b)). For each point, we compute the orientation vector $\mathbf{x}$ of the gripper when placed. This vector is determined as the normalized vector pointing from $\mathbf{p}$ to the nearest edge of the original polygon $\mathcal{F}_i$ (red vectors in 3.1(b)). The identified edge corresponds to the one shared by the two faces between which the transition is performed.

This preprocessing generates a set of potential transition footholds that the later planners will use to find the robot's path through the structure.

### 3.2.2 *Transition Footholds Planner (TFP)*

The objective of this planner is to find a sequence of transition footholds that the robot can use to transition and traverse faces in order to reach the goal foothold $\mathbf{FH}_g$ from the start foothold $\mathbf{FH}_s$. This way, the problem is approached globally, first determining the sequence of faces that the robot must traverse, and then solving the simpler two-dimensional problems of navigating each face in following planners.

In order to find the global transition path (i.e., the sequence of transition footholds), the planner uses a graph-based search algorithm. Specifically, the A* algorithm (Hart et al., 1968) is used, where the nodes of the graph are the potential transition footholds generated in the preprocessing of the structure (Section 3.2.1). The Euclidean distance between the footholds is used as the cost function, and the heuristic function is the Euclidean distance between the foothold and the goal foothold.

Our key contribution to conventional A* methods is the determination of neighbours for each node. For a given foothold, the set of neighbours is the union of two subsets (see Figure 3.3): (1) every other potential transition foothold belonging to the same face, and (2) every potential transition foothold belonging to other faces that are reachable from the current foothold. The identification of

those footholds (or nodes) of subset (2), i.e., those that are reachable from the current foothold, is performed by a reachability test.

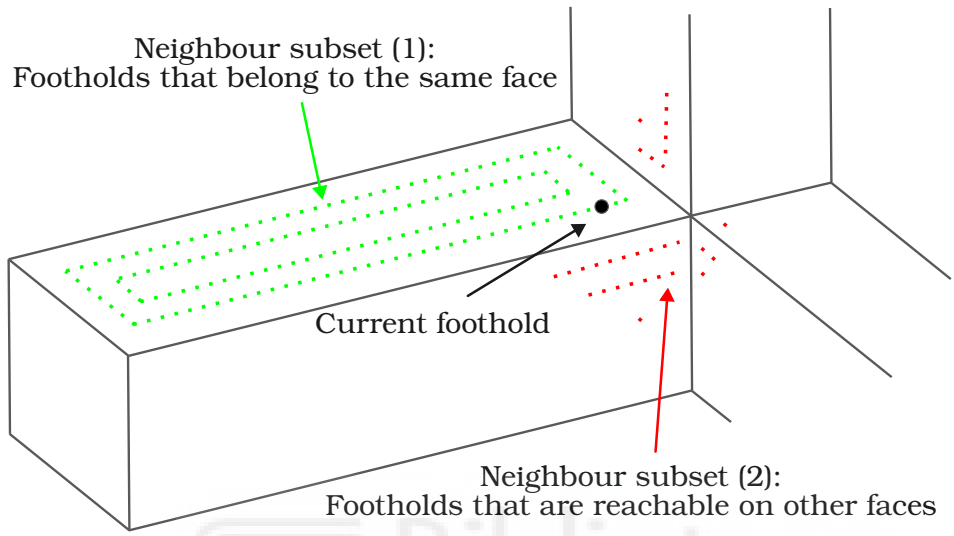


Figure 3.3: Neighbours of a potential transition foothold.

Considering that the fixed gripper is attached to a face, the reachability test must determine whether an input foothold (pose of the free gripper) is reachable from the current foothold of the fixed gripper. In case of non-redundant robots, testing the reachability of a foothold is a simple problem: one must solve the IKP for the tested foothold, and if the solution is real (not complex), does not violate the robot's joint limits, and does not result in a collision, then the foothold is reachable. However, for kinematically redundant robots, such as the HyReCRo robot, there exist an infinite number of joint configurations that map to a specific task point (i.e., the foothold). Consequently, it is not possible to directly verify the reachability of a foothold by solving the IKP, thus requiring a different reachability test approach. The efficiency of the reachability test is crucial for the performance of the algorithm, since it is the fundamental element of the TFP planner. During the development of the algorithm, we have tested different reachability test methods:

- **Iterative reachability test**: For redundant robots with $n$ actuated joints and a task space of $m$ dimensions, $n - m$ joints are selected as redundant joints. In every iteration, redundant joints are given random values within their limits, and the IKP is solved for the remaining $m$ joints (Peidró et al., 2015). The same conditions as in the non-redundant case are checked: the solu-

tion must be real, not violate the joint limits, and not result in a collision. If the conditions are met, the foothold is considered reachable. Otherwise, the process is repeated with new random values for the redundant joints, until a maximum number of iterations is reached (e.g., 300). This method is simple and easy to implement, but it is computationally expensive, not repeatable, and may not find a solution. The variability of the results is considerable, as the method is based on random values, and, in case that the foothold is not reachable, the algorithm may take a long time to determine this (must complete the determined maximum number of iterations).

- **Analytical reachability test**: This method is based on the study of the specific kinematics of each robot in order to determine the reachability of a foothold analytically. For instance, the reachability of a foothold may be determined by checking whether the distance to the tested foothold is lower than the sum of the lengths of the robot's links when fully extended. However, for robots with complex kinematics, such as the HyReCRo robot, such analytical criterion is not easy to formulate, due to the fact that the robot's workspace is not a simple geometric shape, such as a sphere or a cylinder. Therefore, this method presents a strong *ad-hoc* component, tailored for each robot in particular, and is not generalizable.
- **Neural network reachability test**: This approach utilizes a binary classification neural network to evaluate the reachability of a foothold. The forward propagation process (i.e., testing the foothold) is highly efficient and fast. However, generating the training dataset and training the network requires significant time and computational resources. Additionally, it is important to acknowledge that the network's accuracy will never be perfect, meaning there is a possibility of misclassifying a foothold.

Given the complexity of the HyReCRo robot's kinematics, the analytical reachability test is not feasible (Peidró et al., 2015). The iterative reachability test is computationally expensive and not repeatable in terms of results and runtime. Therefore, we have chosen to implement the neural network reachability test.

Solving the IKP for the HyReCRo robot requires differentiating between two situations (Peidró et al., 2015): a regular case, and a singular one, where both **z** axes of the robot's grippers are parallel or antiparallel. When it comes to binary classification, the neural network must also differentiate between these two cases. Hence, two different networks are trained: one for regular poses, usually when footholds pertain to the same face, and another for transition footholds, where **z** axes are parallel or antiparallel, since these poses are the most favourable for the robot to perform a transition (Peidró et al., 2019).

The architecture of the networks is a feedforward neural network with two hidden layers, with 32 and 64 neurons, respectively, as shown in Figure 3.4.

The input of the network is given by the position $(p_x, p_y, p_z)$ and orientation $(r_x, r_y, r_z)$ (coded as **zyx** Euler angles) of the foothold. Both hidden layers use the ReLU activation function, and the output layer uses the sigmoid activation function, which outputs a value between 0 and 1, indicating the reachability of the foothold.

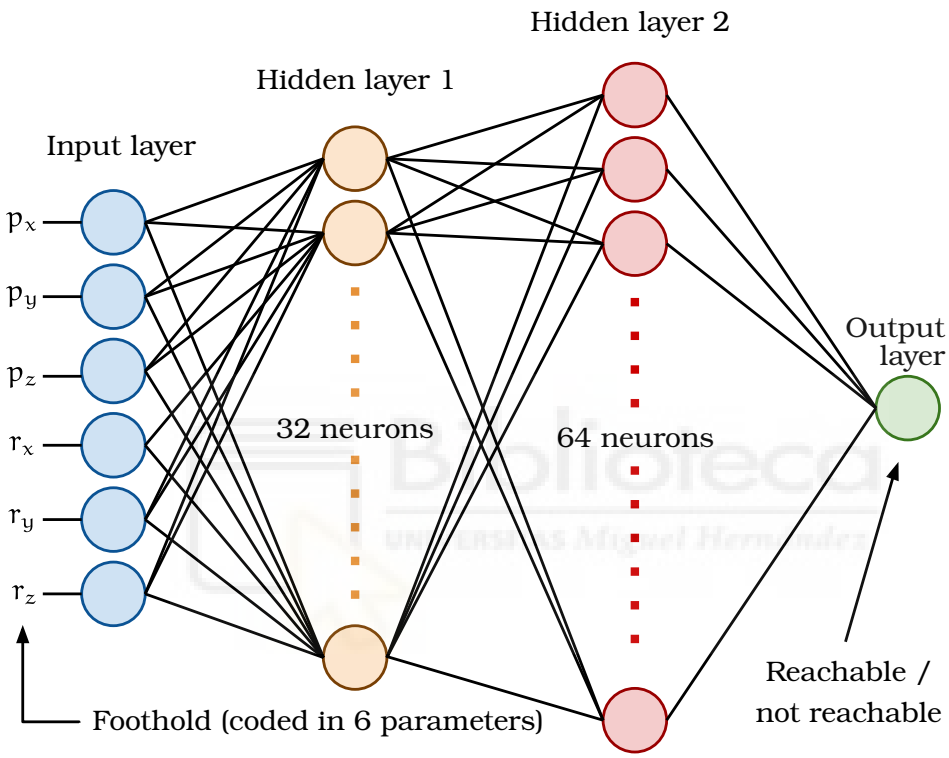


Figure 3.4: Architecture of the neural networks trained for the reachability test.

The training dataset is generated by sampling random footholds within the workspace of the robot, and labelling them as reachable or unreachable, using the iterative reachability test as the ground truth. In this study, $2 \cdot 10^7$ samples have been generated for each network, and the dataset has been split into training and test datasets with a ratio of 80% and 20%, respectively. The networks are trained using the Stochastic Gradient Descent (SGD) optimizer and the binary cross-entropy loss function, with a learning rate of 0.01 and a batch size of 10. The training process is stopped after 200 epochs. The networks are implemented using the PyTorch library (Paszke et al., 2019). The trained networks achieved an

accuracy of 98.32% for regular poses and 97.85% for singular poses on the test dataset, which is considered satisfactory for this study.

With the reachability test implemented, it is now possible to determine the subset (2) of neighbours for a given foothold (recall Figure 3.3). The proposed approach operates as follows. First, the set of all potential footholds reachable from the current foothold is swiftly estimated by a search in a sphere centred at the current foothold. This search is performed efficiently by using a k-d tree data structure (Bentley, 1975). The radius of the sphere must be chosen so that it is large enough to cover the entire workspace of the robot within its limits. Therefore, a prior analysis of the robot's workspace is necessary (Peidró et al., 2017). Note that this step instantly discards all footholds that fall outside the workspace of the robot. Although this approximation of the workspace as a sphere is certainly imprecise, it is used solely to discard points that are undoubtedly unreachable, and not to determine the reachability of a foothold. Points included within the approximation (i.e., within the sphere), but not actually part of the workspace, will be discarded next. Finally, every foothold within the sphere is tested for reachability using the neural network reachability test. If the foothold is deemed reachable, and the joint configuration needed to reach it is collision-free, the foothold is added to the neighbour set.

Incorporating this approach to the conventional A* algorithm, yields a transition foothold path, which is the sequence of footholds that the robot must use to reach the goal foothold $\mathbf{FH}_g$ from the start foothold $\mathbf{FH}_s$. An example of a transition foothold path

$$\mathcal{TP} = \{\mathbf{FH}_s, {}^1\mathbf{FH}_d, {}^2\mathbf{FH}_a, {}^2\mathbf{FH}_d, {}^3\mathbf{FH}_a, {}^3\mathbf{FH}_d, {}^4\mathbf{FH}_a, \mathbf{FH}_g\}$$

is shown in Figure 3.5. If a pair of consecutive transition footholds belong to the same face $F_i$, they are designated as the arrival ${}^i\mathbf{FH}_a$ and departure foothold ${}^i\mathbf{FH}_a$ of said face. The other planner of this hierarchy level, the FFP planner, which is described in the next Section 3.2.3, will plan the footholds that navigate face $F_i$ to connect ${}^i\mathbf{FH}_a$ and ${}^i\mathbf{FH}_d$, as well as $\mathbf{FH}_s$ and ${}^1\mathbf{FH}_d$, and ${}^{last}\mathbf{FH}_a$ and $\mathbf{FH}_g$.

### 3.2.3 *Face Footholds Planner (FFP)*

In the global transition path $\mathcal{TP}$ returned by the TFP planner, a pair of consecutive transition footholds may belong to the same face, as indicate, for example, footholds ${}^2\mathbf{FH}_a$ and ${}^2\mathbf{FH}_d$ in Figure 3.5. In this case, extra footholds on face $F_2$ must be planned in order to connect the arrival and departure footholds (i.e., ${}^2\mathbf{FH}_d$ is not directly reachable from ${}^2\mathbf{FH}_a$). This is the objective of the Face Foothold Planner (FFP), which determines the most efficient path in terms of length and number of footholds to traverse the face.

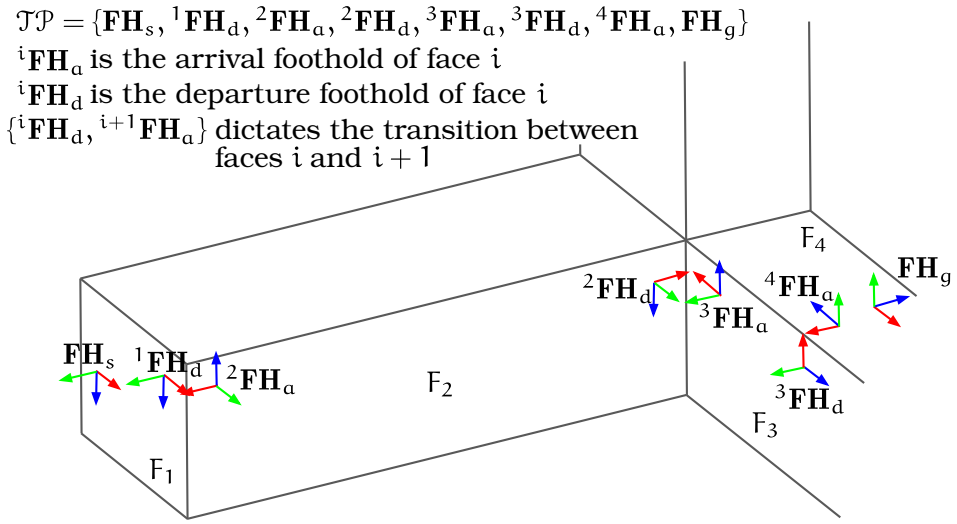


Figure 3.5: Example of a transition foothold path, output of the TFP.

The FFP planner adds a new level of hierarchy to the algorithm, dividing the problem into two subproblems. First, the CFFP finds a continuous path $\mathcal{CP}$ on the face that connects the arrival $\mathbf{FH}_a$ and departure $\mathbf{FH}_d$ footholds of the face. Then, the DFFP utilizes the continuous path in order to derive a discrete path $\mathcal{DP}$, which is a sequence of discrete footholds that the robot can use to reach $\mathbf{FH}_d$ from $\mathbf{FH}_a$.

#### 3.2.3.1 *Continuous Face Footholds Planner (CFFP)*

The objective of the first planner (CFFP) of the FFP hierarchy is to find the shortest continuous path $\mathcal{CP}_i$ that joins the arrival ${}^i\mathbf{FH}_a$ and departure ${}^i\mathbf{FH}_d$ footholds of face $F_i$. To this end, the same process of shrinking the perimeter of the face is performed, as described in Section 3.2.1 and illustrated in Figure 3.1(c). This time, the face is shrunk by a distance $s_s$ that ensures that the grippers of the robot can be placed completely on the face, without being partially on air, as Figure 3.6(a) depicts. This operation produces a *safe polygon* ${}^i\mathcal{F}_s$ that represents the area where the robot can place its grippers on the face $F_i$.

Next, a visibility graph (Lee and Preparata, 1984) is constructed on the safe polygon ${}^i\mathcal{F}_s$, where the vertices are the vertices that define the safe polygon, and the edges are line segments that connect the vertices. The simple geometric shapes that describe the faces of truss structures make the construction of the visibility graph, and the subsequent path solving, a simple and efficient task.

Figure 3.6 (ignore the discrete foothold path $\mathcal{DP}_i$ for now) shows an example of a continuous path $\mathcal{CP}_i$ on face $F_i$, along with the arrival and departure footholds ${}^i\mathbf{FH}_a$ and ${}^i\mathbf{FH}_d$, and the safe polygon ${}^i\mathcal{F}_s$.

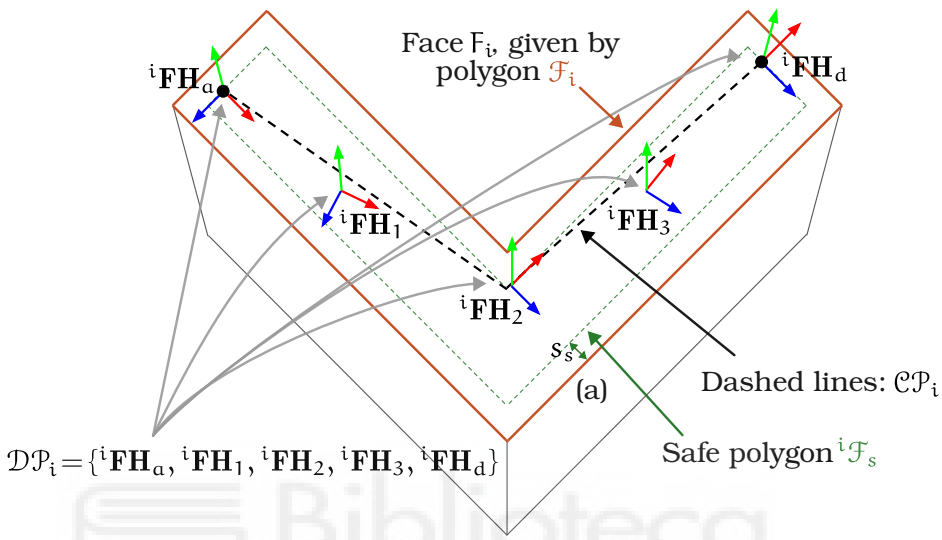


Figure 3.6: Example of a continuous path $\mathcal{CP}_i$, and discrete path $\mathcal{DP}_i$, on face $F_i$. (a) Polygon shrinking operation.

### 3.2.3.2 *Discrete Face Footholds Planner (DFFP)*

The continuous path $\mathcal{CP}_i$ output by the CFFP planner is the shortest sequence of line segments that connect the arrival ${}^i\mathbf{FH}_a$ and departure ${}^i\mathbf{FH}_d$ footholds of face $F_i$. Step-by-step climbing robots, however, cannot move continuously, but must instead move their grippers from one foothold to another. To accomplish this, the DFFP planner utilizes the continuous path $\mathcal{CP}_i$ to generate a discrete sequence $\mathcal{DP}_i$ of footholds that are close to the continuous path. Starting from the arrival foothold ${}^i\mathbf{FH}_a$, the algorithm we present in this section will iteratively compute the next best foothold, employing the continuous path $\mathcal{CP}_i$ as a reference, until the departure foothold ${}^i\mathbf{FH}_d$ is reached.

In order to accelerate the runtime of every iteration of the algorithm below, we propose an *offline* prior analysis of the *planar workspace* of the robot, so that the IKP is not solved in every iteration. The planar workspace is defined as the intersection of the reachable workspace of the robot with the plane that contains the face $F_i$, which will equal the plane defined by axes $\mathbf{x}$ and $\mathbf{z}$ of the footholds

(3.4) when attached to the face. Figure 3.7 illustrates the planar workspace of the robot in perspective view.

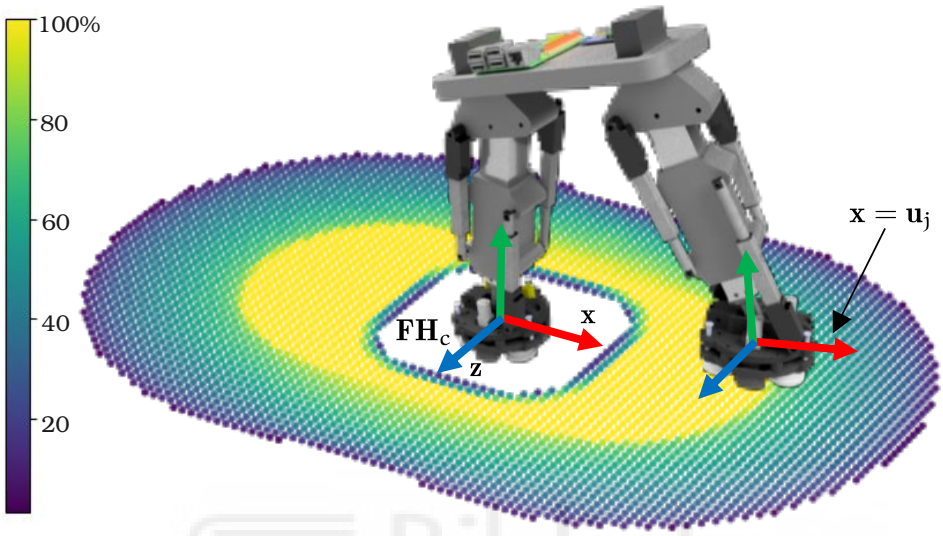


Figure 3.7: Perspective view of the planar workspace of the HyReCRo robot.

The offline analysis of the planar workspace is performed by sampling in a grid the plane defined by axes **x** and **z** of the fixed gripper (i.e., vertical position of the free gripper $p_y = 0$). This grid must be dense enough to obtain an accurate-enough discretization and representation of the workspace. For every point in the grid, reachable orientations of the free gripper are also discretized. Note that not every orientation for a given foothold position is reachable, as Figure 3.7 illustrates. Yellow points indicate that 100% of the orientations are reachable, meaning that vector $\mathbf{u}_j$ could point in any direction, while purple points indicate that few orientations are reachable. The dataset is generated by solving the IKP for every sampled position and orientation, similarly to the iterative reachability test described in Section 3.2.2, and the reachable footholds are stored. As a result, a three-dimensional dataset of reachable footholds is generated, where the first two dimensions define the positions of the footholds in plane **xz**. The third dimension is the orientation of the free gripper, which is given by a set of vectors $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{n_u}\}$, as illustrated in Figure 3.8(a). Every vector $\mathbf{u}_j$ is a unit vector parallel to the face, which would equal the **x** axis of the free gripper when attached to the face at the corresponding position. Moreover, when solving the IKP, a joint configuration out of the infinite number of solutions (due to redundancy) is selected and stored, which, in order to be conservative in terms of collision avoidance, is the one that maximizes the volume occupied

by the robot. The planar workspace of the HyReCRo robot is shown in Figure 3.8, where the percentage of reachable orientations out of the originally sampled orientations for each position is colour-coded. Yellow points indicate that 100% of the orientations are reachable, meaning that vector $\mathbf{u}_j$ could point in any direction, while purple points indicate that few orientations are reachable.

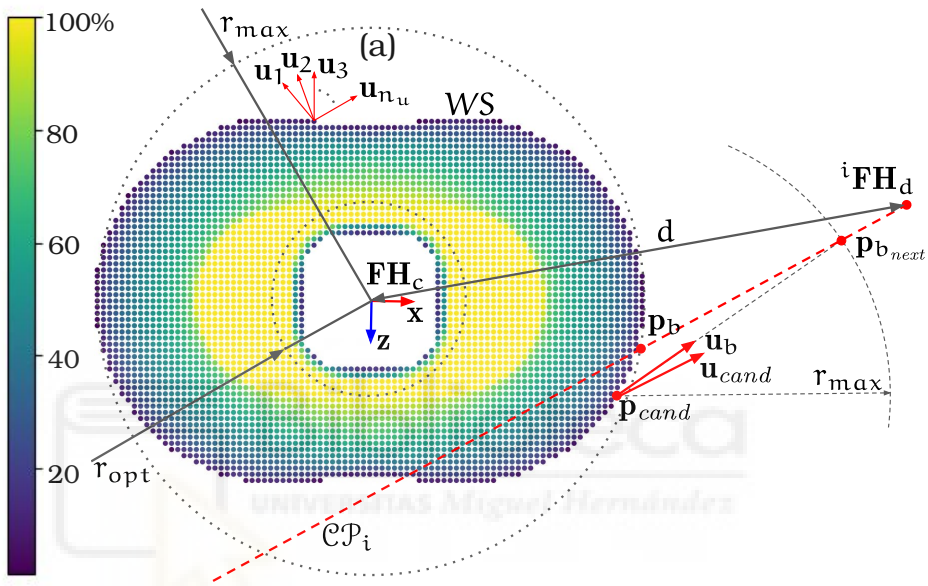


Figure 3.8: DFFP's best foothold selection algorithm and planar workspace of the HyReCRo robot. (a) Discretization of reachable orientations.

Algorithm 3.1 outlines the DFFP planner. Starting from the arrival foothold $^{i}\mathbf{FH}_a$ of face $F_i$, the algorithm iteratively computes the next best foothold $\mathbf{FH}_b$, following the continuous path $\mathcal{CP}_i$ returned by the CFFP planner, until the departure foothold $^{i}\mathbf{FH}_d$ is reached.

The arrival foothold $^{i}\mathbf{FH}_a$ is employed to initialize the discrete foothold path $\mathcal{DP}_i$, and to set the initial current foothold $\mathbf{FH}_c$. The algorithm then iterates until the departure foothold $^{i}\mathbf{FH}_d$ is reached. In every iteration, the algorithm computes the distance $d$ between the current foothold $\mathbf{FH}_c$ and the departure foothold $^{i}\mathbf{FH}_d$. Next, the algorithm computes the desired reach $r$ of the next foothold, which indicates the distance intended to be covered with the next foothold (i.e., the distance covered by the robot in one step):

$$r = \min(r_{max}, d - r_{opt}) \tag{3.5}$$

**Algorithm 3.1** DFFP planner for face $F_i$

1: $\mathcal{DP}_i \leftarrow \{{}^i\mathbf{FH}_a\}$
2: $\mathbf{FH}_c \leftarrow {}^i\mathbf{FH}_a$
3: **while** $\mathbf{FH}_c \neq {}^i\mathbf{FH}_d$ **do**
4: $\quad d \leftarrow$ distance between $\mathbf{FH}_c$ and ${}^i\mathbf{FH}_d$
5: $\quad r \leftarrow \min(r_{max}, d - r_{opt})$
6: $\quad \mathbf{FH}_c \leftarrow$ BestFoothold($\mathbf{FH}_c, r$)
7: $\quad$ Add $\mathbf{FH}_c$ to $\mathcal{DP}_i$
8: **return** $\mathcal{DP}_i$ ▷ Footholds path from ${}^i\mathbf{FH}_a$ to ${}^i\mathbf{FH}_d$

where $r_{max}$ is the maximum reach of the robot, and $r_{opt}$ is the optimal reach. The maximum reach $r_{max}$ is a parameter that defines the maximum possible distance that the robot can cover in one step. It equals to the radius of the circumference, centred at the fixed gripper, that circumscribes the robot's workspace, as illustrated in Figure 3.8. The optimal reach $r_{opt}$ is the radius of the circumference that covers points in the planar workspace where the number of reachable orientations is maximized (see how $r_{opt}$ in Figure 3.8 covers the yellow points). The choice of (3.5) between two possible reaches is made to tackle two situations. The first choice of $r = r_{max}$ is made when the robot is still far from the departure foothold, and regular steps, where the robot covers the maximum distance, are preferable. The second choice of $r = d - r_{opt}$ is made when the robot is approaching the departure foothold. This resulting reach value will produce a foothold that, at the next and final iteration, is able to reach the departure foothold with any orientation, since it will be placed at a distance $r_{opt}$ from it. Not reaching the departure foothold with the desired orientation could result in the robot not being able to perform the transition to the next face, as the orientation of the departure foothold would not be the one expected by the TFP planner.

The core of the algorithm is the BestFoothold function, which computes the best foothold $\mathbf{FH}_c$ for the free gripper, which overwrites the current foothold $\mathbf{FH}_c$ of the fixed gripper, given the desired reach $r$. The function is outlined in Algorithm 3.2 and illustrated in Figure 3.8.

The function starts by computing the best foothold position $\mathbf{p}_b$ as the intersection of the continuous path $\mathcal{CP}$ (computed by the CFFP planner on the current face) and a circumference centred at the current foothold $\mathbf{FH}_c$ with radius $r$. Note that *circ*($\mathbf{p}$, $r$) denotes the circumference centred at point $\mathbf{p}$ with radius $r$.

Next, the planar workspace *WS* contained inside the safe polygon $\mathcal{F}_s$, that defines the attachable surface of the current face, is retrieved. The workspace is sorted by the distance of the points to $\mathbf{p}_b$, so that the closest points are considered first in the following loop.

**Algorithm 3.2** BestFoothold function

```
1:  function BestFoothold(FH_c, r)
2:      p_b ← intersection of CP and circ(FH_c, r)
3:      WS ← planar workspace contained inside F_s
4:      WS_sorted ← WS sorted by distance to p_b
5:      for p_cand in WS_sorted do
6:          p_b_next ← intersection of CP and circ(p_cand, r)
7:          u_b ← unit vector from p_cand towards p_b_next
8:          u_cand ← u_j in p_cand closest to u_b
9:          if u_cand · u_b ⩽ ε and collision-free then
10:             FH_b ← {p_cand, u_cand}
11:             return FH_b
```

For every best-foothold candidate position $\mathbf{p}_{cand}$ in the sorted workspace, the prediction $\mathbf{p}_{b_{next}}$ of the next best foothold (the one that would be reached when attached to $\mathbf{p}_{cand}$) is computed as the intersection of the continuous path $\mathcal{CP}$ and a circumference centred at $\mathbf{p}_{cand}$ with radius $r$ (analogous to the computation of $\mathbf{p}_b$). The unit vector $\mathbf{u}_b$ is then computed as the unit vector pointing from $\mathbf{p}_{cand}$ to $\mathbf{p}_{b_{next}}$. $\mathbf{u}_b$ would correspond to the $\mathbf{x}$ axis of the free gripper when attached to $\mathbf{p}_{cand}$. Given that the reach of the robot is maximized in the direction of the $\mathbf{x}$ axis (see Figure 3.8), this approach maximizes the potential distance covered by the robot in the next iteration of Algorithm 3.1 (when the free gripper is attached to $\mathbf{p}_{cand}$ and becomes the fixed gripper). The unit vector $\mathbf{u}_{cand}$ is then computed as the vector closest to $\mathbf{u}_b$, out of the set of vectors $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{n_u}\}$ of reachable orientations of the free gripper in point $\mathbf{p}_{cand}$, computed during the offline analysis of the planar workspace (Figure 3.8).

Finally, $\mathbf{u}_{cand}$ must be validated to ensure that it is close enough to $\mathbf{u}_b$. This is done by computing the dot product between the two vectors, and checking if it is close to 1 (i.e., the vectors are parallel). If the result of the dot product is larger than the threshold $\varepsilon$ (e.g., 0.998, which corresponds to a difference of approximately $\pm 4°$), and the joint configuration needed to reach $\mathbf{p}_{cand}$, with orientation $\mathbf{u}_{cand}$, is collision-free, then $\mathbf{u}_{cand}$ is considered valid. The foothold $\mathbf{FH}_b$ is then defined as the position $\mathbf{p}_{cand}$ and orientation $\mathbf{u}_{cand}$ as axis $\mathbf{x}$. Recall from the definition of footholds in Section 3.1.2 that the $\mathbf{y}$ axis is always pointing upwards, and is equal to the already-known normal vector of the face $\mathbf{n}$. The remaining axis $\mathbf{z}$ is computed as the cross product between $\mathbf{x} = \mathbf{u}_{cand}$ and $\mathbf{y} = \mathbf{n}$, which ensures that the three axes are orthogonal.

If $\mathbf{u}_{cand}$ is not considered valid, the loop continues with the next candidate position $\mathbf{p}_{cand}$ in the sorted workspace $WS_{sorted}$ until a valid foothold is found.

It is worth noting that the sorting of the planar workspace, as well as the search of nearest positions and orientations, is performed by k-d tree data structures (Bentley, 1975) in order to accelerate the runtime of the algorithm.

Back to Algorithm 3.1, after determining the best foothold $\mathbf{FH}_b$, it is added to the discrete foothold path $\mathcal{DP}_i$. The algorithm iterates until the departure foothold ${}^i\mathbf{FH}_d$ is reached, at which point the discrete foothold path $\mathcal{DP}_i$ is returned.

Finally, in the larger scope of the hierarchical planner, the resulting discrete foothold path $\mathcal{DP}_i$ of every face $F_i$ is then concatenated to form the final path $\mathcal{P}$ of the robot through the structure.

$$\mathcal{P} = \{\mathcal{DP}_1, \mathcal{DP}_2, \ldots, \mathcal{DP}_{n_f}\} \tag{3.6}$$

where $n_f$ is the number of faces that the robot must traverse in order to reach the goal foothold $\mathbf{FH}_g$ from the start foothold $\mathbf{FH}_s$.

## 3.3 COMPARISON WITH PRIOR WORK

The method we propose for path planning in truss structures draws inspiration from the general idea of fragmenting the global three dimensional problem into subproblems on planar patches or faces presented in (Zhu et al., 2021). However, we have proposed novel planners for the different levels of the hierarchy, as well as different advantages in terms of efficiency and generalizability. In this section, we will comment on the differences between our method and the one proposed by Zhu et al. (2021), and experimentally compare via simulations the performance of different planners.

### 3.3.1 *Differences Between Planners*

Regarding the three-dimensional global planner (our TFP planner, Section 3.2.2), the method proposed by Zhu et al. (2021) is based on a three-stage process. First, they analyse the structure to determine every pair of traversable faces by means of their implementation of the reachability test (which will be compared next) in order to build an adjacency matrix. Second, starting from the start face as the root node, they convert the adjacency matrix into a tree structure by continuously exploring the adjacent faces, until the goal face is reached. Finally, they obtain the transition path by solving an optimization problem that minimizes the distance between the transition footholds of the path. Our TFP planner, on the other hand, obtains the transition path by means of a custom A* algorithm that uses the reachability test to determine the neighbours of every potential transition foothold, which allows for solving the global problem in a single step.

The reachability test proposed by Zhu et al. (2021) is based on an analytical study of their robot. While this method is appropriate and efficient, this is only

possible due to the fact that their robot has a simple kinematic structure, and is not kinematically redundant. In fact, when the robot is attached to a face, the reachable workspace is a sphere, which allows for a simple analytical reachability test. However, this method is not generalizable to robots with more complex kinematics, such as the HyReCRo robot, since their workspace is not a simple geometric shape, or a combination of them. Our method, on the other hand, uses a neural network to determine the reachability of a foothold, which can be extended to any robot. This approach is computationally inexpensive in terms of *online* performance, since the forward propagation process is highly efficient, but requires a considerable amount of time and computational resources for generating the training dataset and training the network.

In addition, (Zhu et al., 2021) states that their algorithm requires working with convex faces. This is probably due to the fact that they solve the global problem by means of an optimization problem, for which is complex to handle concave faces, as it could fall into local minima, or need to express polygon constraints non-linearly, in contrast to convex faces, for which a point can be tested to be inside or outside the polygon by means of a set of simple linear inequalities. Our method, on the other hand, does not present this limitation, and is able to handle concave faces, as demonstrated in the experimental results presented in the next Section 3.4.

When it comes to the two-dimensional local planner (our FFP planner, Section 3.2.3), the method proposed by Zhu et al. (2021) also divides the problem into two subproblems.

The first subproblem, equivalent to our CFFP planner (Section 3.2.3.1), is to find the shortest continuous path on a face, which they solve by means of a modified A* algorithm. However, the modification they propose in order to handle obstacles somewhat overcomplicates the algorithm, due to the fact that they include an intricate heuristic function with different conditional values based on taken measures of the obstacles. In our case, obstacles do not need to be considered, as they are already taken into account in TFP planner, which considers them as additional beams of the structure. Note that these beams will not be used as footholds, as the heuristic function used in the TFP planner will prioritize moving around them. On top of that, (Zhu et al., 2021)'s implementation of the A* algorithm to find the continuous path considers the immediate neighbours of the current node for expanding the search, which is not the most efficient approach. This is done in order to account for the potential obstacles. Instead, we take advantage of the simple geometric shapes that define the faces of truss structures, and use the perimeter vertices as nodes of the visibility graph, which allows for a more efficient and straightforward implementation.

The second subproblem, equivalent to our DFFP planner (Section 3.2.3.2), discretizes the continuous path into a sequence of discrete footholds. Their solution

iteratively computes the next best foothold by intuitively intersecting the robot's workspace boundaries with the continuous path returned by the CFFP planner, in order to maximize the distance covered by the robot in one step. Indeed, this approach is appropriate when the kinematics of the robot allow for reaching any position in its workspace with any orientation. However, in complex kinematic structures, such as the HyReCRo robot, this is not the case, as some positions may be reachable with any orientation, while others may not (see Figure 3.8). The solution we propose is to perform an offline analysis of the planar workspace of the robot, and use it in a more sophisticated algorithm that computes the best foothold for the free gripper, given the current foothold of the fixed gripper and the available reach.

### 3.3.2 *Experimental Comparison*

In order to compare the performance of the planners proposed by Zhu et al. (2021) and our method, we have conducted a set of simulations. The comparison of the TFP planners involves implementing the reachability test of each method. However, as we mentioned before, the reachability test proposed by Zhu et al. (2021) is based on an analytical study of their robot, and is not feasible to implement for the HyReCRo robot.

Therefore, we have decided to compare the performance of the FFP planners of each method, formed by the CFFP and DFFP planners. We have established an example concave face (see Figure 3.9), and have run both planners on it.

The first comparison is based on the runtime of the CFFP planner, and the quality (length) of the resulting continuous path. We have implemented the A* algorithm proposed by Zhu et al. (2021) with a neighbourhood of 0.1 meters. Averaged over 100 simulations, the execution time of the A* algorithm was 8.975 milliseconds, with a path length of 2.7 meters. Our CFFP planner, on the other hand, took 0.944 milliseconds to compute the continuous path, with a path length of 2.631 meters. An example of the continuous path obtained by each planner is shown in Figure 3.9(a).

Regarding the DFFP planner, the algorithm we have proposed is more elaborated and generalizable to find the discrete path, as it is more appropriate for robots that, like the HyReCRo robot, cannot reach any position in their workspace with any orientation. Conversely, the DFFP planner proposed by Zhu et al. (2021) is based on an intuitive approach that assumes that the robot can reach any position with any orientation. Figure 3.9(b) shows the discrete path obtained by each planner, where each foothold is represented by a position point, and three vectors that represent the orientation of the free gripper. Solid lines represent the sequence of footholds found by our DFFP planner, while dashed lines represent the sequence of footholds output by the DFFP planner proposed by Zhu et al.

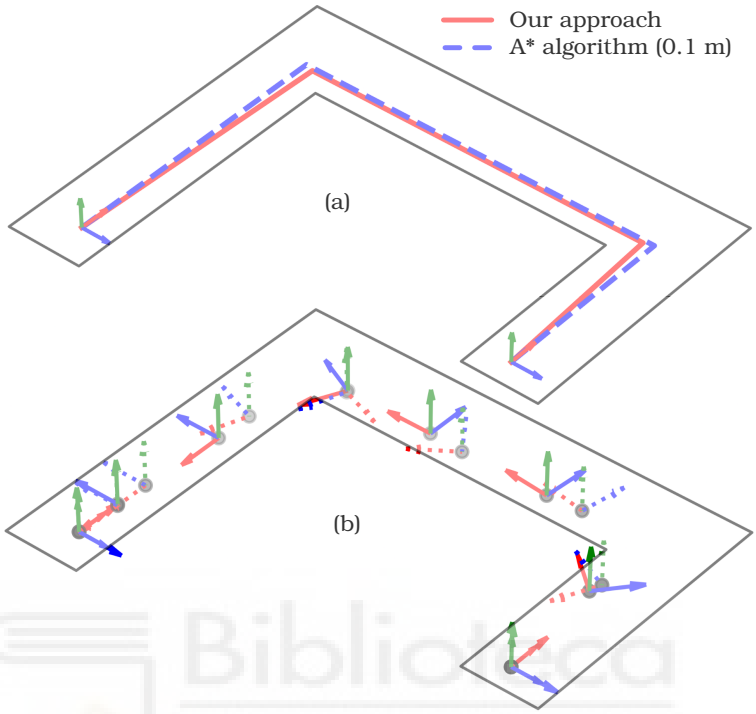


Figure 3.9: Comparison of the paths obtained by the FFP planners of each method. (a) CFFP planners. (b) DFFP planners.

(2021). The DFFP planner proposed by Zhu et al. (2021) outputs a path with 8 footholds in 8.299 milliseconds, while our DFFP planner outputs a path with 7 footholds in 63.643 milliseconds. This slight (tens of milliseconds) difference in runtime is compensated by the fact that our planner will save a step of the robot, which takes a comparatively longer time, as detachment and attachment of the robot to the structure involves an operation that takes around 3 seconds (Peidró et al., 2019). As shown in this example, the proposed algorithm is able to find a more efficient path in terms of number of footholds when the robot traverses faces that, like the one shown in Figure 3.9, have several direction changes. It is worth noting that, if we repeat this comparison for convex faces (without direction changes), both algorithms output paths with the same number of footholds, because, in that case, the robot will not have to change its orientation that much, and considering orientations (as we do in our algorithm) is much less relevant.

In this section, we present the results of a set of simulations that we have conducted in order to validate the performance of the hierarchical path planning method proposed in this chapter. The simulations have been conducted in Python, using an AMD Ryzen 7 5800U processor with 16 GB of RAM.

The simulated scenario consists of a truss structure formed by two perpendicular beams. These beams have lengths of 1 and 2 meters, respectively, with square cross-sections of sides 0.15 meters. This results in a complex structure with concave faces, that requires transitions between faces in order to traverse it, and presents a challenging scenario for the path planning algorithm.

The preprocessing of the structure, detailed in Section 3.2.1, has been conducted with a discretization step of the potential transition footholds of 0.05 meters. The workspace analysis of the HyReCRo robot concluded that the optimal distances for conducting concave ($s_{cc}$) and convex ($s_{cv}$) face transitions are 0.25 and 0.07 meters, respectively. In the CFFP planner, the shrinking distance $s$ has been set to 0.07 meters. Regarding the planar workspace used in the DFFP planner, it has been generated employing a resolution of 0.01 meters for the position dimensions, and 5 degrees for the orientation, as shown in Figure 3.8. These resolutions have been carefully selected after an analysis of the parameters of the algorithm, as discussed later in this section.

In Figure 3.10, we show the results of the simulations, which include the path $\mathcal{P}$ of footholds obtained by the hierarchical path planning method proposed in this chapter, and several instants of the simulation where the robot is attached to the structure. Note that, throughout Figure 3.10, the robot is never colliding, despite the fact that the colouring may suggest so in some images. This is due to how Matplotlib (the library used to plot the results) treats 3D representations: although no interference exists, when two objects are superimposed from the perspective of the camera, both of their colours are combined (structure's grey and robot's blue), which may give the impression of a collision.

The returned path is composed of footholds, which are represented by their position points, and three vectors that represent the orientation of the free gripper (**xyz** axes in (3.4) are coded as RGB, respectively). One can observe that the TFP determines the need to perform 2 transitions between faces to reach the goal point, by means of the transition footholds $\{{}^1\mathbf{FH}_d, {}^2\mathbf{FH}_a, {}^2\mathbf{FH}_d, {}^3\mathbf{FH}_a\}$ labelled in Figure 3.10(a). In Figures 3.10(c) and 3.10(e), the robot is represented performing the transition between faces, using footholds ${}^1\mathbf{FH}_d$ and ${}^2\mathbf{FH}_a$ for moving from face $F_1$ to face $F_2$ (Figure 3.10(c)), and footholds ${}^2\mathbf{FH}_d$ and ${}^3\mathbf{FH}_a$ for moving from face $F_2$ to face $F_3$ (Figure 3.10(e)).

For each face that the TFP planner determines that the robot must traverse, the FFP computes a discrete path of footholds that the robot can use to reach

the departure foothold from the arrival foothold. Foothold ${}^{i}\mathbf{FH}_{j}$ represents the j-th foothold of the discrete path that traverses face $F_i$. As an example, Figure 3.10(d) shows the discrete path that traverses face $F_2$, composed of footholds $\{{}^{2}\mathbf{FH}_{a}, {}^{2}\mathbf{FH}_{1}, {}^{2}\mathbf{FH}_{2}, {}^{2}\mathbf{FH}_{3}, {}^{2}\mathbf{FH}_{d}\}$. Figure 3.10(d) also shows an instant of this traversal (movement between ${}^{2}\mathbf{FH}_{1}$ and ${}^{2}\mathbf{FH}_{2}$).

Looking at Figure 3.10(f), it is clear that, during the traversal of the last face $F_3$, the goal foothold ${}^{3}\mathbf{FH}_{g}$ is not directly reachable from the arrival foothold ${}^{3}\mathbf{FH}_{a}$, even though it is inside a circumference of radius $r_{max}$, centred at the arrival foothold (Figure 3.8). This is because the point does not pertain to the planar workspace of the robot, since it is too close to the fixed gripper, and both grippers (the free and the fixed) would collide. The DFFP planner, however, intelligently detects that an additional foothold ${}^{3}\mathbf{FH}_{1}$ is needed, and positions it at a distance equal to $r_{opt}$ from the goal foothold $\mathbf{FH}_{g}$. This way, from foothold ${}^{3}\mathbf{FH}_{1}$, the robot can reach the goal foothold $\mathbf{FH}_{g}$ with any orientation, thus solving the problem as Figure 3.10(f) shows.

Table 3.1: Averaged runtimes for every stage of the algorithm presented in this chapter.

| Section | Hierarchy level | Runtime (ms) |
|---|---|---|
| 3.2.1 | Structure preprocessing | 136.999 |
| **3.2** | **Entire algorithm *(online)*** | **700.743** |
| 3.2.2 | Transition Footholds Planner | 688.478 |
| 3.2.3 | Face Footholds Planner ($F_1$) | 3.160 |
| | Face Footholds Planner ($F_2$) | 7.792 |
| | Face Footholds Planner ($F_3$) | 1.313 |

The runtime of every planner of the method has been averaged over 100 simulations, and the results are shown in Table 3.1. The preprocessing of the environment (Section 3.2.1) is conducted *offline*, which allows its computation before the robot is deployed on the structure, not disturbing the real-time performance of the algorithm. The average runtime of the entire algorithm presented in this chapter is 700.743 milliseconds, out of which the TFP planner takes 688.478 milliseconds, and the FFP planners 12.265 milliseconds. Since the sum of runtimes is below 1 second, we consider the algorithm to be appropriate for real-time application.

The FFP planner has demonstrated to be considerably efficient, finding discrete paths in a matter of milliseconds, which suggests that there exists little room for improvement in terms of performance. The TFP planner, on the other hand, is a

promising candidate for improvement, as it takes the 98.25% of the total runtime. However, it may be difficult to reduce its runtime without compromising the completeness of the search.

### 3.4.1 *Parameter Analysis*

In order to analyse the influence of the different parameters of the algorithm in its performance, and to select the most appropriate ones, we have conducted a set of experiments. The parameters that we have analysed are the discretization step of the potential transition footholds, the discretization step of the workspace position, and the discretization step of the workspace orientation. Scenario of Figure 3.10 has been maintained, and each parameter has been modified in a range of values. Table 3.2 shows the results of the experiments, where the average runtime of the algorithm and the quality of the resulting path are shown for each parameter configuration. Improvement percentages are computed with respect to the benchmark configuration, which is the one presented in Table 3.1. The quality of the path is given by the number of steps needed to traverse the structure, since this number should be minimized as much as possible, as step-by-step climbing robots invest a considerable amount of time in switching grippers after each step (Peidró et al., 2019), in contrast to continuous climbing robots.

Observing the results of the experiments in Table 3.2, we can conclude, as expected, that the runtime is affected by the discretization step employed in any stage of the algorithm. Be it the discretization of the potential transition footholds, or the planar workspace, a finer resolution will result in a longer runtime. This is due to the fact that the algorithm must analyse more points, and perform more operations overall, which increases the computational cost. Conversely, when the resolution is coarser, the runtime is reduced, but the quality of the path may be compromised, i.e., the number of steps needed to traverse the structure may increase. Note that going below a certain threshold of fine resolution will have no effect on the quality of the path (i.e., the number of steps will not decrease anymore), but the runtime will still increase. Finding these limits for every parameter is the best approach to optimize the values of the parameters, as it will result in the best trade-off between runtime and quality of the path. The results shown in Table 3.1 have been obtained by selecting the best parameters for the algorithm as described above.

## 3.5 CONCLUSIONS

In this chapter, we have presented a hierarchical path planning method for biped climbing robots in truss structures. The different levels of the hierarchy tackle

the problem in a structured and sequential way, decomposing the global three-dimensional problem into simpler planar subproblems. The first planner in the hierarchy, the TFP, determines the faces that the robot will traverse in order to reach the goal foothold. Moreover, the TFP planner computes the precise transition footholds that the robot will use to move between faces. Next, for every face that the TFP planner determines that the robot must traverse, the FFP computes a discrete path of footholds that the robot can use to reach the departure foothold from the arrival foothold. This operation is again decomposed into two subproblems tackled by two different planners: the CFFP and the DFFP. The CFFP planner computes the shortest continuous path between the arrival and departure footholds that the TFP planner (first planner) has determined. The DFFP planner discretizes this continuous path into a sequence of discrete footholds that the robot can use to traverse the face.

The method has been validated through simulations, and statistical analysis of the results has been conducted for selecting the most appropriate parameters of the algorithm. Additionally, the different planners of the algorithm proposed in this chapter have been compared with the ones proposed by Zhu et al. (2021), revealing favourable results.

Regarding future work, it is planned to conduct experimental validation of the method on a real prototype of the HyReCRo robot. The current neural-network model only specifies whether a foothold is reachable or not, future work could include the development of a neural network that predicts the joint configuration of the robot for a given foothold, which would allow for implementations in several parts of the algorithm. Finally, planning the joint trajectories of the robot between successive steps is also crucial and challenging, and will be addressed in future chapters of this thesis. Note that, for simplicity, we have used a linear interpolation of the joint trajectories in the simulations presented in this chapter, which is not always appropriate as it does not account for collisions or other constraints.

## 3.6 PUBLICATIONS RELATED TO THIS CHAPTER

The contributions presented in this chapter are related to the following publication:

- Marc Fabregat-Jaén, Adrián Peidró, Paula Mollá-Santamaría, Francisco José Soler, and Oscar Reinoso (2024). "Planificación jerárquica de movimientos de un robot trepador bípedo en estructuras tridimensionales reticulares." In: *Revista Iberoamericana de Automática e Informática Industrial* 21.3, pp. 262–273. DOI: 10.4995/riai.2024.20779 **(SCI-JCR Impact Factor: 1.2, Q4)**

– This journal extension of a previous conference paper presents a detailed explanation of the hierarchical path planning method, and includes a statistical analysis of the results for selecting the most appropriate parameters of the algorithm. It also includes comparisons with the stages presented by Zhu et al. (2021).

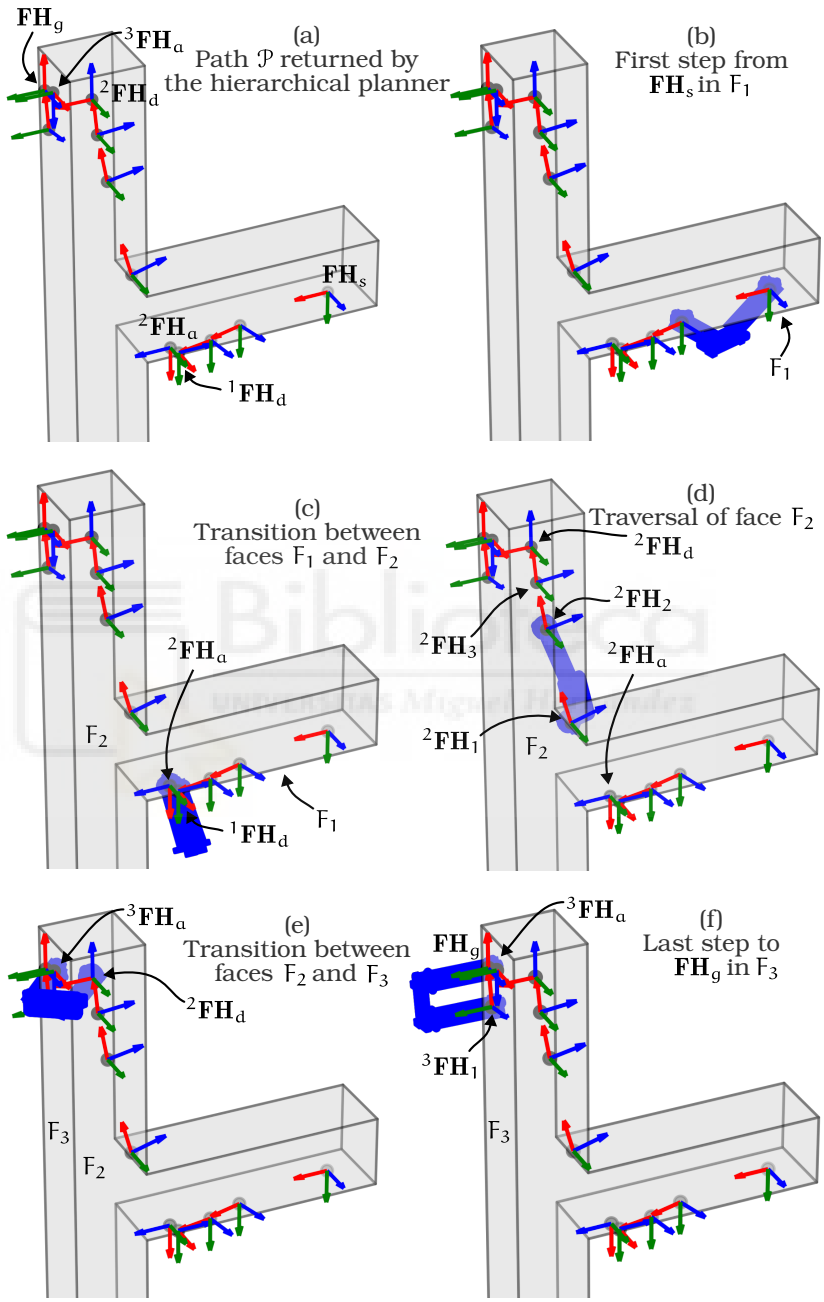


Figure 3.10: Results of the simulations.

Table 3.2: Comparison of different configuration parameters for the scenario of Figure 3.10. Average values of 100 repetitions.

| **Modifications** | | | | **Runtimes** (ms) | | | | | | | **Qualities** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Parameter** | **Original value** | **Increment** | **New value** | **Preprocessing** | **TFP** | **FFP** ($F_1$) | **FFP** ($F_2$) | **FFP** ($F_3$) | **TOTAL** | **Improvement** | **Steps** | **Increment** |
| Original parameters | | | | 136.999 | 688.478 | 3.160 | 7.792 | 1.313 | 837.742 | - | 11 | 0 |
| Potential footholds discretization | 0.050 m | 0.100 m | 0.150 m | 84.498 | 153.899 | 2.932 | 989.954 | 1.121 | 1.232.404 | -32.02% | 13 | 2 |
| | | 0.050 m | 0.100 m | 105.473 | 294.708 | 2.910 | 7.403 | 1.299 | 411.793 | 103.44% | 11 | 0 |
| | | 0.020 m | 0.070 m | 128.936 | 527.592 | 2.946 | 7.539 | 1.318 | 668.331 | 25.35% | 11 | 0 |
| | | -0.020 m | 0.030 m | 271.241 | 1.686.992 | 8.653 | 13.307 | 6.081 | 1.986.274 | -57.82% | 11 | 0 |
| *WS* position discretization | 0.010 m | 0.040 m | 0.050 m | *Algorithm does not converge* | | | | | | | | |
| | | 0.020 m | 0.030 m | 136.981 | 678.223 | 2.194 | 7.003 | 1.106 | 825.507 | 1.48% | 12 | 1 |
| | | -0.005 m | 0.005 m | 135.894 | 690.109 | 9.606 | 27.750 | 7.992 | 871.351 | -3.86% | 11 | 0 |
| *WS* orientation discretization | 5° | 15° | 20° | 134.583 | 682.909 | 1.601 | 6.125 | 1.198 | 826.416 | 1.37% | 12 | 1 |
| | | 5° | 10° | 135.104 | 688.911 | 2.912 | 7.329 | 1.002 | 835.258 | 0.30% | 11 | 0 |
| | | -3° | 2° | 136.004 | 694.291 | 4.928 | 11.583 | 3.136 | 849.942 | -1.44% | 11 | 0 |

# 4

# FEASIBILITY MAPS FOR REDUNDANCY RESOLUTION AND TASK TRAJECTORY GENERATION

The second main objective of this thesis is to solve the redundancy resolution problem. Given a prescribed task trajectory (e.g., an end-effector path) to be executed by a redundant manipulator (i.e., a manipulator with more DoF than the task dimension), the redundancy resolution problem consists in finding a continuous joint-space trajectory that resolves how to use the extra DoF to optimize a given criterion. In the context of this thesis, between each pair of consecutive footholds in the path returned by the algorithm presented in Chapter 3, redundancy resolution is required to generate a continuous joint-space trajectory that allows the robot to move from one foothold to the next one. Redundancy resolution has been approached in two ways in this thesis: using FMs and using SMMs. This chapter presents the first approach, which is based on the concept of FMs to solve the redundancy resolution problem.

After presenting a first solution to redundancy resolution based on FMs in this chapter, later, we will extend the formulation of FMs to the concept of Augmented Feasibility Maps (AFMs), in order to simultaneously solve the redundancy resolution and task trajectory generation problems. The AFM is a novel concept introduced in this thesis, which is based on the idea of augmenting the FM space with the task dimensions, in order to solve both problems in a single step. This allows for not having to rely on a predefined task trajectory, whose feasibility may not be guaranteed, and for generating a task trajectory that is feasible by construction. This is useful, for example, for tracking the discrete foothold path generated by the algorithm presented in Chapter 3. By simply feeding a pair of consecutive footholds to the method we will propose in this chapter, the task trajectory that connects them, as well as a joint trajectory that tracks it, will be generated.

This chapter is organized as follows. Section 4.1 introduces the concept of FMs, which are used to solve the redundancy resolution problem by means of the RRT-based algorithm we propose in Section 4.2. Section 4.2 also presents two examples that successfully solve the redundancy resolution problem using the proposed algorithm. Then, in Section 4.3, the concept of AFMs is introduced as an extension of FMs to solve the redundancy resolution and task trajectory generation problems simultaneously. Section 4.4 presents the algorithm that is used to plan a path in the AFM, and two examples that successfully solve the

redundancy resolution and task trajectory generation problems are presented. Finally, Section 4.5 presents the conclusions of this chapter.

## 4.1 FEASIBILITY MAPS

Consider a manipulator whose joint configuration is given by $\mathbf{q} = [q_1, q_2, \ldots, q_n]^T$, where $n$ is the number of DoF of the manipulator. Its forward kinematics is a function that maps the $n$-dimensional joint-space to the $m$-dimensional task-space, where $m$ is the dimension of the task to be executed:

$$\mathbf{x} = f(\mathbf{q}), \tag{4.1}$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_m]^T$.

It would be desirable to solve the IKP by inverting (4.1) to express the joint-space coordinates $\mathbf{q}$ as a function of the task-space coordinates $\mathbf{x}$:

$$\mathbf{q} = f^{-1}(\mathbf{x}). \tag{4.2}$$

However, deriving an analytical solution for (4.2) is often impossible, even for non-redundant manipulators (i.e., $n = m$), due to the forward kinematics being a non-bijective function, which renders the inverse kinematics mapping multi-valued (i.e., multiple joint configurations can map to the same task-space point).

The notion of *aspects* helps to understand this issue. Borrel and Liégeois (1986) define an aspect as a connected set of joint-space points where the Jacobian matrix remains full rank. In other words, aspects define regions in the joint space where the manipulator can move without encountering singularities. Therefore, the transition between aspects involves a singularity by definition. For manipulators that require passing through singularities to change postures (e.g., elbow-up to elbow-down configurations), and excluding cuspidal manipulators, the IKP problem returns a single unique solution for each joint-space point within an aspect. Therefore, for each aspect, an inverse kinematics function $\mathbf{g}(\cdot)$ can be derived, which maps a task-space point to a unique joint-space point.

Multi-valuedness is further exacerbated in the case of redundant manipulators, where $n > m$, as there exist infinitely many joint-space points $\mathbf{q}$ that map to the same task-space point $\mathbf{x}$. The degree of redundancy is given by $r = n - m$, which represents the number of extra DoF available to the manipulator. In the context of redundant manipulators, the special case of solving the IKP, for which there exist infinitely many solutions, is known as the *redundancy resolution problem*. An approach to redundancy resolution is *task space augmentation*, where the task

space is augmented with $r$ additional dimensions, for which the manipulator becomes non-redundant. A point in the augmented task-space $\mathbf{x}_a$ is given by:

$$\mathbf{x}_a = \begin{bmatrix} \mathbf{x} \\ \mathbf{x}_r \end{bmatrix}, \tag{4.3}$$

where $\mathbf{x}_r$ is the additional $r$-dimensional vector of redundant coordinates. The selection of $\mathbf{x}_r$ is arbitrary, but must be independent of every other component in $\mathbf{x}_a$. $\mathbf{x}_r$ is often chosen to be $r$ joint coordinates or a differentiable function of the joint coordinates, such as the position or orientation of a tracked point on the robot's body. In this thesis, $\mathbf{x}_r$ is assumed to be the former, i.e., $r$ joint coordinates.

Pámanes G et al. (2002) studied the effect that augmenting the task space has on the Jacobian matrix. Since the newly added rows of the Jacobian matrix (those corresponding to $\mathbf{x}_r$) introduced new singularities, a new set of aspects arose, which they named *extended aspects*. Consequently, depending on the choice of $\mathbf{x}_r$, the number of aspects and their domain can change.

Following the same reasoning as in the explanation of $\mathbf{g}(\cdot)$ a couple of paragraphs above, for a given selection of $\mathbf{x}_r$, there exists a unique Inverse Kinematics (IK) function $\mathbf{g}_a(\cdot)$ that maps a point in the augmented task-space to a unique joint-space point:

$$\mathbf{q} = \mathbf{g}_a(\mathbf{x}_a). \tag{4.4}$$

Nevertheless, rather than a single fixed task-space point, the redundancy resolution problem is often posed as the tracking of a trajectory in the task-space. Such trajectory $\mathbf{x}(t)$ is parametrized by an arc-length parameter $t$, which, for simplicity, will be referred to as time hereafter. In this situation, the set of configurations that track the task trajectory can be represented in an $(r+1)$-dimensional space, where the extra dimension corresponds to time $t$, and the remaining $r$ dimensions correspond to the redundant coordinates $\mathbf{x}_r$. This conceptual space is referred to as *FM*, and was first introduced by Wenger et al. (1993). A Feasibility Map $\mathcal{FM}$ is defined as the set of points in the $\left[t, \mathbf{x}_r^T\right]^T$ space for which $\mathbf{g}_a(\cdot)$ yields a real solution that satisfies additional constraints (e.g., joint limits or collision avoidance):

$$\mathcal{FM} = \left\{ \begin{bmatrix} t \\ \mathbf{x}_r \end{bmatrix} \;\middle|\; \mathbf{x}_a = \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{x}_r \end{bmatrix}, \mathbf{q} = \mathbf{g}_a(\mathbf{x}_a), \mathbf{q} \in \mathcal{Q} \right\} \tag{4.5}$$

where $\mathcal{Q}$ denotes the set of real joint-space points that satisfy the additional constraints. Note that, for a given time value $t$ and a set of redundant coordinates $\mathbf{x}_r$, the corresponding augmented task-space point $\mathbf{x}_a$ is uniquely defined, which

is mapped to a unique joint-space point $\mathbf{q}$ by $\mathbf{g}_a(\cdot)$. It is also worth noting that, since the definition of $\mathcal{FM}$ is based on the function $\mathbf{g}_a(\cdot)$, and each $\mathbf{g}_a(\cdot)$ is defined for a specific extended aspect (see previous paragraph), there exist as many FMs as extended aspects, for a given selection of $\mathbf{x}_r$.

Figure 4.1 shows an example of FMs for a 2-DoF manipulator with a 1-dimensional task trajectory, hence $r = 1$, resulting in a 2-dimensional FM. Note that the feasible spaces in the FMs correspond to the uncoloured regions of the map, while the coloured regions correspond to configurations that violate the kinematic constraints. Specifically, the purple regions correspond to points that map to complex (i.e., not real) joint configurations, red regions are points that lead to collisions, and the yellow space corresponds to points that violate the joint limits.

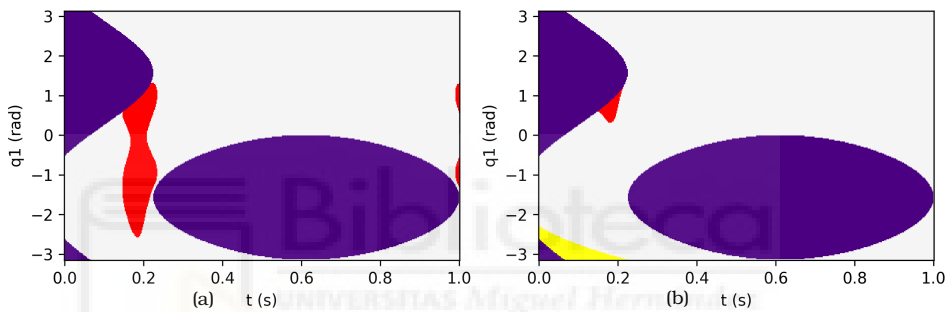


Figure 4.1: Feasibility maps for a 2R planar manipulator with the given 1D task-space trajectory. (a) for $\sigma = -1$, (b) for $\sigma = 1$.

These example FMs correspond to the planar manipulator shown in Figure 4.2, which has two revolute joints, and its joint configuration is given by $\mathbf{q} = [q_1, q_2]^T$. The task is given by the vertical position of the end-effector, $p_y$, with Forward Kinematics (FK) given by $\mathbf{x} = [p_y] = [\sin(q_1) + \sin(q_1 + q_2)]$ (considering unitary link lengths). The task trajectory is given by $\mathbf{x}(t) = p_y(t) = -6.662t^2 + 8.162t - 1.5$, where $t \in [0, 1]$. Additional constraints include joint limits $q_i \in [-\pi, \pi] \, \forall i \in \{1, 2\}$, and the avoidance of the end-effector colliding with an elliptical obstacle given by $\frac{(p_x - 1.1)^2}{1^2} + \frac{(p_y + 0.2)^2}{0.25^2} \leqslant 1$, which is coloured in red in Figure 4.2. The first joint is selected as the redundant coordinate, i.e., $\mathbf{x}_r = q_1$, hence $\mathbf{x}_a = [p_y, q_1]^T$. Therefore, the augmented IK function of Equation (4.4) in this example is given by:

$$\mathbf{q} = \mathbf{g}_a(\mathbf{x}_a) = \begin{bmatrix} q_1 \\ \frac{\pi}{2} + \sigma\left(\frac{\pi}{2} - \arcsin(p_y(t) - \sin q_1)\right) - q_1 \end{bmatrix} \tag{4.6}$$

Note that $\mathbf{g}_a(\cdot)$ has two solutions, depending on the sign of $\sigma$, which is a binary variable $\sigma \in \{-1, 1\}$ that determines which one of the two possible configurations the manipulator adopts, as shown in Figure 4.2. Each one of these solutions corresponds to a different extended aspect, and therefore, a different FM in Figure 4.1. If $\sigma = -1$, the manipulator adopts the configuration shown in solid line in Figure 4.2, which assumes that the angle resulting from the sum $(q_1 + q_2)$ lies in the first or fourth quadrants $(-\frac{\pi}{2} \leqslant q_1 + q_2 \leqslant \frac{\pi}{2})$, and produces the FM shown in Figure 4.1(a). Conversely, if $\sigma = 1$, the manipulator adopts the configuration shown in dashed line in Figure 4.2, which assumes that the angle resulting from the sum $(q_1 + q_2)$ lies in the second or third quadrants $(\frac{\pi}{2} \leqslant q_1 + q_2 \leqslant \frac{3\pi}{2})$, and produces the FM shown in Figure 4.1(b).

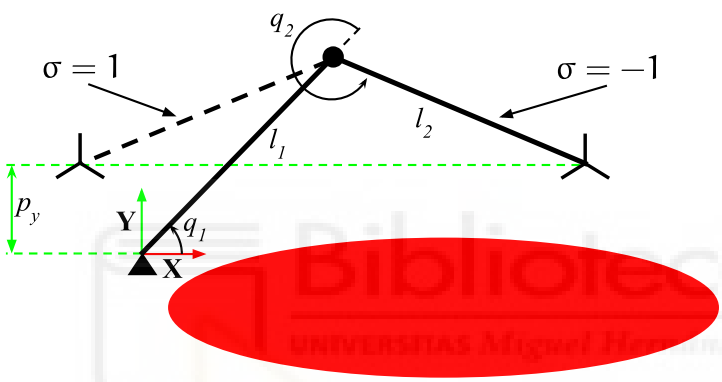


Figure 4.2: 2R planar manipulator with task $\mathbf{x} = p_y$.

## 4.2 FMS FOR REDUNDANCY RESOLUTION

The concept of FMs has been used in the literature to solve the redundancy resolution problem since its introduction by Wenger et al., 1993. The same authors extended their work in (Pámanes G et al., 2002) by employing FMs to plan collision-free paths for redundant manipulators, while minimizing the change in the joint selected as the redundant coordinate. Subsequent work by Reveles, Wenger, et al. (2016) used FMs to plan the trajectory of a redundant parallel manipulator by transitioning between working modes, where each mode corresponds to a different FM. Nevertheless, the planning is not autonomous, as some intermediate waypoints must be manually selected. Finally, (Ferrentino and Chiacchio, 2020) proposed running an exhaustive grid search over every FM to find the optimal solution in terms of a given criterion, while considering transitions between FMs.

The main advantage of using FMs for redundancy resolution is that they provide a global solution to the redundancy resolution problem, as they consider

all possible joint-space configurations that satisfy the task trajectory and the additional constraints. By tackling the redundancy resolution problem in a global manner, using FMs solves the issues presented in Section 2.3.1. The most relevant problem of local methods is that they may not find a solution to the redundancy resolution problem, even if one exists. This is because the exploration of the solution space is led by the Jacobian matrix, whose lack of a global scope (Jacobian matrix can only be considered valid around a specific joint-space point) may lead the algorithm to empty solution sets, due to kinematic constraints or singularities, which emerge as external or internal barriers of the workspace (Peidro et al., 2018). On the other hand, FMs consider the entire solution space, and therefore, are able to find a solution given that the algorithm employed to explore the FM is able to find it.

However, the problem of using FMs is that they are computationally expensive, as they require the evaluation of the IK function $\mathbf{g}_a(\cdot)$ for every point in the FM, and the posterior check of the additional constraints for the given solution $\mathbf{q}$. This may be the reason why the literature has not proposed the use of FMs for degrees of redundancy greater than 1, as the computational cost would increase exponentially with the number of redundant coordinates.

Another issue is that FMs actually are projections of the manifold of solutions $(\mathbf{q}, t)$ for the given task trajectory $\mathbf{x}(t)$ and constraints, which is named the *self-motion domain*, and will be introduced in Chapter 5. Chapter 5 will also provide further details on the problems that arise from these projections, but the main issue is that, in order to transition between FMs (i.e., between different extended aspects), as works like (Reveles, Wenger, et al., 2016) and (Ferrentino and Chiacchio, 2020) propose, the algorithm must pass through singularities of the projection. This fact may produce non-smooth transitions between FMs, as it can be observed in Figure 19 of (Ferrentino and Chiacchio, 2020), where little spikes appear in the trajectory when a transition between FMs is performed.

In this Chapter, a novel approach to the use of FMs for redundancy resolution is proposed. In order to cope with the computational cost of building FMs, and to scale the method to higher degrees of redundancy, an RRT-based algorithm is proposed. The algorithm is able to find a solution to the redundancy resolution problem by exploring the FM in a probabilistic manner, and by considering the additional constraints. The tree is incrementally built by sampling points in the FM, and by connecting them to the nearest point in the tree that satisfies the additional constraints. This way, the algorithm may be run online, without needing to precompute the FM for the given task trajectory, as only the necessary points are computed as the tree is built. This allows the algorithm to scale to higher degrees of redundancy, as Section 4.2.3 will show.

Moreover, in order to avoid the problems arising from FMs being projections of the self-motion domain, we only work with the FM corresponding to the ex-

tended aspect to which the initial joint-space point belongs. The reason for this is that we consider that there exist tools better suited to examine the entire solution set (i.e., the self-motion domain used in Chapter 5), instead of performing transitions between FMs.

### 4.2.1 *RRT-based Planning in FMs*

Consider the same example of the previous section, with the 2R robot shown in Figure 4.2, and the task trajectory given by $\mathbf{x}(t) = p_y(t) = -6.662t^2 + 8.162t - 1.5$, where $t \in [0, 1]$, which produces the Feasibility Map denoted by $\mathcal{FM}$ in Figure 4.1(a) when the first joint is selected as the redundant coordinate $\mathbf{x}_r = q_1$, and $\sigma = -1$ is used for $\mathbf{g}_a(\cdot)$ in Equation (4.6). Given the start configuration of the robot $\mathbf{q}_s$, the redundancy resolution problem consists in finding a continuous joint-space trajectory $\mathbf{q}(t)$ that resolves how to use the extra DoF to optimize a given criterion, while completing the task trajectory $\mathbf{x}(t)$ and satisfying the additional constraints.

The redundancy resolution problem can also be posed as a path planning problem in $\mathcal{FM}$. Recall from Equation (4.5) how any point in $\mathbf{q}(t)$ is mapped to a point in $\mathcal{FM}$, which we will refer to as state $\mathbf{s} = [t, \mathbf{x}_r]^T$. Given the start joint-space point $\mathbf{q}_s$, the corresponding start state in $\mathcal{FM}$ can be computed as $\mathbf{s}_s = [t_s, \mathbf{x}_{rs}]^T$, where $t_s$ is the start time, and $\mathbf{x}_{rs}$ is the initial value of the redundant parameters vector. The final end of the trajectory, however, is not restricted to a specific state in $\mathcal{FM}$. Any point in $\mathcal{FM}$ that corresponds to the goal task-space point $\mathbf{x}(t_g)$ is a valid final state. Since we are working with redundant manipulators, there exist infinitely many final states in $\mathcal{FM}$ that correspond to the same task-space point $\mathbf{x}(t_g)$, where $t_g$ is the goal time in which the task trajectory must be completed. These states are given by the *goal set* $\mathcal{G}$, which is the r-dimensional subset result of slicing $\mathcal{FM}$ with a hyperplane perpendicular to the time axis at $t = t_g$. The goal set $\mathcal{G}$ for the example FM is represented in Figure 4.3 as the green vertical line at $t = t_g = 1$.

Figure 4.3 also shows two possible solutions to the problem. Note that any continuous path starting at $\mathbf{s}_s$ and ending at a point in $\mathcal{G}$ is a valid solution, since it completes the task trajectory $\mathbf{x}(t)$ (joins $t_s = 0$ and $t_g = 1$) and satisfies the additional constraints (the path does not cross any forbidden region). Although the dashed path in Figure 4.3 may seem discontinuous, it is actually continuous as it is the result of wrapping the revolute joint $q_1$ at $\pm\pi$. A robust algorithm should be able to consider these wrappings as long as the joint limits are not violated, since they may lead to a more optimal solution in terms of a given criterion, or avoid obstacles in the FM. Moreover, it is worth mentioning that any path in $\mathcal{FM}$ must be monotonically increasing in time, as going back in $t$

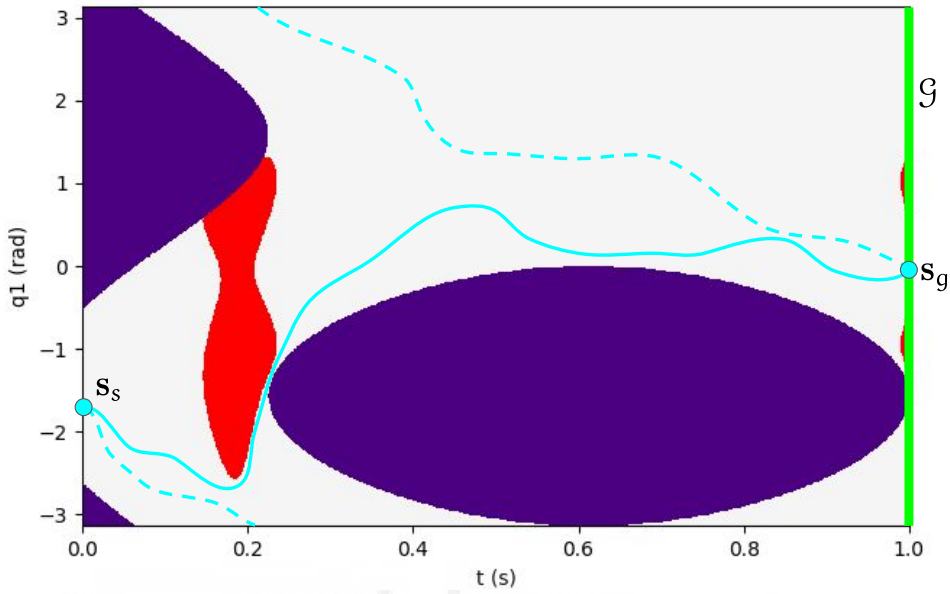


Figure 4.3: Example solution paths for a FM.

would imply that the manipulator is moving backwards in the task trajectory $\mathbf{x}(t)$, which is not allowed.

The proposed algorithm for solving the redundancy resolution problem using FMs is outlined in Algorithm 4.1. The algorithm is based on the RRT algorithm (LaValle, 1998), but adapted to account for the particularities of exploring the FM space.

**Algorithm 4.1** RRT-based Redundancy Resolution on FMs

Initialize tree $\mathcal{T}$ with root node $\mathbf{n}_s = \mathbf{s}_0$
**for** $i = 1, 2, \ldots, i_{max}$ **do**
  $\mathbf{s}_{random} \leftarrow$ SAMPLESTATE()
  $\mathbf{n}_{parent} \leftarrow$ BESTPARENT($\mathbf{s}_{random}$)
  **if** conditions C1, C2 and C3 are met **then**
    Add node $\mathbf{n}_{new} = \mathbf{s}_{random}$ to $\mathcal{T}$ with parent $\mathbf{n}_{parent}$
    $\mathbf{s}_{ext} \leftarrow$ EXTEND($\mathbf{n}_{parent}, \mathbf{n}_{new}$)
    **if** conditions C2 and C3 are met **then**
      Add node $\mathbf{n}_{ext} = \mathbf{s}_{ext}$ to $\mathcal{T}$ with parent $\mathbf{n}_{new}$
$\mathcal{P}_{best} \leftarrow$ BESTPATH($\mathcal{T}, c(\cdot)$)
$\mathcal{P}_{smooth} \leftarrow$ SMOOTHPATH($\mathcal{P}_{best}$)

The first step of the algorithm is to initialize the tree $\mathcal{T}$ with the root node $\mathbf{n}_s = \mathbf{s}_0$. Remember that $\mathbf{s}_0$ is the start state in $\mathcal{FM}$, which is mapped from the start joint-space point $\mathbf{q}_s$ and the start time $t_s$. We define a state $\mathbf{s}$ as a point in $\mathcal{FM}$, and a node $\mathbf{n}$ as a state that has been added to the tree $\mathcal{T}$. A tree is a data structure that consists of nodes connected by edges, where each node has a single parent node, except for the root node, which has no parent. The algorithm then iterates over a maximum number of iterations $i_{max}$.

In each iteration, the algorithm samples a random state $\mathbf{s}_{random}$ in $\mathcal{FM}$ by means of the function SampleState. SampleState returns a state that is uniformly sampled and is contained in $\mathcal{FM}$, meaning that the corresponding joint-space point $\mathbf{q}$ that maps to $\mathbf{s}_{random}$ must not violate the additional constraints.

The algorithm then selects the best parent node $\mathbf{n}_{parent}$ in the tree $\mathcal{T}$ for the random state $\mathbf{s}_{random}$, by means of the function BestParent. The search for the best parent node progresses forward along the time axis, starting from the root node $\mathbf{n}_s$ and visiting nodes in the order of their occurrence in time, until a node $\mathbf{n}_{parent}$ is found that meets a series of conditions:

C1 $t_{parent} < t_{random}$, to ensure that the time axis is monotonically increasing.

C2 The straight-line path between $\mathbf{n}_{parent}$ and $\mathbf{s}_{random}$ completely lies in $\mathcal{FM}$, i.e., the path does not cross any forbidden region.

C3 Other requirements, which may be specific to the problem at hand. For example, the velocity of some component $\dot{q}_i$ lies within its limits: $\dot{q}_{i,min} \leqslant \frac{q_{i,random} - q_{i,parent}}{t_{random} - t_{parent}} \leqslant \dot{q}_{i,max}$, where $t_j$ and $q_{i,j}$ can be directly obtained from $\mathbf{s}_j$ or $\mathbf{n}_j$ (depending on whether j is a state or a node).

The most computationally expensive part of the algorithm is evaluating condition C2, i.e., checking if the straight-line path between $\mathbf{n}_{parent}$ and $\mathbf{s}_{random}$ lies in $\mathcal{FM}$. This is because the FM is not precomputed, meaning that this verification must be performed online, by evaluating the solution $\mathbf{q}_i$ returned by the IK function $\mathbf{g}_a(\mathbf{s}_i)$, and checking whether $\mathbf{s}_i$ pertains to $\mathcal{FM}$ or not (i.e., whether $\mathbf{q}_i$ violates the constraints), for every state $\mathbf{s}_i$ in the segment that connects $\mathbf{n}_{parent}$ and $\mathbf{s}_{random}$. Note that the number of states to check, product of the discretization of the segment that connects $\mathbf{n}_{parent}$ and $\mathbf{s}_{random}$, is dictated by the resolution used. To improve the efficiency of the task, we propose checking the validity of the discretized states in random order, following a uniform random distribution, instead of checking them in order. This approach increases the probability of identifying an invalid state early in the process, which results in a faster rejection of the candidate parent node if invalid. In case all three conditions are met, the algorithm adds the new node $\mathbf{n}_{new}$, which corresponds to the random state $\mathbf{s}_{random}$, to the tree $\mathcal{T}$ with parent $\mathbf{n}_{parent}$.

Next, the algorithm tries to extend the segment that connects $\mathbf{n}_{parent}$ and $\mathbf{n}_{new}$ to reach the goal set $\mathcal{G}$, by means of the function Extend. This is done by elongating the segment from $\mathbf{n}_{parent}$ in the direction of $\mathbf{n}_{new}$, until the segment reaches a state $\mathbf{s}_{ext}$ that is contained in $\mathcal{G}$ (see Figure 4.4 for reference, where the Extend process is illustrated). Conditions C2 and C3 are then checked for the segment that connects $\mathbf{n}_{new}$ and $\mathbf{s}_{ext}$. Note that condition C1 does not need to be checked, since the result of the extension process, will always return a state with time equal to the goal time $t_{ext} = t_g$, which will always be greater than the time of the new node $t_{new}$. If the extended path is feasible (i.e., conditions C2 and C3 are met), the algorithm adds the new node $\mathbf{n}_{ext}$, which corresponds to the extended state $\mathbf{s}_{ext}$, to the tree $\mathcal{T}$ with parent $\mathbf{n}_{new}$.

In Figure 4.4, an example of an iteration of the algorithm is shown, where its most relevant steps are illustrated. A hypothetical $\mathcal{FM}$ is shown, where a 1-dimensional redundancy $\mathbf{x}_r$ is considered, and is bounded by its limits $[\mathbf{x}_{r,min}, \mathbf{x}_{r,max}]$. The hypothetical task trajectory $\mathbf{x}(t)$ that produces the FM starts at $t = t_s$ and ends at $t = t_g$. An obstacle in the workspace produces a circular forbidden region, coloured in red, that does not pertain to $\mathcal{FM}$ (i.e., is part of $\mathcal{FM}'$, which is the complement of $\mathcal{FM}$). The tree $\mathcal{T}$ is half-built, and is formed by the root node $\mathbf{n}_s$ and three nodes $\mathbf{n}_1$, $\mathbf{n}_2$ and $\mathbf{n}_3$, all of which are connected to the root node.

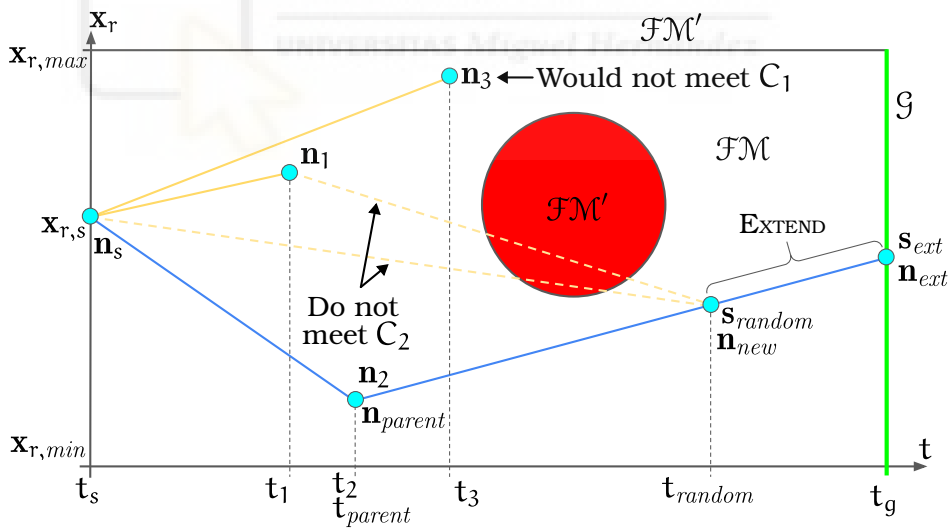


Figure 4.4: Illustration of an iteration of the RRT-based algorithm in a FM.

The illustrative iteration starts by sampling a random state $\mathbf{s}_{random}$ in $\mathcal{FM}$, for which the best parent node $\mathbf{n}_{parent}$ is searched. The search is performed forward

in time, starting from the root node $\mathbf{n}_s$, for which condition C1 is met, but condition C2 is not, as the straight-line path between $\mathbf{n}_s$ and $\mathbf{s}_{random}$ crosses the red forbidden region. The same happens for the next node $\mathbf{n}_1$. The next candidate node $\mathbf{n}_2$ meets conditions C1 and C2, as the straight-line path between $\mathbf{n}_2$ and $\mathbf{s}_{random}$ does not cross forbidden regions. Therefore, the new node $\mathbf{n}_{new}$ is added to the tree $\mathcal{T}$ with parent $\mathbf{n}_2$.

The algorithm then tries to extend the segment that connects $\mathbf{n}_2$ and $\mathbf{n}_{new}$ to reach the goal set $\mathcal{G}$. The extension process elongates the segment in the direction of $\mathbf{s}_{random}$, until the segment reaches a state $\mathbf{s}_{ext}$ that is contained in $\mathcal{G}$. In this case, the extension process is successful, since conditions are met, and the extended node $\mathbf{n}_{ext}$ is added to the tree $\mathcal{T}$ with parent $\mathbf{n}_{new}$. This results in a complete path that joints the start node $\mathbf{n}_s$ and a point in the goal set $\mathcal{G}$, which is a valid solution to the redundancy resolution problem, and is drawn in blue in Figure 4.4.

Back in Algorithm 4.1, the algorithm stops growing the tree after $i_{max}$ iterations, and the best path $\mathcal{P}_{best}$ is extracted from the tree $\mathcal{T}$ by means of the function BestPath. The function evaluates every path that connects the root node $\mathbf{n}_s$ and a node in the goal set $\mathcal{G}$, and returns the path that optimizes a given criterion $c(\cdot)$. This function $c(\cdot)$ evaluates the cost of a connection between a given node and its parent, and should be defined according to the problem at hand. For instance, a simple choice for $c(\cdot)$ is the weighted norm of the segment that connects a node and its parent. For a node $\mathbf{n}_i$, the cost $c(\mathbf{n}_i)$ is given by:

$$c(\mathbf{n}_i) = \sqrt{(\mathbf{n}_i - \mathbf{n}_{parent})^T \mathbf{W} (\mathbf{n}_i - \mathbf{n}_{parent})}, \tag{4.7}$$

where $\mathbf{n}_{parent}$ is the parent node of $\mathbf{n}_i$, which can be obtained from the tree $\mathcal{T}$, and $\mathbf{W}$ is a diagonal matrix, whose diagonal elements are the weights $w_i$ that determine the importance of each component of the FM space. This means that a higher weight $w_i$ will make the algorithm prefer paths that are shorter in the $i$-th dimension of the FM space.

Note that the cost function must be computed recursively, starting from a goal node (i.e., a node in $\mathcal{G}$), and adding its cost to that of its parent, until the root node is reached. The path with the lowest accumulated cost is the best path $\mathcal{P}_{best}$.

Due to the nature of sampling states in the space, every returned path is polygonal, which implies discontinuous velocities and infinite accelerations, which are not feasible for a manipulator. To solve this issue, the function SmoothPath is used to smooth the path $\mathcal{P}_{best}$. Given a path formed by a sequence of $n_n$ nodes $\mathcal{P} = \{\mathbf{n}_1, \mathbf{n}_2, \ldots, \mathbf{n}_{n_n}\}$, the function SmoothPath establishes a set of $n_c$ control points $\mathcal{C}$ along each segment of the path $\mathcal{P}$:

$$\mathcal{C} = \left\{ {}^1\mathbf{c}_1, {}^1\mathbf{c}_2, \ldots, {}^1\mathbf{c}_{n_c}, \ldots, {}^{n_n-1}\mathbf{c}_1, {}^{n_n-1}\mathbf{c}_2, \ldots, {}^{n_n-1}\mathbf{c}_{n_c} \right\}, \tag{4.8}$$

where $n_c$ is the number of control points per segment, and ${}^{i}\mathbf{c}_j$ is the j-th control point of the i-th segment. The control points are regularly spaced along the segment. Note that $n_c$ must be high enough so that the abrupt edges of the original path are smoothed, and the resulting path does not deviate significantly from the original path in order to maintain the feasibility and optimality of the solution. The notation employed and an example of the control points are shown in Figure 4.5.

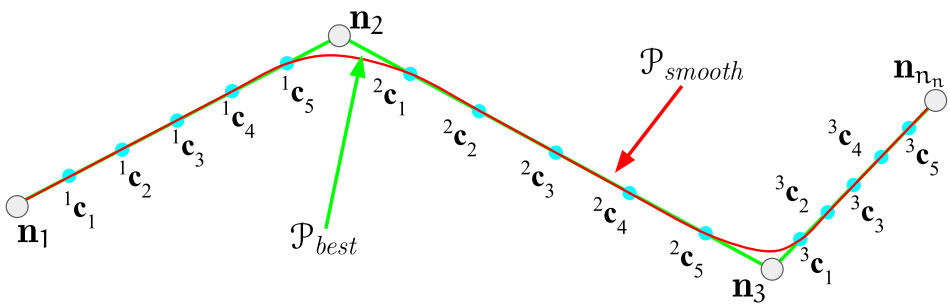


Figure 4.5: Illustration of SmoothPath function.

Once the control points have been established, the function SmoothPath computes a cubic B-spline (De Boor, 1978) that interpolates along the control points, and returns the smoothed path $\mathcal{P}_{smooth}$, as shown in Figure 4.5.

The returned path includes the values of the redundant parameters $\mathbf{x}_r$ in each time step t. By solving $\mathbf{q}$ from $\mathbf{x}_a(t, \mathbf{x}_r)$ by means of Equation (4.6), the joint-space trajectory $\mathbf{q}(t)$ is obtained, solving the problem of redundancy resolution.

### 4.2.2 *Example 1: 2-dimensional FM*

In order to validate the proposed algorithm, a set of simulations were performed. The algorithm was implemented in Python, and the simulations were run in a computer with an Intel Core i5-10400 CPU, which runs at 2.90 GHz, and 8 GB of RAM, running Ubuntu 20.04.

The first experiment corresponds to the example of the 2R robot shown in Figure 4.2, with the task trajectory given by $\mathbf{x}(t) = p_y(t) = -6.662t^2 + 8.162t - 1.5$, where $t \in [0, 1]$. The starting configuration of the manipulator is $\mathbf{q}_s = [-0.698, -0.331]$ rad, and the task trajectory must be completed in $t_g = 1$ s. Since the task trajectory is 1-dimensional, and the robot has 2 DoFs, the redundancy degree is $r = 1$. The first joint is selected as the redundant coordinate of the $(r = 1)$-dimensional redundant parameters vector $\mathbf{x}_r = q_1$. Regarding constraints, joint limits are set as $q_i \in [-2\pi, 2\pi]\, \forall i \in \{1, 2\}$ (in order to allow for angle wrapping at $\pm\pi$), and the end-effector must avoid colliding with the el-

liptical obstacle given by $\frac{(p_x-1.1)^2}{1^2}+\frac{(p_y+0.2)^2}{0.25^2}\leqslant 1$, which is coloured in red in Figure 4.2. Given the initial configuration, the manipulator is in the extended aspect that corresponds to the FM shown in Figure 4.1(a), and the IK function $\mathbf{g}_a(\cdot)$ is given by Equation (4.6) with $\sigma=-1$. Moreover, joint velocities are limited to a maximum value of $\dot{q}_{max}=13$ rad/s.

It is appropriate to recall that the algorithm does not have prior knowledge of the FM, which is computed in real time as needed, which means that only a portion of the FM will be computed by the time the algorithm finishes. However, for illustrative purposes, the entire FM will be shown in the results, even though it is not computed by the algorithm, nor taken into account for the posterior runtime evaluation.

The resolution employed for the evaluation of points in the FM (condition C2) is 0.005 seconds in the time axis. The number of control points per segment is set to $n_c=6$.

In Table 4.1, the average time and cost of the algorithm over 100 runs are shown for different values of $i_{max}$, whose value dictates the number of random states that will be sampled, along with any additional states that may be sampled during the extension process. The cost values respond to the cost function $c(\cdot)$ given in Equation (4.7), with an identity weight matrix $\mathbf{W}=\mathbf{I}$, and are computed for the best path found by the algorithm. The failure rate, which is defined as the percentage of runs in which the algorithm did not find a solution, drops to zero for values of $i_{max}$ greater than 70, which is why such data are not shown in the table.

Some analysis can be made from the results shown in Table 4.1, in order to determine the best value for $i_{max}$. In order to establish the globally optimal path for reference, we run the A* algorithm (Hart et al., 1968) over the entire FM, with the same cost function $c(\cdot)$. The returned path, which has a cost of 3.003, is plotted in magenta in Figure 4.6. Figure 4.7 plots the relative error of the cost of the paths found by the algorithm with respect to the globally optimal path, as a function of $i_{max}$, together with the best-fit hyperbola. This error measures the difference between costs of the paths found by the algorithm and the globally optimal path. Considering the results, the best value for $i_{max}$ is determined as 500, as it provides a good balance between time (2.216 s) and cost (3.258).

Figure 4.6 shows an example run of the algorithm with $i_{max}=500$. The tree is plotted in the figure following a colour coding that indicates, in red, branches of the tree that do not reach the goal set $\mathcal{G}$, and in green, branches that complete the task trajectory. The best smoothed path is plotted in blue. When comparing magenta and blue paths in Figure 4.6, it can be observed that the path found by the algorithm is very close to the globally optimal path, with a cost of 3.258 before smoothing.

Table 4.1: Average time and cost (over 100 runs) for different $i_{max}$

| $i_{max}$ | Average time (s) | Average cost |
|---|---|---|
| 100 | 0.217 | 3.974 |
| 200 | 0.974 | 3.473 |
| 300 | 1.232 | 3.415 |
| 400 | 1.879 | 3.312 |
| **500** | **2.216** | **3.258** |
| 600 | 3.659 | 3.249 |
| 700 | 4.489 | 3.208 |
| 800 | 5.074 | 3.226 |
| 900 | 6.267 | 3.208 |
| 1000 | 7.124 | 3.208 |
| 2500 | 37.442 | 3.173 |
| 5000 | 140.841 | 3.165 |
| 7500 | 302.732 | 3.163 |



### 4.2.3 *Example 2: 3-dimensional FM*

In this second example, an RPR (Revolute Prismatic Revolute) robot, whose joint configuration is given by $\mathbf{q} = [q_1, q_2, q_3]^T$, is considered, with the same task trajectory as in the previous example, and the same ellipsoidal obstacle to avoid. This scenario is shown in Figure 4.8. The links of the robot have lengths $l_1 = 0.5$ m and $l_2 = 1$ m. Since the robot has 3 DoFs, the redundancy degree is $r = 2$, and the redundant parameters vector is selected as $\mathbf{x}_r = [q_1, q_2]^T$. The augmented task vector $\mathbf{x}_a(t, \mathbf{x}_r)$ is therefore given by:

$$\mathbf{x}_a(\mathbf{q}) = \begin{bmatrix} \mathbf{x}(\mathbf{q}) \\ \mathbf{x}_r(\mathbf{q}) \end{bmatrix} = \begin{bmatrix} (l_1 + q_2)\sin(q_1) + l_2 \sin(q_1 + q_3) \\ q_1 \\ q_2 \end{bmatrix} \tag{4.9}$$

and the IK function $\mathbf{g}_a$ is given by:

$$\mathbf{q} = \mathbf{g}_a(\mathbf{x}_a) = \begin{bmatrix} q_1 \\ q_2 \\ \frac{\pi}{2} + \sigma\left(\frac{\pi}{2} - \beta\right) - q_1 \end{bmatrix} \quad \text{for } \sigma = \{-1, 1\} \tag{4.10}$$

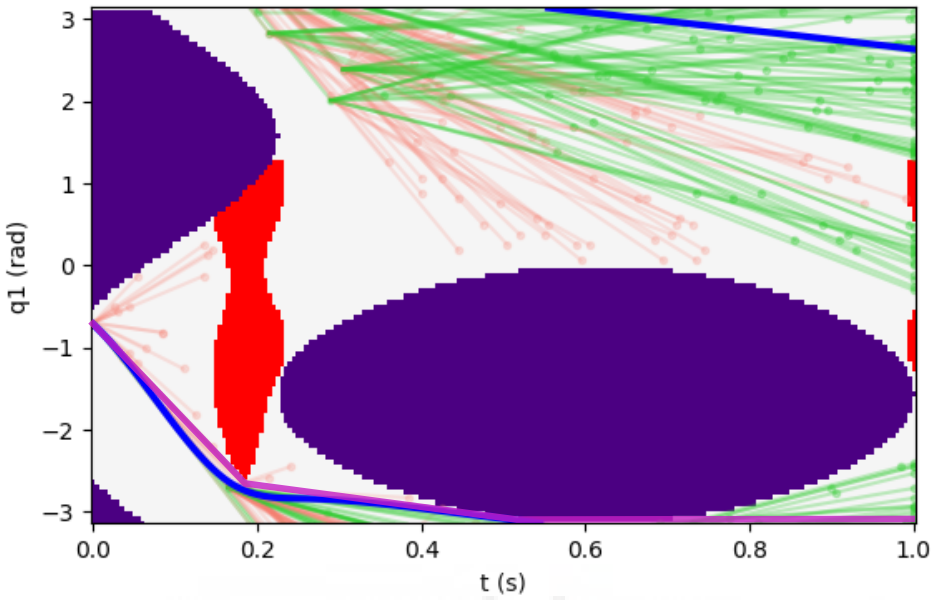


Figure 4.6: Feasibility map and solution paths for the 2R robot and 1-dimensional task trajectory. Includes the globally optimal path (magenta), the best path found by the algorithm (blue), paths that reach the goal set (blue), and paths that do not reach the goal set (red).

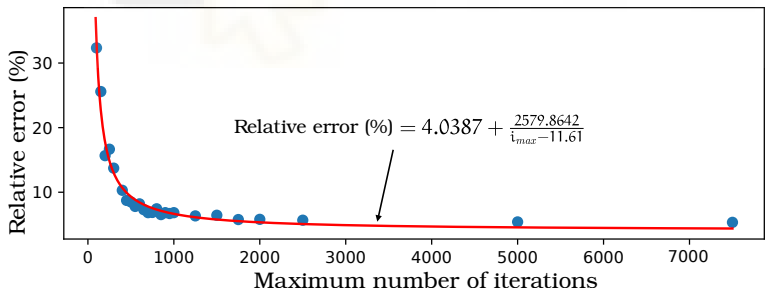


Figure 4.7: Relative error as a function of $i_{max}$.

where $\beta = \arcsin\left(\frac{p_y-(l_1+q_2)\sin(q_1)}{l_2}\right)$. For this example, we have chosen to plan the path in the FM corresponding to the extended aspect to which the solutions with $\sigma = -1$ belong.

The starting joint configuration is set so that the end effector begins at the same task point as in the previous example, $\mathbf{q}_s = [-0.698 \text{ rad}, 0.5 \text{ m}, -0.333 \text{ rad}]$. The joint limits for the revolute joints remain the same as in the previous ex-

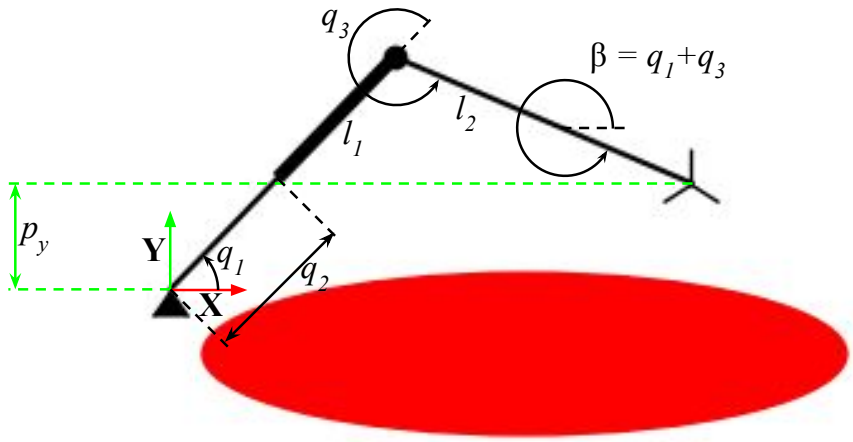


Figure 4.8: RPR robot executing a 1-dimensional task.

ample (i.e., $[-2\pi, 2\pi]$ rad), while the prismatic joint is limited to $q_2 \in [0, 0.5]$ m, ensuring that the maximum extension of the prismatic joint results in the same length as the second link. The velocity limits are also consistent with the previous example, with $\dot{q}_{max} = 13$ rad/s for the revolute joints and $\dot{q}_{2,max} = 0.2$ m/s for the prismatic joint. The resolution for evaluating points in the FM (condition C2) is set to 0.005 seconds along the time axis, and the number of control points per segment is $n_c = 6$. The cost function $c(\cdot)$ and the weight matrix $\mathbf{W} = \mathbf{I}$ are also the same as in the previous example. Additionally, the simulations were run on the same machine.

Table 4.2 gathers the average runtimes, average costs and failure rates for different values of $i_{max}$, over 500 runs. It can be observed how the failure rate drops to zero for values of $i_{max}$ greater than 1500.

This time, however, we have performed a different analysis to determine the best value for $i_{max}$. In Figure 4.9, the cost of the path found by the algorithm is plotted as a function of the runtime, for different values of $i_{max}$ (from 100 to 5900, in steps of 100). In order to determine the best value for $i_{max}$, we have fitted a regression curve to the data (in green), and selected the point in the curve that minimizes the radius of the circle that passes through the point and is centred at the origin (in red). This point is marked in red in the figure, and corresponds to $i_{max} = 2100$, for which the algorithm returns an average cost of 3.642 in an average runtime of 1.367 s, with a failure rate of 0%.

Finally, Figure 4.10 shows an example run of the algorithm with $i_{max} = 2100$. The tree is plotted in the figure following the same colour coding as in the previous example, with red paths that do not reach the goal set $\mathcal{G}$, green paths that reach the goal set, and the best smoothed path in blue. Note that the path

Table 4.2: Average runtimes, average costs and failure rates for different values of $i_{max}$

| $i_{max}$ | **Average runtime (s)** | **Average cost** | **Failure rate (%)** |
|---|---|---|---|
| 100 | 0.040 | 5.967 | 73.6 |
| 500 | 0.230 | 4.743 | 11.2 |
| 1000 | 0.490 | 4.182 | 1.8 |
| 1500 | 0.855 | 3.845 | 0.8 |
| 2000 | 1.260 | 3.709 | 0 |
| **2100** | **1.367** | **3.642** | **0** |
| 2500 | 1.760 | 3.554 | 0 |
| 3000 | 2.300 | 3.480 | 0 |
| 3500 | 2.959 | 3.438 | 0 |

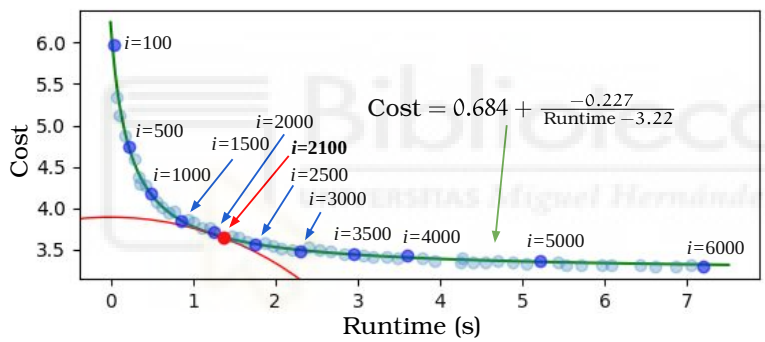


Figure 4.9: Cost as a function of runtime for different values of $i_{max}$.

wraps the revolute joint $q_1$ at $\pm\pi$ through the points $\mathbf{n}_{wrap}$, which are shown in the figure.

It is important to remember that the algorithm does not have prior knowledge of the FM, which is computed in real time as needed, which means that only a portion of the FM will be computed by the time the algorithm finishes. The complete FM is shown in Figure 4.10 only for illustrative purposes, and is not computed by the algorithm, nor taken into account for the runtime evaluation.

## 4.3 AUGMENTED FEASIBILITY MAPS

By means of Feasibility Maps, we have successfully solved the redundancy resolution problem for a given task trajectory. That is, given a task trajectory $\mathbf{x}(t)$,

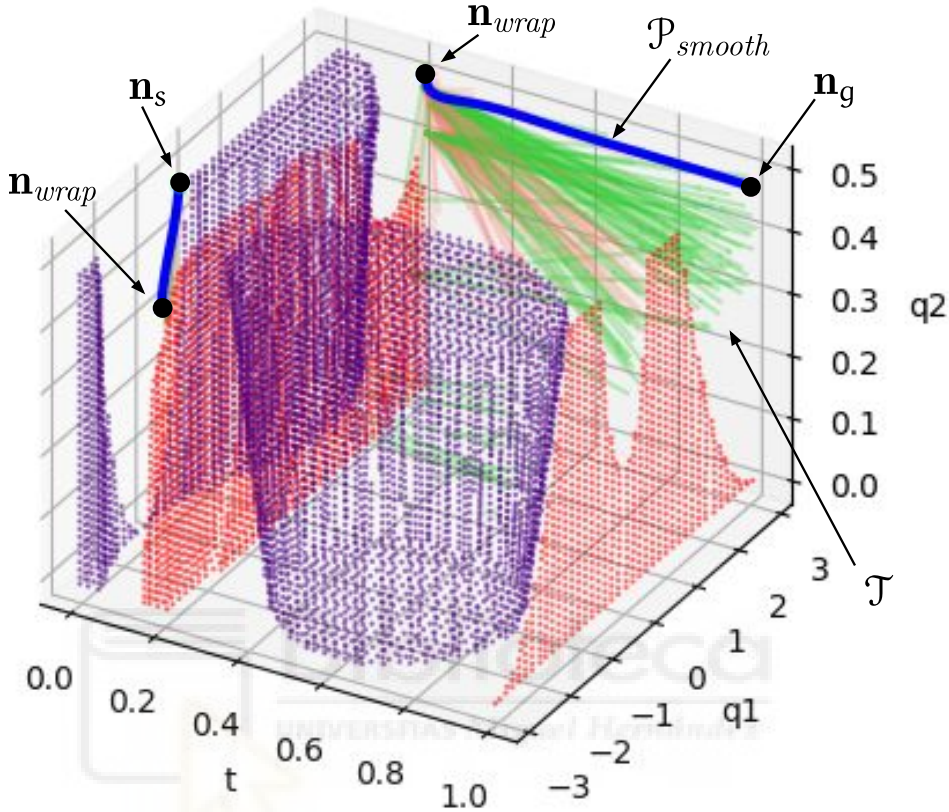


Figure 4.10: Feasibility map and solution paths for the RPR robot and 1-dimensional task, producing a 2-dimensional FM.

the algorithm is able to determine the time history values of the extra (redundant) joints, from which a joint trajectory $\mathbf{q}(t)$ that successfully completes the task trajectory can be obtained.

However, in some applications (e.g., if we were to follow the discrete foothold path given by the algorithm of Chapter 3), only the start $\mathbf{x}_s$ and end $\mathbf{x}_g$ task points are known, and a task trajectory that connects these two points must be generated. The problem of obtaining a task trajectory $\mathbf{x}(t)$ that connects a starting task point $\mathbf{x}_s$ and a goal task point $\mathbf{x}_g$, in a given time $t_g$, is known as *task trajectory generation*.

In the following section, we propose a method to simultaneously solve the redundancy resolution and task trajectory generation problems. This is done by

means of *Augmented Feasibility Maps* (AFMs), which are an extension of FMs that include the task space as additional dimensions. An AFM is defined as:

$$\mathcal{AFM} = \left\{ \begin{bmatrix} t \\ \mathbf{x}_a \end{bmatrix} \;\middle|\; \mathbf{x}_a = \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{x}_r \end{bmatrix}, \mathbf{q} = \mathbf{g}_a(\mathbf{x}_a), \mathbf{q} \in \mathcal{Q} \right\} \tag{4.11}$$

where each individual term is defined as in Equation (4.5).

To better understand the novel concept of AFMs, let us consider the example of the 2R robot of Section 4.2.2. However, instead of having a predefined task trajectory of the vertical position of the end effector, we only know its goal value $p_y(t_g) = 0$ m, to be reached in $t_g = 1$ s. Since $p_y$ can be freely chosen, the task space can be included as an additional dimension in the AFM, which is now 3-dimensional, and results in a 3D map in the space $(t, p_y, q_1)$ (remember that $q_1$ was chosen as the single redundant parameter).

This example AFM is shown in Figure 4.11(b). This map includes three groups of regions that, for clarity, have been represented separately in Figures 4.11(c-e). Figure 4.11(c) shows the regions, coloured in purple, whose points, when substituted into the IK function $\mathbf{g}_a(\cdot)$, return non-real solutions of $\mathbf{q}$. The region of AFM points that map to configurations that lead to collisions with the obstacle is shown in red in Figure 4.11(d). Finally, Figure 4.11(e) shows the parabolic sheet that is parametrically defined as: $\{(t, p_y, q_1) \mid 0 < t < 1, p_y = -6.662t^2 + 8.162t - 1.5, -\pi < q_1 < \pi\}$. This sheet represents the task trajectory specified in the example of Section 4.2.2. In Figure 4.11(e), it is drawn in green, along with the intersections between this sheet and the purple and red forbidden regions. In fact, the projection of this sheet on the plane $(t, q_1)$ (remember that it is the space of the FM) yields the conventional FM shown in Figure 4.1(a).

Note that, by not having a fixed task trajectory $p_y(t)$, the path in the AFM is not constrained to move along the green parabolic sheet, but can move freely in the three-dimensional feasible space that is free of collisions and non-real solutions. This freedom increases the chances of finding a task and joint trajectory that is feasible, which would correspond to a path in the AFM that joins the initial joint configuration of the robot with any feasible point of the *goal set*, whose definition can also be extended to the AFM as: $\mathcal{G} = \{(t, p_y, q_1) \mid t = 1, p_y = 0, -\pi < q_1 < \pi\}$.

Generalizing the previous example, in the following section we propose a method to solve the motion of redundant manipulators to achieve a goal task point $\mathbf{x}_g$ in a given time $t_g$, by means of AFMs. These two constraints define an r-dimensional goal set $\mathcal{G}$, which is a subset of the AFM where the coordinates $t$ and $\mathbf{x}$ are fixed to their goal values $t_g$ and $\mathbf{x}_g$, respectively, and the remaining dimensions of $\mathbf{x}_a$ (i.e., the redundant parameters $\mathbf{x}_r$) are free to vary, as long as

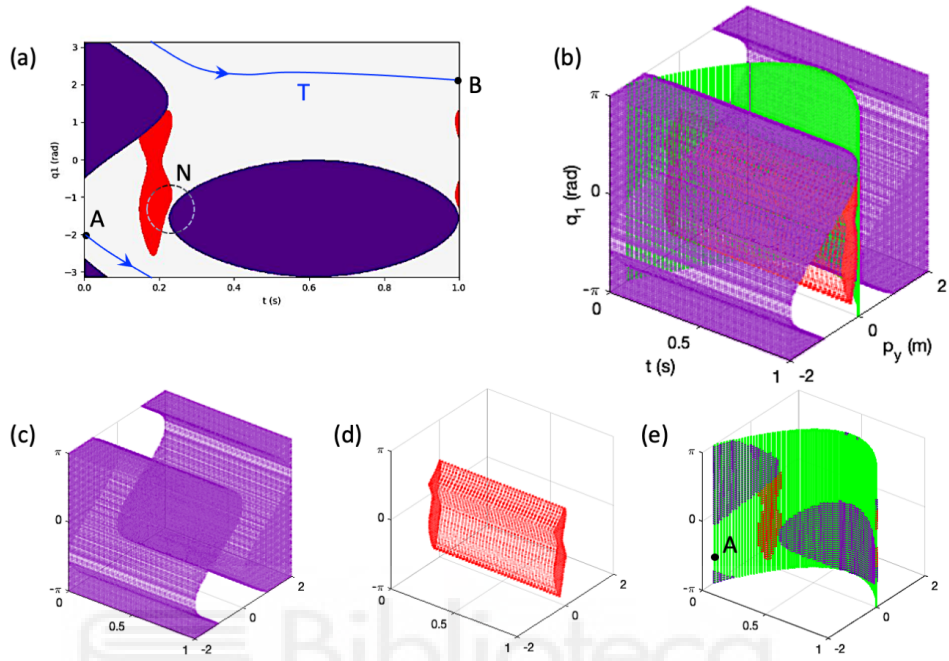


Figure 4.11: (a) Example of an $r = 1$ FM. (b) Example AFM that includes the task coordinate $p_y$. (c) Regions that lead to non-real solutions. (d) Region that leads to collisions. (e) Parabolic sheet corresponding to the task $p_y = -6.662t^2 + 8.162t - 1.5$.

other additional constraints are met (e.g., joint limits, obstacle avoidance, which are dictated by $\mathcal{Q}$):

$$\mathcal{G} = \left\{ \begin{bmatrix} t_g \\ \mathbf{x}_g \\ \mathbf{x}_r \end{bmatrix} \;\middle|\; t = t_g, \mathbf{x} = \mathbf{x}_g, \mathbf{x}_r \in \mathcal{Q} \right\}. \tag{4.12}$$

The method we propose next explores the free space of the AFM $(t, \mathbf{x}, \mathbf{x}_r)$ to find a feasible path connecting the initial joint configuration of the manipulator $\mathbf{q}_s$ with a point in the goal set $\mathcal{G}$, simultaneously solving task trajectory generation (i.e., time history of the task $\mathbf{x}(t)$) and redundancy resolution (i.e., time history of the redundant parameters $\mathbf{x}_r(t)$).

## 4.4 AFMs for simultaneous task trajectory generation and redundancy resolution

By simultaneously solving the redundancy resolution and task trajectory generation problems, we are able to address some the main limitation of the FM approach. That is, the given fixed task trajectory $\mathbf{x}(t)$ may not always be feasible. For example, consider the FM shown in Figure 4.11(a). Assuming that the initial point of the path is the one marked as A $(t = 0, q_1 = -2)$, a valid path in the FM that achieves $\mathbf{x}(t)$ should connect point A with any point in the goal set $(t = 1)$, e.g., point B. The path T shown in the figure is a valid path that connects A with B, but only because we allow for the joint $q_1$ to wrap at $\pm\pi$. If we were to restrict the joint limits to $q_1 \in [-\pi, \pi]$, the path T would not be feasible, and the only feasible path would need to traverse the narrow corridor N. If the task trajectory $\mathbf{x}(t)$ were slightly perturbed, it may occur that the red and purple regions near N touch, and the given task trajectory would not be feasible. By using AFMs, we can additionally explore the space of feasible task trajectories, and guarantee that the returned task trajectory is feasible, if it exists.

Other works have proposed methods to solve the redundancy resolution and task trajectory generation problems simultaneously. For instance, Tassi et al. (2021) proposed a method based on hierarchical quadratic optimization that addresses both problems at once, by augmenting the decision variables of the optimization problem with the task dimensions. The authors of (Zhou et al., 2023) also proposed a method that solves both problems simultaneously, by means of a two-stage approach. First, an RRT is built by sampling points in the task dimensions. Then, for each node of the tree, self-motion manifolds are computed, from which a joint trajectory is derived based on an exhaustive search that minimizes an average performance index (e.g., reciprocal condition number).

We propose a method based on the RRT algorithm, which we have previously used to solve the redundancy resolution problem in FMs. This time, however, we employ the RRT* variant (Karaman and Frazzoli, 2011), which is an extension of the original RRT algorithm that guarantees asymptotic optimality, and is better suited for planning in AFMs. In this section, we present the modifications required to adapt the conventional RRT* to explore the AFM space. The general structure of the conventional RRT* is maintained and showed in Algorithm 4.2, with the main differences being in the specific definitions of the procedures.

Algorithm 4.2 starts by initializing the tree $\mathcal{T}$ with the root node $\mathbf{n}_0 = \mathbf{s}_0$, where $\mathbf{s}_0$ is the initial state of the manipulator. We define a state as a point in the AFM space $\mathbf{s} = [t, \mathbf{x}, \mathbf{x}_r]$, while a node $\mathbf{n}$ is an element of the tree $\mathcal{T}$, which contains a state $\mathbf{s}$, and a pointer to its parent node $\mathbf{n}_{parent}$. Recall that any state $\mathbf{s}$ (i.e., a point in the AFM) can be mapped to a joint configuration $\mathbf{q}$ by means of the augmented IK function $\mathbf{g}_a(\cdot)$.

**Algorithm 4.2** RRT* in AFMs

1: Initialize tree $\mathcal{T}$ with root node $\mathbf{n}_0 = \mathbf{s}_0$
2: **for** $i = 1, 2, \ldots, i_{max}$ **do**
3: $\quad \mathbf{s}_{random} \leftarrow$ SampleState($\alpha$, $\mathcal{G}$)
4: $\quad \mathbf{n}_{near} \leftarrow$ NearestNode($\mathbf{s}_{random}$, $\mathcal{T}$)
5: $\quad \mathbf{n}_{new} \leftarrow$ Steer($\mathbf{n}_{near}$, $\mathbf{s}_{random}$, $\Delta s$)
6: $\quad$ **if** FeasibleConnection($\mathbf{n}_{near}$, $\mathbf{n}_{new}$) **then**
7: $\quad\quad \mathcal{N} \leftarrow$ Neighbours($\mathbf{n}_{new}$, $\mathcal{T}$, rad)
8: $\quad\quad \mathbf{n}_{parent} \leftarrow$ BestParent($\mathcal{N}$, $\mathbf{n}_{new}$)
9: $\quad\quad$ Add $\mathbf{n}_{new}$ to $\mathcal{T}$ with parent $\mathbf{n}_{parent}$
10: $\quad\quad \mathcal{T} \leftarrow$ Rewire($\mathcal{T}$, $\mathcal{N}$, $\mathbf{n}_{new}$)
11: $\mathcal{P} \leftarrow$ BestPath($\mathcal{T}$)

Next, the loop iterates $i_{max}$ times, and, for each iteration, a random state $\mathbf{s}_{random}$ is sampled by means of the SampleState function. This function generally samples a point in the domain of the AFM. The domain of the $t$ dimension is $[0, t_g]$, where $t_g$ is the user-defined duration of the trajectory. The task dimension $\mathbf{x}$ is constrained by the workspace of the robot. Therefore, a prior workspace analysis should be performed in order to determine the sampling domain of $\mathbf{x}$. Finally, the redundant parameters $\mathbf{x}_r$ are constrained by the joint limits of the robot. The SampleState function should sample points in the AFM that are feasible, i.e., that do not lead to non-real solutions or collisions. Consequently, if a sampled point in the domain is deemed as infeasible (i.e., it leads to non-real solutions or collisions), the point is discarded, and a new one is sampled. Additionally, to guide the AFM exploration towards the goal $\mathbf{x}_g$, samples are drawn from the goal set $\mathcal{G}$, which is defined in Equation (4.12), with probability $\alpha$, which typically ranges from 0.05 to 0.2 (i.e., at most, 20% of the sampled states will belong to $\mathcal{G}$).

In line 4, the nearest node $\mathbf{n}_{near}$ to the random state $\mathbf{s}_{random}$ is found by means of the NearestNode function. This function performs a nearest neighbour search in the tree $\mathcal{T}$, and returns the node that is closest to the random state. The distance metric by which the nearest node is found must be given by the user. For example, similarly to the FM case, the weighted norm of the differences between the states could be used as the cost function:

$$c(\mathbf{s}_1, \mathbf{s}_2) = \sqrt{(\mathbf{s}_2 - \mathbf{s}_1)^T \mathbf{W} (\mathbf{s}_2 - \mathbf{s}_1)} \tag{4.13}$$

where $\mathbf{W}$ is a positive definite diagonal matrix that weights the importance of each dimension of the AFM, due to the non homogeneous nature of the space. It is worth noting that the NearestNode procedure must consider the nature of the time dimension $t$ when evaluating the potential nearest node. Since time is

strictly monotonically increasing (i.e., time cannot go backwards), only the states with a time value less than that of the random state $\mathbf{s}_{random}$ should be considered as potential nearest nodes.

The STEER function in line 5 is responsible for generating a new state $\mathbf{s}_{new}$ by displacing the nearest node $\mathbf{n}_{near}$ towards the random state $\mathbf{s}_{random}$ by a distance $\Delta s$. This distance $\Delta s$ is a user-defined parameter that dictates the maximum step size of the exploration. This procedure is exemplified in green in Figure 4.12(a), where $\mathbf{n}_{new}$ is generated by displacing $\mathbf{n}_{near}$ towards $\mathbf{s}_{random}$ by a step size of $\Delta s$.

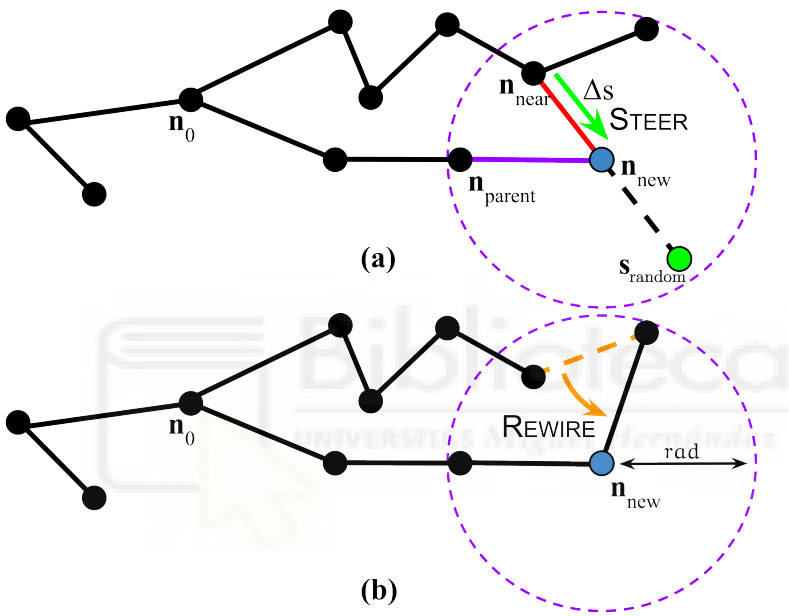


Figure 4.12: Illustration of the procedures of the RRT* algorithm.

Next, procedure FEASIBLECONNECTION (line 6) checks whether the connection between the nearest node $\mathbf{n}_{near}$ and the new node $\mathbf{n}_{new}$ is feasible (red segment in Figure 4.12(a)). This is done by checking if the path connecting the two nodes is free of collisions and non-real solutions, in the same way as the FM algorithm, randomly sampling points along the path. If the connection is feasible, the algorithm proceeds to the two characteristic functions of the RRT* variant: the selection of the best parent, and the rewiring of the tree.

Before selecting the best parent node, the NEIGHBOURS function (line 7) is called to find the set of nodes $\mathcal{N}$ that are within a radius $\mathtt{rad}$ of the new node $\mathbf{n}_{new}$. Figure 4.12 shows the purple circle that encloses the neighbours $\mathcal{N}$ of the new node $\mathbf{n}_{new}$.

Then, the BestParent procedure in line 8 evaluates the nodes in the set $\mathcal{N}$ to select the best parent node $\mathbf{n}_{parent}$. The best parent is the one that minimizes the cost function $c(\cdot)$, which is defined in Equation (4.13), for the entire path from the root node $\mathbf{s}_0$ to $\mathbf{n}_{new}$, routing through $\mathbf{n}_{parent}$. The cost of the entire path is computed by backtracking from $\mathbf{n}_{new}$ to its parent node, and so on, until the root node $\mathbf{s}_0$ is reached, accumulating the cost of each segment. When the best parent is found, the new node $\mathbf{n}_{new}$ is added to the tree $\mathcal{T}$ with the parent $\mathbf{n}_{parent}$ (line 9). In Figure 4.12(a), the node $\mathbf{n}_{new}$ points to the best parent node $\mathbf{n}_{parent}$ (purple segment), instead of the nearest node $\mathbf{n}_{near}$ (red segment), which the original RRT algorithm would have chosen.

Finally, the Rewire function (line 10) is called to rewire the tree $\mathcal{T}$. This function evaluates the nodes in the set $\mathcal{N}$, and checks if the new node $\mathbf{n}_{new}$ is a better parent than the current parent of each node in $\mathcal{N}$. That is, the function checks if the cost of the path from $\mathbf{n}_0$ to $\mathbf{n}_{new}$ is lower than the cost of the path from the root node to the current parent of each node in $\mathcal{N}$. If the path is improved for any neighbour, the parent of such neighbour is updated to be $\mathbf{n}_{new}$, and the tree is rewired accordingly. This procedure is exemplified in Figure 4.12(b).

The algorithm finishes after the loop is completed, and the best path $\mathcal{P}$ is extracted from the tree $\mathcal{T}$ by means of the BestPath function. Computing the total cost in a similar manner to the BestParent and Rewire functions, every path that connects the root node $\mathbf{n}_0$ to the goal set $\mathcal{G}$ is evaluated, and the path with the lowest cost is selected as the best path $\mathcal{P}$.

Similarly to the FM case, the nature of RRT-based algorithms causes the returned path to be non-smooth, presenting discontinuous velocities and infinite accelerations at the connection points between nodes. Therefore, a smoothing procedure is required to obtain a feasible and smooth path. As we did for the FM case, we propose employing a cubic b-spline interpolation along a series of intermediate control points. This time, however, instead of discretizing the path in a predefined number of control points, we propose using the same step size $\Delta s$ that was used to generate the tree, in order to discretize the path in a number of control points that is proportional to the length of the path. Since $\Delta s$ is a small value, the resulting b-spline will not deviate significantly from the original path, reducing the risk that the smoothing procedure provokes entering forbidden regions. This smoothing procedure is shown in Figure 4.13.

### 4.4.1 *Example 1: 3-dimensional AFM*

In the following two examples, we demonstrate the application of the RRT* algorithm in AFMs to solve the redundancy resolution and task trajectory generation problems simultaneously, for different combinations of robots and objectives. The shown examples are simulations that were run on a machine with an Intel

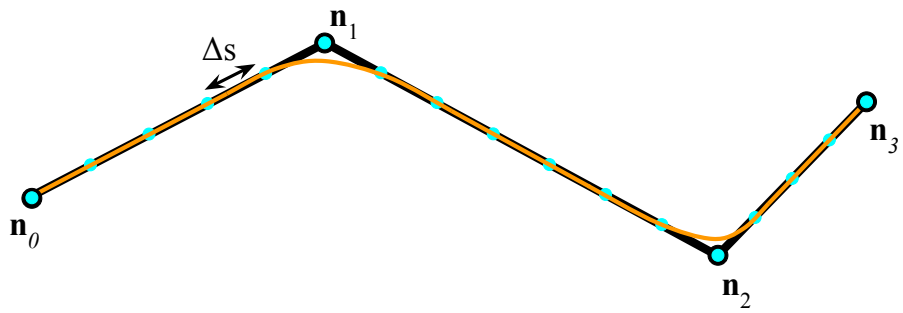


Figure 4.13: Smoothing procedure for the path found by the RRT* algorithm in the AFM.

Core i7-9700F CPU @ 3.00 GHz, with 32 GB of RAM, running Ubuntu 24.04., and implemented in Python 3.12.

In this first example, we consider the same 2R robot as in Section 4.2.2, which is shown in Figure 4.2. However, this time we do not have a predefined task trajectory, but only the goal vertical position of the end effector $p_y(t_g) = 0$ m, to be reached in $t_g = 1$ s. Additionally, this time we consider the joint limits for the revolute joints to be $q_i \in [-\pi, \pi]$ rad, thus preventing the joints from wrapping. The same elliptical obstacle is present in the workspace, which the end-effector must avoid, i.e., $\frac{(p_x-1.1)^2}{1^2} + \frac{(p_y+0.2)^2}{0.25^2} \leqslant 1$. The first joint $q_1$ is still selected as the single redundant parameter in $\mathbf{x}_r$.

For this example, we have set the following RRT* parameters: $\Delta s = 0.2$, $\mathtt{rad} = 0.4$, $i_{max} = 1000$, and $\alpha = 0.2$. The cost function $c(\cdot)$ corresponds to the one defined in Equation (4.13), with $\mathbf{W} = \mathbf{I}$.

Figure 4.14 illustrates the results of the simulation. The AFM is plotted in Figure 4.14(a), but only for illustrative purposes, as the algorithm does not require prior knowledge of the AFM. The tree $\mathcal{T}$ is also plotted in Figure 4.14(a) in blue, with the returned best path in magenta. Additionally, Figure 4.14(b) shows the corresponding robot motion, for different time instants, from $t = 0$ s to $t = 1$ s.

As we did for the FM case, we have performed a runtime and cost analysis for different values of $i_{max}$. In Figure 4.15, the cost of the path found by the algorithm is plotted as a function of the runtime, for different values of $i_{max}$ (from 100 to 2100, in steps of 100). As expected, the execution times proportionally increase with the number of iterations. Regarding the cost of the path, although some variability is observed, the average cost tends to decrease as the number of iterations increases.

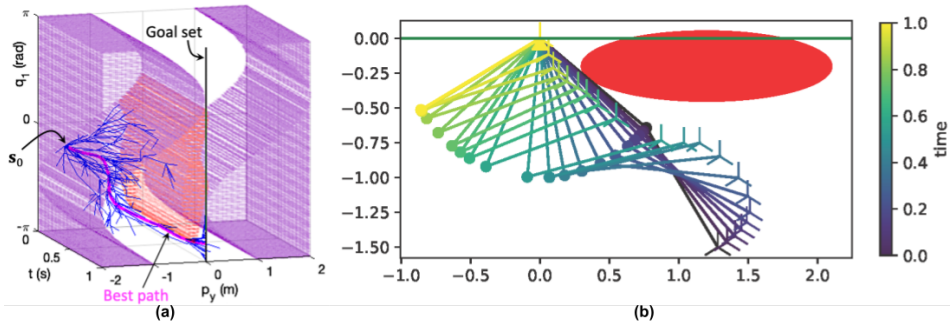


Figure 4.14: (a) AFM and RRT* solution for the 2R robot and the given goal task and duration. (b) Corresponding robot motion.

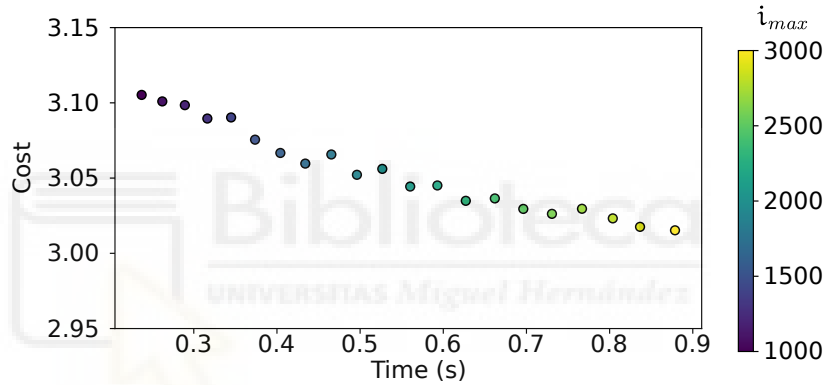


Figure 4.15: Cost as a function of runtime for 21 different values of $i_{max}$, averaged over 1000 runs.

### 4.4.2 *Example 2: 4-dimensional AFM*

The second example considers the same RPR robot as in Section 4.2.3, which is shown in Figure 4.8, and also recalled in Figure 4.16(a). The task remains 1-dimensional, which produces a level of redundancy of $r = 2$ and a 4-dimensional AFM, but, once again, instead of having a predefined task trajectory, we only know the goal vertical position of the end effector $p_y(t_g) = 0$ m, to be reached in $t_g = 1$ s.

This time, however, in order to enhance the complexity and realism of the example, and to showcase the method's efficiency and flexibility, we consider whole-body collision avoidance, rather than just end-effector obstacle avoidance. For this purpose, we have used the Coal library (Pan et al., 2015–2024) to com-

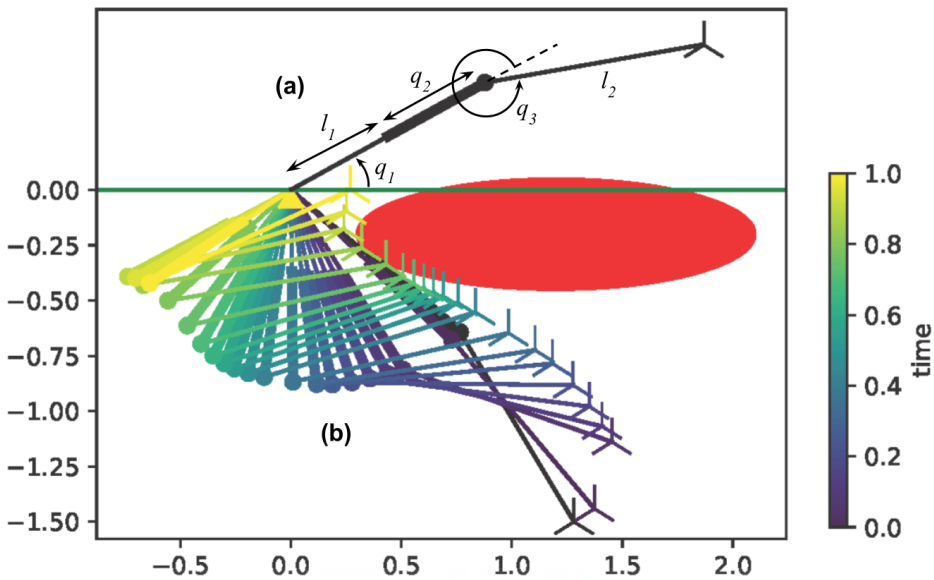


Figure 4.16: Robot motion for the path returned by the algorithm.

pute robot-environment collisions. The obstacle that the whole robot must avoid remains the same as in the previous example.

The prismatic joint $q_2$ is restricted to $q_2 \in [0, 1]$ m, and the revolute joints $q_1$ and $q_3$ are restricted to $q_i \in [-\pi, \pi]$ rad. The redundant parameters $\mathbf{x}_r$ are the joint angles $q_1$ and $q_2$, i.e., $\mathbf{x}_r = [q_1, q_2]^T$.

The same RRT* parameters as in the previous example are used: $\Delta s = 0.2$, $rad = 0.4$, $i_{max} = 1000$, and $\alpha = 0.2$. The cost function also remains unchanged.

Direct visualization of the 4-dimensional AFM is not possible, but different time values of the robot motion for the resulting path are shown in Figure 4.16(b). The average runtime for 100 runs of the algorithm is 97.3 ms, which, considering that whole-body collisions are evaluated (which is a computationally expensive task), is a very efficient runtime. This aligns with the results of the FM case, where the results of the algorithm suggest better performances for higher degrees of freedom, probably due to the percentage of feasible space in the AFM increasing with the number of dimensions.

## 4.5 CONCLUSIONS

In this chapter, we have presented a novel method to solve the redundancy resolution by means of Feasibility Maps. The method is based on the RRT algorithm,

which is used to explore the FM space and find a feasible path that connects the initial joint configuration of the robot with a point in the r-dimensional goal set. Any path that fulfils this condition is a feasible joint trajectory that completes the given task trajectory. The method has been validated in simulations with 2R and RPR robots, and 1-dimensional tasks, producing 2-dimensional and 3-dimensional FMs, respectively. The results of the simulations show that the algorithm is able to find feasible paths in the FM space, and that the performance of the algorithm increases with the number of degrees of redundancy of the robot.

We have also proposed an extension of the FM concept, called Augmented Feasibility Maps, which include the task space as additional dimensions. This allows for simultaneously solving the redundancy resolution and task trajectory generation problems. The method we propose is based on the RRT* algorithm, which explores the AFM space to find a feasible path that connects the initial joint configuration of the robot with a point in the goal set. The method has been validated in simulations with 2R and RPR robots, and 1-dimensional tasks, producing 3-dimensional and 4-dimensional AFMs, respectively. Once again, the results suggest that these methods offer great scalability and efficiency for higher degrees of redundancy, because the free feasible space increases, easing the generation of feasible paths in the AFM.

Future work should focus on testing the proposed methods in real-world scenarios, and on extending the methods to solve the redundancy resolution and task trajectory generation problems for more complex robots and tasks. Additionally, especially in the case of the AFM method, the inclusion of dynamic environments, dynamic constraints, such as torque or acceleration limits, should be considered. Non-holonomic constraints could also be included in the method. Finally, the methods could be extended to consider the possibility of transitioning between different extended aspects, qualifying the method as global redundancy resolution. However, as we will see in the following chapters, we believe that there exist better methods to solve the redundancy resolution problem globally, e.g., using Self-Motion Manifolds.

# 5

# SELF-MOTION MANIFOLDS FOR GLOBAL REDUNDANCY RESOLUTION

In the previous chapter, we presented a solution to the redundancy resolution problem based on the concept of Feasibility Maps. However, as discussed in Section 2.3, approaches that rely on a single FM do not provide a fully-global solution to the problem. A global solution is attainable only by appropriately managing the transitions between every FMs. This limitation arises because an FM represents a specific projection (corresponding to a particular extended aspect) of a broader concept that identifies the infinite set of joint configurations satisfying the constraints of the problem: the SMD.

The SMD can be understood as the manifold obtained by stacking SMMs along the task trajectory. Each SMM describes the set of all possible configurations of a robot that can achieve a specific task point. Consequently, the SMD characterizes the complete set of configurations available to a manipulator for achieving a given task trajectory.

In this chapter, we introduce the concepts of SMM and SMD, describe the algorithms for computing these manifolds, and propose a framework for global redundancy resolution based on the SMD.

The chapter is organized as follows. In Section 5.1, we define the concepts of SMM and SMD and describe important concepts. The main contribution of this chapter is presented in Section 5.2, where we introduce a framework for global redundancy resolution based on the analysis of the SMMs that compose the SMD. First, we describe the generation of the SMMs and the graph that represents transformations between them. Then, we present a method for efficient generation of joint trajectories based on the SMMs and the graph. Finally, we present a method for optimizing the generated trajectories. Additionally, we present variants of the different stages of the framework that can be used to tailor the framework to specific applications. In Section 5.3, we present experimental results demonstrating the performance of the proposed framework on three different examples of different complexity and dimensionality. In Section 5.4, we discuss the global optimality of the proposed framework, analyse its complexity and dimensionality, and present direct comparisons with state-of-the-art methods. Finally, in Section 5.5, we summarize the main contributions of this chapter and present conclusions.

This chapter reproduces the work presented in (Fabregat-Jaén et al., 2025). Some fragments of the text have been reproduced from the original work, while

others have been modified to fit the context of this thesis. Figures, algorithms, and tables from the original work have also been reused.

## 5.1 SELF-MOTION MANIFOLDS

The following Equation is the constraint that describes the relationship between a joint-space configuration $\mathbf{q} \in \mathbb{R}^n$ and a task-space point $\mathbf{x} \in \mathbb{R}^m$:

$$\mathbf{F}(\mathbf{x}, \mathbf{q}) = \mathbf{0}_{m \times 1}. \tag{5.1}$$

The IKP is the problem of finding a joint-space configuration $\mathbf{q}$ that satisfies the constraint (5.1) for a given task-space point $\mathbf{x}$. As we have already discussed in this thesis, for redundant manipulators, where $n > m$, the IKP yields an infinite number of solutions. That is, there exist infinitely many values of $\mathbf{q}$ that satisfy Equation (5.1) for a given $\mathbf{x}$.

Generically, a finite number of $r$-dimensional manifolds gather these solutions, where $r$ is the *degree of redundancy*, defined as the difference between the DoFs of the manipulator and the dimensionality of the task, i.e., $r = n - m$. Burdick (1989) introduced the concept of *self-motion manifolds* to describe these manifolds, since motion along them does not affect the task-space point $\mathbf{x}$. By providing a global description of the infinite set of solutions to the IKP, SMMs are a powerful tool for global redundancy resolution.

Let $\mathbf{q} = [q_1, q_2]^T \in \mathbb{R}^2$ be the joint-space configuration of a hypothetical 2-DoF manipulator, executing a given one-dimensional task trajectory $\mathbf{x}(t) = [x]^T \in \mathbb{R}^1$, represented by the function plotted in Figure 5.1(a). This combination of joint-space and task-space dimensions results in a degree of redundancy of $r = 1$, which produces a one-dimensional SMMs (i.e., curves) in the joint-space for a fixed value of $\mathbf{x}$. Figure 5.1(b) illustrates a hypothetical SMM for the time value $t = f$ of the task trajectory. These curves (SMMs $\mathbf{M}_f^1$ and $\mathbf{M}_f^2$) are the set of joint configurations $\mathbf{q}$ that satisfy the constraint (5.1) for the task-space point $\mathbf{x}(f)$. The existence of two disjoint (not connected) SMMs means that the manipulator will not be able to transition from a joint-space configuration on one SMM to a joint-space configuration on the other SMM without violating the constraint (5.1).

If, instead of a single task-space point $\mathbf{x}(f)$, we consider the entire task trajectory $\mathbf{x}(t)$, where $t$ is an arc-length parameterization of the trajectory (from now on, for simplicity, we will refer to it as *time*), the infinite set of solutions $\mathbf{q}$ that satisfy $\mathbf{F}(\mathbf{x}(t), \mathbf{q}) = \mathbf{0}$ is a $(r+1)$-dimensional manifold. This manifold was coined *self-motion domain* in (Yang et al., 2021), and is illustrated in Figure 5.1(c). Every slice of the SMD at a fixed time $t$ result in the SMMs for the task-space point $\mathbf{x}(t)$ (see how the SMMs in Figure 5.1(b) are slices of the SMD at time $t = f$ in Figure 5.1(c)). The SMMs of three different time values ($t = \{e, f, g\}$) are highlighted in

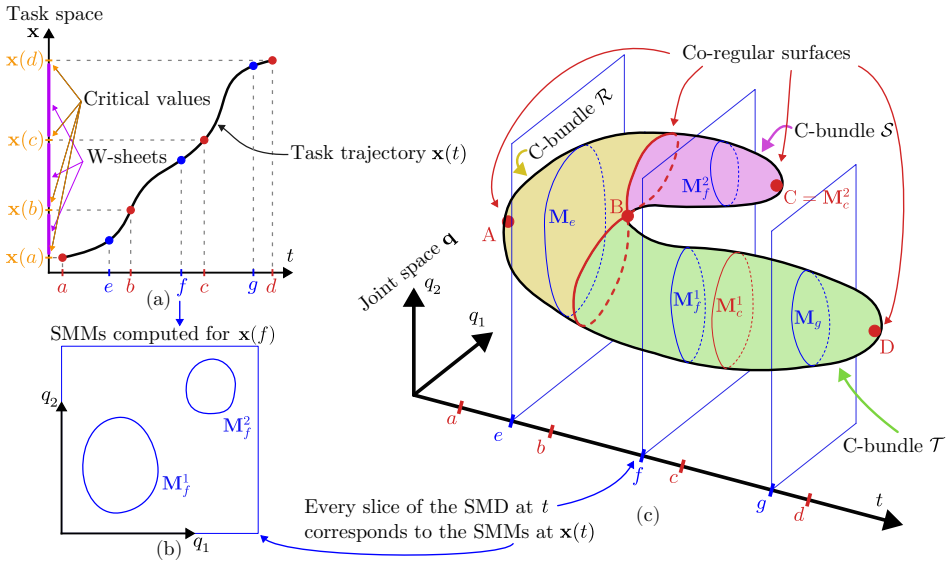


Figure 5.1: (a) Task trajectory $\mathbf{x}(t)$ of a 2-DoF manipulator. (b) Two disjoint SMMs $\mathbf{M}_f^1$ and $\mathbf{M}_f^2$ for the task-space point $\mathbf{x}(f)$. (c) The SMD is the manifold of all possible configurations of the manipulator that satisfy the constraint (5.1) for the entire task trajectory $\mathbf{x}(t)$.

blue in Figure 5.1(c), as a result of the intersection of the SMD with the planes orthogonal to the time axis at the corresponding time values.

When observing the highlighted SMMs in Figure 5.1(c), it is clear that some transformation has occurred between them when advancing in time (i.e., when completing the task trajectory). For time values $t = e$ and $t = g$, the SMMs are single one-dimensional curves, denoted as $\mathbf{M}_e$ and $\mathbf{M}_g$, respectively. Joint configurations on these SMMs map to $\mathbf{x}(e)$ and $\mathbf{x}(g)$, respectively. Therefore, $\mathbf{M}_e$ and $\mathbf{M}_g$ are the *preimages* or solutions to the IKP for the task-space points $\mathbf{x}(e)$ and $\mathbf{x}(g)$, respectively. Conversely, the SMMs at time $t = f$ are two disjoint curves, denoted as $\mathbf{M}_f^1$ and $\mathbf{M}_f^2$, which are the preimages of the task-space point $\mathbf{x}(f)$. Since the time value $t = f$ is between $t = e$ and $t = g$, it can be concluded that some transformation in the SMMs has occurred as the task trajectory was traversed, since the number of disjoint SMMs has changed from one, to two, and back to one.

To better understand the implications of these transformations in the presented framework, it is important to introduce some relevant concepts presented in (Burdick, 1989). Let $\mathbf{J}(\mathbf{q})$ be the Jacobian matrix that relates the joint-space velocities $\dot{\mathbf{q}}$ to the task-space velocities $\dot{\mathbf{x}}$: $\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$. Points in the SMD where the Jacobian matrix $\mathbf{J}(\mathbf{q})$ loses rank are *singularities*. In the SMD of Figure 5.1(c), sin-

gular points are represented by the red dots $\{A, B, C, D\}$. Projecting the singular points onto the time axis $t$ results in time values $\{a, b, c, d\}$. Corresponding task-space points $\{\mathbf{x}(a), \mathbf{x}(b), \mathbf{x}(c), \mathbf{x}(d)\}$ in Figure 5.1(a) are *critical values* of the task trajectory. Figure 5.1(a) depicts one additional concept: *w-sheets*, which are the intervals of the task space bounded by two consecutive critical values.

Two additional concepts can be derived, which are especially relevant for the proposed framework. Preimages of critical values (i.e., the joint-space configurations $\mathbf{q}$ that satisfy the constraint (5.1) for a critical value $\mathbf{x}$) are $r$-dimensional sets, where a *co-regular surface* exists. The term is inspired by the fact that these sets contain both regular points and singularities (as co-regular surface at $t = b$ in Figure 5.1(c) depicts). Note that the use of the term *surface* may be misleading, as the dimensionality of the co-regular surface is $r$ and not always 2, which would produce a surface (e.g., in the case of $r = 1$, the co-regular surface is a curve). Co-regular surfaces may also be formed by a single point, as in the case of singularities A, C, and D in Figure 5.1(c), instead of coexisting with regular points. Finally, similar to the relationship between w-sheets and critical values, *c-bundles* are the disjoint components of the SMD that are bounded by co-regular surfaces. The SMD in Figure 5.1(c) can be divided into three c-bundles: $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{T}$. The c-bundle $\mathcal{R}$ is bounded by the co-regular surfaces at time values $t = a$ and $t = b$, while the c-bundle $\mathcal{S}$ is bounded by the co-regular surfaces at time values $t = b$ and $t = c$. Finally, the c-bundle $\mathcal{T}$ is bounded by the co-regular surfaces at time values $t = b$ and $t = d$. The key feature of c-bundles is that the SMMs that compose them share the same topology, which means that no topological transformations, or change in number of disjoint SMMs, occur when traversing the c-bundle. Consequently, it can be concluded that the transformations of the SMMs occur at the boundaries of the c-bundles, which are the co-regular surfaces. In the presented framework, we will exploit this property to efficiently solve the redundancy resolution problem.

### 5.1.1 *Transformations of self-motion manifolds*

Our proposed redundancy resolution framework is based on the analysis of the transformations of the SMMs that compose the SMD. Four types of transformations can occur in SMMs when traversing co-regular surfaces: *creation*, *merging*, *splitting*, and *vanishing*.

Let us consider the SMD in Figure 5.1(c) when traversed positively in time. For values $t < a$, there are no joint-space configurations that satisfy the constraint (5.1) for the task-space trajectory $\mathbf{x}(t)$. At $t = a$, the first critical value is reached, and the preimage of the critical value $\mathbf{x}(a)$ is a co-regular surface that contains a single point (the singularity $A$). From this time value onwards, the SMD contains a single SMM. Therefore, it can be concluded that the crossing of the co-regular

surface at time $t = a$ results in the creation of a new SMM. This singularity also marks the beginning of the c-bundle $\mathcal{R}$, which is bounded by the co-regular surfaces at time values $t = a$ and $t = b$, and ensures that the SMMs in this c-bundle share the same topology, thus no transformations occur.

At time $t = b$, the preimage of task-space point $\mathbf{x}(b)$ is a co-regular surface that contains singularity B. When crossing this co-regular surface in the positive time direction, the SMM in the c-bundle $\mathcal{R}$ splits into two disjoint SMMs, which also produces the splitting of the c-bundle $\mathcal{R}$ into two new c-bundles: $\mathcal{S}$ and $\mathcal{T}$. Note that this splitting transformation is of particular interest for redundancy resolution, as incorrectly managing the transition may result in leading to an empty solution space, not being able to reach the desired task-space point. For example, if the joint configuration were to end up in the c-bundle $\mathcal{S}$ when traversing time value $t = b$, the robot would not be able to reach the end of the task trajectory at time value $t = d$ without violating the constraint (5.1). The reason for this is that the SMMs in the c-bundle $\mathcal{S}$ would progressively shrink as approaching singularity C, eventually leading to the vanishing of the SMM when crossing $t = c$, and preventing the robot from reaching the task-space point $\mathbf{x}(d)$. On the contrary, if the joint configuration ends up in the c-bundle $\mathcal{T}$, the joint configuration can be smoothly moved along the SMD to the task-space point $\mathbf{x}(d)$.

One additional transformation can be observed when the time dimension is traversed negatively. When crossing the co-regular surface at time $t = b$, the two disjoint SMMs in c-bundles $\mathcal{S}$ and $\mathcal{T}$ merge into a single SMM in the c-bundle $\mathcal{R}$. Although the merging operation is not as relevant for redundancy resolution as the splitting operation, since there is no risk of ending up in an empty solution space, it is still important to consider, as it dictates connectivity between c-bundles, which will be used to generate the graph that represents the SMD in the proposed framework.

## 5.2 Framework for Global Redundancy Resolution Based on Self-Motion Manifolds

The goal of a redundancy resolution method is to find an optimal joint trajectory $\mathbf{q}(t)$ that satisfies the constraint (5.1) for a given task trajectory $\mathbf{x}(t)$. Moreover, additional kinematic constraints, such as joint limits or obstacle avoidance, can be considered.

In this section, we present a framework for global redundancy resolution divided into three main stages. The first stage consists of generating the SMD (and the SMMs that compose it) and a graph that represents the connectivity between the c-bundles of the SMD. Secondly, this graph is searched to extract every possible c-bundle chain (sequence of c-bundles) that completes the task, and a raw joint trajectory is efficiently generated for each c-bundle chain. Finally, the last

stage of the framework consists of optimizing the generated joint trajectories by iteratively solving a constrained Quadratic Programming (QP) optimization problem.

Additionally, we present variants of the different stages of the framework, which allow users to tailor the framework to the specific requirements of their application.

### 5.2.1 *SMD and graph generation*

The first stage of the framework consists of generating the SMD and corresponding c-bundle graph for the given task trajectory $\mathbf{x}(t)$. Our proposal for computing the SMD is a sampling-based approach closely related to the method presented in (Peidro et al., 2018). Algorithm 5.1 describes the proposed method for generating the SMD, where the input is the task trajectory $\mathbf{x}(t)$ and the output is the SMD encapsulated in a graph $\mathcal{G}$.

**Algorithm 5.1** SMD generation

**Initialize:** $\mathcal{X} \leftarrow$ Ordered set $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k\}$
$\mathcal{G} \leftarrow$ Empty graph
**for** $\mathbf{x}_i \in \mathcal{X}$ **do**
  $\mathcal{P} \leftarrow$ SampleSMMs($\mathbf{x}_i$) ▷ Figure 5.2(a)
  $\mathcal{M} \leftarrow$ ClusterSMMs($\mathcal{P}$) ▷ Figure 5.2(b)
  $\mathcal{G} \leftarrow$ MatchSMMs($\mathcal{M}, \mathcal{G}$) ▷ Figures 5.2(c-d)
**return** $\mathcal{G}$

The algorithm begins by discretizing the task trajectory $\mathbf{x}(t)$ into $k$ task-space points, producing the sequence $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_k\}$. This ordered set is distributed along the task trajectory with a time step $\Delta t$, yielding the equivalence $\mathbf{x}_i = \mathbf{x}(i\Delta t)$, with $\Delta t = \frac{t_f}{k-1}$, where $t_f$ is the final time of the task trajectory. The number of discretized task-space points $k$ is a parameter of the algorithm that can be adjusted by the user to control the resolution of the SMD. If $k$ is too small, the SMD may not be accurately represented, while if $k$ is too large, the algorithm may become excessively computationally expensive. Our recommendation is to set $k$ to a value that produces time steps $\Delta t$ of around $0.05$, which is an acceptable refresh rate for robotic applications.

The algorithm then iterates over each task-space point $\mathbf{x}_i$ in the ordered set $\mathcal{X}$. For each task-space point, a three-step procedure is performed, exemplified in Figure 5.2. Similarly to the sampling-based methods for computing SMMs discussed in Section 2.4.2, the manifolds are first sampled using the SampleSMMs function (Figure 5.2(a)), and then clustered using the ClusterSMMs function (Figure 5.2(b)). Finally, the MatchSMMs function (Figures 5.2(c-d)) matches the

clustered SMMs with the existing graph $\mathcal{G}$. We will now describe these three steps in detail.

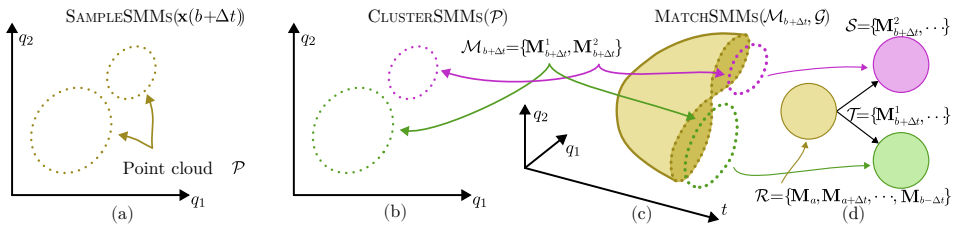


Figure 5.2: SMD generation algorithm. (a) Sampling of the SMMs for the task-space point $\mathbf{x}_i$. (b) Clustering of the sampled SMMs into disjoint manifolds. (c) Matching of the clustered SMMs with the existing graph $\mathcal{G}$. (d) Updating the graph $\mathcal{G}$ with the new c-bundle.

Algorithm 5.2 describes the SampleSMMs function, which generates a densely-populated point cloud $\mathcal{P}$ of joint configurations $\mathbf{q}$ that satisfy the constraint (5.1) for the task-space point $\mathbf{x}_i$. Although an explanation of the algorithm is given in the following paragraph, the reader is encouraged to refer to the original paper (Peidro et al., 2018) for a more detailed description of the method.

**Algorithm 5.2** Sampling of the SMMs at $\mathbf{x}_i$

**procedure** SampleSMMs($\mathbf{x}_i$)
  $\mathcal{P} \leftarrow \emptyset$
  **for** every *different* combination of $r$ redundant joints $\overbrace{(q_i, q_j, \cdots, q_k)}^{r \text{ coordinates}}$ | $i, j, \cdots, k \in \{1, \cdots, n\}$ **do**
    Let $\mathcal{Q}_\nu = [q_{\nu_{min}}, q_{\nu_{max}}]$ be the valid interval of joint $q_\nu$ between its joint limits ($\forall \nu \in \{i, j, \cdots, k\}$)
    Let $\mathbf{Q}_\nu = \{q_{\nu_{min}}, q_{\nu_{min}} + \Delta q_\nu, \cdots, q_{\nu_{max}}\}$ be a discretization of $\mathcal{Q}_\nu$ with step $\Delta q_\nu$
    The Cartesian product $\mathbf{Q}_{red} = \mathbf{Q}_i \times \mathbf{Q}_j \times \cdots \times \mathbf{Q}_k$ is an $r$-dimensional grid of joint coordinates
    **for** every node $\mathbf{q}_{red}$ of the grid $\mathbf{Q}_{red}$ **do**
      Solve $\mathbf{q}$ from $\mathbf{F}(\mathbf{x}_i, \mathbf{q}_{red} = [q_i, q_j, \cdots, q_k]^T) = \mathbf{0}$ (5.1)
      **if** $\mathbf{q}$ is a *feasible* joint configuration **then**
        Add $\mathbf{q}$ to the point cloud $\mathcal{P}$
  **return** $\mathcal{P}$

The algorithm begins by initializing an empty point cloud $\mathcal{P}$. A loop in line 3 (which we will refer to as the *outer loop*) iterates over every *different* combination of $r$ redundant joints, among the $n$ joints of the manipulator. Note that different orderings of the same joints are not considered, i.e., the combination $(q_1, q_2)$ is the same as $(q_2, q_1)$. For every combination of $r$ different joints, the valid interval of values for each joint is computed in line 4 as $\mathcal{Q}_\nu = \left[q_{\nu_{min}}, q_{\nu_{max}}\right]$, where $q_{\nu_{min}}$ and $q_{\nu_{max}}$ are the joint limits of joint $q_\nu$. The valid intervals are then discretized in line 5 into a set of joint coordinates $\mathbf{Q}_\nu = \{q_{\nu_{min}}, q_{\nu_{min}} + \Delta q_\nu, \cdots, q_{\nu_{max}}\}$, where $\Delta q_\nu$ is the step size of the discretization. In line 6, the Cartesian product of the sets of joint coordinates $\mathbf{Q}_\nu$ produces an $r$-dimensional grid of joint coordinates $\mathbf{Q}_{red} = \mathbf{Q}_i \times \mathbf{Q}_j \times \cdots \times \mathbf{Q}_k$.

Finally, the algorithm iterates over every node $\mathbf{q}_{red}$ of the grid $\mathbf{Q}_{red}$ in the so-called *inner loop* (line 7). For each node, the algorithm substitutes the joint coordinates $\mathbf{q}_{red} = \left[q_i, q_j, \cdots, q_k\right]^T$ and the task-space point $\mathbf{x} = \mathbf{x}_i$ into Equation (5.1) and solves for the $m$ remaining joint coordinates (those that are not in the $r$-dimensional grid) that compose joint configuration $\mathbf{q}$ (line 8). Equation (5.1) should be solved using the fastest method available, namely *algebraic elimination*. Note that every possible solution of $\mathbf{q}$ that maps to the task-space point $\mathbf{x}_i$ must be considered when solving the IKP. Line 9 tests if the joint configuration $\mathbf{q}$ is a *feasible* solution, meaning that it satisfies the additional constraints of the problem (e.g., joint limits, obstacle avoidance, etc.). If the joint configuration is feasible, it is added to the point cloud $\mathcal{P}$ (line 10). Otherwise, the joint configuration is stored in a separate point cloud $\bar{\mathcal{P}}$, for its subsequent use in Section 5.2.3.

It is worth discussing the computational effort required by the algorithm, as it produces a number of iterations of the outer loop equal to the number of combinations of $r$ different joints, which is given by the binomial coefficient $\binom{n}{r}$. Similarly, Equation (5.1) must be solved via algebraic elimination for every combination of $r$ joints, which may be a complex manual operation. However, as the experimental results of Section 5.3 will show, we have managed to significantly reduce the computational cost of this algorithm by vectorizing the inner loop of the algorithm, which allows for the simultaneous solving of all nodes in the grid $\mathbf{Q}_{red}$ rather than sequentially iterating over them.

Back to Algorithm 5.1, the second step of the algorithm consists of clustering the sampled SMMs, stored in point cloud $\mathcal{P}$ into a set $\mathcal{M}$ of disjoint manifolds, which is performed by the ClusterSMMs function (Figure 5.2(b)). We propose using the *DBSCAN* clustering algorithm (Ester et al., 1996), which is a density-based clustering algorithm that groups together points that are closely packed together, while marking as outliers points that lie alone in low-density regions. Unlike most other clustering algorithms, DBSCAN does not require the number of clusters to be specified in advance, which aligns with our earlier discussion

about being impossible to know the number of SMMs a priori. Instead, it requires two parameters: the neighbourhood radius $h$ and the minimum number of points in a neighbourhood to form a cluster, which we set to a single point. As for the neighbourhood radius $h$, we derive it from the step size of the discretization of the joint coordinates in the SampleSMMs function as follows:

$$h = \sqrt{\sum_{j=1}^{n} \Delta q_j^2} \tag{5.2}$$

where $\Delta q_j$ is the step size of the discretization of joint $q_j$. The DBSCAN algorithm returns a set of labels, where each integer marks the cluster to which the corresponding point in the point cloud relates. By means of the labels, the algorithm groups the points in the point cloud into disjoint manifolds, which are stored in the set $\mathcal{M}$.

The third step of the algorithm consists of matching the clustered SMMs in $\mathcal{M}$ with the existing SMD up to that time value, which is stored in the graph $\mathcal{G}$. In brief, the MatchSMMs function (Figures 5.2(c-d)) performs a spatial analysis between the newly-generated $\mathcal{M}_t$ and the last computed set of SMMs $\mathcal{M}_{t-1}$ to determine to which c-bundle each SMM in $\mathcal{M}_t$ belongs.

Before describing the MatchSMMs procedure, we believe it is beneficial to revisit the SMD and graph generation process with a simple example. Let us consider the same hypothetical scenario used in Section 5.1 (i.e., $n = 2$, $m = 1$, and $r = 1$) to produce the SMD in Figure 5.1(c). Algorithm 5.1 is executed, and up to time value $t = b$, the SampleSMMs function produces SMMs that the ClusterSMMs function groups into a single manifold. That means that the graph $\mathcal{G}$ contains a single c-bundle $\mathcal{R}$, to which the MatchSMMs (which will be explained next) directly matches every newly generated SMM in each time step.

However, when the algorithm surpasses time value $t = b$ (i.e., at $t = b + \Delta t$), the first transformation occurs, as exemplified in Figure 5.2. For the corresponding task value $\mathbf{x}(b + \Delta t)$, the SampleSMMs function produces a point cloud $\mathcal{P}$ (Figure 5.2(a)), which is clustered into two disjoint manifolds $\mathcal{M}_{b+\Delta t} = \{\mathbf{M}^1_{b+\Delta t}, \mathbf{M}^2_{b+\Delta t}\}$ by the ClusterSMMs function (Figure 5.2(b)). The remaining step of the algorithm (MatchSMMs) should determine the splitting transformation of the SMM that occurs between time values $t = b$ and $t = b + \Delta t$, as Figure 5.2(c) shows. Consequently, the MatchSMMs function should also update the graph $\mathcal{G}$ to include the new c-bundles $\mathcal{S}$ and $\mathcal{T}$, and determine the connectivity of each one of them with the existing c-bundle $\mathcal{R}$ (Figure 5.2(d)).

During the matching stage, correspondences between each SMMs in $\mathcal{M}_{t-\Delta t}$ (already known) and the newly-generated SMMs in $\mathcal{M}_t$ are stored in a mapping, denoted as *MatchTo*. For a given argument, corresponding to a SMM at the current time instant $t$, the mapping structure *MatchTo* returns the set of SMMs at the

previous time instant $t - \Delta t$ that are close to the given SMM. That is, the mapping *MatchTo* returns those SMMs at time $t - \Delta t$ that transform into the given SMM at time $t$. For better understanding of the mapping structure, Table 5.1 summarizes the transformations that occur when traversing the SMD of Figure 5.1(c) in both directions of the time axis $t$. While vanishings are not captured by the mapping (since there would be no SMM at time $t + \Delta t$ to match with), they are implicitly represented in the graph $\mathcal{G}$, as c-bundles without successors that do not reach the end of the task trajectory are vanishing c-bundles.

Table 5.1: Transformations when traversing the SMD of Figure 5.1(c) in both directions of axis $t$.

| Critical point | MatchTo mapping | | Transformation |
|---|---|---|---|
| | $\mathbf{M}$ | MatchTo($\mathbf{M}$) | |
| $t = b$ | $\mathbf{M}^1_{b+\Delta t}$ | $\{\mathbf{M}_b\}$ | Splitting |
| | $\mathbf{M}^2_{b+\Delta t}$ | $\{\mathbf{M}_b\}$ | |
| $t = c$ | $\mathbf{M}_{c+\Delta t}$ | $\{\mathbf{M}^1_c\}$ | - |
| | *Not captured* | | Vanishing |
| *If time were traversed in the negative direction:* | | | |
| $t = b$ | $\mathbf{M}_b$ | $\{\mathbf{M}^1_{b+\Delta t}, \mathbf{M}^2_{b+\Delta t}\}$ | Merging |
| $t = c$ | $\mathbf{M}^1_c$ | $\mathbf{M}_{c+\Delta t}$ | - |
| | $\mathbf{M}^2_c$ | $\emptyset$ | Creation |

Function MatchSMMs is also responsible for updating the graph $\mathcal{G}$ with the new c-bundles and their connectivity. C-bundles are stored in the graph as nodes, and the connectivity between them is stored as edges. The connectivity between c-bundles is determined by the *MatchTo* mapping, which is used to determine the successors of each c-bundle. C-bundles are not only nodes of the graph, but also store ordered sets of SMMs that compose them for each time step. For simplicity, during the explanation of the MatchSMMs procedure, we will refer to the c-bundle that contains a given SMM $\mathbf{M}$ as $\mathcal{C}(\mathbf{M})$.

Algorithm 3 describes the MatchSMMs function for a given set of disjoint SMMs $\mathcal{M}_t$ at time $t$ and the graph $\mathcal{G}$. For clarity, the matching procedure is presented in two parts: the spatial analysis for constructing the mapping structure *MatchTo* (lines 2-7), and the graph update (lines 8-17).

The algorithm begins by retrieving the set of SMMs at time $t - \Delta t$ from the graph $\mathcal{G}$ (line 2). Next, the mapping structure MatchTo is initialized as an empty

**Algorithm 5.3** Matching the set of disjoint SMMs in $\mathcal{M}_t$ to $\mathcal{G}$

1: **procedure** MATCHSMMs($\mathcal{M}_t, \mathcal{G}$)
2: $\mathcal{M}_{t-\Delta t} \leftarrow$ Retrieve the set of SMMs at $t - \Delta t$, stored in $\mathcal{G}$
3: $MatchTo \leftarrow \emptyset$ ▷ Mapping structure
4: **for** $\mathbf{M}_t^i \in \mathcal{M}_t$ **do**
5: **for** $\mathbf{M}_{t-\Delta t}^j \in \mathcal{M}_{t-\Delta t}$ **do**
6: **if** $\mathbf{M}_{t-\Delta t}^j$ is close to $\mathbf{M}_t^i$ **then**
7: Add $\mathbf{M}_{t-\Delta t}^j$ to the set of SMMs close to $\mathbf{M}_t^i$, stored in $MatchTo$ for the argument $\mathbf{M}_t^i$
8: **for** $\mathbf{M}_t^i \in \mathcal{M}_t$ **do**
9: $\mathcal{MT} \leftarrow MatchTo(\mathbf{M}_t^i)$ ▷ Set of SMMs close to $\mathbf{M}_t^i$
10: **if** $|\mathcal{MT}| = 1$ **then** ▷ if the cardinality of $\mathcal{MT}$ is 1
11: $\mathbf{M}_{t-\Delta t} \leftarrow$ the single SMM in $\mathcal{MT}$
12: **if** $\mathbf{M}_{t-\Delta t} \notin MatchTo(\mathbf{M}_t^j) \forall j \neq i$ **then**
13: Add $\mathbf{M}_t^i$ to the c-bundle $\mathcal{C}(\mathbf{M}_{t-\Delta t})$
14: **Continue** ▷ Continue iterating (back to line 9)
15: Initialize new c-bundle $\mathcal{C}_{new} = \{\mathbf{M}_t^i\}$ and add it to $\mathcal{G}$
16: **for** $\mathbf{M}_{t-\Delta t}^j \in \mathcal{MT}$ **do**
17: Add edge in $\mathcal{G}$ from $\mathcal{C}(\mathbf{M}_{t-\Delta t}^j)$ to $\mathcal{C}_{new}$
18: **return** $\mathcal{G}$

structure. The algorithm then iterates over each SMM $\mathbf{M}_t^i$ in the set $\mathcal{M}_t$ (line 4). For each SMM $\mathbf{M}_t^i$, the algorithm iterates over each SMM $\mathbf{M}_{t-\Delta t}^j$ in the set $\mathcal{M}_{t-\Delta t}$ (line 5). If the SMM $\mathbf{M}_{t-\Delta t}^j$ is close to $\mathbf{M}_t^i$, it is added to the set of SMMs close to $\mathbf{M}_t^i$ in the mapping structure $MatchTo$ (line 7). The closeness of the SMMs is the minimum distance between any two points in the two manifolds, which must be less than a threshold $d$ to be considered close. The threshold $d$ is a parameter of the algorithm, similar to $h$ in the CLUSTERSMMs function, but also considers the time dimension:

$$d = \sqrt{\sum_{j=1}^{n} \Delta q_j^2 + \Delta t^2} \tag{5.3}$$

where $\Delta q_j$ is the step size of the discretization of joint $q_j$ (line 5 of Algorithm 5.2) and $\Delta t$ is the time step used to discretize the task trajectory (line 1 of Algorithm 5.1).

Minimum distance computation between two $n$-dimensional point clouds is a broad topic. We have found that a two-step approach is the most efficient for

our application. First, we conduct a broad check using the bounding boxes of the two point clouds. If the bounding boxes are not close, we can skip the finer distance computation, as the two point clouds are not close. If the bounding boxes are close, we compute the distance between every pair of points in the two point clouds. If any pair of points (each belonging to a different point cloud) is closer than the threshold $d$, we consider the two point clouds close to each other (line 6 of Algorithm 5.3). Note that the distance computations are efficiently performed by using k-d tree data structures (Bentley, 1975). Additionally, pairwise distance computations can be performed in parallel to further reduce the computational cost. Upon completion of the nested loop, the mapping structure $\mathtt{MatchTo}$ contains the set of $\mathcal{MT}$ of SMMs at time $t - \Delta t$ that are close to each SMM at time $t$.

The next part of the algorithm (lines 8-17) updates the graph $\mathcal{G}$ with the matching information stored in the mapping structure $\mathtt{MatchTo}$. To do so, the algorithm iterates over each SMM $\mathbf{M}_t^i$ in the set $\mathcal{M}_t$ (line 8). For each SMM $\mathbf{M}_t^i$, the algorithm retrieves the set of close SMMs $\mathcal{MT}$ from the mapping structure for the given argument $\mathit{MatchTo}(\mathbf{M}_t^i)$ (line 9).

If the cardinality of $\mathcal{MT}$ is 1, it means that the SMM $\mathbf{M}_t^i$ is close to a single SMM $\mathbf{M}_{t-\Delta t}$ at time $t - \Delta t$ (line 10). In this case, the algorithm checks if the SMM $\mathbf{M}_{t-\Delta t}$ is not already in the mapping structure for any other SMM $\mathbf{M}_t^j$ (line 11), which would indicate that the SMM $\mathbf{M}_{t-\Delta t}$ is close to multiple SMMs at time $t$ (i.e., a splitting transformation). If line 12 is satisfied, it means the direct extension of the c-bundle $\mathcal{C}(\mathbf{M}_{t-\Delta t})$ (i.e., no topological transformation occurs), the SMM $\mathbf{M}_t^i$ is added to the c-bundle $\mathcal{C}(\mathbf{M}_{t-\Delta t})$ (line 13), and the algorithm continues to the next iteration (back to line 9 for the next SMM).

If the cardinality of $\mathcal{MT}$ is greater than 1 in line 10 (which indicates a merging transformation), or line 12 is not satisfied (which indicates a splitting transformation), the algorithm initializes a new c-bundle $\mathcal{C}_{new}$ and initializes it with the SMM $\mathbf{M}_t^i$ (line 15). Finally, the algorithm iterates over each SMM $\mathbf{M}_{t-\Delta t}^j$ in the set $\mathcal{MT}$ (line 16) and adds an edge in the graph $\mathcal{G}$ from the c-bundle $\mathcal{C}(\mathbf{M}_{t-\Delta t}^j)$ to the new c-bundle $\mathcal{C}_{new}$ (line 17). The algorithm ends by returning the updated graph $\mathcal{G}$ in line 18.

### 5.2.2 *Generation of raw joint trajectories*

The second stage of the framework consists of generating raw joint trajectories from the computed SMD graph $\mathcal{G}$. Tracing a continuous joint trajectory along the computed SMD, from the initial joint configuration $\mathbf{q}(0)$ to the final time value $t_f$, generates a joint trajectory $\mathbf{q}(t)$ that satisfies the constraint (5.1) (and any additional constraints) for the entire task trajectory $\mathbf{x}(t)$. However, in global redundancy resolution, the goal is to find the optimal joint trajectory $\mathbf{q}(t)$ rather

than just any joint trajectory. In this stage we propose a method for efficiently generating preliminary raw joint trajectories from the SMD graph $\mathcal{G}$ which can be subsequently optimized in the next stage of the framework.

By combining the c-bundles in the graph $\mathcal{G}$ in specific orders increasing in time, where the initial c-bundle is the one that contains the initial joint configuration $\mathbf{q}(0)$ and the final c-bundle reaches the final time value $t_f$, we can generate a set of *c-bundle chains*. A c-bundle chain $\mathcal{CBC}$ is a sequence of nodes of the graph $\mathcal{G}$:

$$\mathcal{CBC} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_f\} \tag{5.4}$$

where $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ are connected and timewise successive c-bundles in the graph $\mathcal{G}$, $f$ is the number of c-bundles in the chain, and $\mathcal{C}_1$ and $\mathcal{C}_f$ are c-bundles that exist at the initial and final instant of the task trajectory, respectively. Note that c-bundle chains are simple paths of the graph $\mathcal{G}$ (i.e., paths that do not contain repeated nodes), and therefore the number of c-bundle chains is finite. Figure 5.3(a) shows a hypothetical graph $\mathcal{G}$ where three c-bundle chains are highlighted. The c-bundle chain highlighted in blue is illustrated in Figure 5.3(b). In this representation, the SMD is decomposed into its constituent c-bundles; those that form the c-bundle chain are depicted in full, while the remaining bundles gradually fade out.

To extract every possible c-bundle chain from the graph $\mathcal{G}$, we can use a depth-first search algorithm, as detailed in (Sedgewick, 2001). The algorithm returns all simple paths that connect a given source and target node in the graph. Therefore, the algorithm is run for every pair of source and target nodes in the graph $\mathcal{G}$, where the source and target nodes are c-bundles that exist at the initial and final time values of the task trajectory, respectively. Note that, if the initial joint configuration is restricted (when no prior self-motion is desired), there is only one source node: the c-bundle that contains the initial joint configuration $\mathbf{q}(0)$.

Next, we will describe the procedure for generating a raw joint trajectory from a c-bundle chain $\mathcal{CBC}$. While numerous methods can be used to generate a joint trajectory from a c-bundle chain (as we will discuss in Section 5.2.4.2), we propose a highly efficient approach based on nearest-neighbour queries. Since these generated trajectories will undergo an optimization process in the next stage, we do not need to significantly worry about the quality of the generated trajectories. The approach leverages the same k-d trees already built for the MatchSMMs to efficiently retrieve the closest point in a SMM. Algorithm 5.4 provides the procedure for generating a raw joint trajectory $\mathcal{Q}$ from a c-bundle chain $\mathcal{CBC}$.

Before describing the algorithm, we need to define the concept of *manifold portion*. For every pair of connected c-bundles $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ in the c-bundle chain $\mathcal{CBC}$, the manifold portion is the portion of the co-regular surface that connects the two c-bundles. As discussed in Section 5.1, c-bundles are bounded by co-regular surfaces, which are sets that contain regular and singular points. How-

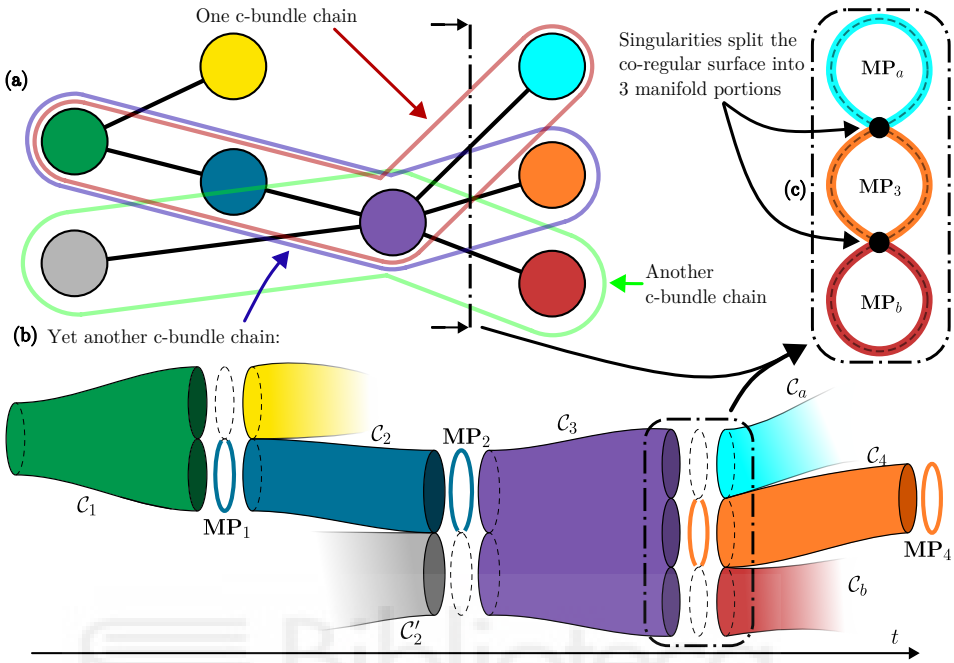


Figure 5.3: (a) A hypothetical graph $\mathcal{G}$. (b) The c-bundle chain highlighted in blue. (c) A co-regular surface with three manifold portions.

ever, co-regular surfaces do not only delimit two c-bundles, but also any other c-bundle that participates in the transformation (see how $\mathcal{C}_3$ is not only connected to $\mathcal{C}_4$, but also to $\mathcal{C}_a$ and $\mathcal{C}_b$ in Figure 5.3). If we use the singularities of the co-regular surface to split it into disjoint portions, we can define the manifold portion as the portion of the co-regular surface that connects two c-bundles $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$. Figure 5.3(c) shows a triple-eight curve co-regular surface with two self-intersections (singularities) that delimit three manifold portions. This co-regular surface is the one where c-bundle $\mathcal{C}_3$ ends, and splits into c-bundles $\mathcal{C}_4$, $\mathcal{C}_a$, and $\mathcal{C}_b$. By removing the self-intersections (singularities) of the co-regular surface, three manifold portions that connect the c-bundles are obtained. Manifold portions $\mathbf{MP}_a$ and $\mathbf{MP}_b$ are open curves that connect c-bundle $\mathcal{C}_3$ with c-bundles $\mathcal{C}_a$ and $\mathcal{C}_b$, respectively. Manifold portion $\mathbf{MP}_3$ is a pair of open curves that connect c-bundle $\mathcal{C}_3$ with c-bundle $\mathcal{C}_4$. For convenience, we denote the manifold portion that connects c-bundles $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ as $\mathbf{MP}_i$, for a given c-bundle chain $\mathcal{CBC}$. A special case occurs for the last c-bundle in the c-bundle chain: since the last c-bundle does not have a successor, $\mathbf{MP}_f$ is the last SMM in the c-bundle $\mathcal{C}_f$ ($\mathbf{MP}_4$ in Figure 5.3).

**Algorithm 5.4** Generation of a raw joint trajectory along a c-bundle chain $\mathcal{CBC}$

1: $\mathcal{Q} = \{\mathbf{q}_0\}$ ▷ Initialize the raw trajectory with $\mathbf{q}_0$
2: **for** $i = 1, 2, \ldots, f$ **do**
3: $\quad \mathcal{C}_i \leftarrow$ i-th c-bundle in $\mathcal{CBC}$
4: $\quad \mathbf{MP}_i \leftarrow$ Manifold portion that connects $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$
5: $\quad \mathbf{q}_a \leftarrow$ Last point in $\mathcal{Q}$
6: $\quad \mathbf{q}_b \leftarrow$ Closest point in $\mathbf{MP}_i$ to $\mathbf{q}_a$
7: $\quad n_i \leftarrow$ Number of SMMs stacked in $\mathcal{C}_i$
8: $\quad \mathcal{Q}_l \leftarrow n_i$-points linear interpolation from $\mathbf{q}_a$ to $\mathbf{q}_b$
9: $\quad$ **for** $j = 1, 2, \ldots, n_i$ **do**
10: $\quad\quad \mathbf{M}_j \leftarrow$ j-th SMM stacked in $\mathcal{C}_i$
11: $\quad\quad \mathbf{q}_l \leftarrow$ Point in $\mathcal{Q}_l$ corresponding to the same $t$ as $\mathbf{M}_j$
12: $\quad\quad \mathbf{q}_c \leftarrow$ Closest point in $\mathbf{M}_j$ to $\mathbf{q}_l$
13: $\quad\quad$ Add $\mathbf{q}_c$ to $\mathcal{Q}$
14: **return** $\mathcal{Q}$

The algorithm begins by initializing the raw joint trajectory $\mathcal{Q}$ with the initial joint configuration $\mathbf{q}_0$ (line 1). Then, it iterates for the number of c-bundles in the c-bundle chain $\mathcal{CBC}$ (line 2). Lines 3 and 4 retrieve the i-th c-bundle $\mathcal{C}_i$ in the c-bundle chain and the manifold portion $\mathbf{MP}_i$ that connects the i-th and $(i+1)$-th c-bundles, respectively. Next, $\mathbf{q}_a$ and $\mathbf{q}_b$ correspond to the last point in the raw joint trajectory $\mathcal{Q}$ and the closest point in the manifold portion $\mathbf{MP}_i$ to $\mathbf{q}_a$, respectively (lines 5-6). The algorithm then computes a linear interpolation $\mathcal{Q}_l$ between the points $\mathbf{q}_a$ and $\mathbf{q}_b$, with $n_i$ points, where $n_i$ is the number of SMMs stacked in c-bundle $\mathcal{C}_i$ (line 7).

In line 9, a second loop iterates over the number of SMMs stacked in c-bundle $\mathcal{C}_i$. For every j-iteration, the algorithm retrieves the j-th SMM $\mathbf{M}_j$ stacked in c-bundle $\mathcal{C}_i$ (line 10), and the point $\mathbf{q}_l$ in the linear interpolation $\mathcal{Q}_l$ that corresponds to the same time value as the SMM $\mathbf{M}_j$ (line 11). The algorithm then retrieves the closest point $\mathbf{q}_c$ in the SMM $\mathbf{M}_j$ to the point $\mathbf{q}_l$ (line 12), and adds it to the raw joint trajectory $\mathcal{Q}$ (line 13).

The algorithm ends by returning the raw joint trajectory $\mathcal{Q}$ in line 14. The generated raw joint trajectory $\mathcal{Q}$ is a sequence of continuously-connected joint configurations that satisfy the constraint (5.1) for the entire task trajectory $\mathbf{x}(t)$.

### 5.2.3 *Optimization of joint trajectories*

The last stage of the framework consists of optimizing the generated raw joint trajectories $\mathcal{Q}$. By iteratively solving a QP problem, every point in the raw joint trajectory is moved along its SMM to minimize a cost function $f(\cdot)$. Similarly to

established methods in the literature for general-purpose trajectory optimization (Kalakrishnan et al., 2011; Ratliff et al., 2009) (not SMM-restricted), we propose to use a cost function that quantifies the sum of the squared joint velocities along the trajectory:

$$f(\varphi) = \frac{1}{2}\|\mathbf{V}\varphi + \mathbf{e}\|^2 \tag{5.5}$$

where $\varphi$ is a column vector corresponding to the flattened joint trajectory $Q$, $\mathbf{V}$ is the finite difference matrix that computes the joint velocities, and $\mathbf{e}$ is a vector that contributes with the fixed initial joint configuration $\mathbf{q}_0$:

$$\begin{aligned}
\varphi &= [q_{11}, q_{12}, \cdots, q_{1n}, q_{21}, q_{22}, \cdots, q_{2n}, \cdots, q_{k1}, q_{k2}, \cdots, q_{kn}]^T \\
\mathbf{V} &= \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}_{k\times k} \otimes \mathbf{I}_{n\times n} \\
\mathbf{e} &= \begin{bmatrix} -\mathbf{q}_0^T & \mathbf{o}_{n(k-1)\times 1}^T \end{bmatrix}^T
\end{aligned} \tag{5.6}$$

where $q_{ij}$ is the j-th joint of the i-th joint configuration in the joint trajectory, $\otimes$ is the Kronecker product, $k$ is the number of joint configurations in the joint trajectory, and the subscript $p \times q$ indicates the size of the matrix.

Similarly to other common optimization techniques, such as Sequential Quadratic Programming (SQP) (Nocedal and Wright, 1999), or Sequential Convex Programming (SCP), the parent problem $\min f(\varphi)$ is not directly solved. Instead, it is solved sequentially by a local approximation of the cost function, which is achieved by a first-order Taylor expansion of the cost function around the current joint trajectory $\varphi_{current}$:

$$f(\varphi) \approx f(\varphi_{current}) + \nabla f(\varphi_{current})^T(\varphi - \varphi_{current}) \tag{5.7}$$

where $\nabla f(\varphi_{current})$ is the gradient of the cost function at the current joint trajectory $\varphi_{current}$, which is computed as:

$$\nabla f(\varphi_{current}) = \mathbf{V}^T(\mathbf{V}\varphi_{current} + \mathbf{e}) \tag{5.8}$$

This local approximation of the cost function is then solved by a QP problem:

$$\underset{\Delta\varphi}{\text{minimize}} \quad \frac{s}{2}\|\Delta\varphi\|^2 + \nabla f^T \Delta\varphi \tag{5.9}$$

where $s$ is a regularization parameter that controls the penalization of large changes in $\Delta\varphi$ (Boyd and Vandenberghe, 2004). In our experiments, we have tested that a value of $s = 10$ provides a good balance between convergence speed and trajectory quality. The QP problem is solved and the increment $\Delta\varphi$ is added to the current joint trajectory $\varphi_{current}$:

$$\varphi = \varphi_{current} + \Delta\varphi \tag{5.10}$$

This QP problem can be further constrained by leveraging the fact that points $\mathbf{q}_i$ in the joint trajectory cannot freely move in the $n$-dimensional joint space, but are restricted to the $r$-dimensional SMM that contains them. This fact reduces the dimensionality of the optimization problem from $n \cdot k$ to $r \cdot k$, by imposing the following constraint:

$$\Delta\varphi = \mathbf{N}\mathbf{S}^T \mathbf{d} \tag{5.11}$$

where $\mathbf{d} = [\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_k]^T = [d_{11}, d_{12}, \cdots, d_{1r}, \cdots, d_{k1}, \cdots, d_{kr}]^T$, and $\mathbf{d}_i$ is the $r$-dimensional vector that quantifies movement along the axes of a null-space basis of the Jacobian matrix at the $i$-th joint configuration $\mathbf{q}_i$. The matrix $\mathbf{NS}$ contains these bases:

$$\mathbf{NS} = \begin{bmatrix} \mathbf{NS}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{NS}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{NS}_k \end{bmatrix}_{rk \times nk} \tag{5.12}$$

where $\mathbf{NS}_i$ is the $r \times n$ matrix that contains an orthonormal basis of the null-space of the Jacobian matrix $\mathbf{J}(\mathbf{q}_i)$ at the $i$-th joint configuration $\mathbf{q}_i$. As Figure 5.4 shows, null-space bases are formed by $r$ linear independent $n$-dimensional vectors that are tangent to the SMM at the point $\mathbf{q}_i$. Movement along these bases *approximates* movement along the SMM. However, the SMM is not a linear subspace, and therefore the movement along the null-space bases is not exactly along the SMM. To overcome this issue, a Newton-based correction is usually necessary for driving the joint trajectory to the SMM. This step is not computationally expensive, as it typically requires only a single iteration to converge, due to restrictions R1 and R2 that will be introduced next.

The final QP problem is then:

$$\begin{aligned} \underset{\mathbf{d}}{\text{minimize}} \quad & \frac{s}{2}\|\mathbf{N}\mathbf{S}^T \mathbf{d}\|^2 + \nabla f^T \mathbf{N}\mathbf{S}^T \mathbf{d} \\ \text{subject to} \quad & \mathbf{d} \in \mathcal{A} \quad \text{(R1)} \\ & \mathbf{d} \in \mathcal{B} \quad \text{(R2)} \end{aligned} \tag{5.13}$$

which is solved at every iteration and its solution is sequentially added to the current joint trajectory $\varphi_{current}$:

$$\varphi = \varphi_{current} + \mathbf{N}\mathbf{S}^{T}\,\mathbf{d} \tag{5.14}$$

Restriction R1 sets a *trust region* around the every joint configuration $\mathbf{q}_i$ in the joint trajectory, which maintains approximations in Equations (5.7) in a valid region, and ensures that a low number of iterations are needed to converge to the SMM due to the local linearity of the SMM (Equation (5.11)). Figure 5.4 represents the trust region in blue for an $r = 2$ example. This trust region is defined as an r-dimensional hypercube centred at the point $\mathbf{q}_i$, with a side length of $2\lambda$, and has its axes aligned with the axes of the null-space basis $\mathbf{NS}_i$. A smaller value of $\lambda$ improves the accuracy of the approximations, but increases the number of iterations needed to converge to the optimal solution of $f(\varphi)$. Conversely, a larger value of $\lambda$ increases the speed of convergence, but compromises the accuracy of the approximations, and leads to potential convergence issues, such as skipping over minima of $f(\varphi)$. Our experiments show that a value of $\lambda = 0.1$ provides a good balance between convergence speed and trajectory quality. Note that the trust region depicted in blue in Figure 5.4 is exaggerated for clarity, and the actual trust region is much smaller with respect to the extents of the SMM.

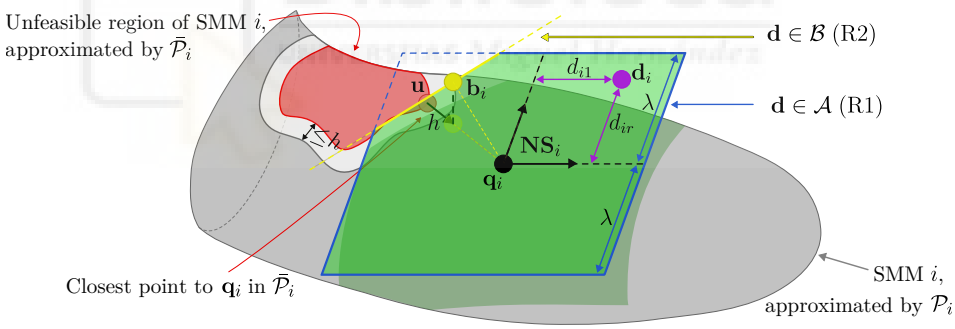


Figure 5.4: QP problem restrictions in a hypothetical 2-dimensional SMM embedded in a 3-dimensional joint space. The trust region is represented in blue (R1), and the estimated boundary restriction is represented in yellow (R2).

The other restriction R2 constraints the values of $\mathbf{d}$ to avoid moving the joint configuration to unfeasible regions, such as the exemplified in Figure 5.4 in red. Without this restriction, the joint configuration could be moved to a region that does not satisfy the constraints, such as joint limits or obstacles. This restriction relies on the previously-computed point cloud of unfeasible joint configurations $\bar{\mathcal{P}}_i$, generated during the sampling phase of the SMD (see Algorithm 5.2 of Section 5.2.1), where points that do not satisfy the constraints are stored. Note that

this process does not involve additional computations, as the points would have been computed and tested anyway during the sampling phase. R2 is established by a simple process to determine the estimated as closest boundary point $\mathbf{b}_i$ to the current joint configuration $\mathbf{q}_i$.

First, the algorithm retrieves the closest point in the point cloud $\bar{\mathcal{P}}_i$ to the current joint configuration $\mathbf{q}_i$. This point is marked in red as $\mathbf{u}$ in Figure 5.4. Figure 5.4 also depicts the *uncertainty region* in white, which is a stripe-shaped region of width $h$ that, due to the step used to sample the SMM, contains points whose membership to the SMM is uncertain. Note that in practice, $h$ is a small value, and therefore the uncertainty region is small, but has been exaggerated in Figure 5.4 for clarity. To make a conservative estimate of the boundary point, the retrieved point $\mathbf{u}$ is shifted towards $\mathbf{q}_i$ by a distance $h$, and projecting the result onto the $r$-dimensional linear subspace spanned by the null-space basis $\mathbf{NS}_i$ (green plane in Figure 5.4) to obtain the point $\mathbf{b}_i$. This estimation of the boundary point may be overly conservative, but it ensures that the joint configuration $\mathbf{q}_i$ is not moved to an unfeasible region. Lastly, the restriction R2 is defined as:

$$\mathbf{b}_i^T \mathbf{d}_i \leqslant \|\mathbf{b}_i\|^2 \tag{5.15}$$

This inequality constraint is the result of linearizing an $r$-sphere centred at $\mathbf{q}_i$ and passing through the point $\mathbf{b}_i$, which produces a hyperplane in the linear subspace spanned by the null-space basis $\mathbf{NS}_i$, as shown in yellow in Figure 5.4. Note that this linearization is performed in order to be able to incorporate restriction R2 in the QP problem, since the $r$-sphere restriction would not be linear and therefore not suitable for a QP problem. Also note that, in Equation (5.15), $\mathbf{b}_i$ is expressed relative to the null-space basis $\mathbf{NS}_i$.

The local minimization of the cost function is performed by iteratively solving the QP problem (5.13) until convergence is reached, as detailed in Algorithm 5.5. In every iteration, the algorithm computes the restrictions R1 and R2 as described above. The algorithm then solves the QP problem in Equation (5.13), subject to restrictions R1 and R2, by means of the QP solver of choice, such as (Bambade et al., 2022) or (Stellato et al., 2020). The solution $\mathbf{d}$ is then used to update the joint trajectory $\varphi$ thanks to the relationship in Equation (5.11). The algorithm ends when the maximum magnitude of the elements in $\mathbf{d}$ is less than a threshold $\varepsilon$ (e.g., $10^{-3}$ has been used in our experiments).

### 5.2.4 *Variants*

One of the main advantages of the proposed framework is its flexibility due to the modularity of the different stages. The framework can be easily adapted to different requirements by modifying any of the stages, to meet the specific needs of the application in terms of generalization to complex kinematic chains,

**Algorithm 5.5** Trajectory optimization

```
while max(|d_ij|) > ε do
    Compute restrictions (R1) and (R2)
    d ← Solve QP problem in Equation (5.13), subject to R1 and R2
    φ ← φ + NS^T d
return φ
```

computational speed, or trajectory quality. In this section, we will briefly discuss some of the possible variants of the framework. Although these variants may be freely combined, we will present some recommended combinations, as well as reasons to choose one variant over another. Table 5.2 summarizes the original approaches, and proposed variants, for the different stages of the framework.

#### 5.2.4.1 *SMD and graph generation*

Section 2.4 presents a review of the state-of-the-art methods for SMM generation. Although any of these methods can be used in the proposed framework, in Section 5.2.1 we have extended the work of (Peidro et al., 2018) to efficiently compute the SMD. However, the main drawback of this method is that it requires solving the IKP by algebraic elimination for every combination of $r$ joints, which may not always be possible, especially for complex kinematic chains. In this section, we will discuss how to implement the continuation-based approaches proposed by (Burdick, 1989; Henderson, 2002) to compute the SMD and the graph $\mathcal{G}$.

Naively, one could take Algorithm 5.1 and simply replace SampleSMMs and ClusterSMMs with any continuation-based SMM generation method from the literature, and leave the rest of the algorithm unchanged. However, this would not be efficient, as these methods, which are already computationally expensive, would be run for every discrete time value $t$ in the task trajectory $\mathbf{x}(t)$. Instead, we propose to run the continuation-based method only for the first time value $t = 0$, and leverage the Jacobian matrices computed during the SMM generation to propagate the SMM to the next time value $t$.

Starting from the initial joint configuration of the robot $\mathbf{q}(0)$, we propose computing the first SMM using a continuation-based method, such as the one proposed by (Burdick, 1989) for degrees of redundancy $r = 1$, or the one proposed by (Henderson, 2002) for $r > 1$. Note that continuation approaches will only compute the SMM at which the initial joint configuration $\mathbf{q}(0)$ lies, and will not compute other disjoint SMMs that may exist. If multiple queries are needed (i.e., solving the redundancy problem for multiple initial joint configurations), numerous seed joint configurations should be sampled until every disjoint SMM is found.

Table 5.2: Proposed variants, briefly summarized, and their sections for the different stages of the proposed framework.

| SMD and graph generation | Generation of raw joint trajectories | Trajectory optimization |
|---|---|---|
| **Sampling** (Section 5.2.1)<br>Sweeping every combination of $r$ joints, and solving the IKP for the remaining. | **Nearest Neighbour (NN)** (Section 5.2.2)<br>Querying nearest neighbours (k-d trees) to efficiently traverse the $\mathcal{CBC}$. | **Standard** (Section 5.2.3)<br>Completely optimizing (until convergence) every raw joint trajectory. |
| **Continuation** (Section 5.2.4.1)<br>(Burdick, 1989; Henderson, 2002) for computing the SMM at $\mathbf{x}(t=0)$. Task propagation, resampling and expansion for the subsequent SMMs. | **Pathfinding Algorithms (PA)** (Section 5.2.4.2)<br>Running pathfinding algorithms (e.g., A* or RRT) in the entire $\mathcal{CBC}$. | **Continuation-friendly** (Section 5.2.4.3)<br>Substituting the SMM boundaries restriction R2 by a joint-limits restriction RV1 and a local approximation of collisions RV2. |
| **Other** (Section 2.4.3)<br>Employing any method for SMM generation for every value of $\mathbf{x}(t)$. | **Hybrid NN-PA** (Section 5.2.4.2)<br>NN method to find the points in every **MP**, PA to establish the trajectory along every c-bundle. | **Online** (Section 5.2.4.3)<br>Simultaneous optimization and execution of the trajectory. |

However, as we discussed in Section 2.4.1, the number of disjoint SMMs cannot be known in advance, and therefore there is no guarantee that the method will find all of them. That is the reason why we recommend using the continuation-based variant for single query applications only, using the known initial joint configuration $\mathbf{q}(0)$ to compute the SMM at that point, since the other disjoint SMMs will not be reachable from the initial joint configuration $\mathbf{q}(0)$ unless traversing time backwards (which is not possible in practice). This can be visualized in Figure 5.3(b). Consider starting from an initial joint configuration $\mathbf{q}(0)$ that lies in c-bundle $\mathcal{C}_2$. To reach c-bundle $\mathcal{C}_2'$, the robot would need to first advance to c-bundle $\mathcal{C}_3$, and then traverse time backwards to reach c-bundle $\mathcal{C}_2'$, which is not possible in practice.

Having computed the initial SMM, Jacobian matrices $\mathbf{J}$ at every point in the SMM have also been computed. These Jacobian matrices can be used to propagate the SMM to the next time value $t$ in the task trajectory $\mathbf{x}(t)$:

$$\mathbf{q}_{i+1} = \mathbf{q}_i + \mathbf{J}^{\dagger}(\mathbf{q}_i)\,(\mathbf{x}(t+\Delta t) - \mathbf{x}(t)) \tag{5.16}$$

This operation is exemplified in Figure 5.5 by the orange arrows, which show the propagation of the SMM from time $t$ to time $t+\Delta t$.

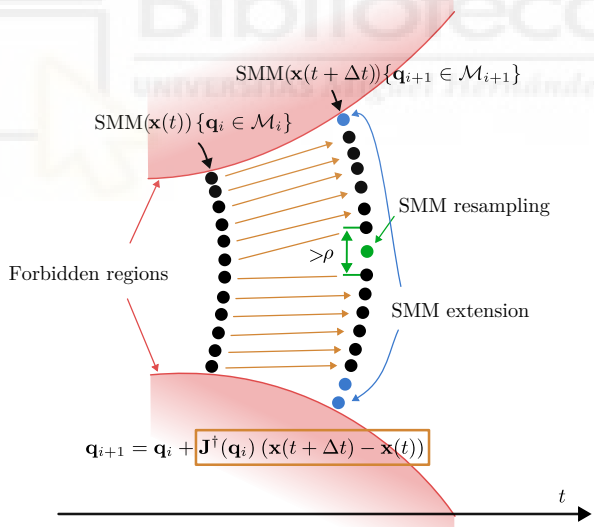


Figure 5.5: Propagation of the SMM from time $t$ to time $t+\Delta t$.

Similarly to continuation methods (Burdick, 1989; Henderson, 2002), Equation (5.16) yields points that do not necessarily belong to the SMM, due to the fact that this approximation is only valid for infinitesimal displacements. Once again, the same Newton correction discussed in Section 5.2.3 should be applied to the

points obtained from Equation (5.16) to ensure that they belong to the SMM. As already discussed, this correction is not computationally expensive, and, on top of that, both of these computations (Equation (5.16) and the Newton correction) can be accelerated by vectorizing the operations. This parallelization allows for the simultaneous computation, and subsequent correction, of all points in the SMM at once, rather than iterating over each point individually.

Finally, the resulting SMMs must be resampled to obtain a densely populated point cloud. As illustrated in Figure 5.5, Equation (5.16) may yield low-density regions (i.e., points that are further apart than the sampling step $\rho$), that need to be resampled like the exemplified green point. Additionally, boundaries of the SMMs should also be tested by trying to expand the SMMs. As Figure 5.5 shows, the red forbidden regions can shrink over time, and therefore the SMM should be extended to include these new regions, as blue points in the figure show. These two operations (resampling and extension) are analogous to the continuation methods proposed by (Burdick, 1989; Henderson, 2002).

By iteratively applying this process at every discretized time value $t$ in the task trajectory $\mathbf{x}(t)$, the SMD is generated. By propagating the SMM to the next time value $t + \Delta t$, information about transformation of the SMMs is obtained, rendering the MatchSMMs function in Algorithm 5.1 unnecessary, as we do not need to calculate the neighbourhood of every new SMM because we already know which SMM it is connected to. That is, the neighbour of SMMs $\mathcal{M}_{i+1}$ are the SMMs $\mathcal{M}_i$, which are the SMMs that were propagated using Equation (5.16) to the next time value $t + \Delta t$.

#### 5.2.4.2 *Generation of raw joint trajectories*

In Section 5.2.2, we proposed a method to generate raw joint trajectories based on efficient *nearest neighbour* (NN) queries. While it is not optimal, it is computationally efficient as it leverages the same k-d trees used in the matching process MatchSMMs in Algorithm 5.1. Since trajectories are post-processed in the optimization stage, this method is sufficient for most applications. However, if the user does not want to post-process the trajectory, it may be interesting to invest extra computational resources to generate the raw joint trajectories more optimally.

Any classical *pathfinding algorithm* (PA) can be used to generate the raw joint trajectories. For example, the A* algorithm (Hart et al., 1968) is a popular pathfinding algorithm that yields the optimal path between two points in a graph. The RRT algorithm (LaValle, 1998) is another popular pathfinding algorithm that is computationally efficient and can be used to generate suboptimal paths in high-dimensional spaces. One could simply run these algorithms on the entire SMD to join the initial joint configuration $\mathbf{q}_0$ to any point that completes the task trajectory $\mathbf{x}(t = t_f)$. However, this would be computationally prohibitive,

as the SMD is a high-dimensional point cloud with an extremely large number of points. Another option would be to run the pathfinding algorithm in the SMD for every c-bundle $\mathcal{C}_i$ in the c-bundle chain $\mathcal{CBC}$, similar to the approach used by Lück (1997), but this would also be computationally expensive when compared to the NN method.

Instead, for these cases, we propose using a *hybrid* combination of the NN and PA methods. First, the NN method is used to determine the points in every manifold portion **MP** of every c-bundle chain $\mathcal{CBC}$ (point $\mathbf{q}_b$ in Algorithm 5.4). Then, inside each c-bundle $\mathcal{C}_i$ in the c-bundle chain $\mathcal{CBC}$, a pathfinding algorithm of choice is run to establish the trajectory along the c-bundle $\mathcal{C}_i$ that joins manifold portion $\mathbf{MP}_{i-1}$ and $\mathbf{MP}_i$. This approach essentially means replacing lines 7-13 of Algorithm 5.4 with a pathfinding algorithm that connects the two points $\mathbf{q}_a$ and $\mathbf{q}_b$. This hybrid method would be a compromise between the efficiency of the NN method and the global optimality of the PA method.

#### 5.2.4.3 *Optimization of joint trajectories*

In Section 5.2.3, we proposed the *standard* method for trajectory optimization, which completely optimizes the generated raw joint trajectories until convergence. This stage is probably the most flexible one, as it can be adapted in different manners, as we will discuss in this section.

When using the *continuation* variant for the SMD generation (Section 5.2.4.1), the point cloud of unfeasible joint configurations $\bar{\mathcal{P}}_i$ is not available, as it is not computed during the sampling phase. Therefore, restriction R2 in the standard method cannot be used. Instead, we propose to use the *continuation-friendly* variant, in which the original restriction R2 is replaced by as many restrictions as additional constraints have been considered. In this work, we have considered joint limits and obstacles, which are the most common constraints in the literature. Therefore, two new restrictions are added to the QP problem to replace R2: RV1 and RV2.

Restriction RV1 is a joint limits restriction that constrains the values of $\mathbf{d}$ to avoid moving the joint configuration to unfeasible regions due to joint limits. This restriction is defined for every $i$-th joint configuration $\mathbf{q}_i$ in the joint trajectory as:

$$\begin{aligned} &\mathbf{q}_i + \mathbf{N}\mathbf{S}_i^T \mathbf{d}_i \geqslant \mathbf{q}_{min} \qquad \text{(RV1)} \\ &\mathbf{q}_i + \mathbf{N}\mathbf{S}_i^T \mathbf{d}_i \leqslant \mathbf{q}_{max} \end{aligned} \tag{5.17}$$

where $\mathbf{q}_{min}$ and $\mathbf{q}_{max}$ are the vectors of lower and upper joint limits, respectively, and the inequalities are element-wise.

Restriction RV2 ensures that the joint configuration $\mathbf{q}_i$ does not collide with obstacles when moving along the SMM. It is a local approximation of the distance

between the robot and the obstacles, as in (Schulman et al., 2013). In every iteration of Algorithm 5.5, the minimum distance between the robot and the obstacles is computed, and the closest points are denoted as $\mathbf{p}_i$ and $\mathbf{o}_i$, for the robot and the obstacles, respectively. Restriction RV2 is then defined as:

$$\mathbf{n}_i^T \mathbf{J}_{\mathbf{p}_i}(\mathbf{q}_i)\, \mathbf{N}\mathbf{S}_i^T\, \mathbf{d}_i \leqslant \|\overrightarrow{\mathbf{p}_i\mathbf{o}_i}\| - d_{safe} \quad \text{(RV2)} \tag{5.18}$$

where subindex $i$ denotes the $i$-th joint configuration in the joint trajectory, $\mathbf{n}_i = \overrightarrow{\mathbf{p}_i\mathbf{o}_i}/\|\overrightarrow{\mathbf{p}_i\mathbf{o}_i}\|$ is the unit normal vector from the robot to the obstacle, $\mathbf{J}_{\mathbf{p}_i}(\mathbf{q}_i)$ is the Jacobian matrix that maps joint velocities to velocities of the robot at the point $\mathbf{p}_i$, and $d_{safe}$ is a safety distance that ensures that the robot does not collide with the obstacles (usually set to zero or a small distance). The QP problem of Equation (5.13) is then modified to include restrictions RV1 and RV2 instead of R2, and the algorithm proceeds as in the standard method.

Another variant is the *online* variant, which is particularly useful for applications that demand immediate responses. Conversely to the standard method, which completely optimizes the joint trajectory until convergence, and then the robot starts moving, the online variant optimizes the joint trajectory while the robot is moving. This is achieved by running the optimization algorithm in parallel with the robot's movement, and updating the joint trajectory at every sampling period of the controller, interrupting Algorithm 5.5 at the end of every sampling period, and applying the first increment $\Delta\varphi$ to move the robot.

For instance, let us consider a robot with a controller operating at 20 Hz, which yields a sampling period of 50 ms. Consider a hypothetical task trajectory $\mathbf{x}(t)$ that has been discretized into $k = 100$ time values, for which the SMD has been generated, and a raw joint trajectory $\varphi$ has been generated. The online optimization variant would start optimizing the joint trajectory until the first sampling instant of the controller is reached, obtaining a partially optimized joint trajectory $\varphi^*$. At this point, the first joint increment $\Delta\mathbf{q}_1$ of $\Delta\varphi^*$ is applied to the robot, initiating the movement. Then, the optimization algorithm is resumed, but with a joint trajectory $\varphi^*$ of 99 points. The optimization algorithm will then continue iterating, until the next sampling instant of 50 ms is reached, at which point the first joint increment $\Delta\mathbf{q}_2$ of the partially optimized joint trajectory $\varphi^*$ is applied to the robot. This process is repeated until the joint trajectory $\varphi$ has been completely optimized, or until the robot has reached the final joint configuration $\mathbf{q}_k$. Due to its nature, the online variant is applied only to the raw joint trajectory with the lowest cost out of all the generated raw joint trajectories.

#### 5.2.4.4 *Variant selection criteria*

The proposed variants can be freely combined, and the user can select the most suitable combination for their application. However, some combinations

are more suitable than others depending on the application. In this section, we will provide a set of criteria to help the user select the most suitable combination of variants for their application.

The proposed framework is formed by three main stages: (1) SMD and graph generation, (2) generation of raw joint trajectories, and (3) optimization of joint trajectories. The chapter has been structured by presenting the original approaches for each stage in Sections 5.2.1, 5.2.2, and 5.2.3, respectively. The original approaches, however, may not be the most suitable for every application, or simply cannot be used in some cases. For these cases, we have proposed alternative variants for each stage throughout Section 5.2.4 in order to address different issues, such as computational speed, global optimality, or the ability to handle complex kinematic chains. It is important to note that some of these variants are interchangeable, but others are complementary. We trust that the explanations provided in the present section, and the examples introduced next, will suffice to help the reader select the most suitable combination of variants for their application. In Table 5.3, we summarize the recommended combinations of variants for different degrees of redundancy, complexity of the kinematic chain, and the desired performance in terms of global optimality and real-time performance.

Regarding generalization to complex kinematic chains, we have already discussed that the sampling approach for generating the SMD (Section 5.2.1) may not be suitable for certain kinematic chains whose IKP cannot be solved by algebraic elimination. For these cases, we recommend employing the continuation-based variant of Section 5.2.4.1, as it is directly applicable to any kinematic chain, regardless of its complexity. However, this variant is computationally expensive for higher degrees of redundancy, and is only capable of providing c-bundle chains that are reachable from the initial joint configuration $\mathbf{q}(0)$.

The dimensionalities of SMMs and SMD are also important factors to consider when selecting the most suitable combination of variants, which directly depend on the degree of redundancy $r$. The SMD generation stage is the stage that is most affected by the degrees of redundancy $r$. For a degree of redundancy $r = 1$, the SMMs within the SMD are 1-dimensional manifolds. In this case, the original continuation-based method, presented by Burdick (1989), is appropriate. However, for higher degrees of redundancy $r > 1$, the extension of the original method presented by Henderson (2002) is necessary. Despite its success in generating SMMs for higher degrees of redundancy, this method is computationally expensive, and therefore the sampling-based method presented in Section 5.2.1 is recommended when $r > 1$. We also recommend employing the sampling-based method for $r = 1$ when collisions are not involved, as we will justify in the experimental results section.

In terms of the second stage of the framework, the generation of raw joint trajectories, we have proposed two methods: the NN method and the hybrid

Table 5.3: Variant selection criteria. Please check Table 5.2 for a summary of every variant and the sections in which they are defined. *: or when solving Equation (5.1) for the remaining values of $\mathbf{q}$ (via algebraic elimination) is not simple and continuation is the only option (complex kinematic chains).

| **Main objective** $\rightarrow$ | **Global optimality of the solution** | **Real-time performance** |
|---|---|---|
| $r = 1$ **with collisions** * | **Continuation**-based SMD computation | **Continuation**-based SMD computation |
| | **Hybrid PA-NN** raw trajectory generation | **NN**-based raw trajectory generation |
| | **Continuation-friendly** + **standard** + **parallelized** optimization | **Continuation-friendly** + **online** optimization |
| $r > 1$ **(or** $r = 1$ **without collisions)** | **Sampling**-based SMD computation | **Sampling**-based SMD computation |
| | **Hybrid PA-NN** raw trajectory generation | **NN**-based raw trajectory generation |
| | **Standard** + **parallelized** optimization | **Online** optimization |

PA-NN method. Using the hybrid NN-PA approach reduces the time required for post-processing the trajectory, but increases the time required to generate the raw joint trajectories. However, when immediate responses are required, the NN method is recommended, as it is computationally efficient and is able to provide a solution in a matter of milliseconds. Therefore, when the application requires immediate responses, the NN method is recommended. Conversely, when the global optimality of the solution is the main objective, the hybrid PA-NN method is recommended.

Finally, the optimization stage can be adapted to provide a balance between global optimality and computational speed. When the global optimality of the solution is the main objective, and the application allows for the sufficient time to optimize the trajectory, the standard method is recommended, as it will provide the globally optimal solution for each c-bundle chain, as Section 5.4.1 will demonstrate. On the contrary, when the application requires the fastest possible response, the online variant, together with the NN method for generating raw joint trajectories, is recommended. This combination provides an almost instantaneous response after the SMD generation stage, and the robot can start moving while the trajectory is being optimized in parallel, refining the trajectory as the robot moves. When the application requires a balance between global optimality and computational speed, we recommend employing the standard method for the optimization stage, but with a restricted time window for the optimization. This means that the optimization algorithm will be interrupted after a certain time, and the partially optimized trajectory will be used to move the robot. This approach could be paired with the online variant when the time window has been reached. Additionally, the user could select the appropriate time window based on the application requirements, gradually transitioning the objective of this last stage from optimizing the trajectory when the time window is large, to simply smoothing the trajectory when the time window is small.

This compromise between global optimality and computational speed also present in the literature. For example, Albu-Schäffer and Sachtler (2023) describe their method as *rather expensive* in terms of computational time. To alleviate this issue, and to extend their method to higher task dimensionalities, they employ neural networks to approximate their solution to the redundancy problem. This approximation inevitably compromise the optimality of the solution, but it allows for a faster response. Conversely, works such as (Ferrentino and Chiacchio, 2020; Lück, 1997) choose not to compromise the optimality of the solution, and therefore their methods are computationally expensive. In our proposed framework, instead of forcing one of these two extremes, we provide the user with the flexibility to select the most suitable combination of variants for their application, allowing them to choose between global optimality and computational speed, or find a balance between the two.

Three different simulated scenarios are presented in this section to evaluate the performance of the proposed framework in solving the redundancy problem. As evaluation metrics, we will use the computational time required to execute the different stages of the framework, and the value of the cost function defined in Equation (5.5) for the generated joint trajectories. The simulations were performed in an Intel Core i7-9700F CPU with 32 GB of RAM, under Ubuntu 24.04. The code was implemented in Python 3.11, and libraries FCL 0.7 and ProxSuite 0.6.2 were used for collision detection and QP solving, respectively.

### 5.3.1 *Example 1: 3R planar manipulator*

The first example is a 3R planar manipulator tasked with following 2-dimensional task trajectory $\mathbf{x}(t) = [x, y]^T$. The end-effector position is required to follow a linear path joining the points $(2.457, -0.793)$ and $(-2.4, 1.5)$, given in meters, relative to the base of the robot. The task must be completed in 5 seconds, with constant velocity. The initial joint configuration of the robot is $\mathbf{q}(0) = [\pi/8, -\pi/4, -\pi/6]^T$ radians. As additional constraints, two ellipses that the entire manipulator must avoid are defined, with the following inequalities:

$$\begin{aligned} \frac{(x-0.5)^2}{1^2} + \frac{(y-0.7)^2}{0.25^2} &\leqslant 1 \\ \frac{(x-1)^2}{0.3^2} + \frac{(y+1)^2}{0.2^2} &\leqslant 1 \end{aligned} \tag{5.19}$$

Due to such demanding collision constraints, the joint limits were relaxed to $[-2\pi, 2\pi]$ radians for all joints.

Regarding the variants selected for this example, we will use the continuation approach for the SMD and graph generation stage, which is suitable when the kinematic equations of the robot are too complex to be solved by algebraic elimination. Note that this is not the case for the 3R planar manipulator, but we start with this choice as an initial benchmark for its later comparison with the sampling-based approach. As for the parameters of the SMD generation, we discretized the trajectory into $k = 100$ time values, resulting in a time step of $\Delta t = 0.05$ seconds, which provides a fine-enough resolution for the problem analysis. A value of $\rho = 0.3$ meters was used for the continuation-based generation of the SMD.

In terms of the optimality of the solution, we selected the standard optimization method to generate the overall globally optimal joint trajectory. Having chosen the continuation-based variant for the SMD generation, the point cloud $\bar{\mathcal{P}}_i$ of unfeasible joint configurations is not available, and therefore the restriction

R2 cannot be used. Instead, we will use the continuation-friendly variant, which replaces restriction R2 with the joint limits restriction RV1 and the local approximation of collisions RV2.

Finally, we chose the NN method for generating the raw joint trajectories, as it is computationally efficient and provides a good enough solution for this example.

Figure 5.6 show the results of the simulation. A video of the simulation can be found at https://ars.els-cdn.com/content/image/1-s2.0-S0094114X25001090-mmc3.mp4. The top two rows (Figure 5.6(a-f)) display six snapshots of joint space (top row) and task space (second row) at different time values t. The different time values correspond to the initial and final instants ($t = 0$ and $t = 5$), as well as the intermediate time values $t = \{0.05, 0.9, 1.55, 3.3\}$: instants at which the task trajectory traverses co-regular surfaces. Additionally, the graph yielded by the SMD generation stage is shown in Figure 5.6(g), where nodes are c-bundles and edges are manifold portions of co-regular surfaces. Note that, for simplicity, we represent c-bundle graphs with only topological information, i.e., nodes are circles without timespan information. However, in this example, we have manipulated the nodes in order to faithfully represent the time dimension. That is, every c-bundle is spanned horizontally (in the time dimension) from the time value at which it starts to the time value at which it ends.

The second stage of the framework determined four possible c-bundle chains, which are highlighted in Figure 5.6(g) with different colours, one for each c-bundle chain. The top row of Figure 5.6(a-f) (i.e., joint space) contains the optimized joint trajectories that traverse each c-bundle chain, as well as the SMMs (dotted curves) at the corresponding time values, with the corresponding configuration $\mathbf{q}_i$ for each joint trajectory marked with a black cross. In the task space (i.e., second row of Figure 5.6(a-f)), the commanded task trajectory is shown in green, while the robot is represented adopting the joint configurations $\mathbf{q}_i$ at the corresponding time values for each joint trajectory.

We have followed two colour schemes for Figure 5.6. The first one is the one used for the c-bundle (nodes of the graph), where each c-bundle chain is represented with a different colour. This scheme is also used for plotting the SMM in the joint space, where each SMM is represented with the same colour as the c-bundle it belongs to. The second colour scheme is used for differentiating every c-bundle chain (and derived joint trajectory). Both the optimal joint trajectory plotted in the joint space and the corresponding robot configuration in the task space are represented with the same colour as the c-bundle chain they belong to.

Table 5.4 gathers the computational times spent in the different stages of the framework, as well as the cost of the generated joint trajectories. The runtimes are given in milliseconds (ms), and the cost of the joint trajectories is given by the

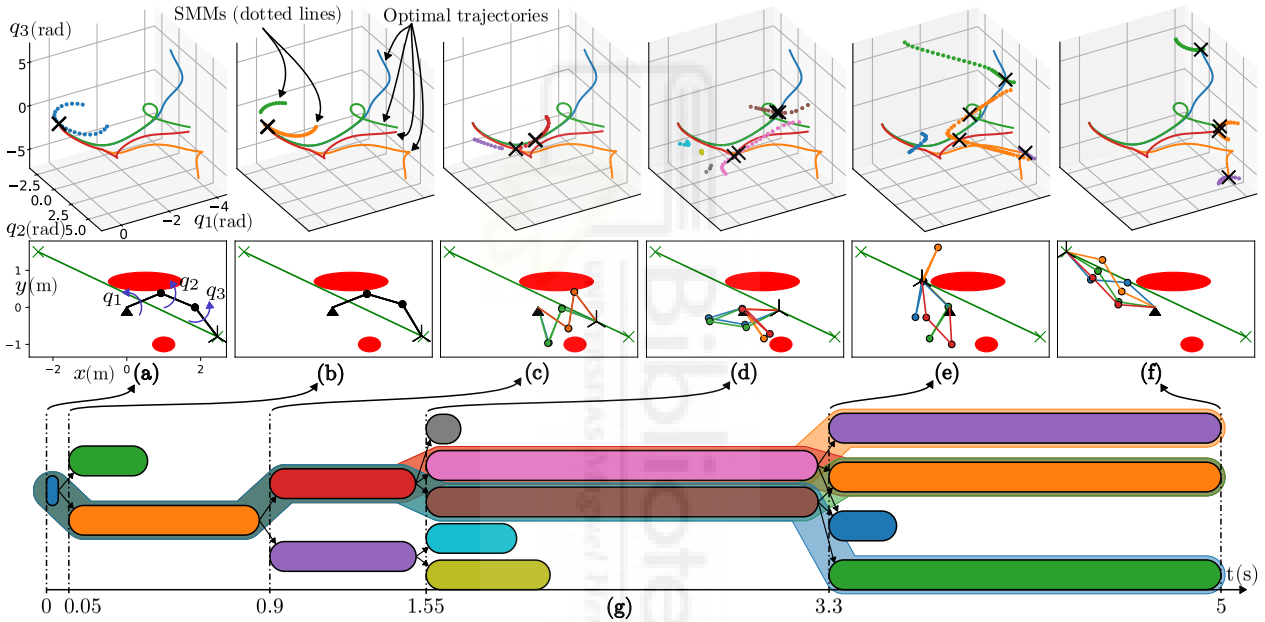


Figure 5.6: (a-f) Snapshots of the simulation. (g) C-bundle graph $\mathcal{G}$ and every possible c-bundle chain.

cost function defined in Equation (5.5), which quantifies the sum of the squared joint velocities along the trajectory. The time values shown in the table are the average of 100 runs of the simulation.

The globally optimal joint trajectory corresponds to the one that traverses c-bundle chain $\mathcal{CBC}_4$ (depicted in red in Figure 5.6), which has the lowest cost of 1.195. Regarding the computational times, the SMD generation stage took 2092 ms using the continuation-based method, which is a reasonable time considering that the SMD characterizes the infinite set of joint configurations that track the task trajectory. We also ran the sampling-based method for the SMD generation stage, which took 2973 ms. We believe that the continuation method outperforming the sampling method in this case is due to the significant overhead of computing whole-body collisions. In the case of the sampling-based method, collisions must be checked for every sampled joint configuration, which is computationally expensive. Conversely, the continuation method will stop checking for collisions as soon as the first collision is detected at each end of the manifold (if it is a curve ($r = 1$)), which produces a significant reduction in the number of collision checks, which outweighs the additional computational cost of this method (i.e., Jacobian's null space computation and Newton). However, this will not be the case for higher degrees of redundancy, where the sampling-based method will outperform the continuation method, as we will show in the next examples.

The generation of raw joint trajectories took 20 ms, a significantly short time as the traversal of the graph is a simple operation, and the NN search is efficient due to the use of k-d trees. The largest time was spent in the optimization stage, which took between 2509 ms and 4056 ms, depending on the c-bundle chain traversed. This suggests that the optimization stage is a suitable candidate for parallelization, given that the CPU possesses enough cores, as the optimization of each c-bundle chain is independent of the others, and the total time could be reduced from the sum of the individual times to the maximum time of all c-bundle chains. In practice, if this strategy was implemented, the total time of the framework would be reduced to 6.2 seconds.

Finally, we conducted an additional test to evaluate the framework's capability to manage collisions among links. We incorporated collisions between the first and third links, emulating a scenario where the two links lied on the same plane of the robot assembly. The results of the experiment can be found in the video at https://ars.els-cdn.com/content/image/1-s2.0-S0094114X25001090-mmc2.mp4, and in Figure 5.7, which shows snapshots of the simulation at the beginning, when crossing co-regular surfaces, and at the end.

Table 5.4: Computational times and costs of the generated trajectories for the 3R planar manipulator.

| **SMD and graph generation** | **Continuation: 2092 ms** | | *Sampling: 2973 ms* | |
|---|---|---|---|---|
| **Generation of raw trajectories** | **20 ms** | | | |
| *C-bundle chains calculation* | *1 ms* | | | |
| | $\downarrow \mathcal{CBC}_1$ (blue) | $\downarrow \mathcal{CBC}_2$ (orange) | $\downarrow \mathcal{CBC}_3$ (green) | $\downarrow \mathcal{CBC}_4$ (red) |
| *Raw trajectories* | *4 ms* | *5 ms* | *4 ms* | *6 ms* |
| **Optimization (standard)** | **4056 ms** | **2509 ms** | **3715 ms** | **4010 ms** |
| **Trajectory cost** | **2.144** | **1.591** | **2.07** | **1.195** |

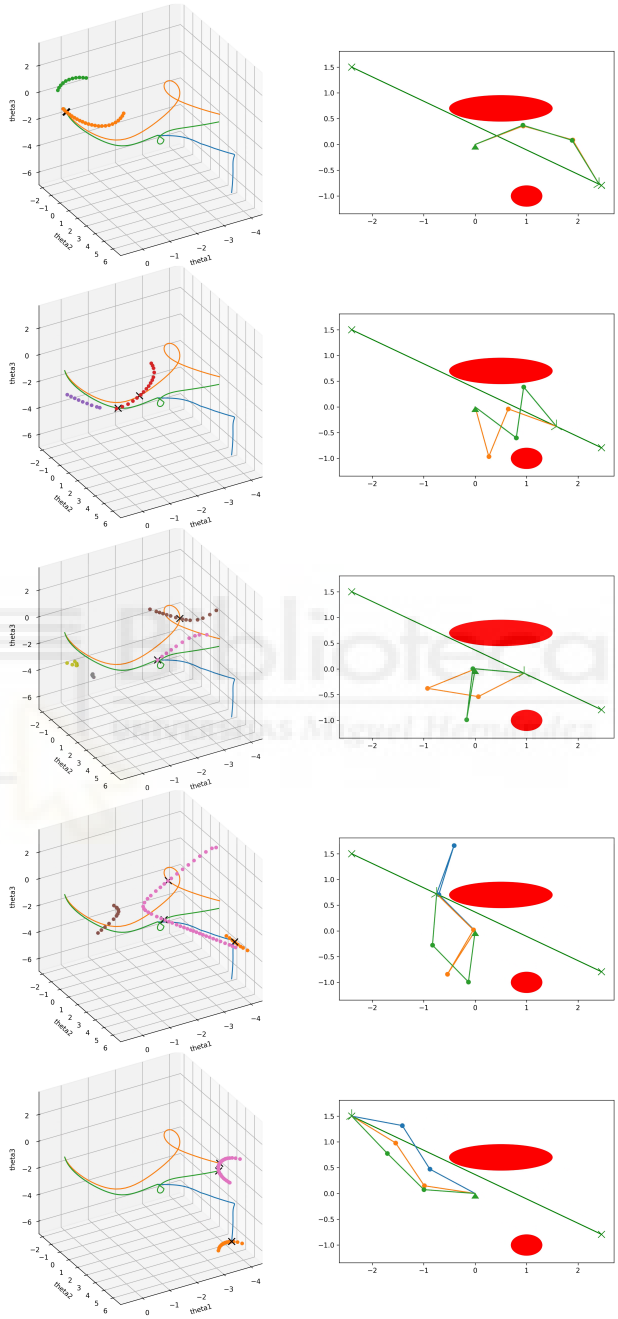


Figure 5.7: Snapshots of the simulation with link-link collisions.

### 5.3.2 *Example 2: HyReCRo robot*

In this example, we consider a more complex scenario, consisting of the serial equivalent of the HyReCRo robot (i.e., RPRRRRPR), which is tasked with following a 6-dimensional trajectory, which includes end-effector position and orientation. The kinematic model and equations are given in (Peidró et al., 2016a).

This results in a significantly demanding scenario, with $n = 8$ and $r = 2$, which necessitates an efficient combination of framework variants in order to keep the runtimes within reasonable limits. Regarding the SMD generation stage, we selected the sampling-based method, as it is the most efficient method for generating the SMD when DoF and redundancy are high. For the optimization stage, we selected the online optimization variant in order to test it against the standard optimization method. In fact, to test the worst-case scenario, we will only perform a single iteration of the online optimization between every sampling period of the controller. Finally, for the raw joint trajectory generation, we selected the hybrid combination of the NN and PA methods. For the PA of choice, we implemented a simple depth-first search (DFS) algorithm to generate the raw joint trajectories. Note that the selection of the hybrid variant is done to yield a better trajectory at this stage, in order to increase the chances of picking the raw trajectory that would become the globally optimal trajectory after the optimization stage.

Two different task trajectories are defined for this example. In the first one, the free gripper of the robot is required to follow 9 waypoints in 5 seconds, which have been interpolated with a cubic spline. This trajectory avoids an obstacle in the workspace while keeping the gripper at a constant orientation. We have also considered joint limits as additional constraints (Peidró et al., 2016a). As with the previous example, the trajectory was discretized into $k = 100$ time values, resulting in a time step of $\Delta t = 0.05$ seconds. The initial configuration was selected arbitrarily as $\mathbf{q}(0) = [-0.055, 0.18, -0.055, -1.505, -3.076, 0.273, 0.187, 0.273]^T$, where revolute joints are in radians and prismatic joints are in meters. Steps of $\Delta q = 0.1$ radians for revolute joints and $\Delta q = 0.05$ meters for prismatic joints were used for the sampling-based method.

The simulation can be found at https://ars.els-cdn.com/content/image/1-s2.0-S0094114X25001090-mmc6.mp4. Figure 5.8(a-e) shows snapshots of the simulation at different time values $t = \{0, 2.3, 2.75, 3.95, 5\}$, corresponding to the initial instant, the traversal of the three co-regular surfaces, and the final instant, respectively. The layout of the figure is also similar to that of Figure 5.6, with the top row showing the joint space and the second row showing the task space. In addition to the single joint trajectory that is output by the online optimization variant, we also show the two joint trajectories that were output by the standard optimization variant for comparison. We also share the same colour scheme as in

Figure 5.6, where (1) the c-bundle chains are represented with different colours, with their respective joint trajectories and robot representations sharing the same colour, and (2) the SMMs are represented with the same colour as the c-bundle they belong to.

Green and orange trajectories correspond to the globally optimal joint trajectories within each c-bundle chain, while the blue trajectory corresponds to the joint trajectory generated by the online optimization variant (which traverses the c-bundle chain drawn in green). It can be observed how the online optimization variant produces a joint trajectory that is visually almost indistinguishable from the globally optimal joint trajectory, as the future performance data will confirm.

It is important to mention that the resulting SMMs are ($r = 2$)-dimensional manifolds in the ($n = 8$)-dimensional joint space, which is why the SMMs are represented as surfaces (approximated by point clouds) in the top row of Figure 5.8(a-e). Since the 8-dimensional joint space cannot be visualized directly, we have projected the SMMs onto the 3-dimensional space of the first, fourth and sixth joints of the HyReCRo robot, which are denoted as $(\varphi_{1A}, \theta_A, \varphi_{2B})$ in the figure. This projection is the reason why the blue and orange SMMs at the initial instants seem to be connected in Figure 5.8(a), when in reality they are not.

Table 5.5 summarizes the computational times and costs of the generated joint trajectories. Note that, in comparison with the previous example, although the dimensionality of the problem has increased significantly, the computational times of the framework are within the same order of magnitude, which is a remarkable result. This is mainly due to the efficient sampling-based method for generating the SMD. Further examples of the performance of sampling-based methods for computing SMMs can be found in (Peidró et al., 2024). Regarding the optimization stage, using the online variant significantly reduced the computational time of the framework, as the movement of the robot could be initiated as soon as the raw joint trajectory was generated, with the optimization algorithm further refining the trajectory in parallel with the robot's movement. It is worth noting that the time taken to perform an iteration remains under 30 ms, which is a reasonable time for a controller operating at 20 Hz. By comparing the last two rows of Table 5.5, we can see that the cost of the joint trajectory generated by the online optimization variant is slightly higher than that of the standard optimization variant, which is expected. However, the difference is negligible, as observed both in the table and in the simulation snapshots, where the joint trajectories are visually indistinguishable.

This experiment, which depicts a scenario more complex than Section 5.3.1, demonstrates the advantages of the online optimization variant, as well as the efficiency and scalability to higher dimensionalities of the sampling-based method. As discussed in Section 5.2.1, the sampling approach requires sweeping or scanning through every possible combination of $r$ joint coordinates, which produces

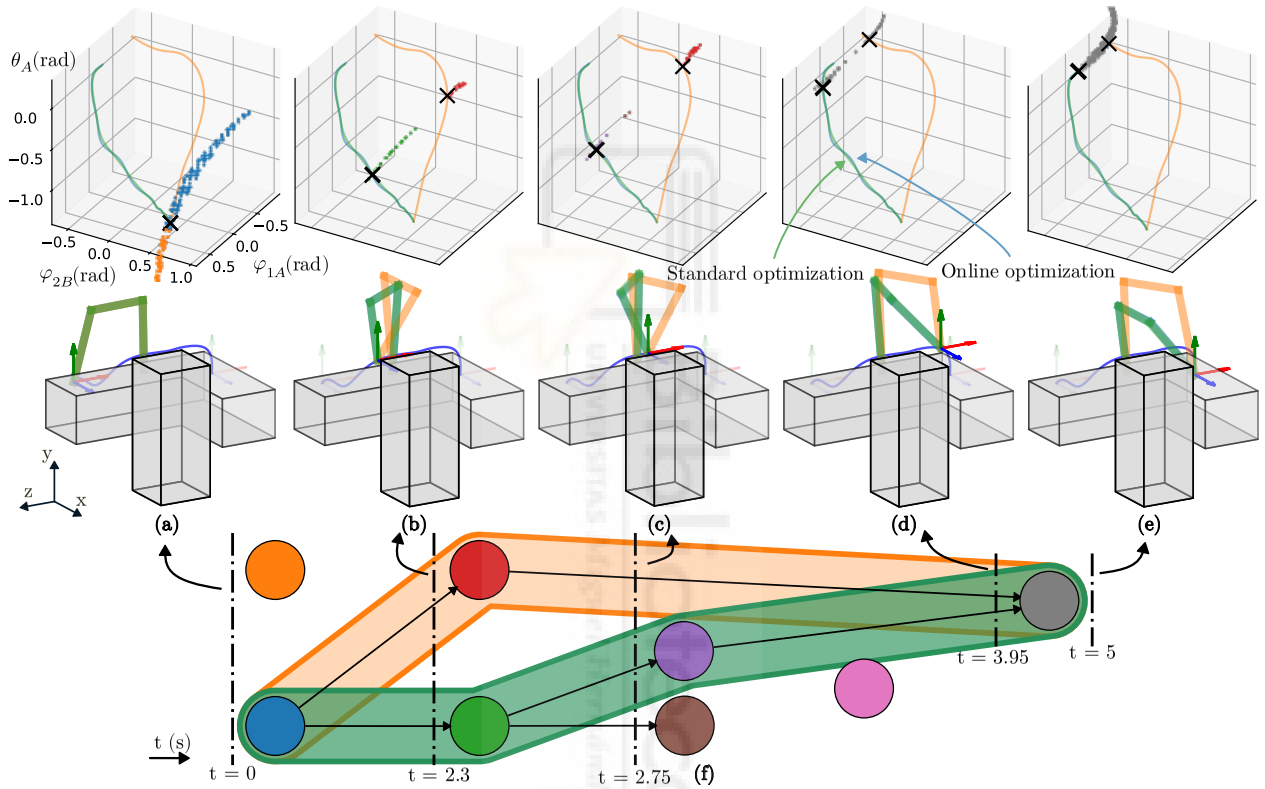


Figure 5.8: (a-e) Snapshots of the simulation. (f) C-bundle graph $\mathcal{G}$ and every possible c-bundle chain.

Table 5.5: Computational times and costs of the generated trajectory for the RPRRRRPR robot.

| | |
|---|---|
| **SMD and graph generation** | **3917 ms** |
| *Sampling* | *2957 ms* |
| *Clustering* | *896 ms* |
| *Matching* | *64 ms* |
| **Generation of raw trajectories** | **16 ms** |
| *C-bundle chains calculation* | *1 ms* |
| *Raw trajectory* | *15 ms* |
| **Online optimization (average time per iteration)** | **26 ms** |
| *Standard optimization* | *6043 ms* |
| **Trajectory cost (online optimization)** | **0.119** |
| *Trajectory cost (standard optimization)* | *0.111* |

a number of scans equal to the binomial coefficient $\binom{n}{r}$. This means that the number of scans grows factorially with the number of DoFs. In the example presented in this section, the binomial coefficient is $\binom{8}{2} = 28$. Generating the SMD for an 8-DoF robot with $r = 2$ in the same order of time as a 3-DoF robot with $r = 1$ is a remarkable result, and demonstrates the efficiency of the sampling-based method.

However, some of these 28 combinations of redundant joint coordinates are unsolvable, due to the fact that, when having a fixed task value $\mathbf{x}$, some redundant joint coordinates become dependent on the others. Specifically, the HyReCRo robot has 3 combination of joint coordinates that are unsolvable, yielding a total of 25 solvable combinations. In addition, when solving some of these combinations, it becomes necessary to find the roots of polynomials of degree higher than 4, which is computationally expensive. In these cases, we chose to skip these combinations and not include them in the SMD generation stage, resulting in a less densely sampled SMD. However, thanks to the nature of the optimization stage, which does not rely on the precalculated SMD, the costs of the final joint trajectories are not affected by this decision. This is due to the fact that, as we will demonstrate in Section 5.4.1, any starting joint trajectory within a c-bundle chain will converge to the globally optimal joint trajectory within that c-bundle chain, regardless of the density of the SMD. In the results we have presented in Figure 5.8 and Table 5.5, the SMD was computed omitting 10 combinations for solving the IKP, significantly reducing the computational time of

the SMD generation stage. Figure 5.9 and the video located at https://ars.els-cdn.com/content/image/1-s2.0-S0094114X25001090-mmc5.mp4 show the same example, but with the SMD generated using every possible combination. It can be observed how the SMD and the extracted joint trajectories are not affected by the omitted combinations. The only difference is that the SMD is denser, and a couple of very small c-bundles are created and instantly vanish, but these could be considered noise, and do not affect the final result.

The second task trajectory given to the HyReCRo robot emulates a concave transition during the exploration of a vertical truss structure that the robot is required to climb. The starting pose of the robot is the same as in the previous example. This time, the free gripper is required to reach a pose located at a vertical beam of the truss structure, to which the robot must attach the gripper.

The video of the simulation can be found at https://ars.els-cdn.com/content/image/1-s2.0-S0094114X25001090-mmc4.mp4, and Figure 5.10 shows snapshots of the simulation at the initial instant, the traversal of co-regular surfaces, and the final instant.

### 5.3.3 *Example 3: RPR planar manipulator*

This last example introduces a brief experiment to demonstrate the exhaustive nature of the framework. In scenarios with substantial constraints, such as very limited joint limits or large obstacles, the resulting graph is often sparse. In these cases, such demanding constraints reduce the feasible regions of the joint space, leading to the removal of many c-bundles, which limits the number of c-bundles and c-bundle chains that successfully achieve the task. Conversely, in scenarios with less demanding constraints, the graph is denser, and the number of c-bundles and c-bundle chains increases. In this example, we aim to illustrate the exhaustive capabilities of the proposed method for handling scenarios with numerous c-bundles and c-bundle chains.

We will use an RPR (Revolute-Prismatic-Revolute) planar manipulator tasked with following a one-dimensional end-effector trajectory in the task space, as shown in Figure 5.11(a). The task is defined as a parabolic trajectory of the $y$ coordinate of the end-effector, with the following equation:

$$y(t) = -6.662t^2 + 8.162t - 1.5 \tag{5.20}$$

to be completed in 1 second. This results in a degree of redundancy of $r = 2$, with SMMs being surfaces. An elliptic obstacle that the end-effector must avoid is defined with the following inequality:

$$\frac{(x-1.1)^2}{1^2} + \frac{(y+0.2)^2}{0.25^2} \leqslant 1 \tag{5.21}$$

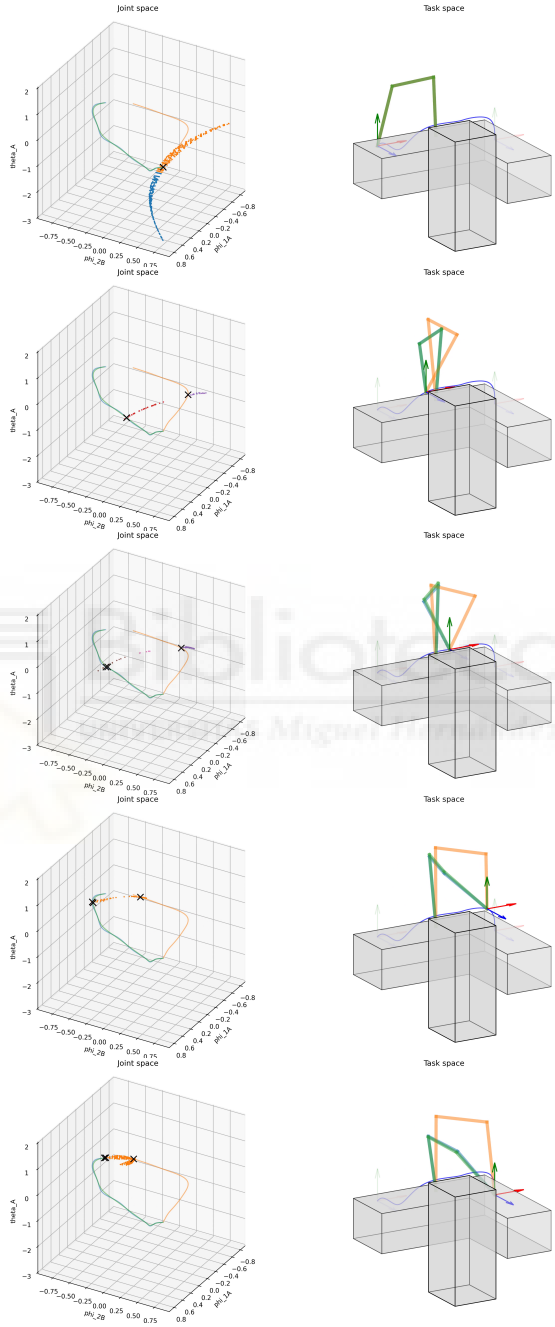


Figure 5.9: Snapshots of the simulation with all combinations of joint coordinates.

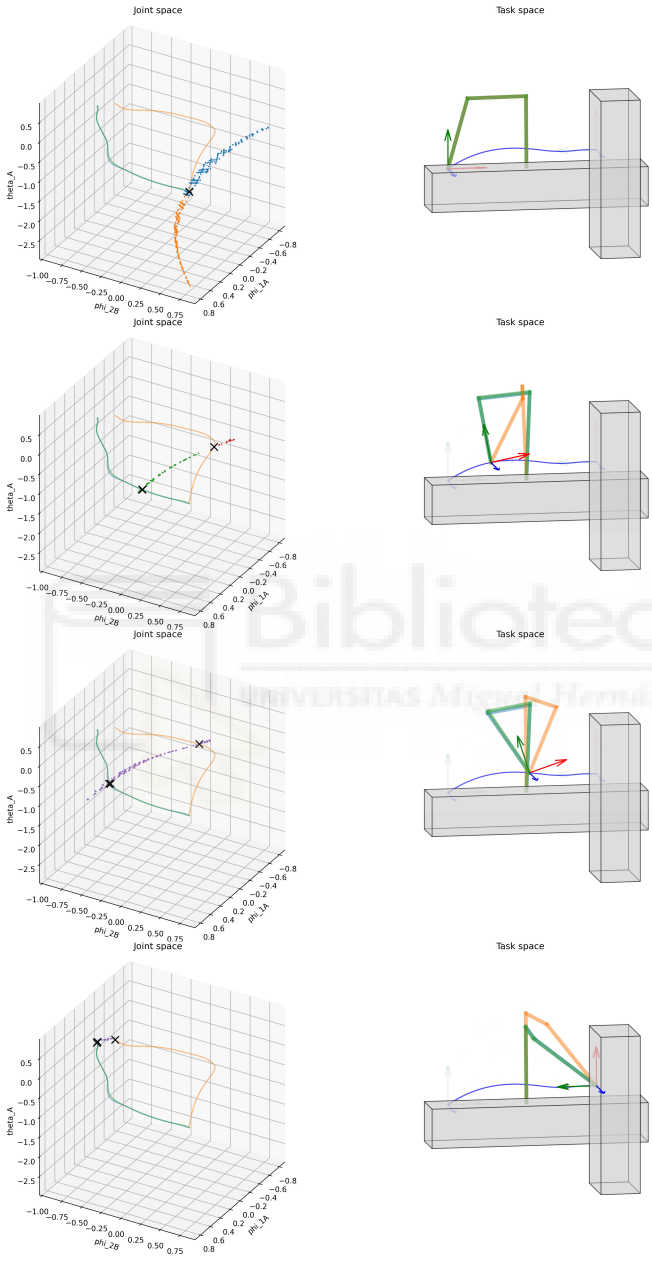


Figure 5.10: Snapshots of concave transition simulation.

Finally, the joint limits are set to $[-2\pi, 2\pi]$ radians for the revolute joints and $[0, 0.5]$ meters for the prismatic joint.

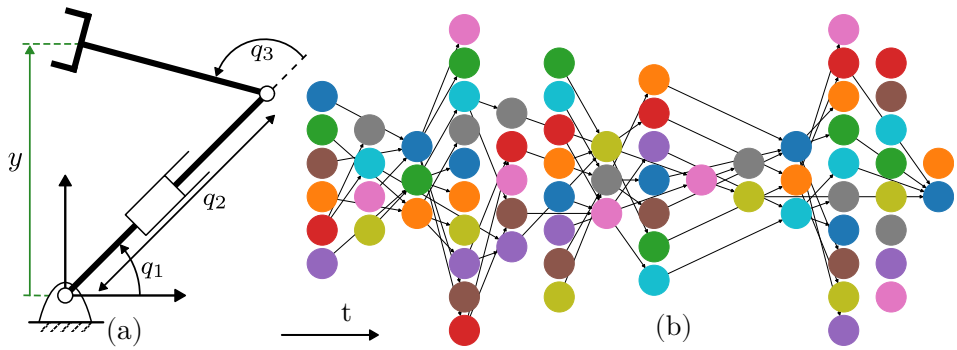


Figure 5.11: (a) RPR planar manipulator. (b) C-bundle graph $\mathcal{G}$.

The obtained graph is shown in Figure 5.11(b). The graph is much denser than the previous examples, due to the less demanding constraints. The graph contains 72 c-bundles connected through 68 manifold portions of co-regular surfaces, which indicate feasible transitions between c-bundles. The rest of the results are omitted, since the focus of this example is to demonstrate the exhaustive nature of the framework.

Additionally, we have conducted a test to evaluate the effectiveness of the implemented vectorization of the SMD generation stage (Section 5.2.1). When employing Algorithm 5.2 sequentially (i.e., without vectorization) to compute a single SMM at task point $y = -1.5$ m, the time taken was 82.909 ms. When the inner loop of the algorithm was vectorized, the time taken was reduced to 1.675 ms, which is a 49.5 times speed-up. When extending this comparison for the entire SMD generation stage of the task trajectory presented at the beginning of this section, the runtime was reduced from 10.614 seconds to 0.194 seconds, which is a 54.7 times speed-up. From this, we can estimate that the vectorization of the SMD generation stage is capable of reducing the time taken to compute the SMD by a factor of approximately 50, which is a significant improvement.

## 5.4 DISCUSSION

This section discusses some of the key aspects of the proposed framework, including its global optimality, complexity and dimensionality analysis, and a comparison with state-of-the-art methods.

### 5.4.1 *Global optimality of the method*

The term *global optimality* refers to the ability of a redundancy resolution method to find the joint trajectory $\mathbf{q}(t)$ that *globally* minimizes a given cost function, such as the one defined in Equation (5.5), for the entire task trajectory $\mathbf{x}(t)$.

Recall that, in the framework proposed in this chapter, we identify the infinite set of joint configurations that can track the task trajectory $\mathbf{x}(t)$, by means of the SMD generation stage. Then, we identify every possible c-bundle chain that traverses the SMD and completes the task. Considering that the SMD will certainly contain the globally optimal joint trajectory (since it contains all possible joint configurations that can track the task trajectory), and that we extract every possible c-bundle chain from the SMD, we can conclude that the globally optimal joint trajectory will correspond to the optimal joint trajectory within one of the c-bundle chains. This means that, if the optimization stage is able to find the globally optimal joint trajectory within each c-bundle chain, then the overall joint trajectory will be globally optimal for the whole SMD. While we are not able to provide a formal proof of this statement, we can offer a statistical demonstration of the global optimality of the method.

We have conducted a simulation similar to the 3R planar manipulator example described in Section 5.3.1, where four different c-bundle chains were extracted that complete the task. However, instead of generating the raw joint trajectories using any of the methods proposed in this chapter, we generated completely random joint trajectories within each c-bundle chain. The random joint trajectories were generated by modifying line 6 of Algorithm 5.4, where instead of selecting the nearest point in $\mathbf{MP}_i$, we randomly selected a point in the manifold portion $\mathbf{MP}_i$. This slight modification allows us to generate random joint trajectories that traverse each c-bundle chain. We then optimized the random joint trajectories using the standard optimization method, until convergence. We run 2350 simulations, and the results are plotted in Figure 5.12 in the form of a histogram per c-bundle chain.

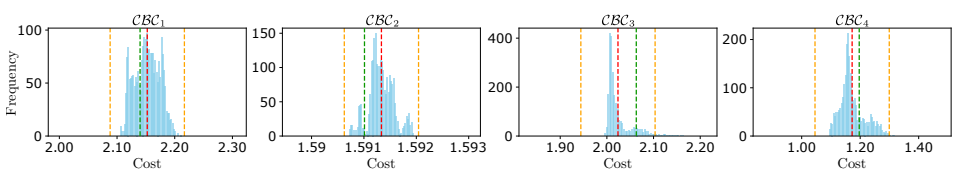


Figure 5.12: Histogram of the cost of the optimized random joint trajectories.

Each histogram shows vertical bars, where the horizontal placement of the bars corresponds to the returned cost of the optimized joint trajectory, and the height of the bars corresponds to the frequency this cost was returned. We in-

dicated the mean cost value with a vertical red line, and the two orange lines indicate three standard deviations from the mean. The results obtained in the experiment of Section 5.3.1 are also shown in the histograms with vertical green lines. The histograms show that every random joint trajectory within each c-bundle chain converged to the same cost value within a small margin of error. This small margin of error is due to the value we selected for the convergence threshold of the optimization algorithm $\varepsilon$, which we set to a large enough value to be able to visualize such distribution.

This demonstrates that the optimization stage is able to find the globally optimal joint trajectory within each c-bundle chain, regardless of the initial joint trajectory. Therefore, we can conclude that the overall joint trajectory generated by the framework is globally optimal. While this is not a formal proof, it is a strong statistical demonstration of the global optimality of the method.

### 5.4.2 *Complexity and dimensionality analysis*

The first step of the framework is the generation of the SMD, which probably is the most computationally expensive stage of the framework. In this stage, we sweep every combination of $r$ of the $n$ joint coordinates, which results in a number of combinations equal to the binomial coefficient $\binom{n}{r}$. Then, for each of these combinations, $N^r$ points are sampled in the joint space (where $N$ is the number of samples per joint coordinate), and the IKP is solved for each of these points, which yields up to "$a$" solutions for a given sampled point. Note that this process is performed for every $k$ total task points in which the task trajectory is discretized. If we consider the worst-case scenario, where every sampled point yields "$a$" solutions, the computational complexity of the SMD generation stage is given by:

$$\mathcal{O}\left(k \cdot \binom{n}{r} \cdot N^r \cdot a\right) \tag{5.22}$$

The same complexity applies to the NN-based raw joint trajectory generation stage, as the complexity of Algorithm 5.4 is dictated by the nearest-neighbour search, whose worst-case complexity equals to the number of points in the current SMM. If this is extended for the entire trajectory, the complexity of the NN-based raw joint trajectory generation stage matches that of the SMD generation stage:

$$\mathcal{O}\left(k \cdot \binom{n}{r} \cdot N^r \cdot a\right) \tag{5.23}$$

If, instead of the NN method, we use the hybrid method, the complexity of the raw joint trajectory generation stage is given by the complexity of the pathfind-

ing algorithm of choice, multiplied by the times it is executed, which is equal to the number of c-bundle chains.

From this analysis, it is clear that the computational complexity of the framework grows factorially with the number of DoFs $n$ and exponentially with the degree of redundancy $r$. In fact, few works have attempted to tackle the global redundancy resolution problem for robots with a degree of redundancy higher than 2, as the curse of dimensionality makes the problem of identifying the infinite set of solutions (i.e., SMMs or FMs) intractable in a reasonable time. Literature approaches for computing SMMs range from hundreds (Yang et al., 2021) (for $r = 1$) to thousands of seconds (Wu et al., 2023) (for $r = 2$) to compute the SMMs at a single task point. In comparison, we have successfully reduced the computational time for computing SMMs to the order of milliseconds (and seconds for the entire SMD), even for an 8-DoF robot with $r = 2$, as demonstrated in the example of Section 5.3.2. Moreover, when working with complex kinematic chains, for which it is necessary to use a generalizable method, such as the continuation-based approach, we have managed to implement the method of (Burdick, 1989) with a computational time similar to that of the sampling-based method for $r = 1$, and the method of (Henderson, 2002) for the $r = 2$ HyReCRo example, with a computational time of around 1 second per SMM.

Additionally, for complex kinematic chains that necessitate a generalizable method (continuation), we implemented the method from (Burdick, 1989), achieving computational times comparable to those of the sampling-based method for $r = 1$. For the $r = 2$ HyReCRo example, we employed the method from (Henderson, 2002), which requires approximately 1 second per SMM.

The computational optimization strategies discussed throughout this chapter, such as the use of efficient data structures like k-d trees, the vectorization of the SMD generation stage, and the parallelization of the optimization stage, are crucial for achieving these computational times, and essential to implement the framework in resource-constrained robotic control systems.

When studying the literature, and extending to higher degrees of redundancy the presented framework, it seems that $r = 3$ is out of reach any method aspiring to solve the global redundancy resolution problem in a reasonable time. The proposed framework is capable of handling up to $r = 2$ in a reasonable time, and we have shown that the sampling-based method is the most efficient method for generating the SMD in this case. A possible suboptimal approach for tackling $r \geqslant 3$ is to virtually remove redundancy levels by sequentially fixing joint coordinates, until the degree of redundancy is reduced to $r = 2$. Then, the framework can be applied to the reduced problem. The same strategy could be applied for sudden joint failures where a joint becomes locked. In this case, the existing SMD could be intersected with a hyperplane defined by $q_j = b$, where $q_j$ is the failed joint, and $b$ is the fixed value of the joint. This would avoid the recomputation

of the SMD, but the graph should still be recomputed, as the c-bundles would change, and subsequent stages of the framework would still be required to be recomputed.

### 5.4.3 *Comparison with state-of-the-art methods*

Out of the state-of-the-art methods for global redundancy resolution discussed in Section 2.3.2, only a few can be directly compared to our approach, as most focus on addressing redundancy across the entire workspace or integrate task trajectory planning. For instance, Ferrentino and Chiacchio (Ferrentino and Chiacchio, 2020) tackle the global redundancy resolution problem for a specific task trajectory, while the authors of (Hauser and Emmons, 2018) propose an algorithm (supplementary to their main contribution) to construct the SMD and establish paths within it. Due to the lack of publicly available repositories or benchmarks for a direct comparison, we have made every effort to faithfully replicate the methods described in these two studies. To this end, we implemented their approaches using the scenario of the 3R planar manipulator presented in Section 5.3.1.

In (Ferrentino and Chiacchio, 2020), redundancy is addressed by using FMs rather than SMMs. To apply their method to our example in Section 5.3.1, we selected $q_1$ as the redundant variable and discretized its range $[-2\pi, 2\pi]$ into $N_u = 144$ points, ensuring a resolution comparable to that in Section 5.3.1; the time axis was similarly discretized into $N_i = 100$ points. This configuration establishes a fair comparison scenario.

For the cost function, we adapted Equation (5.5) to match the formulation in (Ferrentino and Chiacchio, 2020) by defining their function $l$ as the norm of the difference between each pair of tested joint configurations, i.e., $l = \|\mathbf{q}(t_1) - \mathbf{q}(t_2)\|$. The resulting solution is shown in Figure 5.13, where the FMs are displayed in the top row. In these maps, purple regions indicate areas where the configurations yield complex IKP solutions when solving for $q_2$ and $q_3$ (given each value of $q_1$ and $\mathbf{x}(t)$, two solutions exist, which is why two FMs appear in the figure). Red regions denote collisions with the elliptical obstacles, while yellow regions represent pairs $(t, q_1)$ that result in $q_2$ and $q_3$ values violating the joint limits. Uncoloured regions correspond to feasible configurations.

The bottom-left subfigure of Figure 5.13 presents the cost map of the FM that leads to the optimal solution, rendered with the *viridis* colourmap (dark purple indicates the lowest cost, yellow the highest). The bottom-right subfigure compares the optimal joint trajectories: the time histories of $q_1(t)$, $q_2(t)$, and $q_3(t)$ obtained via our implementation of (Ferrentino and Chiacchio, 2020)'s method are plotted with solid lines, while those provided by our method are shown with

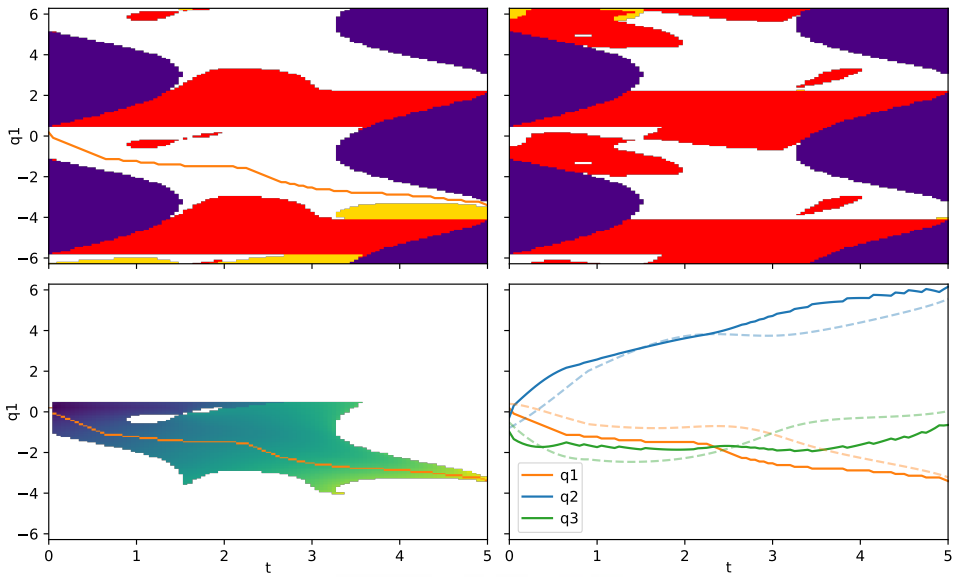


Figure 5.13: Results for the experiment with the 3R planar manipulator using the method proposed by Ferrentino and Chiacchio (Ferrentino and Chiacchio, 2020). The top row displays the two feasibility maps, while the bottom row shows the cost map (left) and the resulting joint trajectories (right).

dashed lines. Additionally, the optimal trajectory of the redundant variable $q_1(t)$ is depicted in both the top-left FM and the bottom-left cost map.

Computing both FMs for the given task and constraints required an average of 6.474 seconds over 100 tests. Our implementation of the algorithm described in Figure 14 of (Ferrentino and Chiacchio, 2020), when applied within these computed FMs, had an average runtime of 10.083 seconds. When we applied Equation (5.5) to the resulting solution, the cost value obtained was 1.44, which is comparable (albeit slightly higher) to the cost (1.195) produced by our method. As shown in the joint trajectory plots in the bottom-right subplot of Figure 5.13, both solutions exhibit the same overall trend, approaching a similar optimal solution. Minor differences can be attributed to variations in resolution, implementation details, or grid discretization.

In terms of computational efficiency, both approaches yield the optimal solution within a similar timeframe of approximately 16 seconds. For the FM-based method, this duration is the sum of the FM generation time (6.474 s) and the FM exploration time for identifying the optimal solution (10.083 s). In our proposed framework, the 16-second timeframe comprises the SMD generation via continuation (2.1 s), the generation of all raw joint trajectories (a negligible amount of

time), and their subsequent optimization (4.1 s + 2.5 s + 3.7 s + 4 s), assuming sequential processing on a single core. However, if the optimization stage is parallelized, the overall runtime is reduced to approximately 6 seconds (2.1 s for SMD generation plus 4.1 s for optimizing the most demanding raw trajectory). For further comparison, if we adjust the resolution $N_u$ in our FM-based implementation to match this 6-second timeframe, a solution with a cost of 2.11 could be achieved.

Beyond similar performance, our approach offers clear advantages over FM-based methods for solving the redundancy resolution problem. A key difference lies in how transitions between FMs are managed. The method proposed in (Ferrentino and Chiacchio, 2020) explores feasible trajectories in the plane $(t, q_1)$, corresponding to different extended aspects, to avoid missing potentially better solutions (i.e., transitioning between FMs). Extended aspects represent alternative solutions of the IKP, obtained when solving for joint coordinates given the task coordinates and redundant variables. In fact, FMs corresponding to different extended aspects are different branches of the projection of the SMD on a hyperplane $(t, \mathbf{q}_r)$, where $t$ is time and $\mathbf{q}_r$ represents $r$ redundant coordinates.

For example, consider an SMD described by the unit sphere $t^2 + q_1^2 + q_2^2 = 1$, where $t \in (-1, 1)$ and $(q_1, q_2)$ are joint coordinates. Selecting $q_1$ as the redundant variable ($\mathbf{q}_r = q_1$) and projecting the sphere onto the $(t, q_1)$ plane results in two FMs corresponding to the two hemispheres (i.e., $q_2 > 0$ and $q_2 < 0$). In (Ferrentino and Chiacchio, 2020), the exploration of both projections is required, with transitions between them occurring at the projection of the equator, which are singularities of the projection. As illustrated in Figure 19 of ref. (Ferrentino and Chiacchio, 2020), these transitions through singularities of the projection may produce small artificial spikes in the obtained trajectories, which should actually be smooth transitions because the robot is not going through kinematic singularities, only singularities of the projection of the SMD on the FMs. In contrast, our framework manages transitions between extended aspects by considering the entire SMD, thereby eliminating discontinuities and ensuring smoother joint trajectories. Returning to the sphere example, our method searches for optimal trajectories directly on the SMD rather than searching in its projections.

Moreover, even if the output trajectory spikes were corrected or mitigated, the FM-based approach may still fail to find a solution when one exists. For instance, consider the same task and robot but remove all obstacles, with joint limits constrained to $[-3, 3]$ for $q_1$ and $[-\pi, -0.5]$ for $q_2$ (with $q_3$ unbounded). This scenario produces the FMs depicted in Figures 5.14(a-b): FM1 in Figure 5.14(a) comprises three connected components (labelled A, B, C), whereas FM2 in Figure 5.14(b) consists of a single connected component. *Transition maps* (Ferrentino and Chiacchio, 2018) plotted over these FMs (with a colourmap where dark blue indicates lower values) represent the angular distance between the two joint

configurations $\mathbf{q}$ associated with each pair $(t, q_1)$ (recall that each $(t, q_1)$ pair yields two solutions for $(q_2, q_3)$). Configurations with angular distances near zero (dark blue) indicate viable transitions between extended aspects (and between FMs), and we term them *gateways*. It is important to note that while different FMs may be represented using distinct colourmaps for clarity in the overall SMD (see Figure 5.14(c)), the lowest angular distance values (dark blue) remain consistent across these maps, as shown in Figure 5.14(d).

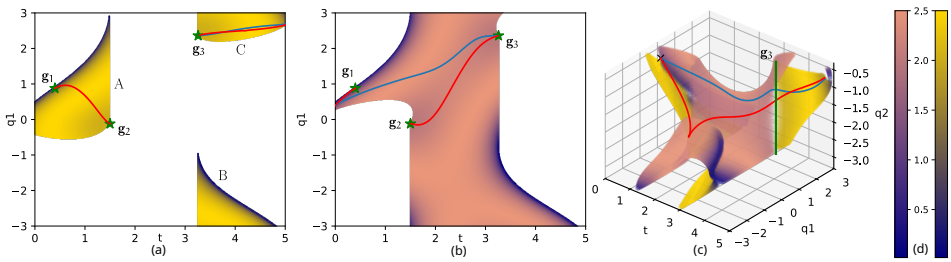


Figure 5.14: (a): FM1. (b): FM2. (c): The SMD projected onto the $(t, q_1, q_2)$ subspace. (d): The colourmaps used for the transition maps.

Relying solely on angular distance to determine possible transitions between FMs suggests that no feasible trajectory can be found using an FM-based approach. However, as demonstrated by the blue and red trajectories obtained with our method, a feasible trajectory does exist. This discrepancy arises because, when starting at $t = 0$ in FM2 as shown in Figure 5.14(b), the only regions with an angular distance near zero (which would permit a transition from FM2 to FM1) lead to components A and B of FM1. Since these components do not extend to $t = 5$, the task cannot be completed. In contrast, a transition into component C of FM1 would yield a complete trajectory; yet, such a transition is not detected when using angular distance as the sole criterion because there are no points with near-zero angular distance linking FM2 with component C.

Nonetheless, as illustrated in Figure 5.14(c), when the SMD is projected onto the subspace $(t, q_1, q_2)$ (omitting $q_3$), a vertical line of gateway points $\mathbf{g}_3$ (represented by a green segment) clearly indicates that this transition is indeed possible. The failure of the FM-based method to detect gateway $\mathbf{g}_3$ is due to the projection of this vertical segment of singularities onto a single point in the $(t, q_1)$ plane (depicted as the rightmost green stars in Figures 5.14(a-b)). When the $(t, q_1)$ plane is discretized into a grid to compute the FM and transition maps, the probability of hitting this singular point is virtually zero. In other words, detecting this singularity via a sweeping of $(t, q_1)$ is nearly impossible, as it would require an exact coincidence between a discretization grid point and the singularity, which is extremely unlikely. Consequently, attempting to find a feasible

trajectory exclusively through FMs fails in such cases because the colour maps used to indicate angular distances do not reveal low values near this critical singularity, unlike other gateways marked in dark blue.

By contrast, when we apply our proposed framework to the same scenario, we obtain two trajectories (plotted in blue and red in Figure 5.14). The blue trajectory represents the optimal solution, while the red trajectory is suboptimal and exhibits a full wrapping in angle $q_3$. These trajectories confirm that feasible joint motions completing the task are attainable and that the desired transition between extended aspects via the otherwise undetected gateway $\mathbf{g}_3$ can be achieved. Specifically, the optimal (blue) trajectory accomplishes the task with a single transition through gateway $\mathbf{g}_3$, whereas the suboptimal (red) trajectory involves three transitions: the first through the FM-detected gateway $\mathbf{g}_1$ (joining component A of FM1 with FM2), followed by transitions through the FM-missed gateways $\mathbf{g}_2$ and $\mathbf{g}_3$.

Supplementary video material (https://ars.els-cdn.com/content/image/1-s2.0-S0094114X25001090-mmc1.mp4) illustrates the globally optimal (blue) solution in motion alongside the secondary suboptimal (red) solution detected by our method. The video also shows the SMD, an animated sequence of SMMs with the corresponding trajectories plotted, and the c-bundle graph from which our method derives feasible c-bundle chains. Notably, both solutions correspond to c-bundle chains formed by the succession of c-bundles coloured blue-purple (for the optimal solution) and blue-red-lightblue (for the suboptimal solution) in the supplementary video.

For (Ferrentino and Chiacchio, 2020)'s algorithm, both the original study and our implementation of their method, have shown that the solution cost is highly dependent on the FM resolution. In contrast, in our presented framework, while a sufficiently fine resolution is still necessary to avoid misinterpreting disjoint SMMs as connected, the cost of the solution remains unaffected by the resolution of the SMD. This is because the optimization stage does not depend on the sampled points of the SMD and will converge to the same solution regardless of the initial discretization, as demonstrated in Section 5.4.1.

Moreover, it is important to note that the demonstration of (Ferrentino and Chiacchio, 2020)'s method was limited to examples with $r = 1$ degree of redundancy, whereas our experiments in Sections 5.3.2 and 5.3.3 have shown the effectiveness of our framework for the more challenging scenario of $r = 2$ degrees of redundancy: a case that few studies have addressed.

Besides FMs, another approach similar to our framework is presented in (Hauser and Emmons, 2018), which proposes an algorithm to generate the SMD and establish paths across it, a method which we have replicated for comparison. Since the algorithm is probabilistically complete (i.e., given sufficient time, it would densely generate the entire SMD), we imposed a runtime limit comparable

to that required by our method when the optimization stage is not parallelized (approximately 16.5 seconds). The results are shown in Figure 5.15(b), where the sampled SMD points are plotted in blue and the segments connecting them in green. For comparison, we also include the SMD obtained via continuation from the experiment in Section 5.3.1, depicted in orange. The optimal trajectory (with a median cost of 1.793, versus our optimal cost of 1.195) is plotted in red.

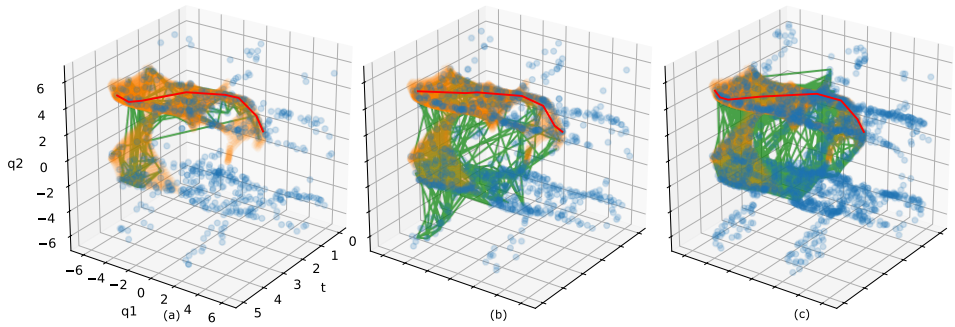


Figure 5.15: Results obtained using the Hauser and Emmons method (Hauser and Emmons, 2018) for SMD generation, applied to the scenario described in Section 5.3.1. (a): Results when (Hauser and Emmons, 2018) is executed with the same computation time as our method when the optimization of raw trajectories is parallelized. (b): Results when (Hauser and Emmons, 2018) is executed in the same time frame as our method without parallelization. (c): Results when (Hauser and Emmons, 2018) is run until the optimal cost matches that of our method.

When parallelization is implemented at the optimization stage, our framework returns the optimized trajectories in around 6 seconds. If (Hauser and Emmons, 2018)'s algorithm is stopped at this shorter runtime, the median cost of its solutions increases to 1.967. An example solution for this configuration is presented in Figure 5.15(a). Although (Hauser and Emmons, 2018)'s algorithm for SMD generation is functional, its efficiency is notably lower; especially in the final stage, which involves connecting the sampled points to form a complete path. For instance, when we allowed the algorithm to run until it achieved the same optimal cost as our method, it required an average runtime of 40 seconds, as shown in Figure 5.15(c). Moreover, we observed substantial variance in the solutions across different runs using the same parameters. For example, Figure 5.15(b) shows that the algorithm sometimes detects that an isolated region of SMD points (in the bottom-left part) is reachable, while in other instances it does not. This inconsistency may stem from the fact that SMD generation was not the primary focus of (Hauser and Emmons, 2018)'s work but rather a supplemen-

tary tool supporting their main contribution: a continuous pseudoinversion of the forward kinematic map.

## 5.5 CONCLUSIONS

In this paper, we present a novel framework for global redundancy resolution that optimally leverages extra DoFs to execute a task while minimizing a cost function subject to kinematic constraints such as joint limits and obstacles. The framework comprises three primary stages: SMD generation, extraction of raw joint trajectories, and subsequent trajectory optimization.

Our method builds on the concept of SMMs to compute the SMD, which represents the infinite set of inverse kinematics solutions that fulfil the imposed constraints along a task trajectory. The SMD is then analysed spatially to generate a graph whose nodes (termed c-bundles) encapsulate SMMs with the same topological characteristics. The graph is traversed to identify preliminary joint trajectories that successfully complete the task, and these trajectories are later refined through optimization.

This framework is highly flexible, allowing for different variants to be selected for each stage based on requirements for computational efficiency, solution optimality, kinematic complexity, and dimensionality. For instance, in the first stage, we recommend generating the SMD via a sampling method for a redundancy degree of $r = 1$ when collision constraints are absent, or for any case with $r > 1$. Conversely, the continuation-based method is more efficient for tracing one-dimensional SMMs when collision constraints significantly limit the traceable region, or when complex kinematic chains render algebraic elimination impractical.

The choice of method for the second stage depends primarily on the desired computational efficiency. For real-time performance, an NN-based approach is suggested to quickly search for raw trajectories. Alternatively, if obtaining the globally optimal solution is the goal, the hybrid NN-PA variant is advised, as it produces more optimal raw trajectories, leading to faster convergence in the final optimization stage. A similar trade-off exists in the third stage: online optimization is recommended when a viable, though not necessarily globally optimal, trajectory is sufficient for real-time applications, whereas standard optimization is preferable when seeking global optimality.

We have evaluated the proposed framework through extensive simulations with several scenarios, including different stage variants, redundancy degrees, DoFs, and task constraints. However, when addressing problems with a redundancy degree greater than 2, processing times become impractical for real-time applications.

In particular, computing SMMs with our continuation method (described in Section 5.2.4.1) can be computationally intensive, especially for redundancy degrees exceeding 1. This method is employed when algebraic elimination (as discussed in Section 5.2.1) becomes overly complex due to complex kinematic chains. In scenarios where continuation is too costly and pure algebraic elimination is intractable, we suggest hybrid approaches, such as the algebraic-numerical method based on constraint curves presented in (Peidró et al., 2020), or the alternative we will propose in the next chapter.

Future work will focus on improving the spatial analysis of c-bundles (currently reliant on time-consuming clustering and matching stages) to enhance the overall efficiency of graph generation. Additionally, real-world experiments with physical robots will be conducted to validate the framework under practical conditions.

## 5.6 PUBLICATIONS RELATED TO THIS CHAPTER

The contributions presented in this chapter are related to the following publications:

- Marc Fabregat-Jaén, Adrián Peidró, Matteo Colombo, Paolo Rocco, and Óscar Reinoso (2025). "Topological and spatial analysis of self-motion manifolds for global redundancy resolution in kinematically redundant robots." In: *Mechanism and Machine Theory* 210, p. 106020. ISSN: 0094-114X. DOI: 10.1016/j.mechmachtheory.2025.106020 **(SCI-JCR Impact Factor: 5.3, Q1)**
  - This journal paper presents the framework presented in this chapter for solving the redundancy problem in redundant robots. It includes the definition of the SMD and the graph generation stage, as well as the generation of raw joint trajectories and the optimization stage. The paper also includes a detailed analysis of the computational complexity of the proposed framework, and a comparison with state-of-the-art methods. The proposed framework is evaluated in a 3R planar manipulator and a 8-DoF serial equivalent of the HyReCRo robot. Part of this paper was done during a 4-month research stay at *Politecnico di Milano*, under supervision of *Dr. Prof. Paolo Rocco*.

# 6

# HOMOTOPY-BASED COMPUTATION OF SELF-MOTION MANIFOLDS

In Chapter 5, we introduced the concept of Self-Motion Manifolds and presented an updated method for their computation by means of analytically solving, via algebraic elimination, the kinematic equations of the robot. This method demonstrated to be efficient for the computation of SMMs, as long as a closed-form solution of the kinematic equations is provided for every combination of redundant parameters. However, complex kinematic chains may render the algebraic elimination method an arduous task, and in some cases, it may not even be possible to obtain a closed-form solution. For these cases, we proposed using continuation methods, which are general, and can be applied to any kinematic chain. Nonetheless, these approaches are less efficient, especially as the number of degrees of freedom increases.

In this chapter, we develop a novel homotopy-based method for computing SMMs that overcomes the limitations of existing techniques. Our approach covers the gap between sampling-based methods (which require analytical inverse kinematics solutions) and continuation methods (which struggle with computational efficiency and finding disconnected manifolds). Using a combination of homotopy sampling and efficient Newton-based propagation, our method does not require closed-form solutions of the robot's kinematic equations, can identify all disjoint manifold components, and demonstrates significantly better performance than traditional continuation methods. Experimental validation across two examples confirms these advantages.

The chapter is organized as follows. In Section 6.1, we recap the definition of SMMs and the sampling-based method for their computation, presented in Chapter 5. Section 6.2 presents a method for the redensification of the SMMs when low-density regions are present. This is the case when not all combinations of $r$ joints can be solved via algebraic elimination. In Section 6.3.1, we introduce the concept of homotopy and its application in resolving the IKP. Then, we present the method for the computation of SMMs based on homotopy in Section 6.3.2. Finally, we present the experimental validation of the proposed method in Section 6.4 and conclude the chapter in Section 6.5.

Let us recap the definition of SMMs and the sampling-based method for their computation presented in Chapter 5, which is based on the work of Peidro et al. (2018). Recall that the following Equation relates a joint configuration $\mathbf{q} \in \mathbb{R}^n$ of a robot, and a point $\mathbf{x} \in \mathbb{R}^m$ in the task space:

$$\mathbf{F}(\mathbf{q}, \mathbf{x}) = \mathbf{0} \tag{6.1}$$

The infinite set of $\mathbf{q}$ that satisfy Equation (6.1) for a fixed task value $\mathbf{x}$ are gathered in a finite number of disjoint $r$-dimensional SMMs, where $r$ is the degree of redundancy, which is defined as the difference between the number of joints $n$ and the number of task space dimensions $m$, i.e., $r = n - m$.

In Section 5.2.1, we presented Algorithm 5.2 for the computation of SMMs for a given $\mathbf{x}$. For every possible combination of $r$ joints (out of the $n$ joints of the robot), the remaining $m$ joints must be analytically solved (via algebraic elimination) for the task value $\mathbf{x}$. Then, for each combination, every point of the resulting $r$-dimensional grid is solved for the remaining $m$ joints. If the resulting joint configuration $\mathbf{q}$ is valid (i.e., it satisfies the additional constraints), it is added to the SMMs. By means of scanning through every combination of $r$ joints, we manage to obtain densely sampled SMMs.

Let us now exemplify the method with a simple 2-DoF robot, which is shown in Figure 6.1. Both links of the robot are of length $l_1 = l_2 = 1$ m. It is tasked to place the vertical coordinate of the end-effector at a fixed value $y = 1$ m. The degree of redundancy is $r = n - m = 2 - 1 = 1$, thus SMMs are 1-dimensional curves in the joint space. For reference, we first show the actual single SMM (when no additional constraints are considered) that exists for the given task value in Figure 6.1(a). We imposed an obstacle as an additional constraint, which translates to a forbidden region in the joint space, drawn in red in Figure 6.2(a). Note how this added constraint affects the SMM in the joint space, rendering the previously valid configurations invalid, as the dotted grey lines show, and splitting the previously single SMM into two disjoint SMMs $\mathbf{M}_1$ and $\mathbf{M}_2$.

With no prior knowledge of the SMMs, we now proceed to compute them using the sampling-based method. Following the approach, every combination of $r = 1$ joints must be solved for the remaining $m = 1$ joints. The number of combinations is given by the binomial coefficient $\binom{n}{r}$, which in this case is $\binom{2}{1} = 2$. One combination is selecting $\mathbf{q}_r = [q_1]$ as the redundant joint, and solving for the remaining joint $\mathbf{q}_m = [q_2]$. After selecting $\mathbf{q}_r$, its axes are discretized with step $\Delta q_j$ between its valid range $[\mathbf{q}_{r,\min}, \mathbf{q}_{r,\max}]$. For each discretized value of $\mathbf{q}_r$, the remaining joint $\mathbf{q}_m$ is solved for the task value $\mathbf{x}$. In this case, $\mathbf{q}_m$ has two solutions for a given pair of $\mathbf{q}_r$ and $\mathbf{x}$, which are added to the SMMs if they are valid (i.e., they satisfy the additional constraints). The result of sweeping

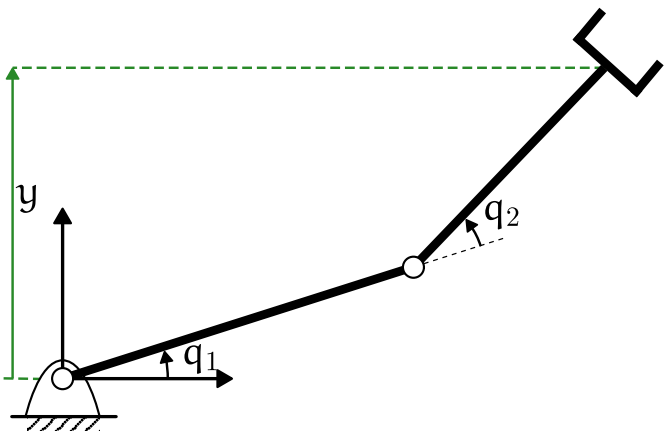


Figure 6.1: 2R planar manipulator with task $\mathbf{x} = y$.

through this combination is shown in Figure 6.2(b). Solutions that are valid are scattered in blue, while invalid solutions are shown in red.

Note that, up to this point (i.e., having computed the SMM for one combination of $r$ joints), the resulting sampled SMMs present low-density regions. Specifically, the regions of the manifolds that are tangent to the vertical lines, orthogonal to the $q_1$ axis, are not sampled densely enough. This could lead to the misinterpretation of an actual SMM as being disconnected, as sets $m_1$ and $m_2$ suggest in Figure 6.2(b).

To solve this, we can repeat the process for the other combination of $r$ joints: in this case, $\mathbf{q}_r = [q_2]$. Figure 6.2(c) shows the result of this second sweep. The same phenomenon occurs, and the resulting SMMs are not sampled densely enough in the regions tangent to the lines orthogonal to the $q_2$ axis.

By combining the results of both sweeps, we can obtain a denser sampling of the SMMs. The resulting SMMs are shown in Figure 6.2(d) after sweeping through all combinations of $r$ joints. Note that the sets of points $m_1$ and $m_2$, that previously seemed disconnected, are now connected and form the single SMM $\mathbf{M}_1$.

This approach ensures that the sampled points of the SMMs are at most $h = \sqrt{\sum_{j=1}^{n} \Delta q_j^2}$ apart, as Figure 6.2(e) exemplifies for the worst case. Continuation methods do not directly provide such a guarantee, but easy resampling approaches can be applied to the resulting SMMs to ensure that the points are at most $h$ apart, as Chapter 5 proposed.

The sampling-based method has proven to be the most efficient approach for computing SMMs, outperforming other methods in the literature, as Section 6.4 will show. One of the reasons for this is that it can be easily vectorized, allowing for parallel computation of the SMMs, as Chapter 5 also discussed. The only

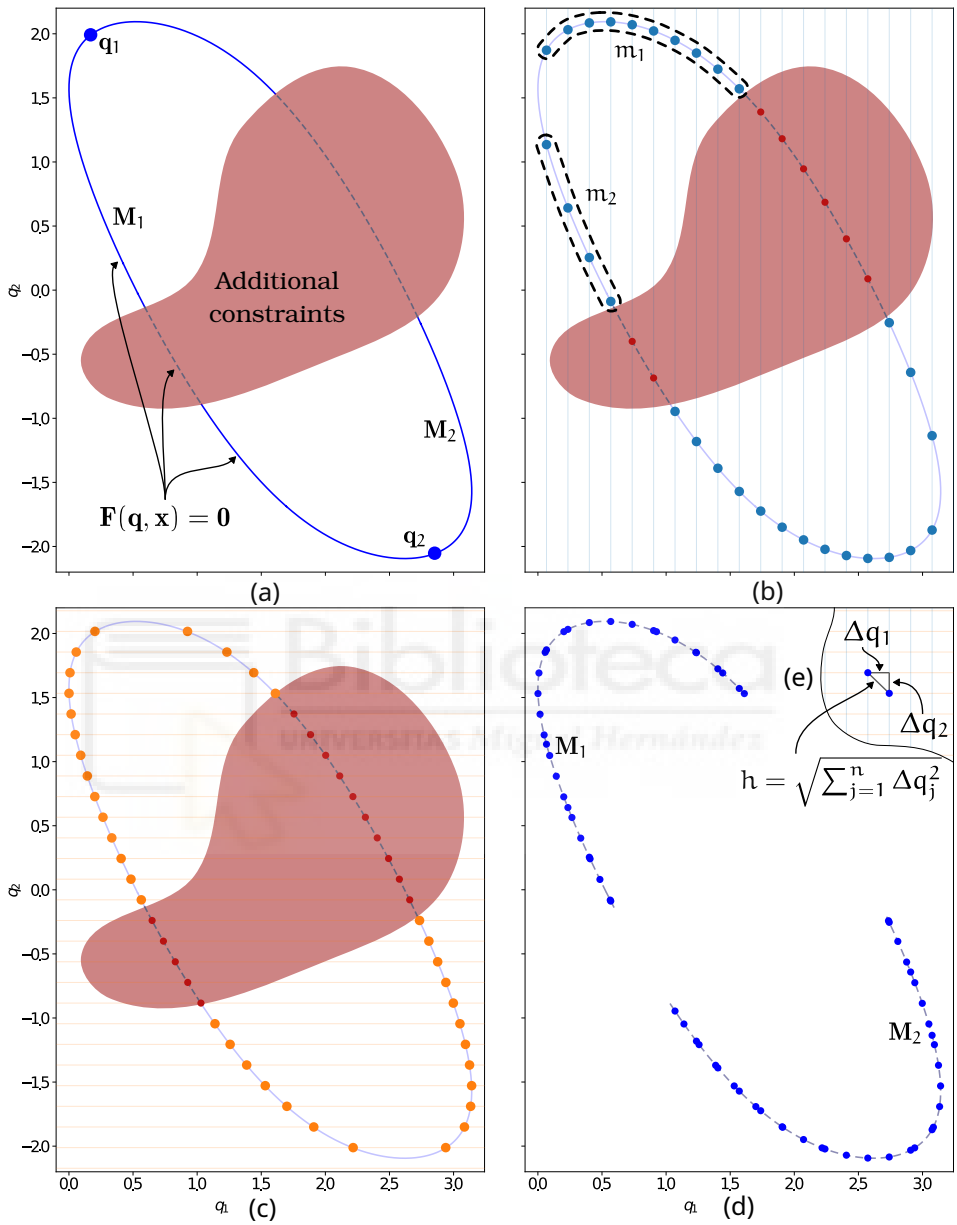


Figure 6.2: (a) SMMs of the 2R manipulator for a fixed task value $y = 1$ m and an arbitrary obstacle (red region). (b) Sampling of the SMMs for the combination $\mathbf{q}_r = [q_1]$. (c) Sampling of the SMMs for the combination $\mathbf{q}_r = [q_2]$. (d) Combination of the results of (b) and (c). (e) Maximum distance between the points of the sampled SMMs.

case in which the continuation method outperforms the sampling-based method is when the SMMs are one-dimensional curves, and obstacles in the workspace render most of the manifolds invalid (see red region in Figure 6.2). This special case and its justification have been discussed in Chapter 5.

Additionally, the method successfully identifies every disjoint SMM in the joint space, as shown in Figure 6.2. Conversely, the continuation method will only identify the SMM to which the given initial configuration belongs. For instance, if we start the continuation method from the configuration $\mathbf{q}_1$ in Figure 6.2(a), the method will only identify SMM $\mathbf{M}_1$. The posterior identification of the other SMM $\mathbf{M}_2$ would require a new initial configuration $\mathbf{q}_2$ that belongs to $\mathbf{M}_2$, which is not always available.

However, the sampling-based method has a major drawback: it necessitates the analytical solution (via algebraic elimination) of the kinematic equations for every combination of $r$ joints. This can be a cumbersome task, especially for complex kinematic chains, and in some cases, it may not even be possible to obtain a closed-form solution. For instance, in the case of the HyReCRo robot example presented in Chapter 5, we chose to discard some combinations of $r$ joints without affecting the final result, as the application allowed for a less dense sampling of the SMM. Nonetheless, there are situations where a dense computation of the SMM is required. In the next section, we present a method for the redensification of the SMMs that does not require the analytical solution of the kinematic equations for every combination of $r$ joints.

## 6.2 MANIFOLD REDENSIFICATION

In the previous section, we discussed the main drawback of the sampling-based method for the computation of SMMs: when not all combinations of $r$ joints can easily be solved via algebraic elimination, or there are too many combinations to exhaustively solve them all, the method does not guarantee a dense sampling of the SMMs. In this section, we present a method for the redensification of the SMMs.

We have identified two reasons for the low density of the sampled SMMs when discarding certain combinations of $r$ joints. The first reason is the projection singularities that arise when sweeping through the combinations of $r$ joints. These singularities occur when the Jacobian matrix of the task $\mathbf{x}$ with respect to the non-redundant joints $\mathbf{q}_m$, becomes rank-deficient. They manifest as regions where the SMM becomes tangent to hyperplanes defined by constant values of $\mathbf{q}_r$, as the apparent discontinuity between sets $m_1$ and $m_2$ in Figure 6.2(b) exemplifies, although the SMM is actually connected in this region. In the sampling process, when projecting the SMM onto the $\mathbf{q}_r$ subspace, these tangent regions either collapse to a point or exhibit very steep gradients in the projected manifold.

Consequently, uniform sampling in the $\mathbf{q}_r$ space results in highly non-uniform sampling on the actual SMM, with sparse coverage in regions of high curvature relative to the $\mathbf{q}_r$ axes.

The second reason is the loss of guarantee regarding the maximum distance between the points of the sampled SMMs being at most $h = \sqrt{\sum_{j=1}^{n} \Delta q_j^2}$. When all combinations of $r$ joints are considered in the sampling process, the worst-case scenario for point spacing is illustrated in Figure 6.2(e). However, when certain combinations of $r$ joints are omitted from the sampling process due to analytical intractability, this guarantee no longer holds, even in regions not necessarily close to the projection singularities mentioned above.

In this section, we present two methods for the redensification of the SMMs that address these two issues.

### 6.2.1 *Redensification of Projection Singularities*

Let $\mathbf{J}_m(\mathbf{q})$ be the Jacobian matrix of $\mathbf{F}(\mathbf{q}, \mathbf{x})$ (Equation (6.1)) with respect to the non-redundant joints $\mathbf{q}_m$. A projection singularity occurs at configurations $\mathbf{q}$ where $\text{rank}(\mathbf{J}_m(\mathbf{q})) < m$. Visually, these singularities correspond to points of the SMM that are tangent to hyperplanes defined by constant values of $\mathbf{q}_r$. Figure 6.3(a) illustrates the phenomenon for the 2-DoF robot example when $\mathbf{q}_r = [q_1]$ is the redundant joint (as in Figure 6.2(b)), with singular points $\mathbf{q}_s$ and tangent lines marked in red. Note that these singularities $\mathbf{q}_s$ have not been identified up to this point, and are only illustrated for explanatory purposes.

These singular points $\mathbf{q}_s$ correspond to the roots of the following system, where the determinant of $\mathbf{J}_m(\mathbf{q})$ being equal to zero is added to the original system $\mathbf{F}(\mathbf{q}, \mathbf{x})$:

$$\mathbf{F}_s(\mathbf{q}, \mathbf{x}) = \begin{bmatrix} \mathbf{F}(\mathbf{q}, \mathbf{x}) \\ |\mathbf{J}_m(\mathbf{q})| \end{bmatrix} = \mathbf{0} \tag{6.2}$$

To locate these singular points, we employ Newton's method to solve the system in Equation (6.2). Starting from previously computed points on the SMM, the method iteratively refines the solution according to:

$$\mathbf{q}_{k+1} = \mathbf{q}_k - \mathbf{J}_s^{-1}(\mathbf{q}_k)\, \mathbf{F}_s(\mathbf{q}_k, \mathbf{x}) \tag{6.3}$$

where $\mathbf{J}_s(\mathbf{q})$ is the Jacobian of the augmented system $\mathbf{F}_s(\mathbf{q}, \mathbf{x})$ with respect to $\mathbf{q}$.

To reduce computational cost, we focus on identifying the sparsely sampled regions that typically occur near singularities. We utilize kernel density estimation (KDE) (Parzen, 1962) to identify points within the lowest $p$ percentile of density in the sampled SMM. These low-density regions, illustrated as green points

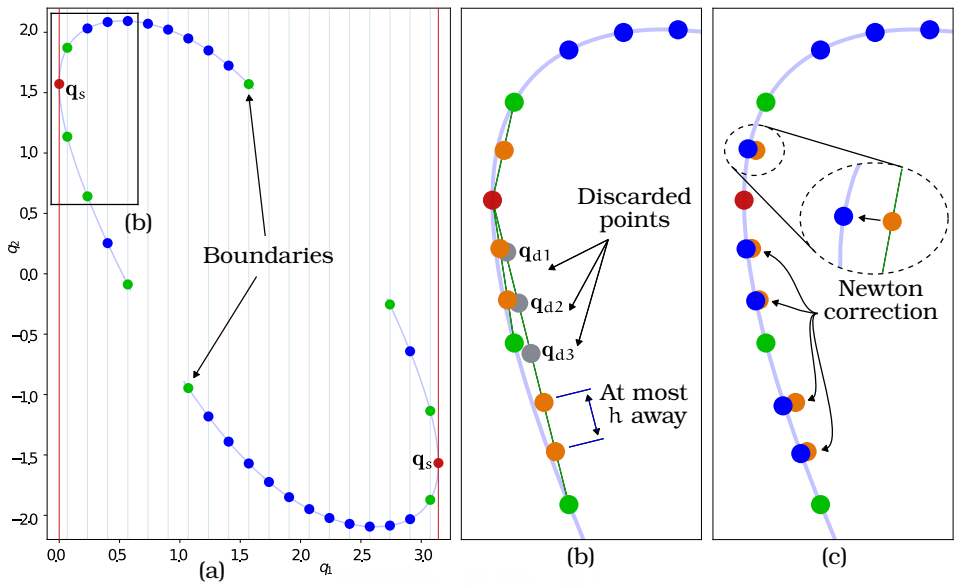


Figure 6.3: (a) Projection singularities $\mathbf{q}_s$ and low-density points. (b) Interpolated points (orange) between low-density points and projection singularities $\mathbf{q}_s$. (c) Redensified points (blue) after Newton correction.

in Figure 6.3, serve as initial seeds for the Newton iterations. While some identified low-density points may actually be at the boundaries of the SMM rather than near projection singularities, the method will still converge to singular points $\mathbf{q}_s$ with sufficient iterations, albeit potentially requiring more computational steps for these boundary cases.

In order to redensify the SMM, we first linearly interpolate points between each identified low-density point and the singular point $\mathbf{q}_s$ to which the Newton method for singularity projections has converged, as black segments in Figure 6.3(b) illustrate. The number of interpolated points is determined so that the maximum separation between them is at most $h$. Out of these interpolated points, we discard those that are within a distance $h$ from any other point, i.e., sampled points during the initial sampling process ($\mathbf{q}_{d3}$) or other interpolated points ($\mathbf{q}_{d1}$ and $\mathbf{q}_{d2}$). Discarded points $\{\mathbf{q}_{d1}, \mathbf{q}_{d2}, \mathbf{q}_{d3}\}$ are shown in grey in Figure 6.3(b).

The remaining interpolated points are shown in orange in Figures 6.3(b-c). These points undergo a Newton correction, similar to the one performed in continuation-based SMM computation methods, to ensure they are valid solutions of the system $\mathbf{F}(\mathbf{q}, \mathbf{x}) = \mathbf{0}$, yielding the final redensified points drawn in

blue in Figure 6.3(c). The redensified points, and the identified singularities $\mathbf{q}_s$, are then added to the previously computed SMM.

### 6.2.2 *Redensification of the Point Cloud*

When not sweeping through all combinations of $r$ joints, the maximum distance between neighbouring points in the sampled SMM is no longer given by the example of Figure 6.2(e). For instance, when performing the sampling process for $\mathbf{q}_r = [q_1]$, it is clear that the points within the set $m_2$ of Figure 6.2(b) are more than $h$ apart. Additionally, the redensification of the projection singularities that we proposed in the previous section does not guarantee that the maximum distance between neighbouring points is at most $h$. Although the interpolated points are guaranteed to be at most $h$ apart, the posterior Newton correction may drift them further apart.

To address this issue, we propose a second method for the redensification of the SMM, based on closest-neighbour queries by means of the efficient k-d tree data structure (Bentley, 1975). Algorithm 6.1 summarizes the method. Figure 6.4 illustrates the method for the 2-DoF robot example, considering that the projection singularities have already been identified and the redensification of the low-density regions has been performed.

**Algorithm 6.1** Redensification of the Point Cloud

$\mathcal{P}$: Point cloud of joint configurations $\mathbf{q}_i$ of the SMM to redensify
**for** $\mathbf{q}_i \in \mathcal{P}$ **do**
  $\mathcal{N} \leftarrow k$ nearest neighbours of $\mathbf{q}_i$ in $\mathcal{P}$
  Remove neighbours that are within a distance $h$ from $\mathbf{q}_i$
  **for** $\mathbf{q}_n \in \mathcal{N}$ **do**
    $\mathcal{I} \leftarrow$ Interpolate $n$ points between $\mathbf{q}_i$ and $\mathbf{q}_n$
    **for** $\mathbf{q}_j \in \mathcal{I}$ **do**
      $\mathbf{q}_j \leftarrow$ Newton correction of $\mathbf{q}_j$ to satisfy $\mathbf{F}(\mathbf{q}, \mathbf{x}) = \mathbf{0}$
      Add the $\mathbf{q}_j$ to the point cloud $\mathcal{P}$
**return** $\mathcal{P}$

For each point $\mathbf{q}_i$ in the point cloud, the algorithm identifies its $k$ nearest neighbours $\mathcal{N}$. The value of $k$ depends on the dimensionality of the redundant joint subspace $\mathbf{q}_r$. Specifically, $k$ equals the number of adjacent points in the $r$-dimensional grid plus one (to account for the point itself, which later will be discarded).

For our 2-DoF robot example with $r = 1$, each point has exactly two adjacent points along the one-dimensional manifold (one in each direction), resulting in $k = 3$ when including the point itself, as Figure 6.4(a) illustrates. For

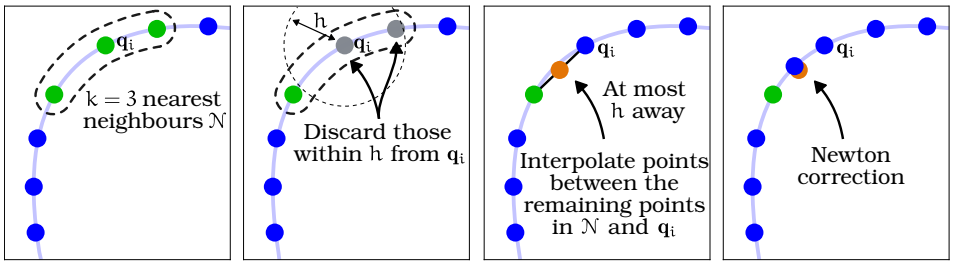


Figure 6.4: Redensification of the point cloud of joint configurations $\mathbf{q}_i$ of the SMM. (a) Nearest neighbours $\mathcal{N}$ of a point $\mathbf{q}_i$ in the point cloud. (b) Discarded points that are within a distance $h$ from $\mathbf{q}_i$. (c) Interpolated points between $\mathbf{q}_i$ and its remaining neighbours in $\mathcal{N}$. (d) Redensified point cloud after the Newton correction.

higher-dimensional redundancy spaces, the value of $k$ corresponds to the chosen connectivity pattern. For instance, when $r = 2$: $k = 5$ when 4-connectivity is considered (considering only vertical and horizontal neighbours); and $k = 9$ with 8-connectivity (including diagonal neighbours).

The algorithm then removes any neighbours that are within a distance $h$ from the point $\mathbf{q}_i$. This process is exemplified in Figure 6.4(b). This ensures that the algorithm only considers neighbours that are sufficiently far apart, thus avoiding redundancy in the redensification process.

Next, for each neighbour $\mathbf{q}_n$ in the set $\mathcal{N}$, the algorithm interpolates $n$ points between $\mathbf{q}_i$ and $\mathbf{q}_n$, where $n$ is determined so that the maximum distance between the interpolated points is at most $h$, as we did in the previous section (see Figure 6.4(c)). The subsequent step also aligns with the previous section: each interpolated point $\mathbf{q}_j$ is corrected using a Newton method to ensure it satisfies the system $\mathbf{F}(\mathbf{q}, \mathbf{x}) = \mathbf{0}$, as Figure 6.4(d) shows. Finally, the corrected points are added to the point cloud $\mathcal{P}$.

The computational efficiency of this algorithm can be substantially improved through two key optimizations. First, the loops can be vectorized since each point $\mathbf{q}_i$ is processed independently, allowing simultaneous operations across multiple points. Second, modern parallel computing frameworks provide efficient implementations of k-d tree data structures that parallelize nearest neighbour queries, which significantly accelerates the search process.

Upon completion, the algorithm outputs the redensified point cloud $\mathcal{P}$, effectively addressing the sparse sampling issues that arise when only a subset of the r-joint combinations can be solved via algebraic elimination.

Nevertheless, a significant limitation remains: if no combination of r joints can easily be solved through algebraic elimination, researchers have traditionally

been forced to resort to continuation methods, with their two major disadvantages: significantly reduced computational efficiency and the inability to identify all disjoint SMMs. To overcome these limitations, we propose a new approach in the following section that leverages homotopy methods to compute the SMMs without requiring the analytical solution of the kinematic equations.

## 6.3 HOMOTOPY-BASED COMPUTATION OF SELF-MOTION MANIFOLDS

In this section, we present a novel method for the computation of SMMs based on homotopy. The presented method addresses the limitations of the two most-established methods for the computation of SMMs. Regarding the sampling-based method, the proposed method does not require the analytical solution of the kinematic equations for every combination of $r$ joints. This is a significant advantage, as it allows for the computation of SMMs in a generalized manner, without the need for tailoring the method to specific kinematic chains. On the other hand, the proposed method is more efficient than continuation methods, and it is able to identify all disjoint SMMs in the joint space.

The proposed method is based on the concept of homotopy to establish initial solutions of the SMM. From there, we propagate the solutions using a Newton-based method to compute the entire disjoint SMM. This process is repeated until the entire redundant space is covered, and every disjoint SMM is identified.

We will first introduce the concept of homotopy and its application in resolving the IKP. Then, we will present the method for the computation of SMMs based on homotopy.

### 6.3.1 *Inverse Kinematics Resolution via Homotopy*

Homotopy is a powerful numerical technique for solving systems where direct methods might struggle (Allgower and Georg, 2012). The core idea behind homotopy methods is to transform a difficult problem into a similar simpler one with known solutions, and then gradually deform the simpler problem back into the original problem while tracking how the solutions evolve. Mathematically, a homotopy can be defined as a continuous function $\mathbf{H}(z, t)$ such that $H(\mathbf{z}, 1) = G(\mathbf{z})$ and $H(\mathbf{z}, 0) = F(\mathbf{z})$, where $G(\mathbf{z}) = \mathbf{0}$ is a simple system with known solutions and $F(\mathbf{z}) = \mathbf{0}$ is the target system we wish to solve. By gradually varying the parameter $t$ from 0 to 1, we can trace solution paths from the known solutions of $G(\mathbf{z})$ to the desired solutions of $F(\mathbf{z})$ (Li, 1997). This technique is known as *path-tracking*. Do not confuse the parameter $t$ with the notation we used in other chapters of this thesis, where $t$ was used to denote the time variable. This path-tracking process is typically implemented using predictor-corrector methods, as we will introduce later. The method is illustrated in Figure 6.5.

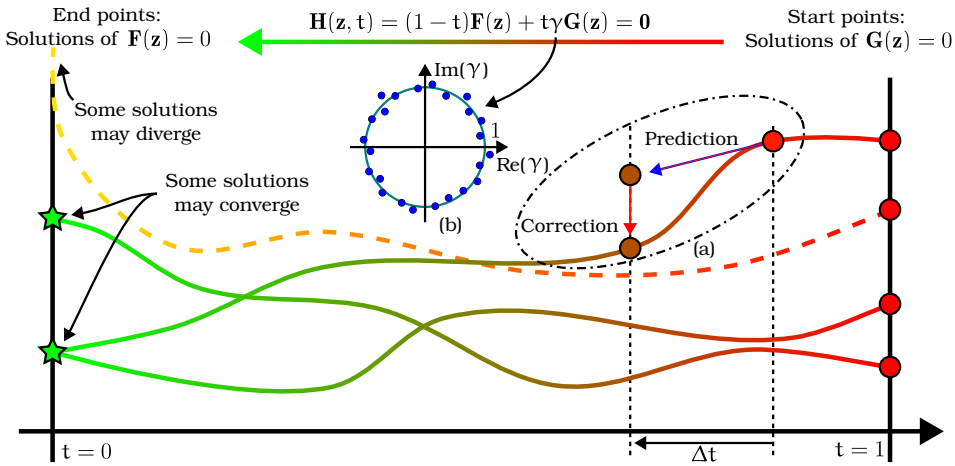


Figure 6.5: Homotopy method illustration. The right-most points correspond to the start system $G(\mathbf{z})$, which has known solutions. The left-most points correspond to the target system $F(\mathbf{z})$, which we want to solve. The path-tracking process is illustrated by the prediction and correction steps.

We will explain the concept of homotopy in the context of resolving the IKP of a robot. Specifically, we will exemplify the method with the 2-DoF robot example presented in Section 6.1. First, we will introduce the construction of the homotopy and the related system $G(\mathbf{z})$ that serves as start points for the path-tracking method. Then, we will present the path-tracking process: the gradual transformation of the system $G(\mathbf{z})$ into the original system $F(\mathbf{z})$, by means of prediction and correction steps.

#### 6.3.1.1 *Homotopy Construction*

The first step in the homotopy method is to define a start system $G(\mathbf{z})$ whose solutions are known. The roots of this system serve as starting points for the path-tracking process, as the right-most ($t = 1$) points in Figure 6.5 illustrate. The idea is to define a polynomial system related to the original system $F(\mathbf{z})$ (for instance, with the same degrees) that can be easily solved. Then, a homotopy function $\mathbf{H}(\mathbf{z}, t)$ can be defined as:

$$\mathbf{H}(\mathbf{z}, t) = (1-t)\mathbf{F}(\mathbf{z}) + t\gamma\mathbf{G}(\mathbf{z}) \tag{6.4}$$

so that the initial (when $t = 1$) solutions of $\mathbf{H}(\mathbf{z}, 1)$ are the solutions of the start system $G(\mathbf{z})$, and the final (when $t = 0$) solutions of $\mathbf{H}(\mathbf{z}, 0)$ are the solutions of the target system $F(\mathbf{z})$. The most common type of homotopy to construct

the system $G(\mathbf{z})$ is the so-called *total-degree homotopy* (Sommese, Wampler, et al., 2005).

Let $f_1, \ldots, f_N$ be the defining $N$ polynomials of $F(\mathbf{z})$, with degrees $d_i = \deg(f_i)$. We then construct a new system $G(\mathbf{z})$ consisting of $N$ polynomials $g_1, \ldots, g_N$, each with the same degree as the corresponding $f_i$. Among the many possible choices, a particularly convenient start system is (Morgan and Sommese, 1987b):

$$g_i(\mathbf{z}) = z_i^{d_i} - 1 \tag{6.5}$$

where $z_i$ is the $i$-th variable in $\mathbf{z}$, and $d_i$ is the degree of $f_i$. Each equation $g_i(\mathbf{z}) = 0$ has $d_i$ distinct roots, corresponding to the $d_i$-th roots of unity, whose computation is trivial. Every possible combination of the roots from all $N$ polynomials yields a unique starting solution for $G(\mathbf{z})$. Thus, the total number of starting solutions is given by the product of the degrees:

$$\prod_{i=1}^{N} d_i \tag{6.6}$$

Note the presence of the parameter $\gamma$ in Equation (6.4). This parameter is a random complex number that is used to contribute with numerical stability to the homotopy method. Following the well-known *gamma trick* (Morgan and Sommese, 1987a), we can set $\gamma$ randomly to a complex number within a small neighbourhood of the unit circle, as Figure 6.5(b) illustrates.

Following the recurring example of the 2-DoF robot, we will consider the task $\mathbf{x} = y$, and will specify $q_1$ as the redundant joint: $\mathbf{q}_r = [q_1]$, as in Figures 6.1 and 6.2(b). Note that the homotopy method requires the system $F(\mathbf{z})$ to be made of polynomial equations. Therefore, we must treat trigonometric functions as the variables in $\mathbf{z}$, rather than their arguments, which are the joint angles $\mathbf{q}$:

$$\mathbf{z} = \begin{bmatrix} s_2 \\ c_2 \end{bmatrix} \tag{6.7}$$

where $s_j = \sin q_j$ and $c_j = \cos q_j$. Note that $q_1$ is not included in $\mathbf{z}$, as it is the redundant joint whose value is fixed and is constant during the homotopy process. This conversion from transcendental equations to polynomial form is crucial, as homotopy continuation methods are rigorously developed for polynomial systems where fundamental theorems like Bézout's theorem apply. The unit circle constraint $s_2^2 + c_2^2 - 1 = 0$ ensures that the solutions correspond to valid joint angles. Therefore, we can express the system $F(\mathbf{z})$ as:

$$\mathbf{F}(\mathbf{z}) = \begin{bmatrix} -y + s_1 + c_1 s_2 + s_1 c_2 \\ s_2^2 + c_2^2 - 1 \end{bmatrix} = \mathbf{0} \tag{6.8}$$

Both polynomials have degree 2, thus $d_1 = d_2 = 2$. We can then construct the corresponding total-degree start system $G(\mathbf{z})$ following Equation (6.5):

$$\mathbf{G}(\mathbf{z}) = \begin{bmatrix} s_2^2 - 1 \\ c_2^2 - 1 \end{bmatrix} = \mathbf{0} \tag{6.9}$$

Both equations have two distinct roots, which produces a total of four starting solutions $(\pm 1, \pm 1)$ to be tracked, as Figure 6.5 illustrates.

Other types of homotopy can be constructed, which, depending on the problem at hand, could reduce the number of paths to be tracked, making the method more efficient. For instance, *polyhedral homotopy* (Huber and Sturmfels, 1995) exploits the theory that gives the maximum number of isolated roots that any system of polynomials can have, reducing the number of paths to be tracked. However, this process presents a high computational complexity due to the large number of combinatorics involved.

Middle-ground functions between total-degree and polyhedral homotopies are *multihomogeneous homotopies* and *linear product homotopies* (Wampler, 1992). However, they require setting up a user-defined homotopy, constructing and solving the start system by other means.

In this work, we will focus on the total-degree homotopy, as it allows for generalization and do not require such elaborate setup. The potential large number of paths to be tracked is compensated by the fact that the method can be easily parallelized, as paths are independent of each other.

#### 6.3.1.2 *Path-tracking*

The second step in the homotopy method is to track the paths from the start system $G(\mathbf{z})$ to the target system $F(\mathbf{z})$. After constructing the start system $\mathbf{G}(\mathbf{z})$ using Equation (6.5) and defining the homotopy function $\mathbf{H}(\mathbf{z}, t)$ in Equation (6.4), we trace each path by gradually decreasing the parameter $t$ from 1 to 0.

The path-tracking process implements a predictor-corrector framework, as illustrated in Figure 6.5(a). Each iteration consists of two stages:

1. A predictor step (blue arrow) that estimates the next point on the path after decreasing $t$ by $\Delta t$
2. A corrector step (red arrow) that refines this estimate to ensure it precisely satisfies $\mathbf{H}(\mathbf{z}, t - \Delta t) = \mathbf{0}$

This process iterates until $t$ reaches 0, at which point we obtain solutions to the original system $\mathbf{F}(\mathbf{z}) = \mathbf{0}$.

The prediction step estimates how solution change as $t$ decreases by $\Delta t$. The simplest approach is Euler's method:

$$\mathbf{z}_{t-\Delta t} = \mathbf{z}_t - \mathbf{J}_{\mathbf{H}}^{-1}(\mathbf{z}_t, t)\frac{\partial \mathbf{H}(\mathbf{z}_t, t)}{\partial t}\Delta t \tag{6.10}$$

where $\frac{\partial \mathbf{H}(\mathbf{z}_t,t)}{\partial t} = \gamma \mathbf{G}(\mathbf{z}_t) - \mathbf{F}(\mathbf{z}_t)$, and $\mathbf{J}_H$ is the Jacobian of $\mathbf{H}$ with respect to $\mathbf{z}$:

$$\mathbf{J}_H(\mathbf{z}, t) = \frac{\partial \mathbf{H}(\mathbf{z}, t)}{\partial \mathbf{z}} = (1-t)\cdot\frac{\partial \mathbf{F}(\mathbf{z})}{\partial \mathbf{z}} + t\gamma \cdot \frac{\partial \mathbf{G}(\mathbf{z})}{\partial \mathbf{z}} \tag{6.11}$$

While computationally inexpensive, Euler's method can introduce significant errors along curved paths, necessitating very small step sizes. For improved accuracy, we implement the Runge-Kutta-Fehlberg method (RKF45) (Fehlberg, 1970), which provides fourth-order accuracy while allowing larger step sizes.

The selection of step size $\Delta t$ represents a trade-off between computational efficiency and numerical stability (Deuflhard, 2011). While adaptive stepping as proposed by Bates et al. (2008) provides theoretical advantages, our experimentation revealed that a fixed-step RKF45 implementation offers an effective balance for typical robotic kinematic chains.

After the prediction step, the corrector refines the predicted point using Newton's method to solve $\mathbf{H}(\mathbf{z}, t-\Delta t) = \mathbf{0}$:

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \mathbf{J}_H^{-1}(\mathbf{z}_k, t-\Delta t)\cdot \mathbf{H}(\mathbf{z}_k, t-\Delta t) \tag{6.12}$$

The Newton iterations continue until the solution converges, typically when $\|\mathbf{H}(\mathbf{z}_k, t-\Delta t)\|$ falls below a specified tolerance, which usually occurs after a few iterations.

In the case of the 2-DoF robot example, the Jacobian $\mathbf{J}_H(\mathbf{z}, t)$ is:

$$\mathbf{J}_H(\mathbf{z}, t) = (1-t)\begin{bmatrix} c_1 & s_1 \\ 2s_2 & 2c_2 \end{bmatrix} + t\gamma \begin{bmatrix} 2s_2 & 0 \\ 0 & 2c_2 \end{bmatrix} \tag{6.13}$$

where the first matrix represents $\frac{\partial \mathbf{F}(\mathbf{z})}{\partial \mathbf{z}}$ and the second matrix represents $\frac{\partial \mathbf{G}(\mathbf{z})}{\partial \mathbf{z}}$.

Each tracked path from $t = 1$ to $t = 0$ leads to one of three possible outcomes:

- A finite real solution to $\mathbf{F}(\mathbf{z}) = \mathbf{0}$.
- Convergence to a complex solution, which are not physically realizable for a robot, but can be useful for the purpose of self-motion manifold computation, as we will explain in the next section.
- Divergence to infinity (as the yellow dashed path in Figure 6.5 illustrates).

It is worth mentioning that we tried to homogenize the variables in the system, in order to reduce the number of paths to be tracked, as Bates et al. (2013) suggest. However, we found that the reduction in the length of the paths to be tracked was minimal, and did not compensate for the increased number of variables to be handled.

### 6.3.2 *Homotopy-based Computation of Self-motion Manifolds*

Having introduced the homotopy method for solving the IKP, we can now present the proposed method for computing SMMs, whose core idea is to leverage the homotopy method to establish initial solutions for every disjoint SMM.

The proposed method is outlined in Algorithm 6.2. The algorithm takes as input the kinematic equations $\mathbf{F}(\mathbf{q}, \mathbf{x}) = \mathbf{0}$, the task value $\mathbf{x}$, and the redundant variables valid interval $\mathcal{R}$, which gathers the valid intervals of the redundant variables, e.g., $\mathcal{R}_i = \left[q_{r_{i,min}}, q_{r_{i,max}}\right]$ denotes the valid interval of the redundant variable $q_{r_i}$. Note that the redundant variables do not necessarily need to be joint values, but we will refer to them as such for simplicity, as it is the most common selection.

**Algorithm 6.2** Homotopy-based Computation of Self-motion Manifolds

**Inputs:** $\mathbf{F}(\mathbf{q}, \mathbf{x}), \mathbf{x}, \mathcal{R}$
$\mathcal{Q}_{r_i} = \{q_{r_{i,min}}, q_{r_{i,min}} + \Delta q_{r_i}, \ldots, q_{r_{i,max}}\}$: a discretization of $\mathcal{R}_i$
$\mathcal{Q}_r = \mathcal{Q}_{r_1} \times \mathcal{Q}_{r_2} \times \ldots \times \mathcal{Q}_{r_r}$: set of points $\mathbf{q}_r$ that form a grid of $\mathcal{R}$
$\mathcal{P} \leftarrow \emptyset$ ▷ Set of joint configurations of the SMMs
**while** not all points $\mathbf{q}_r$ in $\mathcal{Q}_r$ have been processed **do**
  $\mathcal{Q}_h \leftarrow$ HomotopySampling($\mathcal{Q}_r$)
  $\mathcal{Q}_n \leftarrow$ NewtonPropagation($\mathcal{Q}_h$)
  $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{Q}_n$ ▷ Add the new points to the point cloud
$\mathcal{P} \leftarrow$ Redensification($\mathcal{P}$)
$\mathcal{M} \leftarrow$ Cluster($\mathcal{P}$)
**return** $\mathcal{M}$ ▷ Set of disjoint SMMs

The first step of Algorithm 6.2 is to discretize the redundant joint space into a structured grid $\mathcal{Q}r$. This discretization is dictated by the valid intervals $\mathcal{R}$ of the redundant variables, which are provided as input to the algorithm. Each redundant variable $q_{r_i}$ in $\mathcal{R}$ is sampled at regular intervals of size $\Delta q_{r_i}$, creating a discrete set $\mathcal{Q}_{r_i}$. The complete grid $\mathcal{Q}_r$ is then formed by computing the Cartesian product of these individual sets, resulting in a structured sampling of the entire redundant joint space.

The algorithm initializes an empty set $\mathcal{P}$ to store the joint configurations that comprise the SMMs, then enters its main loop which continues until all grid points in $\mathcal{Q}_r$ have been processed. Each iteration invokes two core functions: HomotopySampling and NewtonPropagation, which together form the foundation of the proposed method.

Figure 6.6 demonstrates two iterations of the algorithm applied to our 2-DoF robot example. We will later refer to this figure to explain the two core functions of the algorithm. For the moment, pay attention to the discretization of the redundant space marked by $\mathcal{Q}_r$. In this specific case, since only $q_1$ is designated as the redundant joint ($r = 1$), the grid $\mathcal{Q}_r$ is one-dimensional. For higher redundancy levels, this grid becomes multi-dimensional (e.g., a plane for $r = 2$ in Section 6.4).

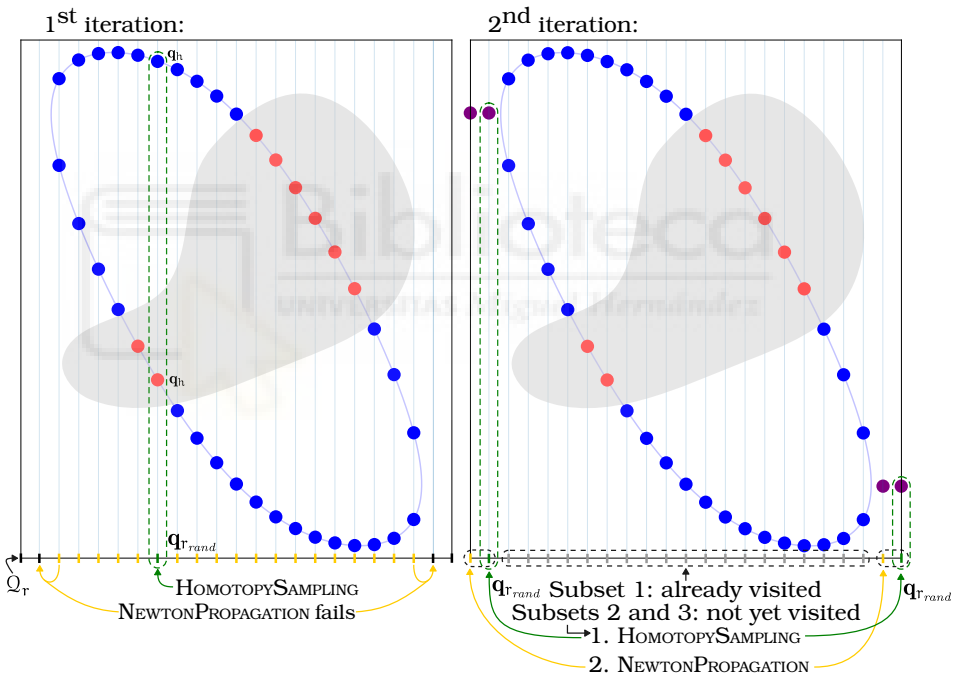


Figure 6.6: Two iterations of the homotopy-based computation of SMMs for the 2-DoF robot example.

In the visualization, different solution types are colour-coded: real solutions satisfying all constraints are shown in blue, the real part of the complex solutions (mathematically valid but physically unrealizable) appear in purple, and real solutions that violate additional constraints (such as collision avoidance) are highlighted in red.

#### 6.3.2.1 *Homotopy Sampling*

The function HomotopySampling is responsible for sampling initial points $\mathbf{q}_h$, from which the SMMs will be propagated. Algorithm 6.3 summarizes the function.

**Algorithm 6.3** Homotopy Sampling

**procedure** HomotopySampling($\Omega_r$)
  $\Omega_h \leftarrow \emptyset$
  $\mathcal{S} \leftarrow$ Clusters of connected *unprocessed* points in $\Omega_r$
  **for** $\mathcal{S}_i \in \mathcal{S}$ **do**
    $\mathbf{q}_{random} \leftarrow$ Randomly select a point in $\mathcal{S}_i$
    Mark $\mathbf{q}_{random}$ as *processed*
    $\mathbf{q}_h \leftarrow$ Homotopy to solve $\mathbf{F}(\mathbf{q}, \mathbf{x}) = \mathbf{0}$ substituting $\mathbf{q}_r$ by $\mathbf{q}_{random}$
    $\Omega_h \leftarrow \Omega_h \cup \mathbf{q}_h$
  **return** $\Omega_h$
**end procedure**

The function first identifies clusters $\mathcal{S}$ of unprocessed points in the redundant space grid $\Omega_r$. Each cluster $\mathcal{S}_i$ represents a continuous region of the redundant space that has not yet been explored. In the first iteration (Figure 6.6(a)), the entire grid $\Omega_r$ forms a single cluster. In the second iteration, illustrated in Figure 6.6(b), the central part of the grid has already been processed, leaving two clusters of points in the grid: the left and right clusters.

For each identified cluster $\mathcal{S}_i$, the algorithm:

1. Randomly selects a point $\mathbf{q}_{random}$ from the cluster
2. Applies the homotopy method to solve $\mathbf{F}(\mathbf{q}, \mathbf{x}) = \mathbf{0}$ with the selected $\mathbf{q}_{random}$ fixed
3. Collects all resulting solutions (which may include multiple configurations for the same $\mathbf{q}_{random}$) in the set $\Omega_h$

The homotopy method returns all possible inverse kinematics solutions for the given redundant joint value, including those that might be physically invalid. For instance, in the first iteration shown in Figure 6.6(a), one solution lies in the forbidden region (red point) while the other is valid (blue point). Despite its invalidity, the red point is retained as it remains significant for its later propagation in order to compute the complete SMM.

Similarly, in the second iteration (Figure 6.6(b)), both clusters yield complex solutions (purple points). While these solutions do not represent physically real-

izable robot configurations, they are crucial in the propagation of the SMM and facilitate the discovery of real solutions in other disjoint components of the SMMs.

This comprehensive sampling approach ensures that all components of the SMMs are discovered, addressing one of the key limitations of traditional continuation methods.

#### 6.3.2.2 *Newton Propagation*

The second core function of the algorithm is NewtonPropagation, which propagates the sampled points $\mathcal{Q}_h$ throughout the SMM. This function efficiently explores the manifold structure by iteratively solving the IKP for adjacent points in the redundant joint space. Algorithm 6.4 presents the approach.

**Algorithm 6.4** Newton Propagation

**procedure** NewtonPropagation($\mathcal{Q}$)
  $\mathcal{N}_r \leftarrow$ Identify *unprocessed* points in $\mathcal{Q}_r$, neighbours to every $\mathbf{q} \in \mathcal{Q}$
  **if** $\mathcal{N}_r = \emptyset$ **then**
    **return** $\mathcal{Q}$ ▷ Termination condition
  Mark all points in $\mathcal{N}_r$ as *processed*
  $\mathcal{Q}_n \leftarrow$ Solve $\mathbf{F}(\mathbf{q}, \mathbf{x}) = \mathbf{0}$ using Newton's method with seeds from $\mathcal{Q}$
  $\mathcal{Q}_n \leftarrow \mathcal{Q}_n \cup \mathcal{Q}$ ▷ Combine new solutions with initial set
  **return** NewtonPropagation($\mathcal{Q}_n$) ▷ Recursive exploration
**end procedure**

The algorithm receives the set of sampled initial points $\mathcal{Q}$ and identifies the neighbouring points $\mathcal{N}_r$ in the redundant space grid $\mathcal{Q}_r$. Note that this determination of neighbours marks the expansion strategy of identification of the SMMs. We have implemented the following strategy:

1. For every point $\mathbf{q}$ in $\mathcal{Q}$, we query the $k$ nearest neighbours in $\mathcal{Q}_r$. Recall from Section 6.2.2 that $k$ is determined by the dimensionality of the redundant joint space, and it is equal to the number of adjacent points in the $r$-dimensional grid, i.e., $k = 3^r - 1$ when diagonal connections are considered. For the 2-DoF example with $r = 1$, this means $k = 2$ neighbouring points.

2. The resulting set of points $\mathcal{N}_r$ is then filtered to remove any points that have been already processed. That is, points $\mathbf{q}_r$ in the redundant space grid $\mathcal{Q}_r$ for which solutions of $\mathbf{F}(\mathbf{q}, \mathbf{x}) = \mathbf{0}$ have already been computed, and are thus already in the point cloud $\mathcal{P}$, either by HomotopySampling or by previous calls to NewtonPropagation.

3. The filtered set of neighbours $\mathcal{N}_r$ is returned.

For each identified neighbouring point $\mathbf{q}_r \in \mathcal{N}_r$, we compute the corresponding complete configuration $\mathbf{q}$ by:

1. Setting the redundant joint values to $\mathbf{q}_r$
2. Using the closest known configuration in $\mathcal{Q}$ as an initial seed
3. Applying Newton's method to solve $\mathbf{F}(\mathbf{q}, \mathbf{x}) = \mathbf{0}$ while keeping $\mathbf{q}_r$ fixed

Newton's method will converge in a few iterations, due to the initial guess being close to the solution. However, convergence is not guaranteed in all cases. For example, in Figure 6.6(a), two propagation attempts fail (marked in the horizontal axis $\mathcal{Q}_r$) because the solution transitions from real to complex across those boundaries. Other singularities may also occur that prevent convergence. Even though Newton's method may still converge to these points from other seeds, we mark them as processed points and do not try to converge to them again in future calls to NewtonPropagation.

The algorithm then collects all the new points $\mathcal{Q}_n$ obtained from the Newton method and adds them to the input set of points $\mathcal{Q}$. Finally, the algorithm recursively calls itself with the updated set of points $\mathcal{Q}_n$.

This process continues until all possible points in the redundant space grid $\mathcal{Q}_r$ have been processed, at which point the algorithm returns the set of points $\mathcal{Q}$. This event occurs when the set of neighbours $\mathcal{N}_r$ is empty, indicating that the current Newton propagation has been exhausted.

This propagation strategy enables efficient exploration of the entire SMM structure while requiring far fewer costly homotopy computations than would be needed to solve the IKP at every grid point independently.

When $\mathcal{N}_r$ becomes empty, it indicates that the current connected component of the SMM has been fully explored. The algorithm then returns to the main Algorithm 6.2, where HomotopySampling will identify seed points for any remaining unexplored components, ensuring complete coverage of all disjoint SMMs.

We have tested other approaches for the propagation of the SMM, such as *parameter homotopies* (Morgan and Sommese, 1989). However, while both methods showcased similar performance in terms of computational efficiency, we found that parameter homotopies were not as robust, since the number of paths that converged to different solutions was not constant (recall path-tracking in Section 6.3.1).

#### 6.3.2.3 *Final Processing Steps*

After completing the main loop of the algorithm, we obtain a point cloud $\mathcal{P}$ containing the processed points of the SMMs. However, this point cloud may contain gaps where Newton's method failed to converge, or regions of low density that

require refinement. To address these issues and ensure complete coverage of the manifolds, we apply a structured redensification process.

Note that these final steps are only performed on the real valid solutions of the IKP (blue points in Figure 6.6). We kept the complex solutions (purple points) and invalid real solutions (red points) for the sake of completeness, but they are not used in the final post-processing since they do not correspond to valid configurations of the robot, and therefore do not belong to the SMMs. From now on, when we refer to the point cloud $\mathcal{P}$, we will be referring to the subset of valid real points in the point cloud, i.e., the blue points in Figure 6.6.

Our redensification approach consists of three targeted steps:

1. **Multiple-seed convergence:** During the calls to the function NewtonPropagation, points where convergence failed were quickly discarded to maintain computational efficiency. We revisit these challenging points by applying Newton's method from multiple alternative seeds. Specifically, from each of the other $k-1$ nearest neighbour configurations. Often, convergence is possible from a different direction, allowing us to fill gaps in the manifold representation efficiently.

2. **Projection singularity treatment:** We apply the redensification technique described in Section 6.2.1 with a computational advantage: rather than searching for projection singularities $\mathbf{q}_s$ from low-density regions, we use the non-convergence information collected during NewtonPropagation. As evidenced by comparing Figure 6.3 with Figure 6.6, the points where Newton propagation failed typically correspond to projection singularities where the manifold becomes tangent to hyperplanes of constant $\mathbf{q}_r$.

3. **Uniform density enforcement:** Finally, we apply the point cloud redensification algorithm from Section 6.2.2 using the same neighbourhood parameter $k$ established during the NewtonPropagation stage. This ensures consistent point density throughout the manifold, particularly in regions of high curvature.

The final step of the algorithm is to classify the point cloud $\mathcal{P}$ into disjoint SMMs. Since we have no a priori knowledge of the number of separate manifolds, we employ density-based clustering techniques- Specifically, DBSCAN (Ester et al., 1996), which does not require pre-specifying the number of clusters. The clustering algorithm groups points in the configuration space based on proximity and connectivity, yielding a set $\mathcal{M}$ of disjoint SMMs.

## 6.4 EXAMPLES

This section presents a comprehensive evaluation of the proposed homotopy-based method for computing SMMs through two representative case studies: a hypothetical robot with 2-sphere SMMs and a practical 5-DoF serial manipulator. We benchmark our approach against established methods in the field to demonstrate its effectiveness, computational efficiency, and robustness in capturing the complete structure of SMMs.

We implemented our algorithm in Python 3.13, with NumPy and SciPy for numerical operations. All experiments were conducted on a machine with an AMD Ryzen 7 5700X3D CPU and 32 GB of RAM, under the Ubuntu 22.04 operating system running on a Windows Subsystem for Linux 2 (WSL2).

The following methods were included in our comparative analysis:

1. **Standard sampling method** (Peidro et al., 2018): The approach described in Section 6.1, using complete manifold discretization across all possible combinations of redundant joints.

2. **Partial sampling with redensification**: A modified sampling method that reduces computational expense by only sweeping one combination of $r$ joints while applying the redensification techniques from Section 6.2.

3. **Higher-dimensional continuation** (Henderson, 2002): A state-of-the-art continuation method for manifold computation.

4. **Our homotopy-based approach**: The method proposed in this chapter.

For each method, we measure computation time and completeness (ability to identify all disjoint manifolds). Every method is executed with parameters that produce similar point densities in the resulting SMMs.

### 6.4.1 *Example 1: 2-Sphere Self-Motion Manifold*

For illustration purposes, our first example is synthetic. That is, it examines a hypothetical 3-DoF robot with joint configuration $\mathbf{q} = [q_1, q_2, q_3]^T$, executing a 1-dimensional task $\mathbf{x} = [x]$. This synthetic example produces a 2-sphere SMM in the joint space:

$$\mathbf{F}(\mathbf{q}, \mathbf{x}) = \left[ q_1^2 + q_2^2 + q_3^2 - 1 \right] = \mathbf{0} \tag{6.14}$$

We selected the first two joints as redundant ($\mathbf{q}_r = [q_1, q_2]^T$), imposed joint limits of $[-1, 1]$ for all joints, and used a discretization step size $\Delta q_{r_i} = 0.07$. This

example provides an analytically known manifold structure, enabling precise verification of our computational results.

Figure 6.7 shows the final result of our homotopy-based method, revealing a uniformly sampled unit spherical manifold. For better understandability of the algorithm's behaviour, Figure 6.8 illustrates the processing sequence with colour-coding of the processing order and highlights the homotopy sampling points (red crosses). Note that this representation excludes the post-processing redensification steps for clarity.

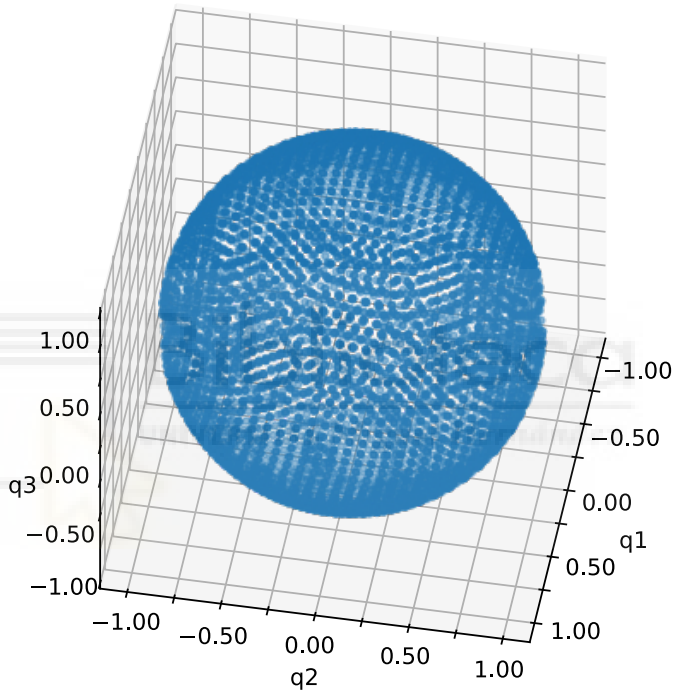


Figure 6.7: Homotopy-based computation of the SMM for the 2-sphere example.

When projected onto the redundant space (the $q_1$-$q_2$ plane), five distinct clusters emerge: one circular region corresponding to real solutions (the unit sphere) and four moon-shaped regions containing complex solutions. The algorithm's exploration strategy is represented through colour progression:

1. The first HomotopySampling call provides a seed point on the actual SMM.
2. NewtonPropagation expands from this seed throughout the manifold until reaching boundaries where solutions transition from real to complex.

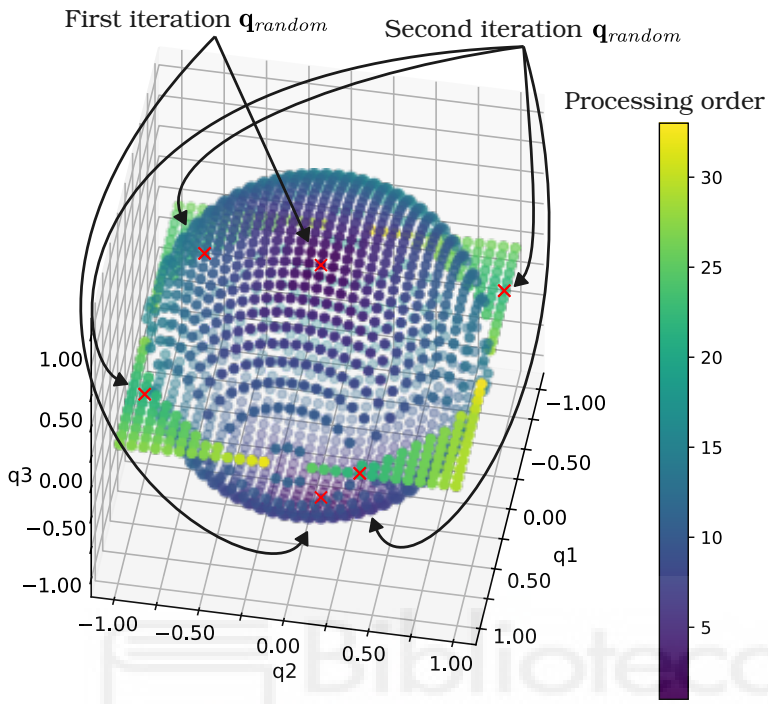


Figure 6.8: Processing order for the SMM for the 2-sphere example.

3. A second HomotopySampling iteration identifies four unprocessed clusters (moon-shaped regions when projected) and samples one point from each.

4. Additional recursive calls to NewtonPropagation completes coverage of the four complex solution regions.

Thanks to the expansion strategy of the NewtonPropagation function, we prove that the algorithm is able to explore the entire SMM with only five calls to the HomotopySampling function, one for each of the five clusters in the redundant space. Note that, independently of the randomness of the sampling, the algorithm is able to cover the entire SMM with the same number of calls to the HomotopySampling function, as transitions between real and complex solutions always render the same number of clusters in the redundant space.

For comparison, Figure 6.9 shows results from alternative approaches. Specifically, we show the results of the continuation method (Henderson, 2002) in Figure 6.9(a), and the standard sampling method (Peidro et al., 2018) in Figure 6.9(b).

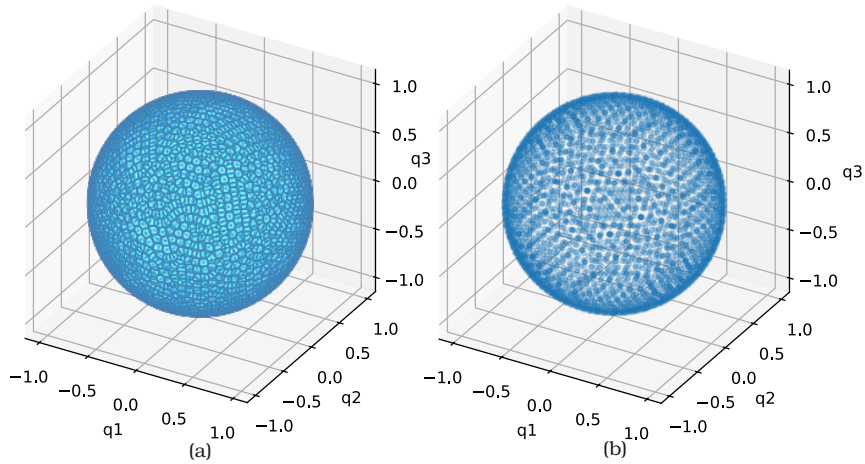


Figure 6.9: Results of the other methods applied to the 2-sphere example: (a) Continuation method (Henderson, 2002). (b) Standard sampling method (Peidro et al., 2018).

Table 6.1 summarizes performance metrics. The sampling-based method is exceptionally fast (3 ms) but requires analytical solutions of every combination of r redundant joints through algebraic elimination, which is a significant limitation for complex kinematic chains. In the following section, we will show that only one of these combinations is needed to compute every SMMs. The continuation method is the slowest of all methods, in addition to the fact that it is not able to compute every disjoint SMM, as it is limited to the one to which the initial given point belongs. The homotopy-based method we presented outperforms the continuation method, and is able to compute all disjoint SMMs in a fraction of the time without the need for any analytical solution.

Table 6.1: Performance of the different methods applied to the 2-sphere example. The computation time is averaged over 100 runs.

| **Method** | **Computation time** |
|---|---|
| Homotopy-based (ours) | 611 ms |
| Sampling-based (Peidro et al., 2018) | 3 ms |
| Continuation-based (Henderson, 2002) | 2717 ms |

It is worth noting that, while the sampling-based method excels in this simple example, its performance advantage diminishes substantially for more complex

kinematic chains with collision constraints or high-degree polynomial equations, as demonstrated in Chapter 5. Under such conditions, continuation-based methods are often preferable for $r = 1$, but still are outperformed by sampling-based methods for $r > 1$.

### 6.4.2 *Example 2: 5R Manipulator*

The second example involves a 5-DoF serial manipulator with joint configuration $\mathbf{q} = [q_1, q_2, q_3, q_4, q_5]^T$ and task value $\mathbf{x} = [x, y, z]^T$, yielding $r = 2$, and a 2-dimensional SMM in the joint space. Specifically, we consider the manipulator denoted as $1^{st}$ *robot* in Table 1 of (Wu et al., 2023), whose kinematic relationships are given by the D-H parameters shown in Table 6.2: The task is to reach the

Table 6.2: D-H parameters of the 5-DoF manipulator.

| **Link** | $a_i$ | $\alpha_i$ | $d_i$ | $\theta_i$ |
|---|---|---|---|---|
| 1 | 0 | $\frac{\pi}{2}$ | 0 | $q_1$ |
| 2 | 1/4 | 0 | 0 | $q_2$ |
| 3 | 1/4 | $\frac{\pi}{2}$ | 0 | $q_3$ |
| 4 | 1/4 | 0 | 0 | $q_4$ |
| 5 | 1/4 | 0 | 0 | $q_5$ |

target position $\mathbf{x} = [0.2, 0, 0.3]^T$. Figure 15(b) in (Wu et al., 2023) shows the SMMs, which are two disjoint 2-dimensional manifolds, computed using their cellular automata-based method.

To replicate this example, we set the joint limits to $[0, 2\pi)$ for all joints, the discretization step size to $\Delta q_{r_i} = 0.1$, and selected the last two joints as redundant ($\mathbf{q}_r = [q_4, q_5]^T$).

Figure 6.10 shows the results of our homotopy-based method, which successfully identifies the two disjoint SMMs. Five calls to the HomotopySampling function were made. Nonetheless, in this example, we have observed that the number of calls to the HomotopySampling function is not constant, as we have encountered regions of singularities not related to the transition between real and complex solutions, which we did not expect, for which, depending on the side from which we approach the singularity, Newton's method may or may not converge, which may lead to different numbers of calls to the HomotopySampling function. Note that this is not a limitation of the algorithm, as the algorithm is still able to cover the entire SMM, but increases the variance of the computational efficiency of the algorithm.

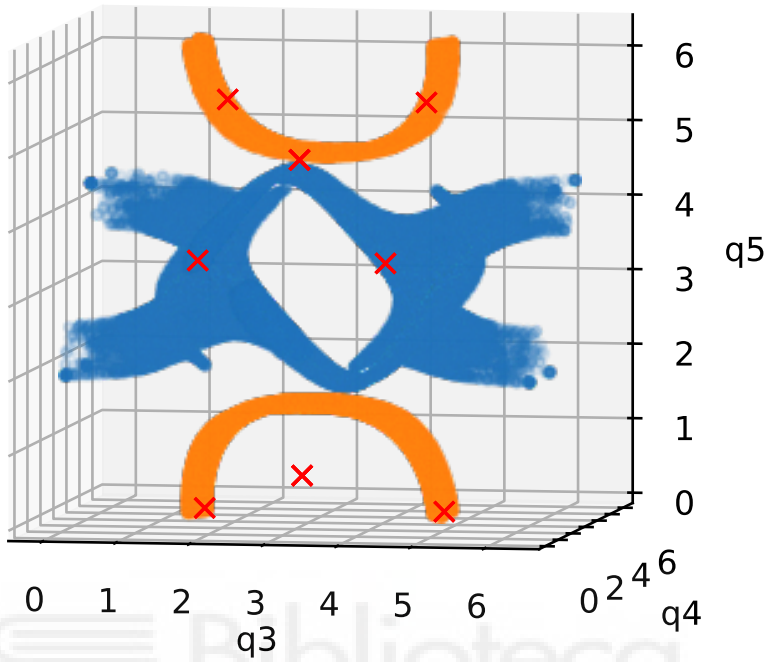


Figure 6.10: Homotopy-based computation of the SMMs for the 5-DoF manipulator.

The results of other methods applied to the 5-DoF manipulator are shown in Figure 6.11. The continuation method (Henderson, 2002) is shown in Figure 6.11(a), and the partial sampling with redensification method is shown in Figure 6.11(b). Note how the continuation method is unable to compute the second SMM (the orange one in Figure 6.10 that is split by the limits of $q_4$), as the provided initial point is located in the first SMM (the blue one in Figure 6.10).

Table 6.3 summarizes the performance of the different methods applied to the 5-DoF manipulator. Note that we present runtimes rounded to seconds, as this example is significantly more computationally expensive than the previous one. The partial sampling with redensification method is the fastest, but it is important to note that it requires solving the IKP for at least one combination of redundant joint values. Out of the methods that do not require solving the IKP, the homotopy-based method is the fastest, outperforming the continuation method by a factor of 7. In addition, the homotopy-based method is able to compute all disjoint SMMs, rather than just one of them, as the continuation method does. Wu et al. (2023) proposed a cellular automata-based method that is able to compute all disjoint SMMs as well, but it is significantly slower than our method, taking more than 3 hours to compute the SMMs of this example.

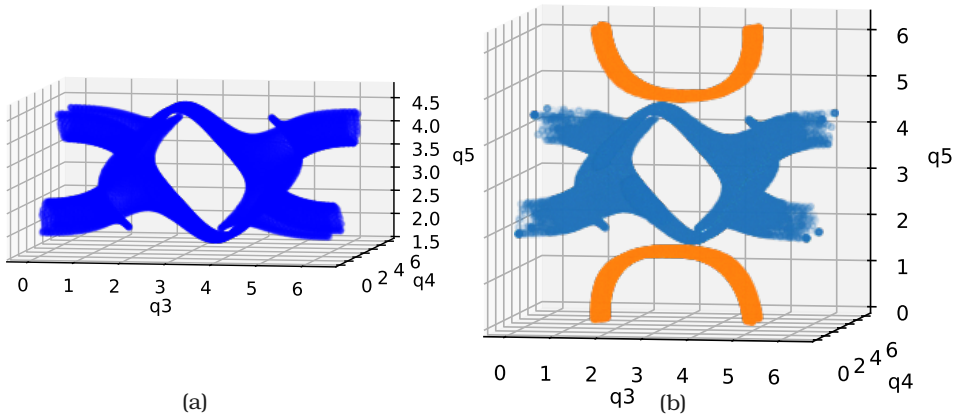


Figure 6.11: Results of other methods applied to the 5-DoF manipulator: (a) Continuation method (Henderson, 2002). (b) Partial sampling with redensification method.

Table 6.3: Performance of the different methods applied to the 5R example. The computation time is averaged over 100 runs, except for the cellular automata method, whose runtime is the one reported in (Wu et al., 2023).

| **Method** | **Computation time** |
|---|---|
| Homotopy-based (ours) | 17 s |
| Partial sampling with redensification | 6 s |
| Continuation-based (Henderson, 2002) | 118 s |
| Cellular automata (Wu et al., 2023) | 11347 s |

It is worth noting that the redensification step in the partial sampling with redensification method is significantly slower than in our homotopy-based method. This is due to the fact that the homotopy-based approach leverages non-convergence information from NewtonPropagation as indicators of projection singularities, eliminating the need for the computationally expensive density-based singularity detection explained in Section 6.2.1. In fact, the version of the identification of projection singularities that we implemented in the partial sampling would benefit from further work, such as vectorizing the operations, and computing the Jacobian analytically.

## 6.5 CONCLUSIONS

In this chapter, we presented a novel homotopy-based method for computing SMMs that addresses the limitations of existing approaches. Our method combines the advantages of both sampling-based and continuation-based approaches while avoiding their respective drawbacks.

Traditional sampling-based methods require analytical solutions of the kinematic equations for multiple combinations of redundant joints, which becomes impractical or impossible for complex kinematic chains. Meanwhile, continuation methods, though generally applicable, suffer from computational inefficiency and the inability to compute all disjoint components of the SMMs.

Our homotopy-based approach offers several key advantages:

- Unlike sampling-based methods, our approach does not require solving via algebraic elimination the kinematic equations of the robot, making it applicable to any kinematic chain regardless of complexity
- The algorithm identifies all disjoint components of the SMMs by tracking both real and complex solutions during propagation, overcoming a critical limitation of traditional continuation methods
- Experimental results demonstrate that our method is significantly faster than continuation methods (7x speed-up for the 5R example) and comparable to sampling-based approaches when analytical solutions are not available
- Our integrated redensification process addresses projection singularities and solves low-density representation of the SMMs, leveraging information collected during the Newton propagation phase to efficiently detect problematic regions

The experimental validation across two representative examples (a hypothetical 2-sphere manifold and a practical 5R manipulator) confirmed these benefits. While the extremely simple spherical example showed the raw speed advantage of analytical sampling methods, the 5R manipulator case demonstrated the true value of our approach for practical robotics applications, especially when compared to other methods.

Future work should focus on conducting experiments in real-world scenarios, and test the algorithm on more complex kinematic chains with additional constraints, such as collision avoidance.

# 7 CONCLUSIONS AND FUTURE WORK

This chapter summarizes the main contributions of this thesis and outlines directions for future research. It also reflects on the broader implications of this work for the field of structure-climbing robots and industrial inspection applications.

## 7.1 CONTRIBUTIONS

This thesis has addressed several fundamental challenges in the planning and control of structure-climbing redundant robots. The main contributions can be summarized as follows:

### 7.1.1 *Hierarchical Path Planning for Structure-Climbing Robots*

In Chapter 3, we presented a hierarchical path planning algorithm for step-by-step biped robots navigating three-dimensional truss structures. The key contributions of this algorithm include:

- A two-level planning strategy that effectively decomposes the complex three-dimensional planning problem into manageable subproblems: global transition planning and local face navigation.
- The Transition Footholds Planner (TFP) that efficiently identifies a sequence of transition footholds connecting different faces of the structure, providing a global path framework.
- The Face Footholds Planner (FFP) that solves the two-dimensional navigation problem within each face, further decomposing the problem into continuous path planning (CFFP) and discrete foothold selection (DFFP).
- Implementation and validation through simulations demonstrating the algorithm's effectiveness in finding optimal paths through complex truss environments, with significant improvements in computational efficiency compared to existing methods.

The hierarchical approach has proven to be both efficient and effective, reducing the computational complexity while maintaining the ability to find high-quality paths through complex structures with both convex and concave transitions.

### 7.1.2 *Feasibility Maps for Redundancy Resolution*

Chapter 4 introduced the concept of Feasibility Maps (FMs) and their application to solving the redundancy resolution problem. The primary contributions in this area include:

- An efficient RRT-based algorithm for exploring Feasibility Maps to generate near-optimal joint-space trajectories that satisfy all constraints.
- Introduction of Augmented Feasibility Maps (AFMs) that extend the FM concept to simultaneously address both redundancy resolution and task trajectory generation, eliminating the need for predefined task trajectories.
- Comprehensive validation through simulations with various redundant robots, demonstrating the method's ability to generate feasible and efficient trajectories in complex environments.

The FM-based approach provides a method for redundancy resolution that addresses many of the limitations of traditional Jacobian-based techniques while being applicable to a wide range of redundant robotic systems.

### 7.1.3 *Global Redundancy Resolution Using Self-motion Manifolds*

In Chapter 5, we developed a comprehensive framework for global redundancy resolution based on Self-motion Manifolds (SMMs) and Self-motion Domains (SMDs). Key contributions of this framework include:

- Algorithms for computing SMDs and analyzing the transformation graph between SMMs, enabling the identification of all disjoint solution components.
- Methods for generating globally optimal joint trajectories based on SMM analysis, ensuring both feasibility and optimality throughout the entire task.
- Implementation and validation through case studies with various redundant robots, demonstrating the framework's ability to identify global solutions that cannot be found by local methods.
- We have managed to run the redundancy resolution algorithm for up to 2 degrees of redundancy in a reasonable time, which few studies have achieved.

This SMM/SMD-based approach represents a significant advancement in redundancy resolution, providing truly global solutions that overcome the limitations of local methods and enabling optimal trajectory planning across the entire task.

### 7.1.4 *Homotopy-based Computation of Self-motion Manifolds*

Chapter 6 introduced a novel homotopy-based method for computing Self-motion Manifolds, addressing a critical gap in the literature. The main contributions of this method include:

- A computational approach that bridges the gap between sampling-based methods (which require analytical inverse kinematics solutions) and continuation methods (which struggle with computational efficiency and finding disconnected manifolds).
- Application of homotopy theory to the inverse kinematics problem, enabling efficient computation of SMMs for complex kinematic chains where analytical solutions are difficult or impossible to obtain.
- A hybrid algorithm that combines homotopy sampling with efficient Newton-based propagation, providing both computational efficiency and the ability to identify all disjoint manifold components.
- Comprehensive validation through multiple examples, demonstrating superior performance compared to other methods, particularly as the number of degrees of freedom increases.

This homotopy-based approach represents a significant methodological advancement in SMM computation, offering a general solution that does not require closed-form kinematic solutions while efficiently identifying all disjoint manifold components.

## 7.2 FUTURE WORK

Despite the significant contributions of this thesis, there are several limitations and open questions that warrant further investigation. These limitations suggest promising directions for future research.

### 7.2.1 *Path Planning for Structure-Climbing Robots*

The hierarchical path planning algorithm presented in Chapter 3 could be enhanced through the following future work:

- **Joint Trajectory Planning:** The current algorithm focuses on foothold planning without fully addressing the joint trajectory planning problem. Future work should integrate the redundancy resolution techniques developed in later chapters with the path planning algorithm to generate complete motion plans.
- **Experimental Validation:** As mentioned in Chapter 3, experimental validation with a physical prototype of the HyReCRo robot is planned for future work. This validation will be crucial for demonstrating the practical applicability of the algorithm in real-world scenarios.
- **Machine Learning Integration:** The development of neural networks for solving the inverse kinematics problem of redundant robots, as mentioned in the future work section of Chapter 3, could significantly enhance the performance of the path planning algorithm. This represents a promising direction for future research.

### 7.2.2 *Redundancy Resolution Methods*

The redundancy resolution approaches presented in Chapters 4, 5, and 6 could be advanced through the following future work:

- **Computational Complexity:** While the proposed methods are more efficient than many existing approaches, the computational complexity still increases significantly with the number of degrees of freedom and redundancy. Future work could focus on further improving the computational efficiency through advanced hybrid techniques, parallel computing, or machine learning approaches.
- **Transitions Between Extended Aspects:** As mentioned in the future work section of Chapter 4, extending the FM-based methods to consider transitions between different extended aspects would qualify them as global redundancy resolution approaches. This integration of local and global methods represents an interesting direction for future research.
- **Improvements to Spatial Analysis:** As noted in Chapter 5, improving the spatial analysis of c-bundles, which currently relies on time-consuming clustering and matching stages, could enhance the overall efficiency of graph generation. This represents a specific technical challenge for future work.
- **Experimental Validation:** As with the path planning algorithm, experimental validation with physical robots will be crucial for demonstrating the

practical applicability of the redundancy resolution methods. This validation should include testing under various conditions and with different types of redundant robots.

## 7.3 CONCLUDING REMARKS

This thesis has developed and validated a comprehensive framework for planning and control of structure-climbing redundant robots, with specific application to the HyReCRo robot. By addressing both path planning and redundancy resolution through novel algorithmic approaches, this work makes significant contributions to advancing the state of the art in robotic systems designed for hazardous inspection and maintenance tasks.

The hierarchical path planning algorithm effectively decomposes the complex problem of navigating three-dimensional structures into manageable subproblems, enabling efficient planning of foothold sequences. The redundancy resolution methods, ranging from approaches based on Feasibility Maps to global frameworks using Self-motion Manifolds, provide the means to execute these plans while satisfying kinematic constraints and optimizing performance.

The novel homotopy-based method for computing Self-motion Manifolds addresses a critical gap in the literature, offering a general approach that does not require closed-form kinematic solutions while efficiently identifying all disjoint manifold components. This methodological advancement has implications beyond the specific application domain of structure-climbing robots, potentially impacting the broader fields of robotics, computer graphics, and biomechanics.

While there are limitations to the current approaches, they also suggest promising directions for future research. These future developments could further enhance the practicality and applicability of structure-climbing robots for industrial inspection and maintenance tasks.

Through these contributions, this thesis advances the field of structure-climbing robotics, bringing these systems closer to practical deployment in real-world industrial applications. By making structure-climbing robots more capable, efficient, and reliable, this work contributes to the broader goal of reducing the need for human workers to perform dangerous inspection and maintenance tasks in hazardous environments.

# 8 CONCLUSIONES Y TRABAJOS FUTUROS

Este capítulo resume las principales contribuciones de esta tesis y expone las líneas de investigación futuras. También reflexiona sobre las implicaciones más amplias de este trabajo para el campo de los robots trepadores de estructuras y las aplicaciones de inspección industrial.

## 8.1 CONTRIBUCIONES

Esta tesis ha abordado varios desafíos fundamentales en la planificación y el control de robots redundantes trepadores de estructuras. Las principales contribuciones se pueden resumir de la siguiente manera:

### 8.1.1 *Planificación Jerárquica de Movimientos para Robots Trepadores de Estructuras*

En el Capítulo 3, se presentó un algoritmo de planificación jerárquica de Movimientos para robots bípedos, de movimiento paso a paso, que trepan estructuras tridimensionales reticulares. Las principales aportaciones de este algoritmo incluyen:

- Una estrategia de planificación en dos niveles que descompone eficazmente el complejo problema de planificación tridimensional en subproblemas manejables: planificación global de transiciones y navegación local en caras.
- El Planificador de Puntos de Transición (TFP), que identifica de manera eficiente una secuencia de puntos de transición que conectan diferentes caras de la estructura, proporcionando un marco de trayectoria global.
- El Planificador de Puntos en Caras (FFP), que resuelve el problema de navegación bidimensional dentro de cada cara, descomponiéndolo a su vez en planificación continua de trayectorias (CFFP) y selección discreta de puntos de pegado (DFFP).
- Implementación y validación mediante simulaciones que demuestran la eficacia del algoritmo para encontrar trayectorias óptimas en entornos complejos reticulares, con mejoras significativas en eficiencia computacional respecto a métodos existentes.

El enfoque jerárquico ha demostrado ser eficiente y efectivo, reduciendo la complejidad computacional y manteniendo la capacidad de encontrar trayectorias de alta calidad en estructuras complejas con transiciones tanto convexas como cóncavas.

### 8.1.2 *Mapas de Factibilidad para la Resolución de Redundancia*

El Capítulo 4 introdujo el concepto de Mapas de Factibilidad (FM) y su aplicación a la resolución del problema de redundancia. Las principales contribuciones en este ámbito incluyen:

- Un algoritmo eficiente basado en RRT para explorar Mapas de Factibilidad y generar trayectorias en el espacio articular cercanas al óptimo que satisfacen todas las restricciones.
- Introducción de los Mapas de Factibilidad Aumentados (AFM), que extienden el concepto de FM para abordar simultáneamente la resolución de redundancia y la generación de trayectorias de tarea, eliminando la necesidad de trayectorias de tarea predefinidas.
- Validación exhaustiva mediante simulaciones con varios robots redundantes, demostrando la capacidad del método para generar trayectorias factibles y eficientes en entornos complejos.

El enfoque basado en FM proporciona un método para la resolución de redundancia que supera muchas de las limitaciones de las técnicas tradicionales basadas en la matriz Jacobiana, siendo aplicable a una amplia gama de sistemas robóticos redundantes.

### 8.1.3 *Resolución Global de Redundancia Usando Variedades de Automovimiento*

En el Capítulo 5, se desarrolló un método integral para la resolución global de redundancia basado en Variedades de Automovimiento (SMM) y Dominios de Automovimiento (SMD). Las principales contribuciones de este método incluyen:

- Algoritmos para el cálculo de SMDs y el análisis del grafo de transformaciones entre SMMs, permitiendo la identificación de todos los componentes disjuntos de la solución.
- Métodos para generar trayectorias articulares globalmente óptimas basadas en el análisis de SMM, garantizando factibilidad y optimalidad a lo largo de toda la tarea.

- Implementación y validación mediante ejemplos con varios robots redundantes, demostrando la capacidad del método para identificar soluciones globales que no pueden ser halladas por métodos locales.
- Se ha conseguido ejecutar el algoritmo de resolución de redundancia para hasta 2 grados de redundancia en un tiempo razonable, lo que pocos estudios han logrado.

Este enfoque basado en SMM/SMD representa un avance significativo en la resolución de redundancia, proporcionando soluciones verdaderamente globales que superan las limitaciones de los métodos locales y permitiendo la planificación óptima de trayectorias a lo largo de toda la tarea.

### 8.1.4 *Cálculo de Variedades de Automovimiento Basado en Homotopía*

El Capítulo 6 introdujo un novedoso método basado en homotopía para el cálculo de Variedades de Automovimiento, abordando una carencia crítica en la literatura. Las principales contribuciones de este método incluyen:

- Un enfoque computacional que soluciona los problemas de los métodos basados en muestreo (que requieren soluciones analíticas de cinemática inversa) y los métodos de continuación (que presentan problemas de eficiencia computacional y de identificación de variedades desconectadas).
- Aplicación de la teoría de homotopía al problema de cinemática inversa, permitiendo el cálculo eficiente de SMMs para cadenas cinemáticas complejas donde las soluciones analíticas son difíciles o imposibles de obtener.
- Un algoritmo híbrido que combina muestreo por homotopía con propagación eficiente basada en el método de Newton, proporcionando tanto eficiencia computacional como la capacidad de identificar todos los componentes disjuntos de la variedad.
- Validación exhaustiva mediante múltiples ejemplos, demostrando un rendimiento superior respecto a otros métodos, especialmente a medida que aumenta el número de grados de libertad.

Este enfoque basado en homotopía representa un avance metodológico significativo en el cálculo de SMM, ofreciendo una solución general que no requiere soluciones analíticas de la cinemática inversa y permitiendo identificar de manera eficiente todos los componentes disjuntos de la variedad.

## 8.2 TRABAJOS FUTUROS

A pesar de las importantes contribuciones de esta tesis, existen varias limitaciones y cuestiones abiertas que merecen ser investigadas. Estas limitaciones sugieren líneas prometedoras para la investigación futura.

### 8.2.1 *Planificación de Trayectorias para Robots Trepadores de Estructuras*

El algoritmo de planificación jerárquica presentado en el Capítulo 3 podría mejorarse mediante los siguientes trabajos futuros:

- **Planificación de Trayectorias Articulares:** El algoritmo actual se centra en la planificación de puntos de pegado sin abordar completamente el problema de la planificación de trayectorias articulares. Trabajos futuros deberían integrar las técnicas de resolución de redundancia desarrolladas en los capítulos posteriores con el algoritmo de planificación de trayectorias para generar planes de movimiento completos.
- **Validación Experimental:** Como se menciona en el Capítulo 3, está prevista la validación experimental con un prototipo físico del robot HyReCRo como trabajo futuro. Esta validación será crucial para demostrar la aplicabilidad práctica del algoritmo en escenarios reales.
- **Integración de Aprendizaje Automático:** El desarrollo de redes neuronales para resolver el problema de cinemática inversa de robots redundantes, como se menciona en la sección de trabajos futuros del Capítulo 3, podría mejorar significativamente el rendimiento del algoritmo de planificación de trayectorias. Esto representa una línea prometedora para la investigación futura.

### 8.2.2 *Métodos de Resolución de Redundancia*

Los enfoques de resolución de redundancia presentados en los Capítulos 4, 5 y 6 podrían avanzar mediante los siguientes trabajos futuros:

- **Complejidad Computacional:** Aunque los métodos propuestos son más eficientes que muchos enfoques existentes, la complejidad computacional sigue aumentando significativamente con el número de grados de libertad y redundancia. Trabajos futuros podrían centrarse en mejorar aún más la eficiencia computacional mediante técnicas híbridas avanzadas, computación paralela o enfoques de aprendizaje automático.

- **Transiciones entre Aspectos Extendidos:** Como se menciona en la sección de trabajos futuros del Capítulo 4, extender los métodos basados en FM para considerar transiciones entre diferentes aspectos extendidos los calificaría como enfoques globales de resolución de redundancia. Esta integración de métodos locales y globales representa una línea interesante para la investigación futura.
- **Mejoras en el Análisis Espacial:** Como se señala en el Capítulo 5, mejorar el análisis espacial de los c-bundles, que actualmente depende de etapas de clustering y matching costosas en tiempo, podría aumentar la eficiencia global de la generación de grafos. Esto representa un reto técnico específico para trabajos futuros.
- **Validación Experimental:** Al igual que con el algoritmo de planificación de trayectorias, la validación experimental con robots físicos será crucial para demostrar la aplicabilidad práctica de los métodos de resolución de redundancia. Esta validación debería incluir pruebas bajo diversas condiciones y con diferentes tipos de robots redundantes.

## 8.3 CONSIDERACIONES FINALES

Esta tesis ha desarrollado y validado un marco integral para la planificación y el control de robots redundantes trepadores de estructuras, con aplicación específica al robot HyReCRo. Al abordar tanto la planificación de trayectorias como la resolución de redundancia mediante enfoques algorítmicos novedosos, este trabajo realiza contribuciones significativas al avance del estado del arte en sistemas robóticos diseñados para tareas de inspección y mantenimiento en entornos peligrosos.

El algoritmo de planificación jerárquica descompone eficazmente el complejo problema de navegar por estructuras tridimensionales en subproblemas manejables, permitiendo la planificación eficiente de secuencias de puntos de pegado. Los métodos de resolución de redundancia, que van desde enfoques basados en Mapas de Factibilidad hasta marcos globales usando Variedades de Automovimiento, proporcionan los medios para ejecutar estos planes cumpliendo las restricciones cinemáticas y optimizando el rendimiento.

El novedoso método basado en homotopía para el cálculo de Variedades de Automovimiento aborda una carencia crítica en la literatura, ofreciendo un enfoque general que no requiere soluciones cerradas de cinemática y permitiendo identificar de manera eficiente todos los componentes disjuntos de la variedad. Este avance metodológico tiene implicaciones más allá del dominio de aplicación específico de los robots trepadores de estructuras, pudiendo impactar los campos más amplios de la robótica, los gráficos por ordenador y la biomecánica.

Si bien existen limitaciones en los enfoques actuales, éstas también sugieren líneas prometedoras para la investigación futura. Estos desarrollos futuros podrían mejorar aún más la practicidad y aplicabilidad de los robots trepadores de estructuras para tareas de inspección y mantenimiento industrial.

A través de estas contribuciones, esta tesis impulsa el campo de la robótica trepadora de estructuras, acercando estos sistemas a su despliegue práctico en aplicaciones industriales reales. Al hacer que los robots trepadores de estructuras sean más capaces, eficientes y fiables, este trabajo contribuye al objetivo más amplio de reducir la necesidad de que los trabajadores humanos realicen tareas peligrosas de inspección y mantenimiento en entornos hostiles.

# BIBLIOGRAPHY


Albu-Schäffer, Alin and Arne Sachtler (2023). "Redundancy resolution at position level." In: *IEEE Transactions on Robotics*.

Allgower, Eugene L and Kurt Georg (2012). *Numerical continuation methods: an introduction*. Vol. 13. Springer Science & Business Media.

Balaguer, Carlos, Antonio Giménez, José Manuel Pastor, VM Padron, and Mohamed Abderrahim (2000). "A climbing autonomous robot for inspection applications in 3d complex environments." In: *Robotica* 18.3, pp. 287–297.

Bambade, Antoine, Sarah El-Kazdadi, Adrien Taylor, and Justin Carpentier (2022). "Prox-qp: Yet another quadratic programming solver for robotics and beyond." In: *RSS 2022-Robotics: Science and Systems*.

Banfield, Ilka and Humberto Rodríguez (2022). "Generation of the Self-motion Manifolds of a Functionally Redundant Robot Using Multi-objective Optimization." In: *Robotics for Sustainable Future: CLAWAR 2021 24*. Springer, pp. 438–452.

Bates, Daniel J, Jonathan D Hauenstein, Andrew J Sommese, and Charles W Wampler (2008). "Adaptive multiprecision path tracking." In: *SIAM Journal on Numerical Analysis* 46.2, pp. 722–746.

Bates, Daniel J, Andrew J Sommese, Jonathan D Hauenstein, and Charles W Wampler (2013). *Numerically solving polynomial systems with Bertini*. SIAM.

Bentley, Jon Louis (1975). "Multidimensional binary search trees used for associative searching." In: *Commun. ACM* 18, pp. 509–517.

Berenson, Dmitry, Siddhartha S Srinivasa, Dave Ferguson, and James J Kuffner (2009). "Manipulation planning on constraint manifolds." In: *2009 IEEE international conference on robotics and automation*. IEEE, pp. 625–632.

Borrel, Paul and Alain Liégeois (1986). "A study of multiple manipulator inverse kinematic solutions with applications to trajectory planning and workspace determination." In: *Proceedings. 1986 ieee international conference on robotics and automation*. Vol. 3. IEEE, pp. 1180–1185.

Boyd, Stephen and Lieven Vandenberghe (2004). *Convex optimization*. Cambridge university press.

Breitenmoser, Andreas and Roland Y. Siegwart (2012). "Surface reconstruction and path planning for industrial inspection with a climbing robot." In: *2012 2nd International Conference on Applied Robotics for the Power Industry (CARPI)*, pp. 22–27.

Burdick, Joel W (1989). "On the inverse kinematics of redundant manipulators: Characterization of the self-motion manifolds." In: *Advanced Robotics: 1989:*

*Proceedings of the 4th International Conference on Advanced Robotics Columbus, Ohio, June 13–15, 1989*. Springer, pp. 25–34.

Cai, Guozhen, Weilin Lin, Shangbiao Wei, Shichao Gu, Haifei Zhu, and Yisheng Guan (2019). "Representation of 3d structure for path planning with biped wall-climbing robots." In: *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, pp. 2250–2255.

Chen, Weinan, Shichao Gu, Yisheng Guan, Hong Zhang, Guanfeng Liu, and Hui Tang (2016). "A multi-layered path planning algorithm for truss climbing with a biped robot." In: *2016 IEEE International Conference on Information and Automation (ICIA)*, pp. 1200–1205.

Chen, Xianlei, Yiping Wu, Huadong Hao, Haolei Shi, and Haocai Huang (2019). "Tracked Wall-Climbing Robot for Calibration of Large Vertical Metal Tanks." In: *Applied Sciences*.

Chung, Wing Kwong and Yangsheng Xu (2011). "A novel frame climbing robot: Frambot." In: *2011 IEEE International Conference on Robotics and Biomimetics*. IEEE, pp. 2559–2566.

De Boor, C (1978). "A practical guide to splines." In: *Springer-Verlag google schola* 2, pp. 4135–4195.

DeMers, David and Kenneth Kreutz-Delgado (1994). "Canonically parameterized families of inverse kinematic functions for redundant manipulators." In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE, pp. 1881–1886.

Deuflhard, Peter (2011). *Newton methods for nonlinear problems: affine invariance and adaptive algorithms*. Vol. 35. Springer Science & Business Media.

Ester, Martin, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *kdd*. Vol. 96. 34, pp. 226–231.

Fabregat-Jaén, M., P. Mollá-Santamaría, E. González-Amorós, A. Peidró, and O. Reinoso (2024). "A SOFTWARE PACKAGE TO LEARN GLOBAL REDUNDANCY RESOLUTION OF REDUNDANT ROBOTS." In: *ICERI2024 Proceedings*. 17th annual International Conference of Education, Research and Innovation. Seville, Spain: IATED, pp. 10201–10207. ISBN: 978-84-09-63010-3. DOI: 10.21125/iceri.2024.2585.

Fabregat Jaen, Marc, Adrián Peidró Vidal, Francisco Jose Soler, Arturo Gil, and Óscar Reinoso (2023a). "Planificación de trayectorias en robots redundantes con mapas de factibilidad y RRT." In: *XLIV Jornadas de Automática*. Universidade da Coruña. Servizo de Publicacións, pp. 581–586. ISBN: 978-84-9749-860-9. DOI: 10.17979/spudc.9788497498609.

Fabregat Jaen, Marc, Adrián Peidró, Paula Molla-Santamaria, Francisco Jose Soler, and Óscar Reinoso (2023b). "Planificación de trayectorias de un robot bípedo trepador de estructuras reticulares." In: *Jornadas de Robótica y Bioin-*

*geniería (Madrid, 14-16 junio de 2023)*. Fundación General de la Universidad Politécnica de Madrid, pp. 171–178. ISBN: 978-84-09-51892-0. DOI: 10.20868/UPM.book.74896.

Fabregat-Jaén, Marc, Adrián Peidró, Paula Mollá-Santamaría, Francisco José Soler, and Oscar Reinoso (2024). "Planificación jerárquica de movimientos de un robot trepador bípedo en estructuras tridimensionales reticulares." In: *Revista Iberoamericana de Automática e Informática Industrial* 21.3, pp. 262–273. DOI: 10.4995/riai.2024.20779.

Fabregat-Jaen, Marc, Adrián Peidro, Arturo Gil, David Valiente, and Oscar Reinoso (2023). "Exploring feasibility maps for trajectory planning of redundant manipulators using RRT." In: *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8. ISBN: 979-8-3503-3991-8. DOI: 10.1109/ETFA54631.2023.10275484.

Fabregat-Jaén, Marc, Adrián Peidró, Óscar Reinoso, Francisco-José Soler, and Luis Miguel Jiménez (2022). "Automatización del pegado de las garras magnéticas de un robot trepador bípedo." In: *Jornadas de Robótica, Educación y Bioingeniería - JREB22*. Vol. 1, pp. 57–63.

Fabregat-Jaén, Marc, Adrián Peidró, Matteo Colombo, Paolo Rocco, and Óscar Reinoso (2025). "Topological and spatial analysis of self-motion manifolds for global redundancy resolution in kinematically redundant robots." In: *Mechanism and Machine Theory* 210, p. 106020. ISSN: 0094-114X. DOI: 10.1016/j.mechmachtheory.2025.106020.

Fabregat-Jaén, Marc, Adrián Peidró, Esther González-Amorós, María Flores, and Óscar Reinoso (2024). "Augmented Feasibility Maps: A Simultaneous Approach to Redundancy Resolution and Path Planning." In: *Proceedings of the 21st International Conference on Informatics in Control, Automation and Robotics - Volume 2: ICINCO*. INSTICC. SciTePress, pp. 166–173. ISBN: 978-989-758-717-7. DOI: 10.5220/0012946000003822.

Fang, Lei, Qinglong Yang, and Ting Yang (2019). "Research on Path Planning Algorithm of Two-Machine Cooperative Wall Climbing and Sanding Robot Based on Ant Colony Algorithm." In: *Lecture Notes in Electrical Engineering*.

Faroni, Marco, Manuel Beschi, Nicola Pedrocchi, and Antonio Visioli (2018). "Predictive inverse kinematics for redundant manipulators with task scaling and kinematic constraints." In: *IEEE Transactions on Robotics* 35.1, pp. 278–285.

Fehlberg, Erwin (1970). "Classical fourth-and lower order Runge-Kutta formulas with stepsize control and their application to heat transfer problems." In: *Computing* 6, pp. 61–71.

Ferrentino, Enrico and Pasquale Chiacchio (2018). "A topological approach to globally-optimal redundancy resolution with dynamic programming." In: *ROMANSY 22–Robot Design, Dynamics and Control: Proceedings of the 22nd*

*CISM IFToMM Symposium, June 25-28, 2018, Rennes, France*. Springer, pp. 77–85.

Ferrentino, Enrico and Pasquale Chiacchio (2020). "On the optimal resolution of inverse kinematics for redundant manipulators using a topological analysis." In: *Journal of Mechanisms and Robotics* 12.3, p. 031002.

Figliolini, Giorgio, Pierluigi Rea, and Marco Conte (2010). "Mechanical design of a novel biped climbing and walking robot." In: *ROMANSY 18 Robot Design, Dynamics and Control: Proceedings of The Eighteenth CISM-IFToMM Symposium*. Springer, pp. 199–206.

Flacco, Fabrizio, Alessandro De Luca, and Oussama Khatib (2015). "Control of redundant robots under hard joint constraints: Saturation in the null space." In: *IEEE Transactions on Robotics* 31.3, pp. 637–654.

Gimenez, A, M Abderrahim, VM Padron, and C Balaguer (2002). "Adaptive control strategy of climbing robot for inspection applications in construction industry." In: *IFAC Proceedings Volumes* 35.1, pp. 19–24.

Guan, Yisheng, Li Jiang, Haifei Zhu, Xuefeng Zhou, Chuanwu Cai, Wenqiang Wu, Zhanchu Li, Hong Zhang, and Xianmin Zhang (2011). "Climbot: A modular bio-inspired biped climbing robot." In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 1473–1478.

Guan, Yisheng, Haifei Zhu, Wenqiang Wu, Xuefeng Zhou, Li Jiang, Chuanwu Cai, Lianmeng Zhang, and Hong Zhang (2013). "A Modular Biped Wall-Climbing Robot With High Mobility and Manipulating Function." In: *IEEE/ASME Transactions on Mechatronics* 18, pp. 1787–1798.

Hart, Peter E, Nils J Nilsson, and Bertram Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths." In: *IEEE transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Haug, Edward J (2024). "Redundant Serial Manipulator Inverse Position Kinematics and Dynamics." In: *Journal of Mechanisms and Robotics* 16.8, p. 081008.

Hauser, Kris and Scott Emmons (2018). "Global redundancy resolution via continuous pseudoinversion of the forward kinematic map." In: *IEEE Transactions on Automation Science and Engineering* 15.3, pp. 932–944.

Henderson, Michael E (2002). "Multiple parameter continuation: Computing implicitly defined k-manifolds." In: *International Journal of Bifurcation and Chaos* 12.03, pp. 451–476.

Hirai, Masahiro, Shigeo Hirose, and Woosub Lee (2013). "Gunryu III: reconfigurable magnetic wall-climbing robot for decommissioning of nuclear reactor." In: *Advanced Robotics* 27, pp. 1099 –1111.

Huang, Haocai, Li Danhua, Zhao Xue, Xianlei Chen, Li Shuyu, Jianxing Leng, and Yan ding Wei (2017). "Design and Performance Analysis of a Tracked Wall-Climbing Robot for Ship Inspection in Shipbuilding." In: *Ocean Engineering* 131, pp. 224–230.

Huber, Birkett and Bernd Sturmfels (1995). "A polyhedral method for solving sparse polynomial systems." In: *Mathematics of computation* 64.212, pp. 1541–1555.

Jaillet, Léonard and Josep M Porta (2012). "Path planning under kinematic constraints by rapidly exploring manifolds." In: *IEEE Transactions on Robotics* 29.1, pp. 105–117.

Jaillet, Léonard and Josep M. Porta (July 2013). "Asymptotically-Optimal Path Planning on Manifolds." In: *Robotics: Science and Systems VIII*. The MIT Press, pp. 145–152. ISBN: 9780262315722.

Jang, Keunyoung, Yun-Kyu An, Byunghyun Kim, and Soojin Cho (2020). "Automated crack evaluation of a high-rise bridge pier using a ring-type climbing robot." In: *Computer-Aided Civil and Infrastructure Engineering* 36, pp. 14 –29.

Kalakrishnan, Mrinal, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal (2011). "STOMP: Stochastic trajectory optimization for motion planning." In: *2011 IEEE international conference on robotics and automation*. IEEE, pp. 4569–4574.

Karaman, Sertac and Emilio Frazzoli (2011). "Sampling-based algorithms for optimal motion planning." In: *The international journal of robotics research* 30.7, pp. 846–894.

Khatib, Oussama (1986). "Real-time obstacle avoidance for manipulators and mobile robots." In: *The international journal of robotics research* 5.1, pp. 90–98.

LaValle, Steven M (1998). "Rapidly-exploring random trees: A new tool for path planning." In: *Research Report 9811*.

Lee, D. T. and Franco P. Preparata (1984). "Euclidean shortest paths in the presence of rectilinear barriers." In: *Networks* 14, pp. 393–410.

Li, Bing, Kenshin Ushiroda, Liang Yang, Qiang Song, and Jizhong Xiao (2017). "Wall-climbing robot for non-destructive evaluation using impact-echo and metric learning SVM." In: *International Journal of Intelligent Robotics and Applications* 1, pp. 255–270.

Li, Tien-Yien (1997). "Numerical solution of multivariate polynomial systems by homotopy continuation methods." In: *Acta numerica* 6, pp. 399–436.

Liegeois, Alain et al. (1977). "Automatic supervisory control of the configuration and behavior of multibody mechanisms." In: *IEEE transactions on systems, man, and cybernetics* 7.12, pp. 868–871.

Lück, Carlos L (1997). "Self-motion representation and global path planning optimization for redundant manipulators through topology-based discretization." In: *Journal of Intelligent and Robotic Systems* 19, pp. 23–38.

Mavrommati, Anastasia, Carlos Osorio, Roberto G Valenti, Akshay Rajhans, and Pieter J Mosterman (2021). "An Application of Model Predictive Control to Reactive Motion Planning of Robot Manipulators." In: *2021 IEEE 17th*

*International Conference on Automation Science and Engineering (CASE)*. IEEE, pp. 915–920.

Morgan, Alexander P and Andrew J Sommese (1989). "Coefficient-parameter polynomial continuation." In: *Applied Mathematics and Computation* 29.2, pp. 123–160.

Morgan, Alexander and Andrew Sommese (1987a). "A homotopy for solving general polynomial systems that respects m-homogeneous structures." In: *Applied Mathematics and Computation* 24.2, pp. 101–113.

Morgan, Alexander and Andrew Sommese (1987b). "Computing all solutions to polynomial systems using homotopy continuation." In: *Applied Mathematics and Computation* 24.2, pp. 115–138.

Nguyen, Son Thanh and Hung Manh La (2021). "A Climbing Robot for Steel Bridge Inspection." In: *Journal of Intelligent & Robotic Systems* 102.

Nigl, Franz, Shuguang Li, Jeremy E Blum, and Hod Lipson (2013). "Structure-reconfiguring robots: Autonomous truss reconfiguration and manipulation." In: *IEEE Robotics & Automation Magazine* 20.3, pp. 60–71.

Nocedal, Jorge and Stephen J Wright (1999). *Numerical optimization*. Springer.

Pagano, David and Dikai Liu (2016). "An approach for real-time motion planning of an inchworm robot in complex steel bridge environments." In: *Robotica* 35, pp. 1280 –1309.

Pagano, David, Dikai Liu, and Kenneth Waldron (2012). "A method for optimal design of an inchworm climbing robot." In: *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, pp. 1293–1298.

Pámanes G, J Alfonso, Philippe Wenger, and José Luis Zapata D (2002). "Motion planning of redundant manipulators for specified trajectory tasks." In: *Advances in Robot Kinematics: Theory and Applications*, pp. 203–212.

Pan, Jia, Sachin Chitta, Dinesh Manocha, Florent Lamiraux, Joseph Mirabel, Justin Carpentier, Louis Montaut, et al. (2015–2024). *Coal: an extension of the Flexible Collision Library*. https://github.com/coal-library/coal.

Parzen, Emanuel (1962). "On estimation of a probability density function and mode." In: *The annals of mathematical statistics* 33.3, pp. 1065–1076.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. (2019). "Pytorch: An imperative style, high-performance deep learning library." In: *Advances in neural information processing systems* 32.

Peidró, A., M. Fabregat-Jaén, A. Gil, D. Valiente, and O. Reinoso (2024). "TEACHING REDUNDANCY RESOLUTION IN REDUNDANT PARALLEL MANIPULATORS WITH AN INTERACTIVE AND GRAPHICAL SIMULATION." In: *INTED2024 Proceedings*. 18th International Technology, Education and Development Conference. Valencia, Spain: IATED, pp. 2677–2686. ISBN: 978-84-09-59215-9. DOI: 10.21125/inted.2024.0740.

Peidró, Adrián, Arturo Gil, José María Marín, Yerai Berenguer, Luis Payá, and Oscar Reinoso (2016a). "Monte-Carlo workspace calculation of a serial-parallel biped robot." In: *Robot 2015: Second Iberian Robotics Conference: Advances in Robotics, Volume 2*. Springer, pp. 157–169.

Peidró, Adrián, Arturo Gil, José María Marín, Yerai Berenguer, and Óscar Reinoso (2016b). "Kinematics, Simulation, and Analysis of the Planar and Symmetric Postures of a Serial-Parallel Climbing Robot." In: *Informatics in Control, Automation and Robotics 12th International Conference, ICINCO 2015 Colmar, France, July 21-23, 2015 Revised Selected Papers*. Springer, pp. 115–135.

Peidró, Adrián, Arturo Gil, José María Marín, and Óscar Reinoso (2015). "Inverse Kinematic Analysis of a Redundant Hybrid Climbing Robot." In: *International Journal of Advanced Robotic Systems* 12.

Peidro, Adrian and Edward J Haug (2023). "Obstacle Avoidance in Operational Configuration Space Kinematic Control of Redundant Serial Manipulators." In: *Machines* 12.1, p. 10.

Peidró, Adrián, Luis Payá, Sergio Cebollada, Vicente Román, and Óscar Reinoso (2020). "Solution of the forward kinematics of parallel robots based on constraint curves." In: *International Conference on Informatics in Control, Automation and Robotics*. Springer, pp. 386–409.

Peidró, Adrián, Óscar Reinoso, Arturo Gil, José María Marín, and Luis Payá (2017). "An improved Monte Carlo method based on Gaussian growth to calculate the workspace of robots." In: *Eng. Appl. Artif. Intell.* 64, pp. 197–207.

Peidro, Adrian, Oscar Reinoso, Arturo Gil, José María Marín, and Luis Paya (2018). "A method based on the vanishing of self-motion manifolds to determine the collision-free workspace of redundant robots." In: *Mechanism and Machine Theory* 128, pp. 84–109.

Peidró, Adrián, Oscar Reinoso, Arturo Gil, José María Marín, Luis Payá, and Yerai Berenguer (2016c). "Calculation of the Boundaries and Barriers of the Workspace of a Redundant Serial-parallel Robot using the Inverse Kinematics." In: *ICINCO (2)*, pp. 412–420.

Peidró, Adrián, Mahmoud Tavakoli, José María Marín, and Óscar Reinoso (2019). "Design of compact switchable magnetic grippers for the HyReCRo structure-climbing robot." In: *Mechatronics*.

Quin, Phillip, Gavin Paul, Alen Alempijevic, and Dikai Liu (2016). "Exploring in 3D with a climbing robot: Selecting the next best base position on arbitrarily-oriented surfaces." In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5770–5775.

Ratliff, Nathan, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa (2009). "CHOMP: Gradient optimization techniques for efficient motion planning."

In: *2009 IEEE international conference on robotics and automation*. IEEE, pp. 489–494.

Reveles, Daniel, Philippe Wenger, et al. (2016). "Trajectory planning of kinematically redundant parallel manipulators by using multiple working modes." In: *Mechanism and Machine Theory* 98, pp. 216–230.

Rochat, Frederic, Ricardo Beira, Hannes Bleuler, and Francesco Mondada (2011). "Tremo: an inspection climbing inchworm based on magnetic switchable device." In: *World Scientific*, pp. 421–428.

Saltaren, Roque, Rafael Aracil, Oscar Reinoso, and Maria Antonieta Scarano (2005). "Climbing parallel robot: A computational and experimental study of its performance around structural nodes." In: *IEEE Transactions on Robotics* 21.6, pp. 1056–1066.

Schmidt, Daniel and Karsten Berns (2013). "Climbing robots for maintenance and inspections of vertical structures—A survey of design aspects and technologies." In: *Robotics and Autonomous Systems* 61.12, pp. 1288–1305.

Schulman, John, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel (2013). "Finding locally optimal, collision-free trajectories with sequential convex optimization." In: *Robotics: science and systems*. Vol. 9. 1. Berlin, Germany, pp. 1–10.

Sedgewick, Robert (2001). *Algorithms in c, part 5: graph algorithms*. Third. Addison-Wesley Professional. ISBN: 9780768685329.

Shvalb, Nir, Boaz Ben Moshe, and Oded Medina (2013). "A real-time motion planning algorithm for a hyper-redundant set of mechanisms." In: *Robotica* 31.8, pp. 1327–1335.

Siciliano, Bruno, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo (2009). *Force control*. Springer.

Slotine, Siciliano B and B Siciliano (1991). "A general framework for managing multiple tasks in highly redundant robotic systems." In: *proceeding of 5th International Conference on Advanced Robotics*. Vol. 2, pp. 1211–1216.

Sommese, Andrew J, Charles W Wampler, et al. (2005). *The Numerical solution of systems of polynomials arising in engineering and science*. World Scientific.

Stellato, Bartolomeo, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd (2020). "OSQP: An operator splitting solver for quadratic programs." In: *Mathematical Programming Computation* 12.4, pp. 637–672.

Stumm, Elena, Andreas Breitenmoser, F. Pomerleau, Cédric Pradalier, and Roland Y. Siegwart (2012). "Tensor-voting-based navigation for robotic inspection of 3D surfaces using lidar point clouds." In: *The International Journal of Robotics Research* 31, pp. 1465 –1488.

Tassi, Francesco, Elena De Momi, and Arash Ajoudani (2021). "Augmented hierarchical quadratic programming for adaptive compliance robot control."

In: *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 3568–3574.

Tavakoli, Mahmoud, Lino Marques, and Anibal T. de Almeida (2011). "3DCLIMBER: Climbing and manipulation over 3D structures." In: *Mechatronics* 21, pp. 48–62.

Tavakoli, Mahmoud, Mohammad Reza Zakerzadeh, GR Vossoughi, and S Bagheri (2005). "A hybrid pole climbing and manipulating robot with minimum DOFs for construction and service applications." In: *Industrial Robot: An International Journal* 32.2, pp. 171–178.

Úbeda, David, José María Marín, Arturo Gil, and Óscar Reinoso (2012). "Design and postures of a serial robot composed by closed-loop kinematics chains." In: *Serial and Parallel Robot Manipulators-Kinematics, Dynamics, Control and Optimization*. IntechOpen.

Viegas, Carlos and Mahmoud Tavakoli (2014). "A single dof arm for transition of climbing robots between perpendicular planes." In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 2867–2872.

Wampler, Charles W (1992). "Bezout number calculations for multi-homogeneous polynomial systems." In: *Applied mathematics and computation* 51.2-3, pp. 143–157.

Ward, Peter Kenneth (2016). *Design of a biologically inspired climbing robot and an adhesion mechanism for reliable and versatile climbing in complex steel structures*. University of Technology Sydney (Australia).

Wenger, Philippe (1992). "A new general formalism for the kinematic analysis of all nonredundant manipulators." In: *Proceedings 1992 IEEE International Conference on Robotics and Automation*. IEEE, pp. 442–447.

Wenger, Philippe, Patrick Chedmail, and Fabienne Reynier (1993). "A global analysis of following trajectories by redundant manipulators in the presence of obstacles." In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE, pp. 901–906.

Whitney, Daniel E (1969). "Resolved motion rate control of manipulators and human prostheses." In: *IEEE Transactions on man-machine systems* 10.2, pp. 47–53.

Wu, Tong, Jing Zhao, and Biyun Xie (2023). "A novel method for computing self-motion manifolds." In: *Mechanism and Machine Theory* 179, p. 105121.

Yang, Chia-Han John, Gavin Paul, Peter Ward, and Dikai Liu (2016). "A path planning approach via task-objective pose selection with application to an inchworm-inspired climbing robot." In: *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 401–406.

Yang, Yajue, Yuanqing Wu, and Jia Pan (2021). "An Interval Branch-and-Bound-Based Inverse Kinemetics Algorithm Towards Global Optimal Redundancy Resolution." In: *arXiv preprint arXiv:2104.12183*.

Yue, Ronggang, Jizhong Xiao, Shaoping Wang, and Samleo L. Joseph (2010). "Three-Dimensional Path Planning of a Climbing Robot Using Mixed Integer Linear Programming." In: *Advanced Robotics* 24, pp. 2087 –2118.

Zanchettin, Andrea Maria and Paolo Rocco (2017). "Motion planning for robotic manipulators using robust constrained control." In: *Control Engineering Practice* 59, pp. 127–136.

Zhou, Zhenyong, Jing Zhao, Ziqiang Zhang, and Xiaohui Li (2023). "Motion Planning Method of Redundant Dual-Chain Manipulator with Multiple Constraints." In: *Journal of Intelligent & Robotic Systems* 108.4, p. 69.

Zhu, Haifei, Junhua Lu, Shichao Gu, Shangbiao Wei, and Yisheng Guan (2021). "Planning Three-Dimensional Collision-Free Optimized Climbing Path for Biped Wall-Climbing Robots." In: *IEEE/ASME Transactions on Mechatronics* 26, pp. 2712–2723.

# A

# SET OF PUBLICATIONS

The major contributions made in the present thesis are supported by two papers published in journals ranked in JCR (Science Edition). The metadata of these journal papers are presented next:

- Marc Fabregat-Jaén, Adrián Peidró, Matteo Colombo, Paolo Rocco, and Óscar Reinoso (2025). "Topological and spatial analysis of self-motion manifolds for global redundancy resolution in kinematically redundant robots." In: *Mechanism and Machine Theory* 210, p. 106020. ISSN: 0094-114X. DOI: 10.1016/j.mechmachtheory.2025.106020
- Marc Fabregat-Jaén, Adrián Peidró, Paula Mollá-Santamaría, Francisco José Soler, and Oscar Reinoso (2024). "Planificación jerárquica de movimientos de un robot trepador bípedo en estructuras tridimensionales reticulares." In: *Revista Iberoamericana de Automática e Informática Industrial* 21.3, pp. 262–273. DOI: 10.4995/riai.2024.20779

These publications are appended next.

Mechanism and Machine Theory 210 (2025) 106020

Contents lists available at ScienceDirect

Mechanism and Machine Theory

journal homepage: www.elsevier.com/locate/mechmt

Research paper

# Topological and spatial analysis of self-motion manifolds for global redundancy resolution in kinematically redundant robots

Marc Fabregat-Jaén [a],*, Adrián Peidró [a], Matteo Colombo [b], Paolo Rocco [b], Óscar Reinoso [a,c]

[a] Instituto de Investigación en Ingeniería de Elche, Universidad Miguel Hernández, Avda. de la Universidad s/n, Elche, 03202, Alicante, Spain
[b] Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo Da Vinci 32, Milano, 20133, Lombardia, Italy
[c] ValgrAI: Valencian Graduate School and Research Network of Artificial Intelligence, Camíde Vera s/n, Edificio 3Q, Valencia, 46022, Valencia, Spain

ARTICLE INFO

*Keywords:*
Self-motion manifolds
Global redundancy resolution
Redundant manipulators
Motion planning
Obstacle avoidance

ABSTRACT

This paper introduces a novel framework for global redundancy resolution in kinematically redundant robots, which have more degrees of freedom than the dimensions required to complete their task. The method is based on the concept of self-motion manifolds (SMMs), which are subsets of the joint space where the robot can move without affecting the task. Given a task trajectory, a sequence of SMMs is generated by building a graph where each node represents a c-bundle, which are sets of SMMs that share the same topology. The graph is then explored to establish feasible paths, from which preliminary joint trajectories are derived. The joint trajectories undergo an iterative optimization process that moves each joint trajectory point along the SMM of the associated task instant. The method is capable of handling kinematic constraints, such as joint limits and collisions, and it is designed to be adaptable to the kinematic complexity of the robot, real-time requirements, or optimality. The effectiveness and global optimality of the method in solving redundancy is validated through simulations with different robots and degrees of redundancy.

## 1. Introduction

*Kinematic redundancy* arises when the degrees of freedom (DoFs) of a robot outnumber the dimensions of the task space. This phenomenon leads to an infinite number of potential solutions to the inverse kinematics problem (IKP) for a given task $\mathbf{x}$. The process of choosing solutions from this infinite set is known as *redundancy resolution*.

When a redundant robot is tasked with executing a prescribed task trajectory $\mathbf{x}(t)$, a continuous joint-space time history $\mathbf{q}(t)$ that resolves how to use the extra DoFs (i.e., redundancy resolution) must be found. The joint-space trajectory $\mathbf{q}(t)$ must not only satisfy task constraints, like the desired end-effector trajectory, but also employ the redundant DoFs to satisfy other kinematic constraints, such as joint limits [1] or collision avoidance [2].

The redundancy resolution problem has been extensively studied in the literature, and a broad collection of methods have been proposed to address it. Depending on the approach used and the scope of the method, these can be categorized in a spectrum that ranges from *local* to *global* redundancy resolution methods, as depicted in Fig. 1.

* Corresponding author.
*E-mail address:* mfabregat@umh.es (M. Fabregat-Jaén).

https://doi.org/10.1016/j.mechmachtheory.2025.106020
Received 19 December 2024; Received in revised form 18 February 2025; Accepted 27 March 2025
Available online 8 April 2025

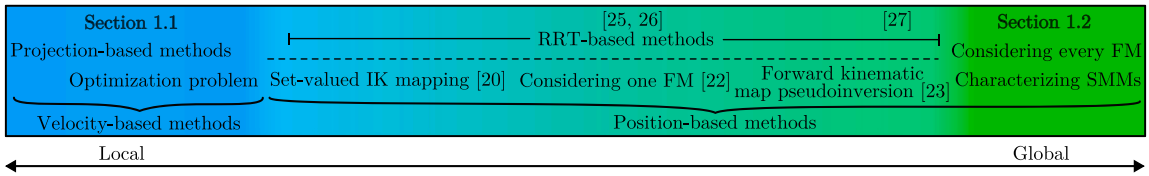


**Fig. 1.** Spectrum of redundancy resolution methods depending on the scope.

### 1.1. Local redundancy resolution

Redundancy resolution is traditionally approached by means of the velocity differential equation $\dot{\mathbf{x}} = \mathbf{J}\,\dot{\mathbf{q}}$, where $\mathbf{J}(\mathbf{q})$ is the configuration-dependent non-square Jacobian matrix that maps joint velocities $\dot{\mathbf{q}}$ to task velocities $\dot{\mathbf{x}}$. *Velocity-based* methods are local techniques that only consider the vicinity of the current configuration $\mathbf{q}$, given that the Jacobian matrix $\mathbf{J}$ depends on $\mathbf{q}$. Depending on how such Jacobian matrix is used, local redundancy resolution methods can be further classified into two categories: *projection-based* and *optimization-based* methods.

Projection-based methods aim to resolve redundancy by pseudoinverting the Jacobian matrix: $\dot{\mathbf{q}} = \mathbf{J}^{\dagger}\,\dot{\mathbf{x}}$ [3], where $(\cdot)^{\dagger}$ denotes the Moore–Penrose pseudoinverse, plus an additional null-space projection term that accomplishes secondary objectives without affecting the main task [4]. Other works have built upon this idea. For example, the authors of [5] proposed a framework for incorporating multiple hierarchically ordered tasks using the null-space projection. Flacco et al. [6] extended this framework into the SNS method, which allows for the simultaneous resolution of multiple tasks by progressively utilizing the residual DoFs that remain from higher-priority tasks.

On the other hand, optimization-based methods formulate the redundancy resolution as an optimization problem, where the objective function is typically a performance index that quantifies the quality of the solution. Rather than pseudoinverting the Jacobian, these methods employ the velocity differential equation $\dot{\mathbf{x}} = \mathbf{J}\,\dot{\mathbf{q}}$ as an equality constraint in the optimization problem. This way, typical problems that arise from pseudoinverting the Jacobian, such as singularities, are avoided. Optimization problems can be solved using a variety of techniques, such as sequential quadratic programming [7], model predictive control [8], or nonlinear versions of those [9], among others. Some works have tried to address the locality of these methods by adding a random component to the optimization process [10,11], but the problems we will discuss next are still present.

Despite the computational efficiency and flexibility of local redundancy resolution techniques, they present certain challenges. The authors of [12] explored these issues for projection-based approaches, some of which may be extended to all local redundancy resolution methods. Given their dependence on the pseudoinversion of the Jacobian matrix, projection-based techniques may produce non-cyclic solutions for the same periodic task trajectory. In industrial settings, where repeatability is crucial, such unpredictability may be considered unacceptable, as it is expected that a closed workspace cycle should return the robot to its original configuration. Another issue is that local methods may not always find a solution, even when one exists. This is due to the fact that the exploration is led by the Jacobian matrix, whose lack of a global scope may lead the algorithm to empty solution sets, due to kinematic constraints or singularities, which materialize as external or internal barriers of the workspace of the robot [13]. Finally, local methods may find a solution that is locally optimal with respect to a given cost function, but not necessarily globally optimal. Once again, this is due to the fact that the scope of these techniques is limited to the proximity of the current configuration, which may prevent them from exploring the entire solution space, missing potentially better solutions.

### 1.2. Global redundancy resolution

On the other end of the spectrum, global redundancy resolution methods are *position-based* approaches that address the issues of local techniques by characterizing the entire solution space. We consider that the global scope of a method is determined by its ability to identify all possible solutions to the IKP. This is typically achieved by calculating the *self-motion manifolds* (SMMs) of the task, which are the subsets of the joint space where the robot can move without affecting the values of the task variables [14], and include all possible solutions to the IKP. Further details on SMMs and their generation will be provided in Section 3.

For instance, Lück [15] proposed building a graph of sets of SMMs for its later exploration in order to extract a joint-space trajectory. [16] presented an RRT-based path planning approach that employs SMMs to determine the continuity between sampled points, and to derive a joint trajectory. The authors of [12] introduced a formal strategy for redundancy resolution at the position level, employing foliations that are orthogonal to the SMMs to establish a global alternative to the null-space projection term of projection-based local methods.

Another approach to global redundancy resolution is the use of *feasibility maps* (FMs) [17]. In contrast with SMMs, which are pointwise representations of the solution space (i.e., SMMs are solutions for a given task-space point), FMs are trajectory-wise representations (i.e., FMs are solutions for a given task-space trajectory). FMs are based on task-space augmentation: for a given task trajectory $\mathbf{x}(t)$, a redundant-coordinates vector $\mathbf{r}$ is added to the task coordinates. FMs are the set of points $(t, \mathbf{r})$ that satisfy the kinematic equations of the robot and the imposed constraints. For every point $(t, \mathbf{r})$, the joint configuration $\mathbf{q}$ that satisfies the desired task $\mathbf{x}(t)$ is solved and checked for feasibility. Every solution $\mathbf{q}$ will lie on an *extended aspect* [18], and there will exist as many FMs as extended aspects, as each one is associated with a possible solution to the IKP. Given that every FM is considered, the method characterizes the entire solution space, thus qualifying as a global redundancy resolution tool. For example, Ferrentino and Chiacchio [19] run an exhaustive dynamic program that explores every FM to find the optimal solution.

### 1.3. In-between approaches

There exist a range of position-based approaches to redundancy resolution that, while not strictly global because they do not exhaustively scan all possible solutions to the IKP for a given task value, have a broader scope than local velocity-based techniques.

Haug [20] presents a set-valued inverse kinematic mapping of redundant manipulators at configuration (or position) level, where the $n$ joint coordinates $\mathbf{q}$ are parameterized in terms of the $m$ task coordinates $\mathbf{x}$ and $n-m$ self-motion coordinates $\mathbf{v}$ that account for redundant degrees of freedom. This mapping is built around a given configuration $\bar{\mathbf{q}}$ of the manipulator by extracting a basis $\mathbf{V}$ of the nullspace of $\mathbf{J}(\bar{\mathbf{q}})$. While in general this mapping is valid only around $\bar{\mathbf{q}}$, its validity can be extended as the trajectory of the manipulator progresses by updating $\mathbf{V}$ whenever the parameterization becomes ill-conditioned. Moreover, depending on the manipulator, the range of validity of such mapping may even become very broad without requiring reparameterizations at all [20]. This mapping elegantly solves deficiencies of traditional generalized-inverse velocity-based approaches such as lack of cyclicity [20], and performs much better than these approaches when dealing with constraints that are better defined at configuration than at velocity level, such as avoidance of obstacles [21].

Another example is the work [22], which proposed a motion planning algorithm that solves the redundancy by exploring one of the FMs associated with the task. The exploration is performed by an RRT-based algorithm that samples points in the FM (instead of doing it in the task or joint spaces) and builds a tree that connects the sampled points. Since only one FM is considered, the method's global scope is restricted to the extended aspect of the solution associated with that FM.

Hauser and Emmons [23] proposed a probabilistic roadmap that concurrently samples points in both the task space and the associated SMM for each task point. Then, a heuristic local three-stage search algorithm is employed to construct a pseudoinversion of the forward kinematic map. Since this pseudoinversion returns a unique $\mathbf{q}$ for each task point $\mathbf{x}$ (instead of all $\mathbf{q}$ that map to $\mathbf{x}$), this may limit finding potentially better solutions. For this reason, although we classify this near the completely global methods in the spectrum of Fig. 1, it is not classified in Section 1.2 with other completely global methods that do obtain all solutions for each task $\mathbf{x}$.

Approaches that utilize sampling-based path planning algorithms cannot be strictly categorized in a single point of the spectrum of Fig. 1, as the scope of the method depends on the exploration strategy. A popular choice is the Rapidly-exploring Random Tree (RRT) algorithm [24], upon which the authors of [25] based their solution, projecting each sampled configuration onto the constraint manifold (i.e., pairs of $(\mathbf{x}, \mathbf{q})$ that satisfy the kinematic constraints of the robot), which allowed for the direct incorporation of additional constraints into the exploration process. Jaillet and Porta [26] proposed a similar approach, where the constraint manifold is described by an atlas, which is iteratively constructed and coordinated, simultaneously with the expansion of the RRT. The AtlasRRT* method, introduced in [27], extends upon the previous work and its asymptotic optimality potentially qualifies it as global redundancy resolution, as it will eventually identify the necessary parts of the constraint manifold. The large amount of existing RRT-based algorithms for redundancy resolution makes it difficult to categorize them all, as they cover a wide range of the spectrum.

### 1.4. Contributions

In this work, we propose a novel framework for global redundancy resolution. The contributions of this paper can be summarized as follows:

- Similarly to [15], we build a graph of c-bundles (i.e., collections of close SMMs sharing the same topology), but instead of building this graph for the entire task space, we focus the computational effort on creating the graph spanned by a desired task trajectory $\mathbf{x}(t)$, which is a more practical approach for real-time applications, where one only needs to solve a concrete trajectory rather than the complete configuration space of the robot. The application of the graph is also unique. While Lück [15] uses the information of the graph to discretize the entire joint and task spaces, we use it to establish the set of feasible graph paths that achieve completion of the desired task trajectory. From each graph path, which we will later define as *c-bundle chains*, we extract a preliminary raw joint trajectory. Finally, these raw joint trajectories are optimized using a novel optimization approach that, for each time instant $t$, keeps the joint configuration of the robot on the SMM corresponding to the task point $\mathbf{x}(t)$, while respecting kinematic constraints such as joint limits and obstacle avoidance.
- We present a variant of the sampling-based method for SMM generation presented in [13]. The performance of the method is improved by vectorizing the sampling of the SMMs, which allows for a significant reduction in the computational cost, and the clustering and matching operations are re-defined. In addition, we extend the method in order to "glue" SMMs along the time dimension to generate the so-called *self-motion domain* (SMD) of the task trajectory, from which the c-bundle graph is inferred.
- We employ the SMD to solve the redundancy resolution problem, and obtain a joint trajectory that is globally optimal in terms of some performance index, while respecting kinematic constraints. The global optimality of the method is discussed and validated through experimental results.
- The presented framework is highly flexible, as it allows the user to select from various computational alternatives at each stage of the algorithm. Among the three stages of the framework, users can select from various alternatives for computation of the SMD, raw trajectory generation, and the optimization stages. This adaptability can address a variety of user-specific needs, whether they are related to generalization (adaptability to complex kinematic chains), redundancy dimension, real-time requirements, or global optimality.

- Our SMM computation method is significantly faster than those presented in the literature. In fact, the method is capable of generating the entire SMD (collection of SMMs discretized along the task trajectory) in a fraction of the time required by other methods to generate SMMs at a single task point.

The remainder of this paper is organized as follows. In Section 2, we provide an overview of the state of the art on global redundancy resolution. Section 3 provides an in-depth explanation of SMMs and their generation, as well as some concepts that are relevant to the proposed method, which is presented in Section 4. In Section 5, we validate the method through simulations. Section 6 discusses the method's optimality, efficiency and scalability. Finally, Section 7 concludes the paper and suggests future directions.

## 2. Related work

Global redundancy resolution has been addressed by a variety of methods in the literature. We consider that the global scope of a method is determined by its ability to identify all possible solutions to the IKP (e.g., the SMMs for a given task $\mathbf{x}$). In this section, we discuss some global redundancy resolution methods.

The recent work in [12] presents a formal strategy to redundancy resolution at position level, employing foliations that are orthogonal to the SMMs. They propose a coordinate-growing approach to generate the foliations, but it is only applicable to one-dimensional tasks, and is described as rather expensive in execution time. For tasks of higher dimensions, which are the norm in practice, the authors employ neural networks to approximate the foliations. However, it is not without its drawbacks, such as the loss of strict orthogonality and the large training time required for the neural networks.

In [15], Lück proposed another approach. He builds a graph where each node represents a c-bundle, and each edge represents a feasible transition between them. Each element of the graph (nodes and edges) corresponds to a significant concept related to SMMs, that will be introduced in Section 3. The information contained in the graph is then discretized, both in the joint and task spaces, resulting in a connectivity graph. The latter graph is finally explored following the A* algorithm, obtaining the globally optimal solution on the discretized space. Specifically, the algorithm outputs a discrete solution in the joint space (the centroid of each discretized cell). While continuity can be achieved by connecting the nodes, it will result in a non-smooth trajectory with high acceleration peaks.

The work of Zhou et al. [16] also presents a method that concurrently addresses path planning and redundancy resolution. They propose a two-stage approach. First, an RRT is built in the task space, determining continuity between sampled points based on the length of the SMMs at each task point. Consequently, the construction of the RRT may necessitate a sufficiently small maximum step size. In the second stage, the authors connect the sampled SMMs in the joint space and derive the joint trajectory based on an exhaustive search that evaluates an averaged performance index (e.g., reciprocal condition number) along each path (i.e., sequence of SMMs).

In the context of FMs, the work of Ferrentino and Chiacchio [19] proposes a global redundancy resolution method. The authors first generate the set of FMs (one per extended aspect) associated with the prescribed task. Then, they run an exhaustive dynamic program that explores every FM to find the optimal solution. The method is computationally expensive, as it requires the evaluation of numerous points within each FM and consider potential combination among FMs. Additionally, the previous generation of multiple FMs is necessary, which is also time-consuming.

These methods have been successful in solving the global redundancy resolution problem, but they present some limitations. Namely, they offer execution times for the global redundancy resolution that are too expensive. In addition, except for the work of Albu-Schäffer and Sachtler [12], none of the methods have been tested with degrees of redundancy higher than one. With our method, we aim to address these limitations, providing a global redundancy resolution method that is capable of globally solving higher degrees of redundancy in a reasonable timeframe.

## 3. Self-motion manifolds

Let Eq. (1) be the constraint that defines the relationship between the joint space $\mathbf{q} \in \mathbb{R}^n$ and the task space $\mathbf{x} \in \mathbb{R}^m$:

$$\mathbf{F}(\mathbf{x}, \mathbf{q}) = \mathbf{0}_{m\times 1} \tag{1}$$

The IKP is defined as finding a joint configuration $\mathbf{q}$ that satisfies $\mathbf{F}(\mathbf{x}, \mathbf{q}) = \mathbf{0}$ for a given task point $\mathbf{x}$. For redundant manipulators, where $n > m$, the IKP yields an infinite number of solutions $\mathbf{q}$ that satisfy the equation. Generically, these solutions lie on a finite number of $r$-dimensional disjoint manifolds in the joint space. Burdick [14] coined the term *self-motion manifolds* (SMMs) to refer to these manifolds, as moving the joint configuration along them does not affect the task-space variables. These representations are particularly useful for redundancy resolution, as they provide a global view of the infinite solutions to the IKP.

Consider a hypothetical 2-DoF $\mathbf{q} = \left[q_1, q_2\right]^{\mathrm{T}}$ redundant manipulator executing a one-dimensional task trajectory $\mathbf{x}(t) = [x]$, represented in Fig. 2(a). The resulting degree of redundancy is $r = 1$, which produces one-dimensional SMMs (i.e., curves) in the joint space for a fixed $\mathbf{x}$. A hypothetical example of the resulting SMMs for the task value $\mathbf{x}(t = f)$ is shown in Fig. 2(b). The plotted curves (SMMs $\mathbf{M}_f^1$ and $\mathbf{M}_f^2$) in the joint space are the set of joint configurations $\mathbf{q}$ that satisfy Eq. (1) for that specific task point $\mathbf{x}(f)$. The presence of two disjoint SMMs at that task point, denoted as $\mathbf{M}_f^1$ and $\mathbf{M}_f^2$, implies that the robot will not be able to transition from a configuration pertaining to one of the SMMs to a configuration within the other without violating Eq. (1).

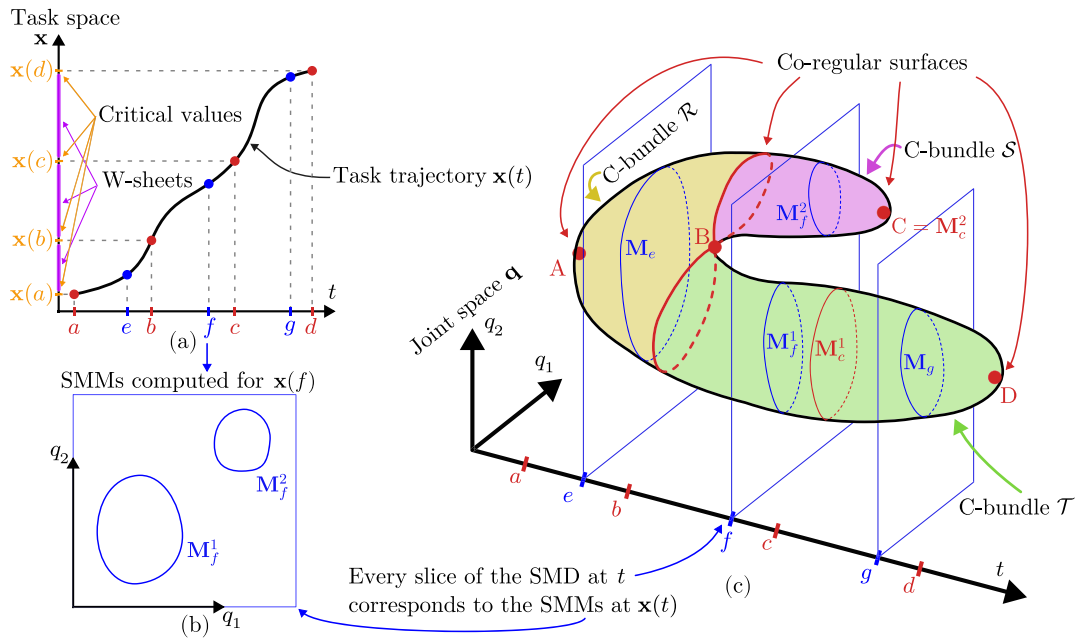


**Fig. 2.** (a) Task trajectory $\mathbf{x}(t)$. (b) One-dimensional SMMs (curves) at task point $\mathbf{x}(f)$. (c) SMD for task trajectory $\mathbf{x}(t)$.

Every SMM corresponds to the infinite joint configurations $\mathbf{q}$ that satisfy Eq. (1) for a fixed task point $\mathbf{x}$. If a task trajectory $\mathbf{x}(t)$ is considered, where $t$ is an arc-length parameter (hereafter referred to as time for simplicity), the solutions $\mathbf{q}$ that satisfy $\mathbf{F}(\mathbf{x}(t), \mathbf{q}) = \mathbf{0}$ form a manifold of $(r+1)$ dimensions, where every slice of the manifold at a specific time $t$ corresponds to the SMM for that task point $\mathbf{x}(t)$ (see Fig. 2(b) and the slice at $t = f$ in Fig. 2(c)). Such $(r+1)$-dimensional manifold, named *self-motion domain* (SMD) in [28], is exemplified in Fig. 2(c) for a representative task trajectory $\mathbf{x}(t)$. The SMMs of three different time values $t = \{e, f, g\}$ are plotted in blue, which result from intersecting the SMD with the planes orthogonal to the axis $t$ at each instant.

Some analysis can be performed based on the observed SMMs. For $t = e$ and $t = g$, there are single one-dimensional manifolds, denoted as $\mathbf{M}_e$ and $\mathbf{M}_g$, respectively. All joint configurations within these manifolds are mapped to $\mathbf{x}(e)$ and $\mathbf{x}(g)$, respectively, when projected to the task space. Consequently, $\mathbf{M}_e$ and $\mathbf{M}_g$ are the *preimages* or solutions to the IKP for task points $\mathbf{x}(e)$ and $\mathbf{x}(g)$, respectively. On the other hand, $\mathbf{M}_f^1$ and $\mathbf{M}_f^2$, which are the preimage manifolds of $\mathbf{x}(f)$ (intersection of the SMD with the plane $t = f$), are two disjoint one-dimensional SMMs. Therefore, it can be inferred that some transformation in the SMMs has occurred as the task trajectory progresses, since the number of disjoint SMMs has changed from one at $t = e$, to two at $t = f$, and back to one at $t = g$.

To better comprehend these transformations, it is beneficial to introduce some paramount concepts presented in [14]. Let $\mathbf{J}$ be the Jacobian matrix that maps joint velocities $\dot{\mathbf{q}}$ to task velocities $\dot{\mathbf{x}}$ (i.e., $\dot{\mathbf{x}} = \mathbf{J}\dot{\mathbf{q}}$). Points of the SMD where $\mathbf{J}$ is not full rank are *singularities*. In the SMD under consideration, the joint configurations that are singularities are represented by the red points $\{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$ in Fig. 2(c). The projections of these singular points onto the time axis $t$ correspond to the values $\{a, b, c, d\}$. The corresponding task values of each singular point (in this case $\{\mathbf{x}(a), \mathbf{x}(b), \mathbf{x}(c), \mathbf{x}(d)\}$ in Fig. 2(a)) are known as *critical values*. Fig. 2(a) also depicts *w-sheets* in purple, which are continuous regions of the task space between critical values.

From critical values, two more important concepts can be derived that play a significant role in our presented redundancy resolution method. Preimages of critical values (i.e., the set of $\mathbf{q}$ that satisfy Eq. (1) for a critical value $\mathbf{x}$) are $r$-dimensional sets, where a *co-regular surface* exists. Indeed, co-regular surfaces can be interpreted as sets where singularities coexist with regular points, thus inspiring the term 'co-regular' (see co-regular surface at $t = b$ in Fig. 2(c)). Note that the term 'surface' may be misleading, as they can be of any dimension (e.g., curves ($r = 1$) in this example). Co-regular surfaces may also correspond to a single singular point, instead of coexisting with regular points, as is the case for singularities A, C and D. A paramount concept of co-regular surfaces is that they divide the SMD into disjoint components, referred to as *c-bundles*, denoted as $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{T}$ in Fig. 2(c). A key characteristic of c-bundles is that they are sets of stacked SMMs that share the same topology, which means that no topological transformations in the SMMs occur within a c-bundle. Therefore, it can be assured that the topological transformations of the SMMs are limited to the boundaries of the c-bundles, which are the co-regular surfaces.

### 3.1. Transformations of self-motion manifolds

Studying the transformations of SMMs is crucial for our proposed redundancy resolution method. The SMD depicted in Fig. 2(c) is composed of three c-bundles: $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{T}$. Several types of transformations can be extracted when studying how the SMMs within the c-bundles evolve.

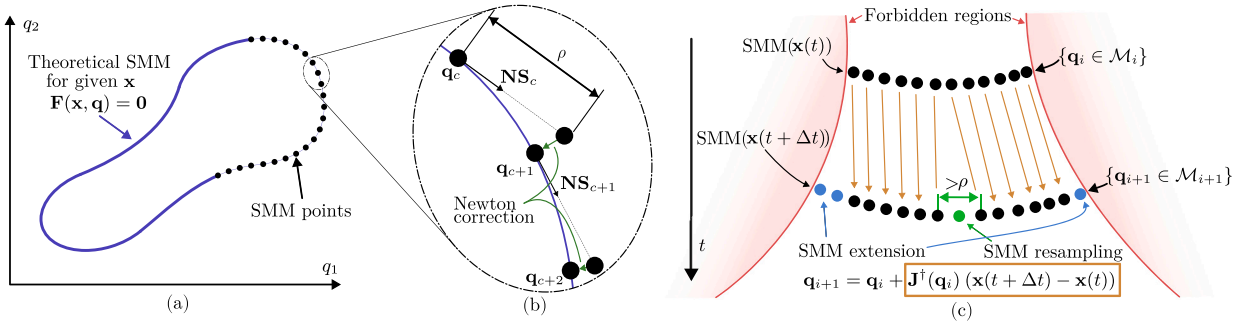


**Fig. 3.** (a) One-dimensional SMM (curve). (b) Continuation method. (c) Task propagation.

Let us analyse the transformations that occur in the SMMs when positively traversing the time axis $t$. The first critical value encountered is $\mathbf{x}(a)$, whose preimage is the singular point A. For values $t < a$, there are no feasible joint configurations that map to $\mathbf{x}(t)$ (i.e., no SMMs exist). At $t = a$, the *creation* of an SMM occurs, which is stated by the appearance of the singularity. This singularity marks the beginning of the c-bundle $\mathcal{R}$. Between instants $t = a$ and $t = b$, the SMMs within the c-bundle $\mathcal{R}$ remain topologically unchanged.

When the critical value $\mathbf{x}(b)$ is reached, intersecting the SMD with the plane $t = b$ returns a co-regular surface. When positively crossing (in the direction of $t$) this co-regular surface, the single SMM that existed in c-bundle $\mathcal{R}$ splits into two disjoint SMMs, which also produces the splitting of the c-bundle $\mathcal{R}$ into two c-bundles: $\mathcal{S}$ and $\mathcal{T}$. Note that the *splitting* operation is of paramount importance for redundancy resolution: if the joint configuration ends up lying in the c-bundle $\mathcal{S}$ when traversing the co-regular surface at $t = b$, the robot will not be able to reach the time instant $t = d$ without violating Eq. (1) (i.e., the task will not be feasible). This phenomenon occurs as the SMMs within c-bundle $\mathcal{S}$ will progressively shrink as they approach the singularity C, which will eventually lead to the *vanishing* of the SMM when crossing $t = c$, thus preventing the robot from reaching the time instant $t = d$. On the other hand, the task would be feasible if the joint configuration was planned to lie in the c-bundle $\mathcal{T}$, as it would be able reach the time instant $t = d$, where the SMM shrinks to the singular point D and then vanishes.

An additional operation can be identified when the SMD is traversed in the opposite direction. When crossing the co-regular surface at $t = b$ in the negative direction of $t$, the two disjoint SMMs merge into a single SMM. Although the *merging* operation is not as relevant for redundancy resolution as the splitting operation, as it does not affect the feasibility of the task, it is still important to consider it, as it dictates the connectivity between different c-bundles, which will be crucial for the spatial analysis of the c-bundles that will be presented later in the paper.

### 3.2. Computation of self-motion manifolds

In this section, we provide an overview of the methods that have been proposed to compute SMMs.

#### 3.2.1. Continuation methods

Along with the introduction of the SMM concept, Burdick [14] proposed the basis for a method to compute them. The method, which we will refer to as the *continuation method*, is based on the continuation in the Jacobian's null-space, which is tangent to the SMM, as is illustrated in Fig. 3(b), where $\mathbf{NS}$ is a null-space basis.

Starting from an initial joint configuration $\mathbf{q}_0$, the method iteratively computes the SMM by displacing the joint configuration a small distance $\rho$ along the null-space of the Jacobian. In Fig. 3(a), a one-dimensional SMM embedded in a 2-DoF joint space is exemplified, where the theoretical SMM is depicted in blue, and the computed points are represented by the black dots. Fig. 3(b) provides a detailed view of the computation process to form the SMM.

In Fig. 3(b), $\mathbf{NS}_c$ corresponds to an orthonormal basis of the null-space of the Jacobian at the $c$th computed point $\mathbf{q}_c$. Adjacent points are computed by displacing the joint configuration $\mathbf{q}_c$ along the null-space basis $\mathbf{NS}_c$ by a distance $\rho$. However, since the Jacobian's null space is a linear approximation of the SMM at each point, the computed adjacent points, which should pertain to the manifold, may not be accurate due to this approximation, especially in curved regions of the manifold, as depicted in Fig. 3(b). A correction must be applied to the computed points to ensure that they lie on the SMM. For this purpose, a Newton-based correction is employed, which is illustrated by the green arrows in Fig. 3(b). The correction is performed by iteratively solving the equation $\Delta\mathbf{x} = \mathbf{J}(\mathbf{q})\,\Delta\mathbf{q}$, where $\Delta\mathbf{x}$ is the difference between the task values of the computed point and the SMM, and $\Delta\mathbf{q}$ is the correction to be applied to the joint configuration. Note that by solving this equation via the Moore–Penrose pseudoinversion of $\mathbf{J}$, the correction $\Delta\mathbf{q}$ is applied perpendicularly to the SMM, which ensures a faster convergence.

Due to the heavy computational operations involved at each iteration, the method is computationally expensive. More importantly, its major limitation is that only the SMM to which the initial joint configuration $\mathbf{q}_0$ belongs will be computed. For task values yielding multiple disjoint SMMs (e.g., the SMMs in Fig. 2(b)), one should provide a joint configuration that acts as a seed for each SMM. Therefore, for applications where every SMM is required, the method may be impractical as it demands the

generation of successive joint configurations until all SMMs are identified, which is complex due to the uncertainty of the exact number of SMMs. Yet, even with this approach, completely identifying all SMMs is not guaranteed unless the number of SMMs equals the maximum number of solutions for the IKP of a non-redundant manipulator of the same class (spherical, regional, spatial, etc.) [14].

In [29], the author proposed a generalization of the original method that allows for the computation of manifolds of higher dimensions. The method is based on the concept of an atlas, which is a collection of charts that cover the manifold. Each chart defines a local coordinate system that is used to parameterize the manifold in a neighbourhood of a point. Similar to the original method, the atlas is built by iteratively computing the neighbouring charts of a given chart until the entire manifold is covered. While it is a generalized approach for any dimension of the manifold, the method shares the same limitations as the original method. Furthermore, with higher degrees of redundancy, the dimensionality of the SMMs increases, and so does the computation time, which makes these methods not suitable for applications where the entirety of the SMD must be generated quickly.

#### 3.2.2. Sampling methods

Another family of methods to compute SMMs are based on sampling. These methods typically involve two stages: the sampling of the joint space to obtain a point cloud that approximates the SMMs, and a clustering stage to identify the disjoint SMMs from the point cloud.

DeMers and Kreutz-Delgado [30] proposed generating the SMMs by randomly sampling the entire joint space, and selecting the joint configurations that are mapped (by forward kinematics) sufficiently close to the task-space value of interest. Then, all joint samples contained in the same task region are clustered by solving a Minimum Spanning Tree to identify the disjoint SMMs. The method is computationally demanding, as it requires the dense sampling of the entire joint space, which is not efficient, since SMMs are lower-dimensional manifolds embedded in the joint space, meaning that the vast majority of the sampled points will not belong to the SMM of interest.

Following a similar approach, Peidró et al. [13] presented a more efficient method. Instead of sampling the entire joint space, the authors generate the point cloud by sweeping every redundant joint that exceeds the number of task dimension and solving the IKP to obtain the remaining joint variables. This produces a densely sampled point cloud, whose points actually pertain to the SMMs, and not the entire joint space. For the clustering stage, the authors propose a recursive method that employs $k$-d trees to efficiently identify the disjoint SMMs. The specific challenge of the method is the need to express $\mathbf{F}(\mathbf{x}, \mathbf{q})$ in terms of every combination of $r$ redundant variables, which is not always possible (i.e., solving $\mathbf{q}$ from $\mathbf{F} = \mathbf{0}$ is not always possible). The method presents several advantages, including computational efficiency, as it directly samples the SMMs rather than the entire joint space; and the guaranteed identification of all manifolds. It outperforms continuation methods in terms of speed, making it suitable for applications where SMMs must be generated in a short time, at least up to a certain degree of redundancy.

The detailed explanation of this family of methods will be left for Section 4.1, where we will present a method to compute SMDs that is based on the sampling approach and is closely related to the method proposed in [13].

#### 3.2.3. Other methods

Other methods to compute SMMs that do not fit into the two most common categories have been presented in the literature. We will briefly discuss these methods, without going into detail, as they are not directly related to our proposed approach.

In [31], the authors propose a novel method to compute SMMs based on the concept of cellular automata. However, the presented computation times displayed are prohibitive for real-time applications, even when computing SMMs at a single task point, let alone the entire SMD.

Zhou et al. [16] presented a continuation approach to compute SMMs by implementing the Artificial Bee Colony Algorithm. This way, the authors are able to compute the SMMs without the need for the Jacobian matrix. Nevertheless, similarly to the continuation methods, it requires a seed joint configuration for each SMM, which is not practical for applications where the entire SMD must be generated, as the exact number of SMMs is unknown (Section 3.2.1).

Similarly, the authors of [32] proposed a method to compute SMMs based on multi-objective optimization. Finally, [28] presented a method to compute SMMs based on interval branch-and-bound. The methods are capable of successfully computing the SMMs for a given task point, but they are computationally expensive.

## 4. Proposed method for global redundancy resolution

As already mentioned, given a defined task trajectory $\mathbf{x}(t)$, the goal of the redundancy resolution problem is to find an optimal joint trajectory $\mathbf{q}(t)$ that satisfies $\mathbf{F}(\mathbf{x}(t), \mathbf{q}(t)) = \mathbf{0}$ in Eq. (1) and the imposed kinematic constraints. The kinematic constraints considered in this work include joint limits and the avoidance of collisions with obstacles.

We propose a three-stage novel framework for global redundancy resolution. The first stage involves the simultaneous generation of the SMD for the given task trajectory, and a graph that shows the connectivity of c-bundles along the SMD. Then, the graph is explored to establish every possible c-bundle chain (sequence of c-bundles) that satisfies the task, deriving a raw joint trajectory for each c-bundle chain. Finally, the raw joint trajectories are optimized by iteratively solving a constrained quadratic programming (QP) optimization problem that moves each discretized point along the SMM they belong to.

Furthermore, we introduce a range of variants for each stage of the algorithm, which allow users to select from various alternatives, depending on the specific requirements of their application.

### 4.1. SMD and graph generation

The first step of the proposed method is the generation of the SMD and graph for the given task trajectory. The SMD is computed using a sampling-based approach, which is closely related to the method proposed by Peidró et al. [13]. The steps are outlined in Algorithm 1. The graph $\mathcal{G}$ is composed of nodes that represent the c-bundles of the SMD for the task trajectory, and edges that represent the transformations that occur in the SMMs when transitioning between adjacent c-bundles.

**Algorithm 1** SMD generation

**Initialize:** $\mathcal{X} \leftarrow$ Ordered set $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k\}$, $\mathcal{G} \leftarrow$ Empty graph
**for** $\mathbf{x}_i \in \mathcal{X}$ **do**
  $\mathcal{P} \leftarrow$ SampleSMMs($\mathbf{x}_i$) ▷ Sample a point cloud that approximates the SMMs at $\mathbf{x}_i$ (Fig. 4(a))
  $\mathcal{M} \leftarrow$ ClusterSMMs($\mathcal{P}$) ▷ Cluster the point cloud into a set of disjoint SMMs (Fig. 4(b))
  $\mathcal{G} \leftarrow$ MatchSMMs($\mathcal{M}, \mathcal{G}$) ▷ Match (or glue) the set of disjoint SMMs to the SMD (Fig. 4(c-d))
**return** $\mathcal{G}$

First, the task trajectory $\mathbf{x}(t)$ is discretized into $k$ task-space points, resulting in the sequence $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$, distributed along the trajectory with a step size $\Delta t$ (i.e., $\mathbf{x}_i = \mathbf{x}(i\,\Delta t)$, with $\Delta t = \frac{t_f}{k-1}$). The number of points $k$ is a user-defined parameter that determines the granularity of the SMD. If $k$ is too small, the SMD may not be accurately represented, while if $k$ is too large, the computational cost will increase. We recommend selecting a value of $k$ that results in time steps $\Delta t$ of approximately 0.05 s, which is an acceptable refresh rate for real-time applications.

Similarly to the methods for computing SMMs discussed in Section 3.2.2, the manifolds are sampled and subsequently clustered to identify the set of disjoint SMMs for each $\mathbf{x}_i$. In the process of generating the SMD, we loop over each task-space point $\mathbf{x}_i$ in $\mathcal{X}$, computing the SMMs at each task value.

The SampleSMMs function generates a densely populated point cloud $\mathcal{P}$ that approximates the SMMs at task point $\mathbf{x}_i$. The procedure is similar to [13]. An outline of the steps is provided in Algorithm 2, but the reader is referred to the original work for further explanation.

**Algorithm 2** Sampling of the SMMs at $\mathbf{x}_i$

1: **procedure** SampleSMMs($\mathbf{x}_i$)
2: $\mathcal{P} \leftarrow \emptyset$
3: **for** every *different* combination of $r$ redundant joints $\overbrace{(q_i, q_j, \cdots, q_k)}^{r \text{ coordinates}} \mid i, j, \cdots, k \in \{1, \cdots, n\}$ **do**
4: Let $\mathcal{Q}_v = \left[q_{v_{\min}}, q_{v_{\max}}\right]$ be the valid interval of joint $q_v$ between its joint limits ($\forall\, v \in \{i, j, \cdots, k\}$)
5: Let $\mathbf{Q}_v = \{q_{v_{\min}}, q_{v_{\min}} + \Delta q_v, \cdots, q_{v_{\max}}\}$ be a discretization of $\mathcal{Q}_v$ with step $\Delta q_v$
6: The Cartesian product $\mathbf{Q}_{red} = \mathbf{Q}_i \times \mathbf{Q}_j \times \cdots \times \mathbf{Q}_k$ is an $r$-dimensional grid of joint coordinates
7: **for** every node $\mathbf{q}_{red}$ of the $r$-dimensional grid $\mathbf{Q}_{red}$ **do**
8: Substitute $\mathbf{q}_{red} = \left[q_i, q_j, \cdots, q_k\right]^{\mathrm{T}}$ and $\mathbf{x} = \mathbf{x}_i$ into Eq. (1) and solve $\mathbf{q}$
9: **if** $\mathbf{q}$ is a *feasible* joint configuration **then** ▷ e.g., joint limits, collision avoidance
10: Add $\mathbf{q}$ to the point cloud $\mathcal{P}$
11: **return** $\mathcal{P}$

The algorithm will store the feasible sampled points in a point cloud $\mathcal{P}$, that is initialized empty. A loop (outer loop) is then started to select every *different* combination of $r$ joint coordinates among the $n$ available. Note that different orderings of the same joint coordinates are not considered, i.e., for $r = 2$, the combination $(q_1, q_2)$ is considered the same as $(q_2, q_1)$. For every said combination, the valid interval $\mathcal{Q}_v$ of each joint coordinate is determined in line 4. The valid interval $\mathcal{Q}_v$ is defined by the joint limits of the robot, and the discretization $\mathbf{Q}_v$ is obtained in line 5 by dividing the interval $\mathcal{Q}_v$ into a set of discrete values with a step $\Delta q_v$. The Cartesian product of the discretized joint coordinates $\mathbf{Q}_{red}$ is then computed in line 6 to generate an $r$-dimensional grid of joint coordinates. Finally, the grid is swept in lines 7–10 (inner loop), where $\mathbf{q}$ is solved for the remaining $m$ joint coordinates that are not in the $r$-dimensional grid. The $m$ remaining joint coordinates in $\mathbf{q}$ are solved by substituting the $r$ joint coordinates $\mathbf{q}_{red}$ into Eq. (1), along with the task point $\mathbf{x}_i$, obtaining every possible solution of $\mathbf{q}$ that maps to the task point $\mathbf{x}_i$. If the resulting joint configuration $\mathbf{q}$ is feasible (e.g., joint limits are met, and no collision is produced), it is added to the point cloud $\mathcal{P}$. Otherwise, the joint configuration is stored in a separate point cloud $\bar{\mathcal{P}}$, for its later use in Section 4.3. This procedure is computationally expensive, as it requires solving the IKP (preferably via algebraic elimination, obtaining all solutions) for every combination of $r$ joint coordinates, producing in practice a number of outer loops (those produced by line 3) equal to the binomial coefficient $\binom{n}{r}$. Although the inner loop (line 7) is presented sequentially in both the presented algorithm and the original work [13], *vectorization* can be employed to bypass it, significantly accelerating the computation. Instead of iterating over each node in the grid $\mathbf{Q}_{red}$ and solving the IKP for each node, vectorization allows for the simultaneous solving of all nodes in the grid.

Next, back in Algorithm 1, the ClusterSMMs function clusters the point cloud $\mathcal{P}$ into a set $\mathcal{M}$ of disjoint SMMs. For this purpose, we utilize DBSCAN [33], a density-based clustering algorithm. Notably, this algorithm does not require specifying the number of

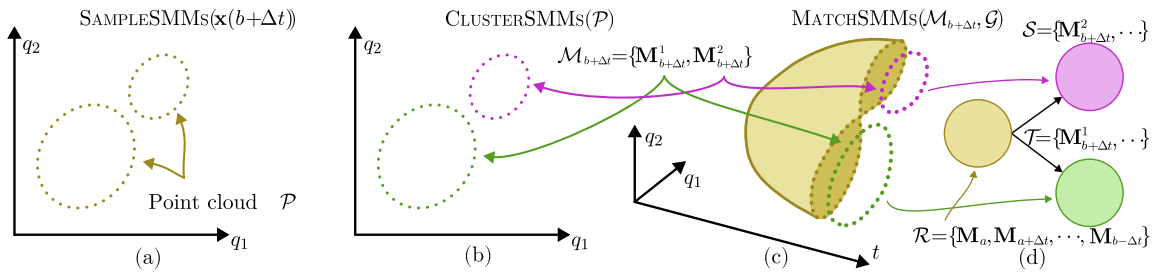


**Fig. 4.** (a) Sampling. (b) Clustering. (c) Matching in the SMD. (d) Matching in the graph.

**Table 1**
Transformations when traversing the SMD of Fig. 2(c) in both directions of axis $t$.

| Traversed critical point | *MatchTo* mapping | | Transformation |
|---|---|---|---|
| | $\mathbf{M}$ | *MatchTo*($\mathbf{M}$) | |
| $t = b$ | $\mathbf{M}^1_{b+\Delta t}$ | $\{\mathbf{M}_b\}$ | Splitting |
| | $\mathbf{M}^2_{b+\Delta t}$ | $\{\mathbf{M}_b\}$ | |
| $t = c$ | $\mathbf{M}_{c+\Delta t}$ | $\{\mathbf{M}^1_c\}$ | – |
| | *MatchTo does not capture vanishings* | | Vanishing |
| *If time were traversed in the negative direction:* | | | |
| $t = b$ | $\mathbf{M}_b$ | $\{\mathbf{M}^1_{b+\Delta t}, \mathbf{M}^2_{b+\Delta t}\}$ | Merging |
| $t = c$ | $\mathbf{M}^1_c$ | $\mathbf{M}_{c+\Delta t}$ | – |
| | $\mathbf{M}^2_c$ | $\emptyset$ | Creation |

clusters, aligning with our earlier discussions about only knowing the maximum number of potential disjoint SMMs, rather than the exact number (Section 3.2.1). The only parameters that must be defined are the maximum distance between two points to be considered part of the same cluster, and the minimum number of points required to form a cluster, which we set to 1. We denote the maximum distance as $h$, derived from the sampling process as follows: $h = \sqrt{\sum_{j=1}^{n} \Delta q_j{}^2}$, where $\Delta q_j$ is the step in joint $j$ during the discretization of its interval as indicated in line 5 of Algorithm 2. The DBSCAN algorithm returns a set of integer labels that indicate the SMM (cluster) to which each point in $\mathcal{P}$ belongs. Utilizing these labels, the point cloud is partitioned into a set of disjoint SMMs, which are stored and returned as the set $\mathcal{M}$.

Finally, the MatchSMMs function will match each disjoint SMM in $\mathcal{M}$ to the SMD generated up to that point (stored in graph $\mathcal{G}$). The matching is performed by comparing the newly generated $\mathcal{M}_t$ with the last computed set of SMMs in the SMD (i.e., SMMS in $\mathcal{M}_{t-\Delta t}$).

Before explaining the matching procedure, we have considered appropriate to exemplify the SMD and graph generation process with a simple example, in order to provide a better understanding of the method, and to introduce a couple of important concepts that will be used throughout the explanation of the last stage of the algorithm.

Consider the same robot and task trajectory $\mathbf{x}(t)$ used in Section 3 to explain SMMs (i.e., $n = 2$, $m = 1$, and $r = 1$), which produces the SMD depicted in Fig. 2(c). As we traverse the axis $t$ in a positive direction, for each instant $t$ between $t = a$ and $t = b$, there is a single SMM. Given that no topological transformations occur in the SMMs (as discussed in Section 3.1), the matching stage should generate the c-bundle $\mathcal{R}$, associating each subsequently generated SMM at $t$ with the single SMM at the previous time instant $t - \Delta t$.

Upon surpassing the critical value $\mathbf{x}(b)$ (i.e., at $t = b + \Delta t$), the sampling stage generates a point cloud $\mathcal{P}$, approximating the two disjoint SMMs resulting from the splitting operation, as illustrated in Fig. 4(a). The clustering stage, depicted in Fig. 4(b), identifies the set $\mathcal{M}_{b+\Delta t} = \{\mathbf{M}^1_{b+\Delta t}, \mathbf{M}^2_{b+\Delta t}\}$ of disjoint SMMs at $t = b + \Delta t$. Subsequently, the matching stage should associate the disjoint SMMs with the SMD stored in $\mathcal{G}$, as shown in Fig. 4(c). The procedure should identify two associations, each corresponding to an SMM at $t = b + \Delta t$, which should be matched to the same SMM $\mathbf{M}_b$ (strictly speaking, $\mathbf{M}_b$ is a co-regular surface, and not a manifold because it self-intersects), as they originate from the splitting operation. This results in the creation of two new c-bundles, $\mathcal{S}$ and $\mathcal{T}$, which are incorporated into the graph $\mathcal{G}$ with edges extending from the c-bundle $\mathcal{R}$, as depicted in Fig. 4(d).

In the matching stage, associations between the newly-created SMMs and the already-computed part of the SMD are stored in a mapping, which we denote as *MatchTo*. The argument of the *MatchTo* mapping are individual SMMs at the current instant, and the value returned is the set of SMMs at the previous instant that are close to the argument SMM (i.e. those SMMs at $t - \Delta t$ that transform into the current SMM at $t$). Table 1 provides a summary of the transformations observed in the SMD of Fig. 2(c) when traversing the time axis $t$ in both directions, as well as the corresponding values of the *MatchTo* mapping. Note that the creation of manifold $\mathbf{M}^2_c$ (shown in Fig. 2(c)) is a vanishing when traversing $t = c$ in the negative direction. While vanishings are not captured by the *MatchTo* mapping, they are implicitly contained in the final graph $\mathcal{G}$, as c-bundles without successors (nodes without successor edges) that do not reach the final instant of the task are considered vanishings.

The matching stage is responsible for the creation of new c-bundles and the establishment of edges between them. C-bundles are stored in the graph $\mathcal{G}$ as nodes, and their internal structure is represented as an ordered set of stacked SMMs identified as part of the same c-bundle. We denote the c-bundle that contains the SMM $\mathbf{M}$ as $C(\mathbf{M})$.

After exemplifying the generation and matching procedure of SMD of Fig. 2(c), we will now explain the matching algorithm in detail. Algorithm 3 displays the steps of the matching stage, which takes the set of disjoint SMMs $\mathcal{M}_t$ at instant $t$ as input, in order to match them to the existing SMMs of the SMD at the previous instant.

**Algorithm 3** Matching the set of disjoint SMMs in $\mathcal{M}_t$ to the SMD in $\mathcal{G}$

```
1:  procedure MatchSMMs(M_t, G)
2:      M_{t-Δt} ← Retrieve the set of SMMs at t − Δt, stored in G
3:      MatchTo ← ∅                                   ▷ Mapping to store the future matching between SMMs
4:      for M_t^i ∈ M_t do
5:          for M_{t-Δt}^j ∈ M_{t-Δt} do
6:              if M_{t-Δt}^j is close to M_t^i then
7:                  Add M_{t-Δt}^j to the set of SMMs close to M_t^i, stored in MatchTo for the argument M_t^i
8:      for M_t^i ∈ M_t do                            ▷ Now, actually process the matching
9:          MT ← MatchTo(M_t^i)                       ▷ Retrieve from MatchTo the set of SMMs that are close to M_t^i
10:         if |MT| = 1 then                          ▷ if the cardinality of MT is 1, M_i matches to a single SMM
11:             M_{t-Δt} ← the single SMM in MT
12:             if M_{t-Δt} ∉ MatchTo(M_t^j)∀j ≠ i then   ▷ M_{t-Δt} is not close to any other SMM
13:                 Add M_t^i to the c-bundle C(M_{t-Δt})
14:                 Continue                          ▷ Continue iterating (skip next lines)
15:         Initialize new c-bundle C_new = {M_t^i} and add it to G
16:         for M_{t-Δt}^j ∈ MT do
17:             Add edge in G from C(M_{t-Δt}^j) to C_new
18:     return G
```

The matching procedure can be divided into two parts for better comprehension: the spatial analysis (lines 2–7) and the matching process (lines 8–17). The algorithm is started by retrieving the set of disjoint SMMs $\mathcal{M}_{t-\Delta t}$ at the previous task instant from the graph $\mathcal{G}$ in line 2. Then, the spatial analysis is performed to determine which SMMs at instant $t-\Delta t$ are close to the newly computed SMMs $\mathcal{M}_t$. This is achieved by a nested loop that iterates over each pair of SMMs, one from $\mathcal{M}_t$ and one from $\mathcal{M}_{t-\Delta t}$, and checks if they are close to each other. The closeness between two SMMs is determined by $d$, which is a parameter similar to $h$, but that also considers the time step to incorporate the extra dimension: $d = \sqrt{\sum_{j=1}^{n} \Delta q_j{}^2 + \Delta t^2}$, where $\Delta q_j$ is the step in joint $j$ during the discretization of its interval in line 5 of Algorithm 2, and $\Delta t$ is the step resulting from the discretization of the task trajectory $\mathbf{x}(t)$ in line 1 of Algorithm 1.

There exist a variety of approaches to distance computation between point clouds in an $n$-dimensional space. We have found that a two-step approach is the most efficient: first, we perform a broad check by evaluating the distance between the bounding boxes of both point clouds, and if they are close enough (i.e., closer than $d$), we proceed to the second step, which involves a more detailed check that computes the distance between every pair of points in both point clouds. The pairwise distances are computed by efficiently querying the $k$-d tree of the point cloud. If any pair of points (each belonging to a different point cloud) is closer than $d$, the SMMs are considered to be close, reaching line 7 of Algorithm 3. If this is the case, the SMM $\mathbf{M}_{t-\Delta t}^j$ is added to the set of SMMs close to $\mathbf{M}_t^i$, which is stored in $MatchTo(\mathbf{M}_t^i)$. Upon completion of the nested loop, the mapping $MatchTo$, initialized in line 3, will collect the set $\mathcal{MT}$ of SMMs from the previous instant that are close to each SMM $\mathbf{M}_t^i$ at the current instant.

When the spatial analysis for each SMM is completed, the actual matching process is executed between lines 8–17. For every SMM $\mathbf{M}_t^i$ at the current instant, the set $\mathcal{MT}$ of close SMMs is retrieved from $MatchTo(\mathbf{M}_t^i)$. The rest of the iteration will serve to match $\mathbf{M}_t^i$ to the manifolds in $\mathcal{MT}$. In line 10, the algorithm verifies whether the SMM $\mathbf{M}_t^i$ has come from a single manifold at the preceding instant (i.e., the cardinality of $\mathcal{MT}$ is 1). Should this be true, the SMM $\mathbf{M}_{t-\Delta t}$ is retrieved from the set $\mathcal{MT}$, and the algorithm checks if $\mathbf{M}_{t-\Delta t}$ has not been matched to any other SMM at the current instant (line 12). If such condition is met, the SMM $\mathbf{M}_t^i$ is incorporated into the graph $\mathcal{G}$ as a direct extension (i.e., no topological transformation occurs) of the c-bundle $C(\mathbf{M}_{t-\Delta t})$ to which $\mathbf{M}_{t-\Delta t}$ pertains. When line 13 is reached, the algorithm skips the next lines and continues to the next SMM in $\mathcal{M}_t$.

Otherwise, when $\mathbf{M}_t^i$ matches to multiple SMMs at the previous instant (condition at line 10 returns false) or the single SMM in $\mathcal{MT}$ belongs to $MatchTo(\mathbf{M}_t^j)$ for any other manifold $\mathbf{M}_t^j$ different from $\mathbf{M}_t^i$ (condition at line 12 is not met), the algorithm initializes a new c-bundle $C_{new}$, as an ordered set of SMMs, with $\mathbf{M}_t^i$ as the only initial component (line 15). Then, the algorithm iterates over every manifold $\mathbf{M}_{t-\Delta t}^j$ in the set $\mathcal{MT}$, adding an edge in the graph $\mathcal{G}$ from the c-bundle $C(\mathbf{M}_{t-\Delta t}^j)$ to the new c-bundle $C_{new}$.

Finally, the algorithm returns the graph $\mathcal{G}$, which is composed of nodes that correspond to the c-bundles, and edges that represent portions of the co-regular surfaces that connect these c-bundles. Note that the SMD can subsequently be retrieved at any time from the graph $\mathcal{G}$ by extracting the ordered set of SMMs stacked within each c-bundle.

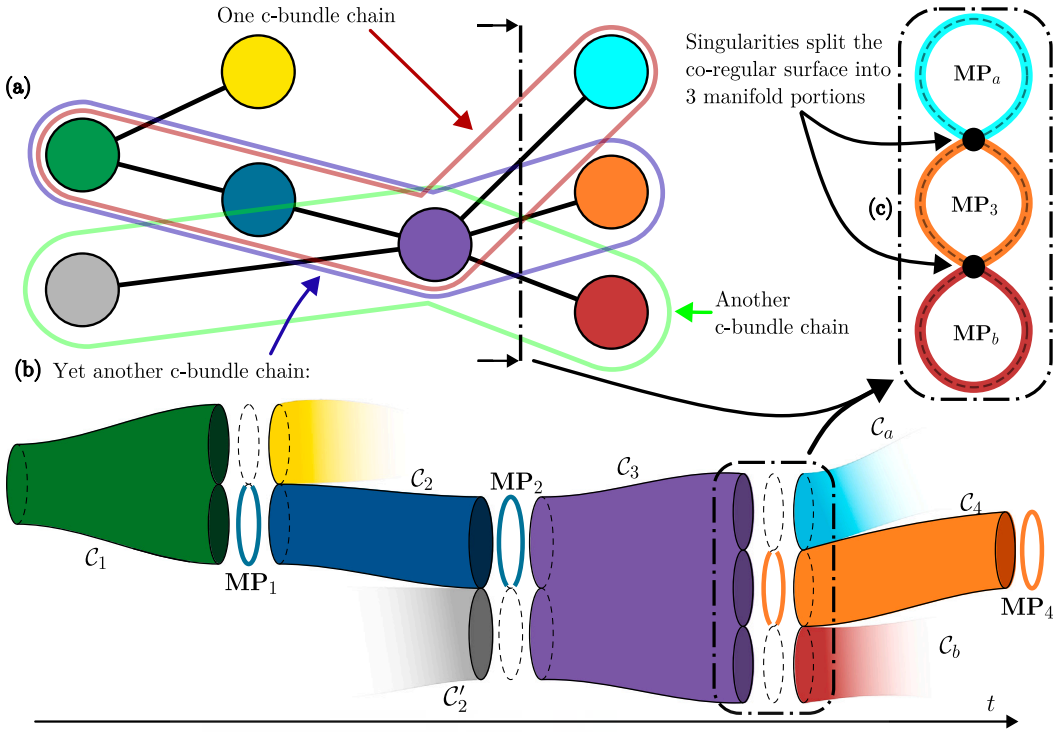


**Fig. 5.** (a) Graph $\mathcal{G}$ and three possible c-bundle chains. (b) Corresponding dismembered SMD for one of the c-bundle chains. (c) Splitting of the co-regular surface into manifold portions.

### 4.2. Generation of raw joint trajectories

The second stage of the proposed method involves the generation of the raw joint trajectories that will be optimized in the final stage. By tracing a continuous path along the generated SMD, from the initial to the final time instant, a joint trajectory that satisfies the task trajectory and kinematic constraints can be generated. We propose generating raw trajectories along the SMD, which will serve as initial trajectories for the subsequent optimization stage, bypassing the need for direct tracing of the globally optimal path in the SMD.

In the previous stage, we constructed a graph $\mathcal{G}$ that represents the SMD, where each node corresponds to a c-bundle. By combining the c-bundles in a specific order, where the initial c-bundle contains the SMM at the initial instant and the final c-bundle contains an SMM at the final instant, a *c-bundle chain* is formed. A c-bundle chain $\mathcal{CBC}$ is a sequence of nodes in the graph $\mathcal{G}$ that connects the initial and final instants: $\mathcal{CBC} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_f\}$, where $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ are adjacent and timewise successive nodes in the graph $\mathcal{G}$, $f$ is the number of c-bundles in the chain, $\mathcal{C}_1$ contains an SMM at the initial instant and $\mathcal{C}_f$ contains an SMM at the final instant ($\mathcal{C}_f$ extends until the final time value). Note that since the c-bundle chains are simple paths (i.e., a path with no repeated nodes) of the graph $\mathcal{G}$, the number of c-bundle chains is finite. Three possible c-bundle chains for a hypothetical graph $\mathcal{G}$ are illustrated in Fig. 5(a). Fig. 5(b) shows the corresponding SMD, dismembered to show the c-bundles in one of the c-bundle chains.

While an infinite number of joint trajectories can be generated from the SMD, our focus is on the extraction of a finite number of joint trajectories, each corresponding to a c-bundle chain. To establish every c-bundle chain, we employ a depth-first search algorithm, as detailed in [34]. The algorithm identifies all simple paths from a source node and a target node within a graph. Consequently, it must be executed for each pair of source and target nodes in the graph $\mathcal{G}$. Here, the source nodes represent the c-bundles to which the SMMs generated at the initial instant belong ($\mathcal{C}_1$ in $\mathcal{CBC}$), and the target nodes correspond to the c-bundles containing every SMMs computed at the final instant ($\mathcal{C}_f$ in $\mathcal{CBC}$). If the initial joint configuration is restricted, as is most often the case in real-time applications where no prior self-motion is desired, the source node will be the c-bundle that includes the SMM that contains the initial joint configuration. For the purpose of this explanation, we assume that the initial joint configuration $\mathbf{q}_0$ is restricted (only one source node is considered), noting that extension to the unrestricted case is trivial.

Next, we propose generating raw joint trajectories along each c-bundle chain. While various methods can be utilized for this purpose, as we will discuss in Section 4.4.2, we present a simple, yet highly efficient approach based on nearest-neighbour queries. Given that the quality of the raw joint trajectories is not crucial, as they will undergo optimization in the final stage of the framework, this method is sufficient. It leverages the same $k$-d trees used in the matching stage of the graph construction (Section 4.1) to efficiently query the closest point in an SMM. The details of this algorithm are provided in Algorithm 4.

Before getting into the details of the algorithm, we must define the concept of a *manifold portion*. For every adjacent pair of c-bundles $C_i$ and $C_{i+1}$ in the c-bundle chain $\mathcal{CBC}$, the manifold portion corresponds to a portion of the co-regular surface that connects the c-bundles. As we reviewed in Section 3, every c-bundle is delimited by co-regular surfaces, which are sets that contain regular and singular points. However, a co-regular surface does not only delimit the c-bundles $C_i$ and $C_{i+1}$, but also any other c-bundle that results from a topological transformation (see how $C_3$ is connected to $C_a$, $C_4$ and $C_b$ in Fig. 5(b)). Co-regular surfaces are split by singularities into what we call "manifold portions", as exemplified in Fig. 5(c). Fig. 5(c) shows a co-regular surface that is a triple-eight curve with two self-intersections. This is where c-bundle $C_3$ ends and splits into new c-bundles $C_a$, $C_4$ and $C_b$. By removing the self-intersections from this co-regular surface, manifold portions that connect adjacent c-bundles are obtained: manifold portion $\mathbf{MP}_a$ is an open curve that connects c-bundle $C_3$ with $C_a$, $\mathbf{MP}_3$ is a pair of open curves that connect $C_3$ with $C_4$, and $\mathbf{MP}_b$ is another open curve that joins $C_3$ with $C_b$. Regarding the notation employed in Algorithm 4, $\mathbf{MP}_i$ is the manifold portion that joins c-bundle $C_i$ with $C_{i+1}$ in the considered c-bundle chain. A special case occurs when the last c-bundle in the chain is reached, since there is no next c-bundle, the manifold portion $\mathbf{MP}_f$ is the last SMM in the c-bundle $C_f$ ($\mathbf{MP}_4$ in Fig. 5(b)).

**Algorithm 4** Generation of a raw joint trajectory along a c-bundle chain $\mathcal{CBC}$

```
1:  Q = {q_0}                                   ▷ Initialize the raw trajectory with the initial joint configuration
2:  for i = 1, 2, ..., f do
3:      C_i ← i-th c-bundle in CBC
4:      MP_i ← Manifold portion that connects C_i and C_{i+1} in CBC
5:      q_a ← Last point in Q
6:      q_b ← Closest point in MP_i to q_a
7:      n_i ← Number of SMMs stacked in C_i
8:      Q_l ← Linear interpolation from q_a to q_b, generating n_i points
9:      for j = 1, 2, ..., n_i do
10:         M_j ← j-th SMM stacked in C_i
11:         q_l ← Point in Q_l corresponding to the same instant as M_j
12:         q_c ← Closest point in M_j to q_l
13:         Add q_c to Q
14: return Q
```

The algorithm receives as input the c-bundle chain $\mathcal{CBC}$ and the initial joint configuration $\mathbf{q}_0$, which initializes the raw joint trajectory $\mathcal{Q}$ in line 1. Then, the algorithm iterates over every c-bundle in the c-bundle chain $\mathcal{CBC}$ (lines 2–3). For each c-bundle $C_i$, the algorithm retrieves the manifold portion $\mathbf{MP}_i$ that connects the c-bundles $C_i$ and $C_{i+1}$ in the c-bundle chain $\mathcal{CBC}$ (line 4). If $C_i$ is the last c-bundle in the chain, $C_{i+1}$ does not exist, and $\mathbf{MP}_i$ simply corresponds to the last SMM in $C_i$.

During each iteration, the algorithm retrieves the last point (up to that moment) in the raw trajectory $\mathcal{Q}$, denoted as $\mathbf{q}_a$. Then, the nearest point in $\mathbf{MP}_i$ is queried and stored as $\mathbf{q}_b$. At this point of the iteration, both $\mathbf{q}_a$ and $\mathbf{q}_b$ represent points located in limiting SMMs of the currently processed c-bundle $C_i$. If a straight line is drawn between $\mathbf{q}_a$ and $\mathbf{q}_b$, it will represent the shortest path that traverses the c-bundle $C_i$, starting from a fixed initial point ($\mathbf{q}_a$). This straight-line path, however, does not accurately represent the actual trajectory, as the straight-line segment that connects $\mathbf{q}_a$ and $\mathbf{q}_b$ will not belong to the SMD in general. Therefore, this straight line serves merely as a reference and is not expected to depict a realistic joint trajectory. To produce this guiding path, the algorithm linearly interpolates between $\mathbf{q}_a$ and $\mathbf{q}_b$, producing $n_i$ points, where $n_i$ corresponds to the number of discretized SMMs comprising the c-bundle $C_i$. To correct this line segment and "project" it onto the SMD, lines 7–13 are executed. First, recall that, for Algorithm 4, c-bundle $C_i$ is an ordered collection of $n_i$ homotopic SMMs stacked along the time dimension. Thus, the straight segment between $\mathbf{q}_a$ and $\mathbf{q}_b$ is discretized into a set $\mathcal{Q}_l$ of $n_i$ points, one per each SMM $\mathbf{M}_j$ (line 8). A final loop then iterates over each SMM $\mathbf{M}_j$ in the c-bundle $C_i$ (lines 9–13). For each SMM $\mathbf{M}_j$, the algorithm retrieves the point $\mathbf{q}_l$ in the linearly interpolated path $\mathcal{Q}_l$ that corresponds to the same instant as $\mathbf{M}_j$. The algorithm queries the nearest point $\mathbf{q}_c$ in the SMM $\mathbf{M}_j$ to the interpolated point $\mathbf{q}_l$ and incorporates it into the raw trajectory $\mathcal{Q}$. This query is executed efficiently using the $k$-d tree of the SMM, which was computed during the matching stage of the graph construction (Section 4.1).

Finally, the algorithm returns the raw joint trajectory $\mathcal{Q}$, which is formed by the points that approximate a joint trajectory along the specified c-bundle chain $\mathcal{CBC}$ that satisfies the task trajectory.

### 4.3. Trajectory optimization

The final stage of the proposed method involves the optimization of the joint trajectory generated for each c-bundle chain. The optimization is performed by iteratively solving a constrained QP problem that moves each joint trajectory point along the SMM of the associated instant. Starting from an initial joint trajectory $\mathcal{Q}$, the aim is to minimize a cost function. In line with the popular methods for general purpose (not SMM-restricted) trajectory optimization [10,11], we employ a cost function that quantifies the sum of the squared velocities along the trajectory:

$$\mathbf{f}(\boldsymbol{\varphi}) = \frac{1}{2}\|\mathbf{V}\boldsymbol{\varphi} + \mathbf{e}\|^2 \tag{2}$$

where $\boldsymbol{\varphi}$ is a column vector that corresponds to the flattened joint trajectory $\mathcal{Q}$, $\mathbf{V}$ is the finite difference matrix that computes the velocities from the joint trajectory, and $\mathbf{e}$ is a vector that contributes with the invariant initial joint configuration:

$$\begin{aligned}\boldsymbol{\varphi} &= \left[q_{11}, q_{12}, \dots, q_{1n}, q_{21}, q_{22}, \dots, q_{2n}, \dots, q_{k1}, q_{k2}, \dots, q_{kn}\right]^{\mathrm{T}} \\ \mathbf{V} &= \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}_{k\times k} \otimes \mathbf{I}_{n\times n} \\ \mathbf{e} &= \begin{bmatrix} -\mathbf{q}_0^{\mathrm{T}} & \mathbf{0}_{n(k-1)\times 1}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}\end{aligned} \tag{3}$$

where $q_{ij}$ denotes the $j$th joint coordinate at the $i$th point in the joint trajectory, $\otimes$ denotes the Kronecker product, $k$ is the number of points in the joint trajectory $\mathcal{Q}$, $\mathbf{q}_0$ is the initial joint configuration, and the subscript $p\times q$ denotes the dimensions of the matrix to which it accompanies.

The parent problem (minimization of $\mathbf{f}(\boldsymbol{\varphi})$) is solved sequentially by a local approximation. This approach is analogous to other common optimization techniques, such as the ones employed in Sequential Quadratic Programming (SQP) or Sequential Convex Programming (SCP). A first-order Taylor expansion is used to locally approximate $\mathbf{f}(\boldsymbol{\varphi})$:

$$\mathbf{f}(\boldsymbol{\varphi}) \approx \mathbf{f}(\boldsymbol{\varphi}_{current}) + \nabla\mathbf{f}(\boldsymbol{\varphi}_{current})^{\mathrm{T}}(\boldsymbol{\varphi} - \boldsymbol{\varphi}_{current}) \tag{4}$$

where $\boldsymbol{\varphi}_{current}$ is the current joint trajectory, and $\nabla\mathbf{f}(\boldsymbol{\varphi}_{current})$ is the gradient of the cost function with respect to the joint trajectory, which is computed as $\nabla\mathbf{f}(\boldsymbol{\varphi}_{current}) = \mathbf{V}^{\mathrm{T}}(\mathbf{V}\boldsymbol{\varphi}_{current} + \mathbf{e})$.

The local approximation is then minimized by solving the following QP problem:

$$\underset{\Delta\boldsymbol{\varphi}}{\text{minimize}} \quad \frac{s}{2}\|\Delta\boldsymbol{\varphi}\|^2 + \nabla\mathbf{f}^{\mathrm{T}}\Delta\boldsymbol{\varphi} \tag{5}$$

where $s$ is a regularization constant that determines the penalization of large changes in $\Delta\boldsymbol{\varphi}$ [35]. We have set $s = 10$ in our experiments, as it provides a good balance between convergence speed and solution quality. Finally, the solution is added to the current joint trajectory following $\boldsymbol{\varphi} = \boldsymbol{\varphi}_{current} + \Delta\boldsymbol{\varphi}$.

The QP problem can be further constrained to ensure that, for each instant $i$, the joint coordinates $\mathbf{q}_i$ remain in the corresponding $r$-dimensional SMM, instead of allowing free movement in the $n$-dimensional joint space. This constraint reduces the number of decision variables from $nk$ to $rk$. The restriction is imposed following the equality:

$$\Delta\boldsymbol{\varphi} = \mathbf{N}\mathbf{S}^{\mathrm{T}}\,\mathbf{d} \tag{6}$$

where $\mathbf{d} = \left[d_{11}, d_{12}, \dots, d_{1r}, \dots, d_{k1}, \dots, d_{kr}\right]^{\mathrm{T}} = \left[\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_k\right]^{\mathrm{T}}$ is the $rk$-dimensional decision vector that signifies movement along the axes of the null-space basis of the Jacobian matrix at each joint configuration. These bases are contained in $\mathbf{NS}$ as follows:

$$\mathbf{NS} = \begin{bmatrix} \mathbf{NS}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{NS}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{NS}_k \end{bmatrix}_{rk\times nk} \tag{7}$$

where $\mathbf{NS}_i$, of size $r\times n$, is an orthonormal basis of the null-space of the Jacobian matrix $\mathbf{J}(\mathbf{q}_i)$ at the $i$th joint trajectory point $\mathbf{q}_i$. As depicted in Figs. 3 and 6, the null-space basis comprises $r$ linearly independent $n$-dimensional vectors that are tangent to the SMM at the joint configuration $\mathbf{q}_i$. Movement along the null-space basis approximates movement along the SMM. However, moving along tangent vectors does not guarantee that the trajectory points will remain on the SMM. To ensure that trajectory points precisely align with the SMM, a Newton-based correction is necessary, as discussed in Section 3.2.1. This step does not significantly impact computation, as the method typically converges in a single iteration, since the restrictions R1 and R2 that will be introduced in the final QP problem ensure that points do not deviate significantly from the current joint trajectory.

The final QP problem that is solved at each iteration is then:

$$\begin{aligned} \underset{\mathbf{d}}{\text{minimize}} \quad & \frac{s}{2}\|\mathbf{N}\mathbf{S}^{\mathrm{T}}\,\mathbf{d}\|^2 + \nabla\mathbf{f}^{\mathrm{T}}\,\mathbf{N}\mathbf{S}^{\mathrm{T}}\,\mathbf{d} \\ \text{subject to} \quad & \mathbf{d} \in \mathcal{A} \quad \text{(R1)} \\ & \mathbf{d} \in \mathcal{B} \quad \text{(R2)} \end{aligned} \tag{8}$$

and the solution is added to the current joint trajectory following $\boldsymbol{\varphi} = \boldsymbol{\varphi}_{current} + \mathbf{N}\mathbf{S}^{\mathrm{T}}\,\mathbf{d}$.

The restriction R1, which establishes a trust region around the current joint trajectory point, ensures the validity of the approximation in Eq. (4). This trust region is an $r$-dimensional hypercube centred at $\mathbf{q}_i$ and has its axes aligned with basis $\mathbf{NS}_i$. It is represented in Fig. 6 by its blue boundaries. This region also prevents excessive null-space movement, thus reducing the number of iterations required for the Newton-based correction required by Eq. (6). The side length of the hypercube is $2\lambda$. A smaller $\lambda$ enhances the accuracy of the cost function approximation in Eq. (4) and the validity of the null-space basis, tangent to the SMM, as an approximate movement along the SMM, but increases the number of iterations needed to minimize the parent problem $\mathbf{f}(\boldsymbol{\varphi})$. Conversely, a larger $\lambda$ reduces the number of iterations but compromises the approximation accuracy, leading to more iterations

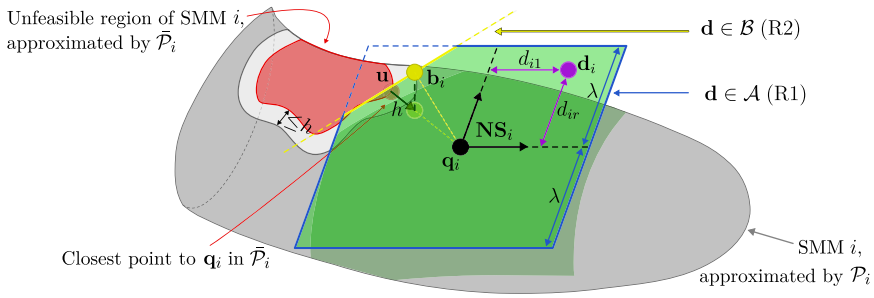


**Fig. 6.** QP problem restrictions in a hypothetical ($r = 2$)-dimensional SMM embedded in an ($n = 3$)-dimensional joint space. Trust region in blue (R1), SMM boundary in yellow (R2).

in the Newton-based correction and potential convergence issues, such as skipping over minima of $\mathbf{f}(\boldsymbol{\varphi})$. Our experiments indicate that a value of $\lambda = 0.1$ exhibits a good balance between accuracy and convergence speed. The trust region depicted in Fig. 6 is exaggerated for illustrative purposes.

The restriction R2 further constraints the movement of the joint trajectory points, to ensure that they remain within the boundaries of their associated SMM. Without this restriction, the points could potentially drift into unfeasible regions of the joint space (i.e., out of the SMM), such as those violating joint limits or causing collisions. A previous process must be performed in order to determine the boundaries of the SMM at the graph construction stage (Section 4.1). Specifically, during the sampling phase, any joint configurations determined as unfeasible (i.e., pertain to the SMM but produce collisions or joint limit violations) are stored in a point cloud $\bar{\mathcal{P}}_i$ that represents the invalid region of the SMM associated with the $i$th task trajectory point. Note that this process does not require additional computational effort, as these points are already computed during the graph construction stage, independently of this restriction, and the overhead of simply storing them is negligible.

In Fig. 6, the SMM and the unfeasible region of the SMM are represented in grey and red, respectively. Since we have used a small yet finite step in the discretization process, the actual boundary of the SMM is unknown and will lie somewhere within the white stripe, situated between the grey and red regions. Therefore, we can only rely on a conservative estimation of the boundary. For this purpose, we propose a simple three-step process to determine the conservative boundary point $\mathbf{b}_i$ for each joint trajectory point $\mathbf{q}_i$.

First, the nearest point in the unfeasible point cloud $\bar{\mathcal{P}}_i$ for each joint trajectory point $\mathbf{q}_i$ is queried, marked as a red point $\mathbf{u}$ in Fig. 6. Given that the maximum width of the white stripe is determined by the maximum distance between discretized points during the graph generation stage (denoted as $h$ in Section 4.1), point $\mathbf{u}$ will be at most a distance of $h$ from the actual SMM boundary. Note that the representation of $h$ has also been exaggerated in Fig. 6 for illustrative purposes. In practice, $h$ is a small value, and such stripe is extremely narrow. The second step is to shift point $\mathbf{u}$ towards $\mathbf{q}_i$ by a distance of $h$, and projecting the result to the $r$-dimensional subspace spanned by $\mathbf{NS}_i$ (green plane in Fig. 6) to obtain $\mathbf{b}_i$. This adjustment, although over-conservative in most cases, ensures that the joint trajectory points do not invade the forbidden regions $\bar{\mathcal{P}}$ of the SMMs. Lastly, the following inequality imposes restriction R2:

$$\mathbf{b}_i^{\mathrm{T}}\,\mathbf{d}_i \leq \|\mathbf{b}_i\|^2 \tag{9}$$

This inequality is obtained by linearizing an $r$-dimensional sphere that is centred at $\mathbf{q}_i$ and passes through $\mathbf{b}_i$, which produces a hyperplane in the subspace spanned by $\mathbf{NS}_i$, depicted in Fig. 6 in yellow. Note that such linearization is done to be able to express the restriction as a linear inequality, which is a requirement for QP problems.

The minimization of the cost function is achieved through an iterative process, as detailed in Algorithm 5. In every iteration, the restrictions R1 and R2 are computed first. Then, the QP problem in Eq. (8) is solved subject to R1 and R2, by employing a QP solver such as [36] or [37]. The solution is added to the current joint trajectory, and the process is repeated until the cost function converges. The convergence is dictated by the maximum magnitude of $\mathbf{d}$, which must be less than a small value $\varepsilon$ (e.g., a value of $10^{-3}$ is used for the experiments of this work).

**Algorithm 5** Trajectory optimization

```
while max(|d_ij|) > ε do
    Compute restrictions (R1) and (R2)
    d ← Solve QP problem in Eq. (8) subject to (R1) and (R2)
    φ ← φ + NS^T d
return φ
```

**Table 2**
Proposed variants, briefly summarized, and their sections for the different stages of the proposed framework.

| SMD and graph generation | Generation of raw joint trajectories | Trajectory optimization |
|---|---|---|
| **Sampling** (Section 4.1)<br>Sweeping every combination of $r$ joints, and solving the IKP for the remaining. | **Nearest Neighbour (NN)** (Section 4.2)<br>Querying nearest neighbours ($k$-d trees) to efficiently traverse the $\mathcal{CBC}$. | **Standard** (Section 4.3)<br>Completely optimizing (until convergence) every raw joint trajectory. |
| **Continuation** (Section 4.4.1)<br>[14,29] for computing the SMM at $\mathbf{x}(0)$. Task propagation, resampling and expansion for the subsequent SMMs. | **Pathfinding Algorithms (PA)** (Section 4.4.2)<br>Running pathfinding algorithms (e.g., A* or RRT) in the entire $\mathcal{CBC}$. | **Continuation-friendly** (Section 4.4.3)<br>Substituting the SMM boundaries restriction R2 by a joint-limits restriction (RV1) and a local approximation of collisions (RV2). |
| **Other** (Section 3.2.3)<br>Employing any method for SMM generation for every value of $\mathbf{x}(t)$. | **Hybrid NN-PA** (Section 4.4.2)<br>NN method to find the points in every **MP**, PA to establish the trajectory along every c-bundle. | **Online** (Section 4.4.3)<br>Simultaneous optimization and following of the trajectory. |

### 4.4. Variants

The proposed method can be adapted to satisfy different requirements in terms of computational speed, optimality or generalization. In this section, we present several variants for every stage of the presented framework, which can be combined to satisfy the specific requirements of the application. Furthermore, we discuss the selection of the most suitable variant for each stage, based on the application's requirements. Table 2 summarizes the proposed variants for each stage of the framework.

#### 4.4.1. SMD and graph generation

Even though any method presented in Section 3.2 can be employed to generate the SMMs in the graph construction stage, we have selected a *sampling*-based method due to its efficiency. However, our proposed method for SMD generation, extended from the method presented in [13] for SMM generation, requires solving the IKP by algebraic elimination for every combination of $r$ joint coordinates, which may not always be possible, especially in complex kinematic chains. In such cases, we propose employing a *continuation*-based method to generate the SMD, which is more computationally expensive, but is generally applicable to any kinematic chain.

One could simply adapt the algorithm presented in Algorithm 1, substituting the SampleSMMs and ClusterSMMs functions with any continuation-based method from the literature, and leaving the rest of the algorithm unchanged. Nevertheless, we present a more efficient alternative that leverages the Jacobian matrices that are computed in the continuation-based method, in order to propagate the already computed SMMs to generate the graph.

Starting from the initial joint configuration $\mathbf{q}_0$, conventional continuation methods, such as [14] ($r = 1$) or [29] (when $r > 1$), can be employed to compute the SMM at the initial task value $\mathbf{x}(0)$. Note that this will only compute the SMM to which the initial joint configuration belongs, and not every disjoint SMM. If the entire SMD is required, numerous seed joint configurations should be sampled to ensure that all disjoint SMMs are identified. However, as there is no guarantee that every existing disjoint SMM will be identified (Section 3.2.1), we recommend working with the initial SMM only, and expand the SMD from there. This is acceptable when using our method for single-query applications, as the other SMMs for $\mathbf{x}(0)$ that are not identified would belong to nodes (c-bundles) of the graph that will not be visited during the execution of the task because this would require traversing time backwards (e.g., like starting in c-bundle $C_2$ in Fig. 5(b), advancing in time to c-bundle $C_3$ in the same figure, and then going back in time to c-bundle $C_2'$).

Once the initial SMM has been generated, since the Jacobian matrices are already computed for every joint configuration sampled in the SMMs, the evolution of the SMMs over time can be efficiently calculated by solving:

$$\mathbf{q}_{i+1} = \mathbf{q}_i + \mathbf{J}^{\dagger}(\mathbf{q}_i)\,(\mathbf{x}(t + \Delta t) - \mathbf{x}(t)) \tag{10}$$

which effect propagating points from the SMM at $\mathbf{x}(t)$ to the SMM at $\mathbf{x}(t + \Delta t)$ is exemplified in Fig. 3(c) by the orange arrows.

However, similarly to the continuation method, the obtained points do not exactly belong to the SMM at $\mathbf{x}(t + \Delta t)$, as this approximation is only valid for infinitesimal displacements. A Newton-based correction is required to ensure that the points lie exactly on their SMM, as in Section 3.2.1. Fortunately, both of these computations can be accelerated by vectorizing the operation, which allows for the simultaneous calculation and subsequent correction of all points within the SMM, instead of performing these operations point by point.

Then, the resulting SMMs must be resampled to guarantee a dense representation within the graph (as shown in Fig. 3(c), when propagating each joint-space point $\mathbf{q}_i$ with Eq. (10), there may appear low-density regions in the SMM that must be filled with the depicted green points). Moreover, the boundaries of the SMMs should be expanded to account for the potential expansion of the SMMs from one instant to the next (again, as shown in 3(c), propagation with Eq. (10) may result in low-density regions that must be populated by the blue points to determine where the SMM ends due to entering a forbidden region). The process of resampling and boundary expansion is analogous to the continuation method under discussion.

The SMD is generated by applying Eq. (10) iteratively to every joint configuration $\mathbf{q}_i$ at every instant. This process yields the SMMs for the subsequent instant, which are then resampled, and their boundaries extended. Furthermore, the matching stage presented in Section 4.1 is not necessary, since the neighbourhood of each SMM is determined by the SMMs from which the propagation

originated. That is, the neighbour of SMM $\mathcal{M}_{i+1}$ is the SMM $\mathcal{M}_i$, which is the SMM containing the points $\{\mathbf{q}_i \in \mathcal{M}_i\}$ that were propagated using Eq. (10) to generate the points $\{\mathbf{q}_{i+1} \in \mathcal{M}_{i+1}\}$.

Note that Eq. (10) is not a direct replacement for the matching stage presented in Algorithm 3, as the latter operates without prior knowledge of the SMMs (i.e., $\mathbf{q}_i$ that lead to $\mathbf{q}_{i+1}$ are not known), and they both serve different purposes. In summary, the sampling method presented in Section 4.1 and the continuation method are not fully interchangeable, in the sense that one method does not perfectly substitute the other. Moreover, as said previously, continuation can only generate SMMs that are expanded from the current ones. If there are new manifolds that are disjoint from the current ones, they would not be generated by continuation, whereas the sampling method of Section 4.1 would generate them.

#### 4.4.2. Generation of raw joint trajectories

In the generation of raw joint trajectories, we introduced a method that, while not optimal, is significantly efficient. This method, which we will refer to as the *nearest neighbour* (NN) approach, leverages the same $k$-d trees used in the matching stage of graph construction to efficiently query the closest points in the SMMs. We opted for this method due to its efficiency, sacrificing the optimality of these trajectories, since it is not crucial at this stage, as they will be optimized in the subsequent stage. However, if the application, due to generalization demands, requires the use of the graph construction method discussed in Section 4.4.1, or a short window of time is allocated for the trajectory optimization stage, it may be more beneficial to invest the extra computational effort in generating the raw joint trajectories more optimally at this stage.

Since the discretized points that form the SMD can be easily converted into a graph, any classical *pathfinding algorithm* (PA) can be employed to generate the raw joint trajectories. The A* algorithm, for example, is a popular choice for path planning in graphs, and can be employed to generate the raw joint trajectories optimally. However, the computational cost of employing such algorithms is significantly higher than the nearest neighbour method, and may not be suitable for real-time applications. The RRT algorithm, on the contrary, is a more efficient alternative that can be employed to generate the raw joint trajectories in a more optimal manner than the nearest neighbour method, but with a lower computational cost than A*. While utilizing such pathfinding algorithms should be the most rigorous and general method for connecting a point in $t = 0$ to a point in $t = t_f$ (final time), the computational effort required for their generation may be prohibitive when planning on the entire $CBC$.

To mitigate this issue, we propose a *hybrid* combination of the two methods, employing the NN-based method to generate the points that belong to manifold portions that separate c-bundles of the c-bundle chain, and then employing a PA to generate the intermediate points along the c-bundles that form the c-bundle chain. This would mean the replacement of lines 7–13 in Algorithm 4 with the PA of choice (e.g., any RRT-based algorithm), which returns a path between the points $\mathbf{q}_a$ and $\mathbf{q}_b$. This approach would be a middle ground between the efficiency of the NN method and the optimality of the PA method, and would be particularly useful when the application required that the raw joint trajectories were near-optimal, but computational speed were also a concern.

#### 4.4.3. Trajectory optimization

The optimization of the raw joint trajectories is performed by iteratively solving a constrained QP problem that moves the trajectory along the SMD, but restricting every point to remain in its corresponding SMM. We will refer to the method presented in Section 4.3 as the *standard* optimization procedure, since it optimizes the entire joint trajectory until convergence. However, this method can be adapted to satisfy different requirements in terms of computational speed or generalization.

When selecting the continuation-based method presented in Section 4.4.1 as a more general variant of the graph construction, it will not be possible to store the point cloud $\bar{\mathcal{P}}$ of unfeasible joint configurations that represent the forbidden regions of the SMM during the graph construction stage, rendering the restriction R2 in the standard optimization stage not applicable. For such cases, we present the *continuation-friendly* variant of the method, in which the restriction R2 is replaced by two new restrictions, RV1 and RV2, that ensure that the joint trajectory points remain within the permitted boundaries of the SMM without the need for the point cloud $\bar{\mathcal{P}}$.

Boundaries of SMMs appear as a result of entering a region that does not satisfy the kinematic constraints of the robot, such as joint limits or collisions. Such boundaries can be approximated by two new restrictions. The first restriction RV1 ensures that the joint trajectory points do not violate the joint limits, and is imposed by the following inequalities at every $i$th trajectory point:

$$\begin{aligned} &\mathbf{q}_i + \mathbf{N}\mathbf{S}_i^{\mathrm{T}}\,\mathbf{d}_i \geq \mathbf{q}_{\min} \qquad \text{(RV1)} \\ &\mathbf{q}_i + \mathbf{N}\mathbf{S}_i^{\mathrm{T}}\,\mathbf{d}_i \leq \mathbf{q}_{\max} \end{aligned} \tag{11}$$

where $\mathbf{q}_{\min}$ and $\mathbf{q}_{\max}$ are the vectors that contain the lower and upper joint limits, respectively, and the inequalities are coordinate-wise.

The second restriction RV2 ensures that the joint trajectory points do not violate the collision constraints. It is based on a local approximation of the distance between the robot and the obstacles, as in [38]. In every iteration of the optimization process, the points $\mathbf{p}$ of the robot and $\mathbf{o}$ of the obstacle that are closest are queried. Then, the following restriction is imposed:

$$\mathbf{n}_i^{\mathrm{T}}\,\mathbf{J}_{\mathbf{p}_i}(\mathbf{q}_i)\,\mathbf{N}\mathbf{S}_i^{\mathrm{T}}\,\mathbf{d}_i \leq \|\overrightarrow{\mathbf{p}_i\mathbf{o}_i}\| - d_{\text{safe}} \quad \text{(RV2)} \tag{12}$$

where the subindex $i$ denotes the $i$th trajectory instant; $\mathbf{n}_i = \overrightarrow{\mathbf{p}_i\mathbf{o}_i}/\|\overrightarrow{\mathbf{p}_i\mathbf{o}_i}\|$ is the unit common normal from robot towards obstacle; $\mathbf{J}_{\mathbf{p}_i}(\mathbf{q}_i)$ is the Jacobian matrix that maps the joint velocities to the velocities of the point $\mathbf{p}_i$; and $d_{\text{safe}}$ is the safety distance that the robot must maintain with the obstacles, which is usually set to zero or a small distance. The QP problem in Eq. (8) should be updated to include these two new restrictions RV1 and RV2, replacing in practice restriction R2.

**Table 3**
Variant selection guidelines. Please check Table 2 for an overview of all variants and the sections in which they are defined.

| Main objective → | Global optimality of the solution | Real-time performance |
|---|---|---|
| $r = 1$ with collisions[a] | **Continuation**-based SMD computation<br>**Hybrid PA-NN** raw trajectory generation<br>**Continuation-friendly** + **standard** + **parallelized** optimization | **Continuation**-based SMD computation<br>**NN**-based raw trajectory generation<br>**Continuation-friendly** + **online** optimization |
| $r > 1$ (or $r = 1$ without collisions) | **Sampling**-based SMD computation<br>**Hybrid PA-NN** raw trajectory generation<br>**Standard** + **parallelized** optimization | **Sampling**-based SMD computation<br>**NN**-based raw trajectory generation<br>**Online** optimization |

[a] Or when solving Eq. (1) for the remaining values of $\mathbf{q}$ (via algebraic elimination) is not simple and continuation is the only option (complex kinematic chains).

Another variant is the *online* optimization, which is particularly useful when the application requires real-time operation. In contrast to the standard optimization, which optimizes the entire joint trajectory $\varphi$ until convergence, and then the robot starts moving, the online optimization method optimizes the joint trajectory concurrently with the robot's movement. The principle behind this variant is to optimize the trajectory as much as possible within each sampling period of the controller, interrupting Algorithm 5 at the end of the sampling period even before it has converged, and then apply the first partially-optimized joint increment to move the robot.

For example, assume that the robot's controller operates at a sample time of 50 ms, and a hypothetical raw joint trajectory $\varphi$ of 100 points must be optimized. The online optimization method would execute as many iterations of Algorithm 5 as possible in the 50 ms time window, obtaining a partially optimized trajectory $\varphi^*$. Then, the first joint increment $\mathbf{q}_1$ of $\varphi^*$ would be applied to move the robot, and $\mathbf{q}_1$ would be removed from $\varphi^*$, shortening the trajectory to 99 points. This process would be repeated in the following sampling periods, optimizing the shortened trajectory as much as possible within each period, applying the first sample to move the robot and removing it from the trajectory, until the final point is reached or the trajectory has been shortened so much that it can be completely optimized within the sampling period (i.e., when Algorithm 5 converges due to $\max\left(|d_{ij}|\right) \leq \varepsilon$).

Due to its online nature, the online optimization variant should be applied only to the raw trajectory with the lowest cost (instead of applying it to all raw trajectories). Therefore, it is advised to combine this method with the hybrid NN-PA method for the generation of raw joint trajectories, to ensure that the chosen raw trajectory is as close as possible to the globally optimal one.

#### 4.4.4. Variant selection guidelines

Given the high flexibility of this framework, users might be uncertain of which variant combination to select. In this subsection, we provide a set of guidelines to assist users in identifying the most suitable combinations for their specific applications.

The framework is composed of three stages: (1) SMD and graph generation, (2) search of raw joint trajectories, and (3) trajectory optimization. The paper has been structured by proposing a baseline approach for each one of the stages in Sections 4.1, 4.2, and 4.3, respectively. However, there exist situations where these baseline methodologies are not the most appropriate, or simply cannot be applied. For these cases, we have proposed alternative variants for each stage, in order to address different requirements (limited hardware resources, very complex kinematic chains, etc.). Some of these alternatives are interchangeable, while others are complementary. We trust that the explanation of the variants throughout the present section, and the examples shown below, will provide the necessary guidance for understanding how to combine them to achieve the desired results. In Table 3, we propose combinations of variants depending on the degrees of redundancy, the complexity of the kinematic chain, and the desired trade-off between computational cost and optimality.

In terms of generalization, when a complex kinematic chain is involved, which may not permit solving the IKP by algebraic elimination as the sampling method described in Section 4.1 requires, we recommend employing the continuation-based variant presented in Section 4.4.1, as it is directly applicable to any robot, regardless of the complexity of its kinematic chain, since it only requires the computation of the Jacobian matrix at each joint configuration.

The dimensionality of the SMMs and, consequently, the SMD, is influenced by the degrees of redundancy inherent in the task and robot. This dimensionality primarily impacts the graph generation stage. For a redundancy of $r = 1$, the SMMs within the SMD are one-dimensional manifolds. In this case, the original continuation-based method for SMM generation, presented by Burdick [14], can be utilized. However, for $r > 1$, the higher-dimensional continuation alternative, introduced in [29], becomes necessary. Despite its successful extension to higher-dimensional SMMs, this method is computationally demanding and unsuitable for real-time applications. In such scenarios, the sampling-based method for graph generation, which offers greater computational efficiency and scalability, is recommended. The sampling-based method is also recommended for redundancy $r = 1$ where collisions are not involved. This will be justified by means of the examples of Section 5.

Regarding the generation of raw joint trajectories, using the hybrid NN-PA approach reduces the time required for post-processing optimization, but it also increases the initial computation time. This trade-off makes hybrid NN-PA more suitable for scenarios where higher-quality final trajectories are preferred. These are the scenarios that benefit from using hybrid NN-PA: if trajectories are going to be fully optimized, hybrid NN-PA is more efficient overall. Otherwise, if a fast solution is needed, and the trajectories are not going to be fully optimized, NN is faster.

Regarding the balance between computational cost and optimality, the choice of which optimization stage to implement presents a trade-off between these two factors. At one end, when optimality is paramount and the application permits sufficient processing time, the standard optimization procedure delineated in Section 4.3 is recommended. This method identifies the globally optimal

joint trajectory in every c-bundle chain. Conversely, when computational efficiency is crucial and real-time operation is necessary, the online optimization variant discussed in Section 4.4.3 is advised. This approach enables an almost immediate commencement of movement following the graph generation stage, with the optimization process operating concurrently, refining the joint trajectory as the allocated time window permits. When a balance between computational cost and optimality is sought, a larger time window for the online optimization could be allocated, or the standard optimization could be conducted with a restricted number of iterations, gradually transitioning the objective of the optimization stage from optimizing to smoothing trajectories as the maximum number of iterations of Algorithm 5 allowed in a window decreases.

This compromise between computational cost and optimality is also present in other state-of-the-art methodologies for global redundancy resolution. For instance, the authors of [12] describe their algorithm as rather expensive in execution time. To solve this issue (and to extend it to higher task dimensionality), they propose the use of neural networks, which are always associated with some degree of approximation, thus compromising the optimality of the solution. On the contrary, works that prioritize the optimality of their solution, not compromising it for computational efficiency, such as [15,19], are computationally expensive. In our framework, we offer the user the possibility to prioritize either computational efficiency or optimality, or to find a balance between the two, by selecting the most suitable variant for each stage.

## 5. Experimental results

A series of simulations have been conducted for the evaluation of the proposed framework for global redundancy resolution. For evaluating the performance of the proposed method, we have employed the cost function defined in Eq. (2), and the runtime of the different stages of the framework, as assessment metrics. The experiments were run on a machine equipped with an Intel Core i7-9700F CPU and 32 GB of RAM, under the Ubuntu 24.04 operating system, using Python 3.11. FCL 0.7 [39] was employed for collision detection, while the QP solver used was ProxSuite 0.6.2 [36].

### 5.1. Example 1: 3R planar manipulator

The first experiment considered corresponds to a 3R planar manipulator tasked with following a 2-dimensional end-effector trajectory. Two ellipses were placed in the workspace, representing obstacles that the entire manipulator must avoid [$(x-0.5)^2/1^2 + (y-0.7)^2/0.25^2 \leq 1$ and $(x-1)^2/0.3^2 + (y+1)^2/0.2^2 \leq 1$]. Due to such demanding obstacle-avoidance constraints, the joint limits were set to $[-2\pi, 2\pi]$ radians for all joints. The task trajectory, corresponding to the end-effector position, was defined as a linear path from $(2.457, -0.793)$ to $(-2.4, 1.5)$ (in metres, relative to the base of the robot), to be completed in 5 s, with constant speed. The starting joint configuration was set to $\mathbf{q} = \left[\frac{\pi}{8}, -\frac{\pi}{4}, -\frac{\pi}{6}\right]^{\mathrm{T}}$ radians.

For this experiment, we used the continuation-based approach for the graph generation stage, which is the best choice when the kinematic equations of the robot are difficult to solve via algebraic elimination methods as the sampling method described in Section 4.1. Note that this is not necessary for the 3R manipulator, but we start with this choice so that it can be compared later with the sampling method. In addition, we seek to completely optimize the joint trajectories regardless of the computational cost.

The task trajectory was discretized into $k = 100$ points, resulting in a time step of $\Delta t = 0.05$ s, which provides sufficient resolution for the task analysis. A value of $\rho = 0.3$ was used for the continuation-based generation of the SMD.

Regarding optimality, we aimed to obtain the optimal trajectory for the task. Therefore, we selected the standard optimization method, which optimizes every joint trajectory until convergence. Having employed the continuation-based graph-generating method, which does not produce the point cloud $\bar{\mathcal{P}}$ of unfeasible joint configurations (i.e., R2 is not applicable), the restrictions R1, RV1, and RV2 were imposed in the QP problem, as discussed in Section 4.4.3. We selected the nearest-neighbour-based method for the generation of raw joint trajectories, since it is significantly more efficient than the pathfinding algorithms, and the quality of the raw joint trajectories is not crucial at this stage, since they will be optimized in optimization process.

Fig. 7(a–f) illustrate the results of the simulation in the form of snapshots of a video, which is available in the supplementary material (Video S3). They display six instants of the simulation, which correspond to the initial and final moments of the task trajectory ($t = 0$ and $t = 5$), as well as four intermediate instants, which correspond to every transition between two c-bundles ($t = 0.05$, $t = 0.9$, $t = 1.55$, and $t = 3.3$). Fig. 7(g) shows the resulting graph $\mathcal{G}$, where the nodes represent the c-bundles of the SMD, and the edges represent the connections between them. Four distinct c-bundle chains that have been found for the task are highlighted in the graph. For simplicity, c-bundle graphs have been represented throughout the paper as graphs with only topological information (connections between c-bundles), representing them as circles without timespan information. However, in this example, for clarity, we have faithfully represented the time instants at which each c-bundle starts and ends, so that it is clear to the reader why some c-bundles do not complete the desired task, due to vanishing before reaching the end time.

Each subfigure presents both the joint space $(q_1, q_2, q_3)$, at the top of Fig. 7(a–f), and the task space $(x, y)$, at the bottom. In the joint space, the optimized joint trajectories that traverse each c-bundle chain are represented along with the SMMs (dotted lines) at the corresponding instant, with the joint configuration of the optimized trajectory at each instant marked with an 'X'. In the task space, the task trajectory is represented in green, and the robot configuration when traversing each c-bundle chain is depicted.

Two different colour schemes are employed in Fig. 7. The first colouring is dictated by the nodes of the graph (c-bundles of the SMD). Every c-bundle is given a different colour in the graph, and the SMMs that are depicted in the joint space (in dotted lines) share the same colour as the c-bundle they pertain to. The other colour scheme corresponds to every obtained c-bundle chain, which are the wide lines marked in the graph. Both the optimal trajectory plotted in the joint space and the corresponding robot representation share the same colour as the c-bundle chain they traverse.

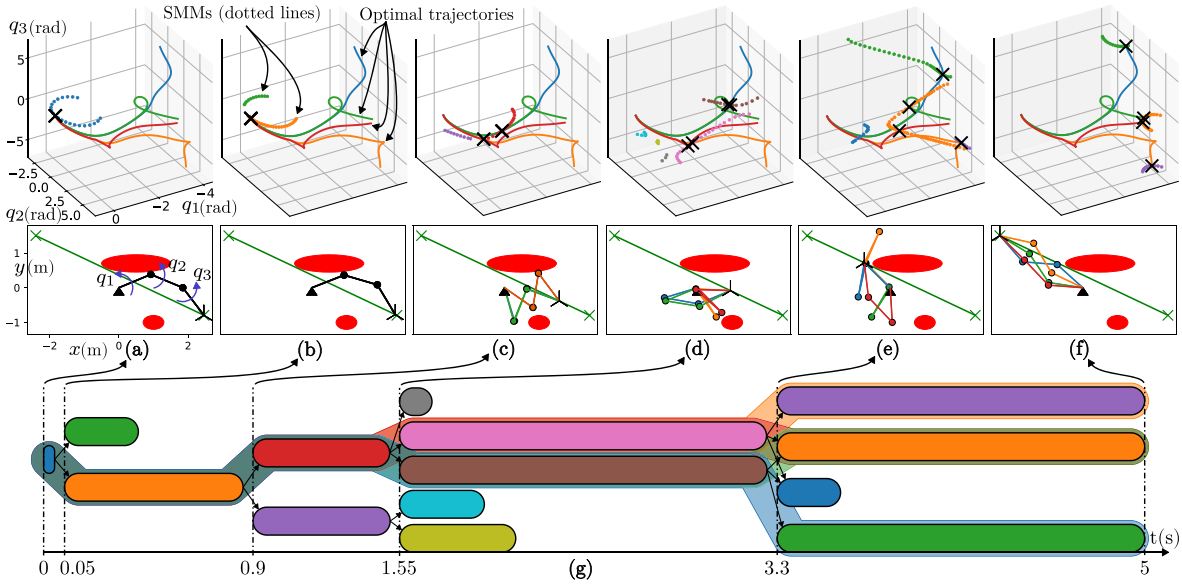


**Fig. 7.** (a–f) Snapshots of the simulation. (g) C-bundle graph $\mathcal{G}$ and every possible c-bundle chain.

**Table 4**
Computational times and costs of the generated trajectories for the 3R planar manipulator.

| **Graph generation** | **Continuation: 2092 ms** | | *Sampling: 2973* ms | |
|---|---|---|---|---|
| **Generation of raw trajectories** | **20 ms** | | | |
| *C-bundle chains calculation* | *1* ms | | | |
| | $\downarrow CBC_1$ (blue) | $\downarrow CBC_2$ (orange) | $\downarrow CBC_3$ (green) | $\downarrow CBC_4$ (red) |
| *Raw trajectories* | *4* ms | *5* ms | *4* ms | *6* ms |
| **Optimization (standard)** | **4056 ms** | **2509 ms** | **3715 ms** | **4010 ms** |
| **Trajectory cost** | **2.144** | **1.591** | **2.07** | **1.195** |

In Table 4, the computational times for each stage of the proposed framework are presented, along with the cost of the generated trajectories. The computational times are given in milliseconds, and the cost of the trajectories corresponds to the cost function in Eq. (2), which quantifies the sum of the squared joint velocities along each trajectory. The provided data corresponds to the average values obtained from 100 independent simulations.

Regarding the cost of the returned trajectories, the results show that the globally optimal joint trajectory corresponds to the c-bundle chain $CBC_4$ (depicted in red in Fig. 7(a–f)), with a cost of 1.195. The SMD is computed in a reasonable time considering that it characterizes the infinite number of solutions to the IKP for the entire task trajectory. In this experiment, the continuation-based method outperforms the sampling-based method. We believe that this is due to the significant overhead that computing whole-body collisions introduces in the sampling-based method, where such computation must be performed for every sampled point. On the contrary, the continuation-based method will stop checking for collisions as soon as the first collision is detected at each end of the manifold (if it is a curve ($r = 1$)), producing a significant reduction in the collision-checking time, which outweighs the additional computational cost of this continuation method (which involves the Jacobian's null-space computation and Newton correction). From the results, we can conclude that, for $r = 1$, when collisions are considered, the continuation-based method is the most efficient method for the generation of the SMD. However, for $r > 1$ (or when $r = 1$ and collisions are not considered), the sampling-based method will be more efficient, as we will demonstrate in the next experiments.

The generation of the raw joint trajectories is significantly fast, as the traversal of the graph is a simple process and the nearest-neighbour search is extremely efficient due to the $k$-d trees. The largest time consumption is observed in the optimization stage, as the QP problem must be iteratively solved for every derived joint trajectory. It is worth noting that this stage is a good candidate for parallelization, given that the CPU employed possesses enough cores, as the optimization of each joint trajectory is independent of the others. In practice, the computational time of the optimization stage could be reduced to the time required to optimize the most demanding joint trajectory, which, in the case of Table 4, corresponds to the trajectory through $CBC_1$, which results in an overall time of 6.2 s for the complete execution of the proposed framework.

Lastly, we have conducted an additional experiment to demonstrate the method's capability to handle link–link collisions. Specifically, we incorporated collision constraints between the first and third links, which correspond to the links that lie on the same plane in the real robot. The results of this experiment are presented in the supplementary material (Video S2). This capability was implemented by querying the closest points among the involved links, in addition to the closest points between the robot and

the obstacles, and imposing the restriction RV2 on the QP problem. This demonstrates that the collision formulation of this paper can equally handle self-collisions between different parts of the robot, or collisions between the robot and the environment.

### 5.2. Example 2: RPRRRRPR robot

The second experiment considered corresponds to a more complex scenario, where an RPRRRRPR serial robot is assigned with following a 6-dimensional end-effector trajectory, including both position and orientation. This kinematic chain models the HyReCRo robot, which is a biped climbing robot that moves by adhering one of its feet to a surface, and moves the other foot to a new location, adhering it to the surface again. The kinematic model and equations of the RPRRRRPR robot are detailed in [40].

Such a computationally demanding scenario ($n = 8$, $r = 2$) necessitates an efficient combination of variants in order to ensure that the computational time remains within acceptable limits. The sampling-based method for graph generation was selected, as it is the most efficient method for the generation of the SMD when both the DoF and redundancy degree increase. For the optimization stage, the online optimization variant was employed, as the computational time required for the entire optimization of every joint trajectory would suppose a significant overhead, although still manageable. We will only perform a single iteration of the online optimization between discrete instants, even though more iterations could be performed, thus a better trajectory could be obtained, but we want to showcase the worst-case scenario. For the trajectory optimization stage, we chose the hybrid combination of the nearest-neighbour-based method and a pathfinding algorithm. Specifically, we implemented a Depth-first search (DFS) algorithm to generate the points along the c-bundles that connect the co-regular surfaces. This selection results in a slightly more expensive raw trajectory generation stage, but the quality of the raw joint trajectories is significantly improved. Such a choice is done to increase the chances of picking the raw trajectory that will become the globally optimal one after applying the online optimization.

The free foot of the robot is tasked with following a trajectory that has been defined as a cubic spline that interpolates 9 waypoints in 5 s while avoiding an obstacle, with the orientation of the free foot being kept constant, considering joint limits as kinematic constraints [40]. Similarly to the previous experiment, the task trajectory was discretized into $k = 100$ points, resulting in a time step of $\Delta t = 0.05$ s. The starting joint configuration was arbitrarily set to $\mathbf{q} = [-0.055, 0.18, -0.055, -1.505, -3.076, 0.273, 0.187, 0.273]^{\mathrm{T}}$, where the revolute joints are measured in radians and the prismatic joints in metres. Joint steps of $\Delta q = 0.1$ radians were employed for the revolute joints, and $\Delta q = 0.05$ m for the prismatic joints, during the sampling-based generation of the SMD and graph.

Fig. 8(a–e) illustrate different moments of the simulation, while Fig. 8(f) displays the generated graph. The video of the animated simulation is also available in the supplementary material (Video S4). The included snapshots correspond to the initial and final moments of the task trajectory ($t = 0$ and $t = 5$), and three intermediate instants ($t = 2.3$, $t = 2.75$, and $t = 3.95$), which correspond to the transitions between c-bundles. Although the online optimization variant was selected, meaning that only the best raw trajectory is optimized, we showcase the optimized joint trajectory resulting from every c-bundle chain in the graph, for the sake of comparison. The same colour scheme as in the previous experiment is employed, i.e.: on the one hand, the nodes of the graph in Fig. 8(f) and the SMMs (in the top of Fig. 8(a–e)) share the same colour as the c-bundle they pertain to. On the other hand, the robot (in the bottom of Fig. 8(a–e)) and the optimized joint trajectory (in the top of Fig. 8(a–e)) share the same colour as the corresponding c-bundle chains of Fig. 8(f). Green and orange trajectories correspond to the fully optimized joint trajectories. Also, the top of Fig. 8(a–e) show the trajectory optimized following the online variant; this is represented as a blue continuous line that is almost indistinguishable from the fully optimized green trajectory (see the difference in the top of Fig. 8(d)). This demonstrates that the less computationally expensive online optimization yields results that are extremely close to the fully optimized trajectory.

It is worth noting that the plotted SMMs are ($r = 2$)-dimensional manifolds embedded in the ($n = 8$)-dimensional joint space, which is why they are represented as surfaces (approximated by point clouds) in the top part of Fig. 8(a–e). Since an 8-dimensional space cannot be visualized directly, we chose to project the SMMs onto the subspace of the first, fourth and sixth joints of this robot [40], which are denoted as $(\varphi_{1A}, \theta_A, \varphi_{2B})$ in Fig. 8(a–e). This is the reason why the blue and orange c-bundles, which are present at the initial instants, seem to be connected in Fig. 8(a), when in reality they are not connected in the full 8-dimensional joint space.

Table 5 presents the computational times for each stage of the proposed framework, along with the cost of the trajectory. Although the number of DoF and redundancy have been increased, it is remarkable that the computational time of the graph generation stage remains within the same order of magnitude as the previous experiment, thanks to the efficiency of the sampling-based method. Further examples of efficiency of sampling the SMMs are given in [41]. The online optimization of the best raw joint trajectory significantly reduces the computational time of the optimization stage, as only a single iteration of the QP problem is required to start the movement of the robot. Moreover, the time taken to perform an iteration remains under 30 ms, which allows for the online optimization of the trajectory within the 50 ms between successive time steps. By comparing in the last two rows of Table 5 the cost of the online-optimized trajectory with that of its completely optimized counterpart, it is evident that the online optimization variant produces a trajectory of slightly higher cost. However, the difference in cost is not significant, as can be observed in the joint-space trajectories in Fig. 8(d) (top), due to employing an efficient pathfinding algorithm to generate the raw joint trajectories.

This experiment demonstrates the efficiency of the sampling-based method for the graph generation in scenarios involving robots with higher DoF. As outlined in Section 4.1, the process of sweeping or scanning through every possible combination of $r$ joint coordinates corresponds to a number of scans equal to the binomial coefficient $\binom{n}{r}$. Consequently, the number of combinations scales exponentially with the increase in DoF $n$. A method lacking efficiency would fail to generate the SMD for high DoF robots within a reasonable timeframe. In the case at hand, the number of combinations corresponds to $\binom{8}{2} = 28$. Resolving the SMD for

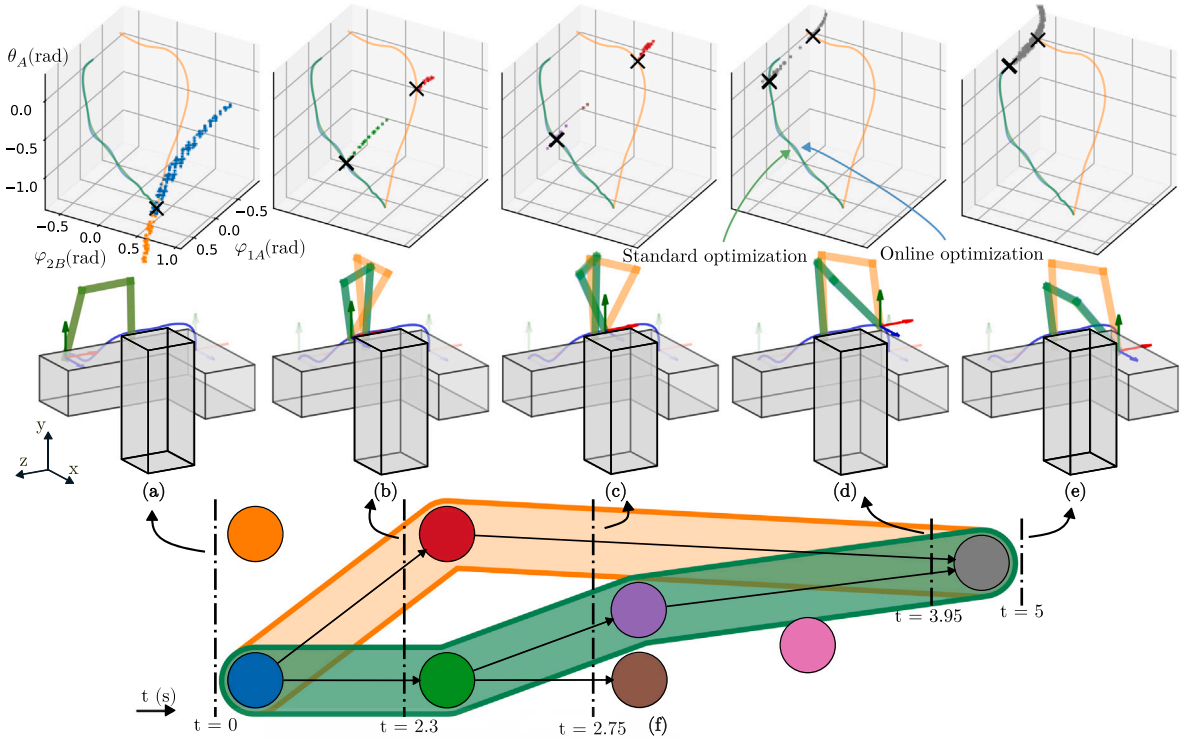


**Fig. 8.** (a–e) Snapshots of the simulation. (f) C-bundle graph $\mathcal{G}$ and every possible c-bundle chain.

**Table 5**
Computational times and costs of the generated trajectory for the RPRRRRPR robot.

| | |
|---|---|
| **Graph generation** | **3917 ms** |
| *Sampling* | *2957* ms |
| *Clustering* | *896* ms |
| *Matching* | *64* ms |
| **Generation of raw trajectories** | **16 ms** |
| *C-bundle chains calculation* | *1* ms |
| *Raw trajectory* | *15* ms |
| **Online optimization (average time per iteration)** | **26 ms** |
| *Standard optimization* | *6043* ms |
| **Trajectory cost (online optimization)** | **0.119** |
| *Trajectory cost (standard optimization)* | *0.111* |

an 8-DoF robot and an $r = 2$ redundancy degree in the same order of magnitude as the 3-DoF robot with redundancy $r = 1$ is a testament to the efficiency of the proposed method.

Nevertheless, some of such combinations of joint coordinates render the IKP unsolvable due to the fact that, when fixing the task $\mathbf{x}$, some joint coordinates become dependent in the kinematic equations of the robot. Specifically, the RPRRRRPR robot has 3 combinations that are unsolvable, yielding a total of 25 solvable combinations. When solving some of these 25 combinations using algebraic elimination methods, it is necessary to find the roots of polynomials of degrees higher than 4, which can increase the computational time of the graph generation stage. To mitigate this issue, these combinations were omitted from the SMD computation stage, resulting in a less densely populated SMD. However, thanks to the optimization stage, which does not rely on the previously sampled points of the SMD, the quality of the final trajectory is not affected by this omission. This is because, as we will demonstrate in Section 6.1, the paths converge to the same optimal solution independently of the density of the SMD or the initial raw trajectories. Specifically, in the results we have shown, we have omitted 10 combinations for solving the IKP, significantly reducing the computational time of the graph generation stage, while maintaining the quality of the final trajectory. A video of the simulation with every possible combination is available in the supplementary material as Video S5, replicating the simulation results presented in Fig. 8, where it can be observed that the SMD and extracted optimal joint trajectories are not affected by the omitted combinations. The only difference corresponds to a couple of very small c-bundles that are created and instantly vanish, but these can be considered noise, and do not affect the final result.

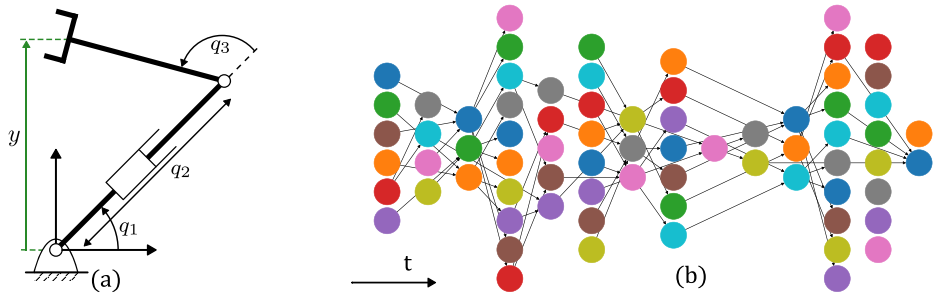


**Fig. 9.** (a) RPR planar manipulator. (b) Generated graph.

Another experiment was conducted in order to simulate a concave transition during the exploration of a vertical structure that the robot must climb. The starting pose of the robot remains the same as in the previous experiment, while the end pose corresponds to a pose located at a vertical beam that the robot must adhere to. The results are presented in the supplementary material in the form of an animation of the robot configuration and the extracted optimal joint trajectory (Video S6).

### 5.3. Example 3: RPR planar manipulator

This brief section introduces an additional experiment designed to demonstrate the exhaustive nature of the proposed method for global redundancy resolution. In scenarios with substantial constraints, such as those presented in the preceding experiments, the resultant graph is often sparse. This sparsity is due to the limited number of c-bundle chains that successfully complete the task, as the constraints imposed by obstacles and joint limits significantly reduce the feasible joint configurations, leading to the elimination of many previously feasible c-bundles. Conversely, in scenarios with less-restrictive constraints, the graph is denser, increasing the number of c-bundles in the SMD. This experiment aims to highlight the exhaustive capabilities of the proposed method, demonstrating its competence in identifying and managing every c-bundle and co-regular surface that arises from the task constraints.

The experiment consists of an RPR (Revolute-Prismatic-Revolute) planar manipulator (Fig. 9(a)) tasked with following a one-dimensional end-effector trajectory. The task trajectory is defined as a parabolic path of the $y$ coordinate of the end-effector, following the equation $y = -6.662t^2 + 8.162t - 1.5$, to be completed in 1 s ($t \in [0, 1]$). This results in degree of redundancy $r = 2$, i.e., SMMs are surfaces. To enhance the complexity and realism of the task, the end-effector is required to avoid an elliptic obstacle in the workspace, defined by the inequality $(x-1.1)^2/1^2 + (y+0.2)^2/0.25^2 \leq 1$. Every position coordinate is given in metres, relative to the base of the robot. The joint limits are set to $[-2\pi, 2\pi]$ radians for the revolute joints, and $[0, 0.5]$ m for the prismatic joint.

The resulting graph is presented in Fig. 9(b). The graph is significantly denser than the previous examples, as the constraints imposed by the task are less restrictive. The graph is composed of 72 nodes, each corresponding to a c-bundle, and 68 edges, which represent the connections between the c-bundles.

The rest of the simulation results are omitted, as the focus of this experiment is the graph itself, which demonstrates the capability of our method to correctly handle SMDs with highly dense and intricate c-bundle connections.

In addition, we have conducted a simulation experiment to assess the effectiveness of the vectorization of the computation of the SMD (Section 4.1). When employing Algorithm 2 sequentially to compute the SMMs at $y = -1.5$ m, the runtime is 82.909 ms. When vectorizing the inner loop of the algorithm, the runtime is reduced to 1.675 ms, which results in a speedup of 49.5 times. When extending this experiment to the entire SMD for the same task as in Fig. 9, the runtime is reduced from 10.614 s to 0.194 s, which results in a speedup of 54.7 times. From this, we can estimate that the vectorization of the SMD computation results in a speedup of approximately 50 times in this example.

## 6. Discussion

This section presents a discussion of some of the key aspects of the proposed framework, including its global optimality, complexity and dimensionality analysis, and comparisons with state-of-the-art methods.

### 6.1. Global optimality

In the context of redundancy resolution, the term *global optimality* refers to the capability of the method to identify the joint trajectory $\mathbf{q}(t)$ for which a given cost function results in the minimum value, from the infinite number of solutions that satisfy a given task trajectory and constraints.

In the proposed framework, we characterize the infinity of solutions by generating the SMD. Then, we identify the c-bundle chains that successfully complete the task in order to extract a joint trajectory for each chain. Considering that the SMD will certainly contain the globally optimal joint trajectory (since it characterizes the infinite set of possible solutions), and that we extract every

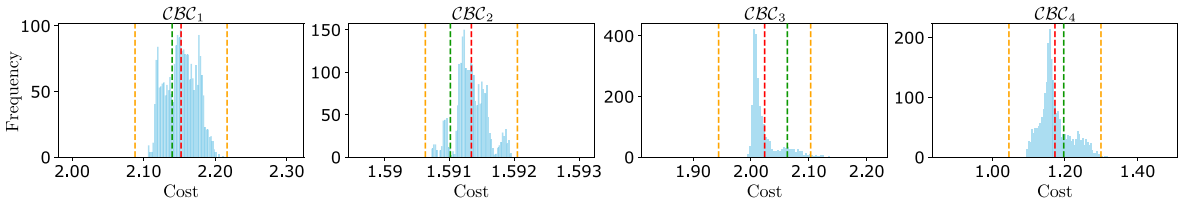


**Fig. 10.** Global optimality analysis.

possible c-bundle chain from the SMD, one should expect that the globally optimal joint trajectory will coincide with the optimal trajectory in one of the extracted c-bundle chains. While we are not able to provide a formal proof of the global optimality of the proposed method, we can offer a statistical demonstration of the optimality of the extracted joint trajectories.

Consider the experiment with the 3R planar manipulator presented in Section 5.1, where we extracted four different optimal joint trajectories, one for each c-bundle chain. To demonstrate optimized trajectories converge to the optimal solutions for each c-bundle chain, we conducted an additional experiment. Instead of generating the raw joint trajectory by any of the introduced methods, we generated completely random raw joint trajectories within each extracted c-bundle chain by modifying line 6 of Algorithm 4 by randomly selecting a point in $\mathbf{MP}_i$. This slight modification produces completely random raw joint trajectories, which are then optimized to compare the costs of the resulting trajectories. We conducted 2350 iterations of this experiment, and the histograms of the cost function values of the resulting trajectories for each c-bundle chain are presented in Fig. 10.

The mean cost function value is plotted as a vertical red line, while two orange lines represent three standard deviations from the mean. The results shown in Table 4 correspond to the vertical green lines. It can be observed that every optimized trajectory for each c-bundle chain converges to the same cost value within a small deviation, and that no significant outliers are present. The slight variations in the cost function values are due to the value selected for the convergence parameter $\epsilon$ in the optimization stage (Algorithm 5). We have selected a large-enough value in order to be able to visualize such distribution. This experiment indicates that the extracted joint trajectories converge to the optimal solutions for each c-bundle chain, which in turn demonstrates that the globally optimal joint trajectory is among the extracted trajectories. While this statistical demonstration does not provide a formal proof of the global optimality of the proposed method, it does offer a strong indication.

### 6.2. Complexity and dimensionality analysis

It is beneficial to analyse the complexity of the proposed algorithms. The first stage of our framework is the computation of the SMD, which is the most computationally intensive part of the process. In Algorithm 2, every combination of $r$ of the $n$ joint coordinates is swept, producing a number of sweeps equal to the binomial coefficient $\binom{n}{r}$. For each of these combinations, $N^r$ points are sampled (where $N$ is the number of points in which every joint coordinate is discretized), and for each point, $\mathbf{q}$ is solved from Eq. (1), obtaining a maximum of $a$ different solutions. If we consider the worst-case scenario over all combinations for generating the SMD, the computational complexity of the sampling-based method is $\mathcal{O}(k\binom{n}{r}N^r a)$, where $k$ is the number of points in which the task trajectory is discretized. The same complexity is present in the NN-based raw trajectory generation stage, as the complexity of Algorithm 4 is dictated by the nearest-neighbour search, whose worst-case complexity equals the maximum possible number of points in the SMD: $\mathcal{O}(k\binom{n}{r}N^r a)$. If the hybrid version of the raw trajectory generation stage is employed, the complexity is that of the employed PA, multiplied by the number of times it is executed, which equals to the number of c-bundles in the c-bundle chain.

It is clear that the computational complexity of the entire framework is dictated by, and grows exponentially with the redundancy degree $r$, and factorially with the increase in DoF $n$. In fact, few studies exist that address the global redundancy resolution problem for robots with a redundancy degree greater equal or greater than 2, as the curse of dimensionality renders the problem of identifying the global set of infinite solutions (e.g., SMMs or FMs) intractable in a reasonable timeframe. State-of-the-art methods for the generation of SMMs take from hundreds [28] ($r = 1$) to thousands of seconds [31] ($r = 2$) to generate the SMMs at a single task point. We have reduced the computational time of the SMM generation process to the order of milliseconds, even for robots with a redundancy degree of $r = 2$, as demonstrated in the experiments of Section 5.2, by using the proposed sampling-based method described in Section 4.1. Moreover, when working with robots with complex kinematic chains, whose kinematic equations cannot be solved by algebraic elimination, we have proposed a continuation-based method that is capable of generating the SMMs for $r = 1$ in the same order of magnitude of time as the sampling-based method, and we tested that, for example, for the RPRRRRPR robot experiment of Section 5.2, it is able to generate SMMs in about 1 s.

The computational optimization strategies that we have discussed throughout this work, such as vectorization of the SMD computation, efficient data structures like $k$-d trees, or parallelization among available CPU cores, are essential to implement the framework on resource-constrained robotic control systems.

However, for the moment, $r = 3$ seems to be out of reach for every method aspiring to completely map three-dimensional (or higher-dimensional) self-motion manifolds. A possible suboptimal approach to tackle $r \geq 3$ problems would be to virtually remove redundancy levels by fixing joints, and then solving the IKP for the remaining $r = 2$ redundancy degree. The same strategy could be applied for sudden joint failures where a joint becomes locked, as the already-computed SMD could be intersected with a hyperplane defined by $q_j = c$, where $q_j$ is the failed joint, and $c$ is the value at which it has been locked. This would avoid the recomputation of the SMD, but subsequent stages of obtaining the graph and solving trajectories would still be needed to adapt the solution to the new constraints.

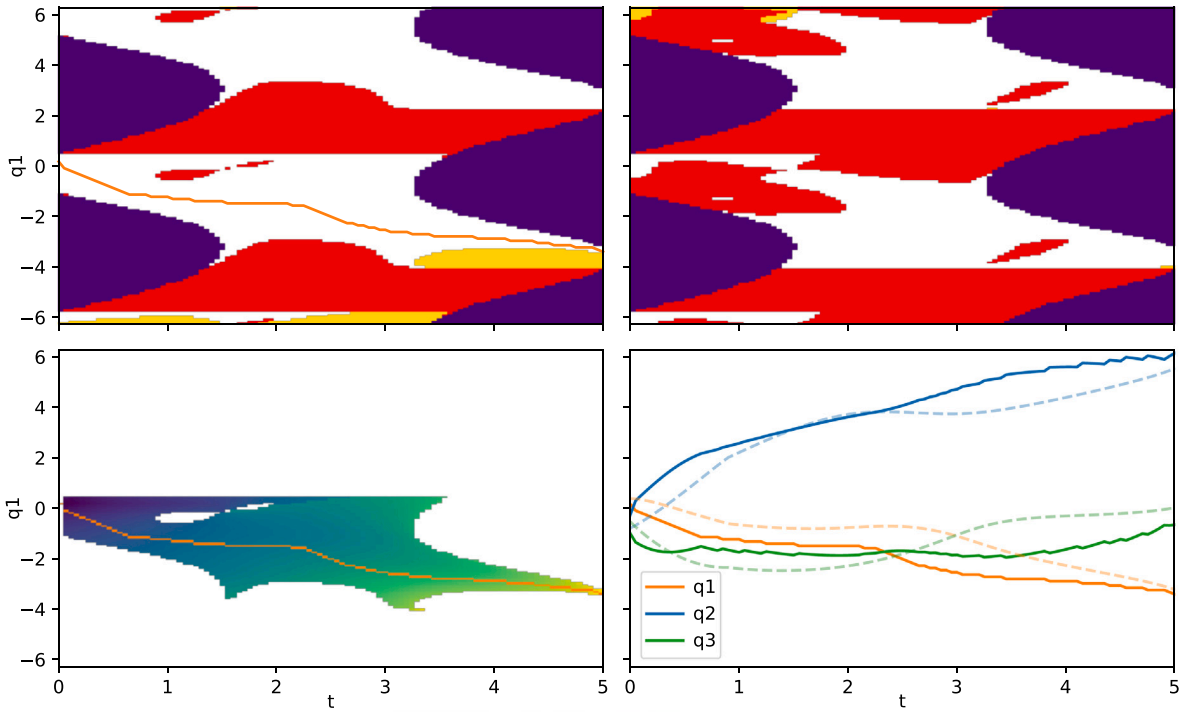


**Fig. 11.** Results of the experiment with the 3R planar manipulator using the method of Ferrentino and Chiacchio [19]. The top row shows the two feasibility maps, while the bottom row shows the cost map (left) and the output joint trajectories (right).

### 6.3. State-of-the-art comparison

Out of the state-of-the-art methods for global redundancy resolution reviewed in Section 2, few are directly comparable to the proposed method, since they usually study the redundancy for the whole workspace, or also plan the task trajectory. Ferrentino and Chiacchio [19], however, do solve the global redundancy resolution problem for a given task trajectory. In addition, the authors of [23] sketch, as a supplementary tool to their main contribution, an algorithm that builds the SMD and establishes paths across it. Since no publicly available repositories or benchmarks exist for a direct comparison, we have made every effort to replicate the methods described in these two studies faithfully. We replicated the scenario of the 3R planar manipulator of Section 5.1 with our implementation of the methods proposed in [19,23].

In [19], the authors resolve the redundancy by means of feasibility maps instead of SMMs. When applying the method of [19] to our example of Section 5.1, we chose $q_1$ as the redundant variable, and discretized its range of values $[-2\pi, 2\pi]$ into $N_u = 144$ points, which provides a resolution comparable to the one that we used in Section 5.1, whereas the time axis is discretized into $N_i = 100$ points, as in Section 5.1. This sets a scenario that is as fair as possible for comparison purposes. For the cost function, we adapted Eq. (2) to align with the formulation in [19], and set their function $l$ as the norm of the difference between each pair of tested joint configurations $\mathbf{q}$ (i.e., $l = \|\mathbf{q}(t_1) - \mathbf{q}(t_2)\|$). The obtained solution is given in Fig. 11, where the FMs are plotted in the top row. Purple regions indicate areas where configurations lead to complex solutions to the IKP, i.e., when solving $q_2$ and $q_3$ for each value of $q_1$ and $\mathbf{x}(t)$ (there are two possible solutions for $q_2$ and $q_3$ for each pair $(t, q_1)$, that is why there are two FMs in the first row of Fig. 11). Red regions represent collisions with the elliptical obstacles, Finally, uncoloured regions denote feasible configurations. The bottom-left subfigure of Fig. 11 shows the cost map of the FM that leads to the optimal solution, and is represented using the *viridis* colourmap, with dark purple indicating the lowest cost and yellow the highest. The optimal joint trajectories are plotted in the bottom-right part of Fig. 11, where the time history values of $q_1(t), q_2(t)$, and $q_3(t)$ obtained by our implementation of [19]'s algorithm are plotted in solid lines, while the solution provided by our method is plotted in dashed lines. The optimal trajectory of the redundant variable $q_1(t)$ is also plotted in the top-left FM and the bottom-left cost map.

Computing both FMs for the given task and constraints took an average of 6.474 s over 100 tests. Running our implementation of the algorithm described in Fig. 14 of [19] in the obtained FMs yielded an average runtime of 10.083 s. When applying Eq. (2) to the obtained solution, it resulted in a cost value of 1.44, which is comparable to the one returned by our method (1.195), if slightly larger. In fact, it can be visualized in the joint trajectories of the bottom-right subplot of Fig. 11 that both solutions share the same tendency, and approach the same optimal solution, as any small differences are likely attributable to resolution variations, implementation differences, or grid discretization. In terms of computational efficiency, both methods appear to share similar performances, as they are able to return the optimal solution in the same timeframe range, namely, around 16 s. For the FM

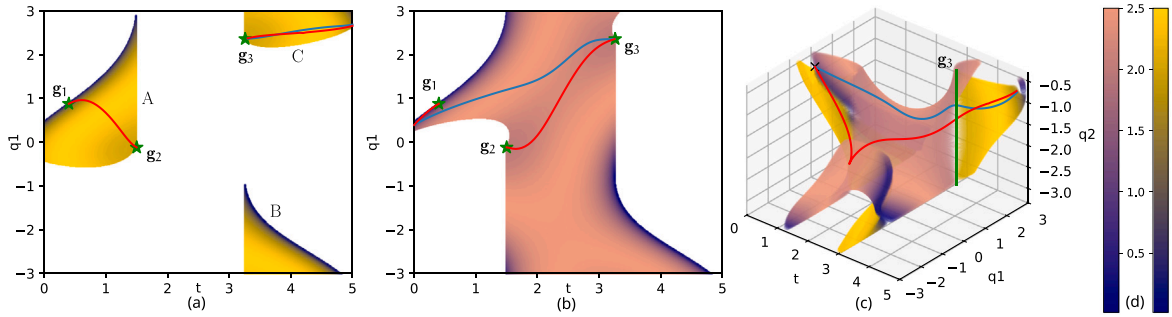


**Fig. 12.** (a): FM1. (b): FM2. (c): SMD projected onto the subspace $(t, q_1, q_2)$. (d): Colourmaps employed to plot the transition maps.

method, these 16 s are the sum of the time required to generate the FMs (6.474 s) and to explore the FMs for the optimal solution (10.083 s). For our proposed framework, these 16 s are the sum of all times recorded in Table 4, counting SMD generation via continuation (2.1 s), generation of all raw joint trajectories (negligible), and their optimization (4.1 + 2.5 + 3.7 + 4 s), assuming that all trajectories are optimized sequentially on a single core. However, if we consider that our optimization stage is parallelized, the runtime of our method is lowered to around 6 s, corresponding to the sum of SMD generation (2.1 s) and optimization of the most demanding raw trajectory (4.1 s). Modifying the resolution $N_u$ of our implementation of the method based on FMs in order to match these 6 s, it is able to provide a solution of cost 2.11 for similar timeframes.

In any case, while performance-wise both approaches seem similar, there exist some clear advantages of using the SMD for solving the redundancy resolution problem over FMs. Firstly, a notable distinction appears in how transitions between FMs are handled. [19] evaluates the possibility of performing transitions between FMs associated with different extended aspects, so that every possible feasible trajectory can be explored and potentially better solutions are not missed. Extended aspects are different solutions of the IKP when one solves the joint coordinates from the kinematic equations of the robot for known values of the task coordinates and redundant variables. In fact, FMs associated to different extended aspects are different branches of the projection of the SMD on a hyperplane $(t, \mathbf{q}_r)$, where $t$ is time and $\mathbf{q}_r$ are $r$ redundant coordinates (where $r$ is the degree of redundancy). For example: imagine a SMD that is a unit sphere $t^2 + q_1^2 + q_2^2 = 1$, where $t$ is time $(-1 < t < 1)$ and $(q_1, q_2)$ are joint coordinates. If we pick $q_1$ as the redundant coordinate $(\mathbf{q}_r = q_1)$ and project this sphere on plane $(t, q_1)$, we get two FMs associated to two different extended aspects: one projection due to one hemisphere $(q_2 > 0)$ and another projection due to the other hemisphere $(q_2 < 0)$. In the method proposed in [19], one would explore the plane $(t, q_1)$ for feasible trajectories and, in order to avoid missing potentially better solutions, it would require to explore both projections of this sphere, which can be connected at the projection of the equator, which are singularities of the projection. As it can be observed in Fig. 19 of Ref. [19], these transitions through singularities of the projection may produce small artificial spikes in the obtained trajectories, which should actually be smooth transitions because the robot is not going through kinematic singularities, only singularities of the projection of the SMD on the FMs. In contrast, our framework naturally handles transitions between extended aspects by considering the entire SMD, eliminating discontinuities and ensuring smoother joint trajectories. This is a direct result of our approach, which considers the global set of inverse solutions holistically, rather than partitioning them into projections based on extended aspects. Returning to the illustrative example of the sphere, our would method searches for optimal trajectories by moving directly on the sphere SMD, instead of working with its projections.

More importantly, even if we omit or somehow correct these spikes in the output trajectories, the method based on FMs is not robust to certain situations, as it may not find a solution even if one exists. For instance, take the same task and robot, but remove every obstacle, and simply constraint joint limits to $[-3, 3]$ for $q_1$ and $[-\pi, -0.5]$ for $q_2$ ($q_3$ remains unbounded). This produces the FMs depicted in Fig. 12(a-b), where the FM1 of Fig. 12(a) consists of three connected components $(A, B, C)$, whereas the FM2 of Fig. 12(b) consists of a single connected component. *Transition maps* [42] are plotted over these FMs following a colourmap, where dark blue indicates lower values (see Fig. 12(d)). These values correspond to the angular distance between the two joint configurations $\mathbf{q}$ that correspond to each pair $(t, q_1)$ (recall that each pair $(t, q_1)$ has two solutions for $(q_2, q_3)$, which produce two different $\mathbf{q}$). Values close to 0 (dark blue) indicate that transitions between extended aspects (and FMs) are possible through those points, which are singularities of the projection of the SMD onto plane $(t, q_1)$. We will refer to these 0-distance configurations as *gateways* between FMs. Note that each FM has been represented with a different colourmap in order to differentiate them when plotting the entire SMD (Fig. 12(c)), but they share the same dark blue colouring for angular distance values close to 0 (see how the lowest values of both colourmaps are similar in Fig. 12(d)).

If we rely on this angular distance to determine possible transitions between FMs, it would appear that no solution can be found when planning the trajectories using feasibility maps (but a trajectory is indeed possible, as shown by the blue and red trajectories, which were obtained by our proposed method). This happens because, if we start at $t = 0$ in the FM2 shown in Fig. 12(b), the only regions with a distance close to 0 (those that allow for a transition from FM2 to FM1) correspond to gateways that lead to components $A$ and $B$ of FM1, which impedes completion of the task since those components do not reach $t = 5$. On the contrary, if we were able to transition into component $C$ of FM1, the trajectory would be completed. Nevertheless, this desired transition seems

not possible when relying on angular distance to identify the mentioned gateways, as there do not exist points with values close to 0 (dark blue) that join FM2 with component $C$ of FM1.

However, as shown in Fig. 12(c), when visualizing the SMD projected onto the subspace $(t, q_1, q_2)$ ($q_3$ is omitted), we clearly see that the said transition is indeed possible through a vertical line of gateway points $\mathbf{g}_3$ that is represented as a green segment, therefore it is a singularity of the projection. This problem of FMs not detecting this gateway $\mathbf{g}_3$ arises because this vertical segment of singularities, which allows for the transition between FMs (both plotted in different colourmaps), projects onto a single point in the $(t, q_1)$ plane (represented as the rightmost green stars $\mathbf{g}_3$ in Fig. 12(a-b)), and this point will be missed with probability one when discretizing the $(t, q_1)$ plane into a grid for computing the FM and transition maps. In other words, detecting this singularity while sweeping $(t, q_1)$ is nearly impossible, as it would require that one of the points of the discretization grid coincides exactly with this singularity, which is extremely unlikely. That is the reason why attempting to find a feasible trajectory by means of FMs would fail in situations like this, because, as Fig. 12 shows, the colourmaps used to indicate angular distances do not suggest small values near this special singularity, unlike for other gateways marked in dark blue.

If we run our proposed framework for the same scenario, we obtain two trajectories that are plotted in blue and red in Fig. 12. The blue trajectory corresponds to the optimal solution, and the red trajectory corresponds to a suboptimal solution that suffers a complete wrapping in angle $q_3$. It can be observed how feasible joint trajectories that complete the task are actually possible, and perform the desired transition between extended aspects through the otherwise-missed gateway $\mathbf{g}_3$. The blue trajectory completes the task with a single transition between extended aspects through the said gateway $\mathbf{g}_3$, while the blue trajectory performs three transitions: the first one through the FM-detected gateway $\mathbf{g}_1$ that joins component $A$ of FM1 with FM2, and the other two through FM-missed gateways $\mathbf{g}_2$ and $\mathbf{g}_3$. A supplementary video material (Video S1) shows this globally optimal solution in motion (blue configuration), together with the secondary suboptimal solution detected by our method (in red). It also shows the SMD, and the animated sequence of SMMs, where the returned trajectories are plotted. Additionally, the c-bundle graph from which our method derives the feasible c-bundle chains is also represented. Note that both solutions correspond to the c-bundle chains formed by the succession of c-bundles coloured blue–purple and blue–red–lightblue in the said supplementary video, for the optimal and suboptimal solutions, respectively.

Another issue that arises is the sensitivity to the resolution (i.e., size of the discretization step) used to compute the FM. With [19]'s algorithm, it has been observed, both in the original paper and in our experiments, that the cost of the solution strongly depends on the resolution of the FM. In our case, while a fine-enough resolution is still required not to misinterpret disjoint SMMs as connected, the cost of the solution is not affected by the resolution of the SMD, as the optimization stage does not rely on the sampled points of the SMD, and will converge to the same solution independently of the raw trajectories produced by the sampled points of the SMD, as demonstrated in Section 6.1.

In addition to these advantages, it should be noted that the demonstration of the method of [19] was limited to examples with $r = 1$ degree of redundancy, whereas our examples in Sections 5.2 and 5.3 have demonstrated the effectiveness of our proposed framework for the more difficult case of $r = 2$ degrees of redundancy, which few research papers address.

Besides FMs, another method that is similar to our proposed framework is the one presented in [23], which proposes an algorithm to generate the SMD and establish paths across it, which we have replicated for comparison. Since the algorithm is probabilistically complete (i.e., given enough time it would densely generate the whole SMD), we had to stop it after some time running, which we set to a comparable time to what our method requires when parallelization at the optimization stage is not performed (around 16.5 s). The results are presented in Fig. 13(b). The sampled points of the SMD are plotted in blue, while the segments that represent connections between them are shown in green. We also plotted in orange the SMD obtained via continuation in the experiment of Section 5.1 for comparison. The optimal trajectory, whose median cost is 1.793 (versus our optimal cost of 1.195), is plotted in red. When parallelization is implemented at the optimization stage, our framework returns the optimized final trajectories in around 6 s. If [23]'s algorithm is stopped when reaching this runtime, the median cost of their solutions is raised to 1.967. An example solution for this configuration is shown in Fig. 13(a). While [23]'s algorithm for SMD generation was functional, its efficiency was significantly lower (particularly during the final stage, which involves connecting sampled points to form a complete path). For example, we run it again, but instead of stopping it when it has taken the same computation time as our method, we stop it when it achieves the same optimal cost as our method. In that case, it took an average of 40 s of runtime to return a solution of the same cost as our method, obtaining the result shown in Fig. 13(c). Also, we have encountered that the variance of the solution is quite high for different runs using the same parameters. For instance, it can be observed in Fig. 13(b), that the algorithm detects that an isolated region of SMD points (in the bottom-left part of the subfigure) is reachable, while in the other examples it does not. This may be due to the fact that this algorithm for SMD generation was not the primary focus of [23]'s work, but rather a supplementary tool supporting their main contribution: a continuous pseudoinversion of the forward kinematic map.

## 7. Conclusions and future work

In this paper, we have introduced a novel framework for global redundancy resolution. It is capable of determining the globally optimal utilization of additional degrees of freedom of a robot to execute a task. The framework minimizes a cost function while subject to kinematic constraints, such as joint limits and obstacles. It is composed of three stages: the generation of the SMD, the extraction of raw joint trajectories, and the optimization of said trajectories.

The proposed method is based on the concept of self-motion manifolds to compute the self-motion domain, which characterizes the infinite number of solutions to the inverse kinematics problem that satisfy the imposed constraints along a task trajectory. The self-motion domain is then spatially analysed to generate a graph whose nodes, the c-bundles, encapsulate self-motion manifolds that

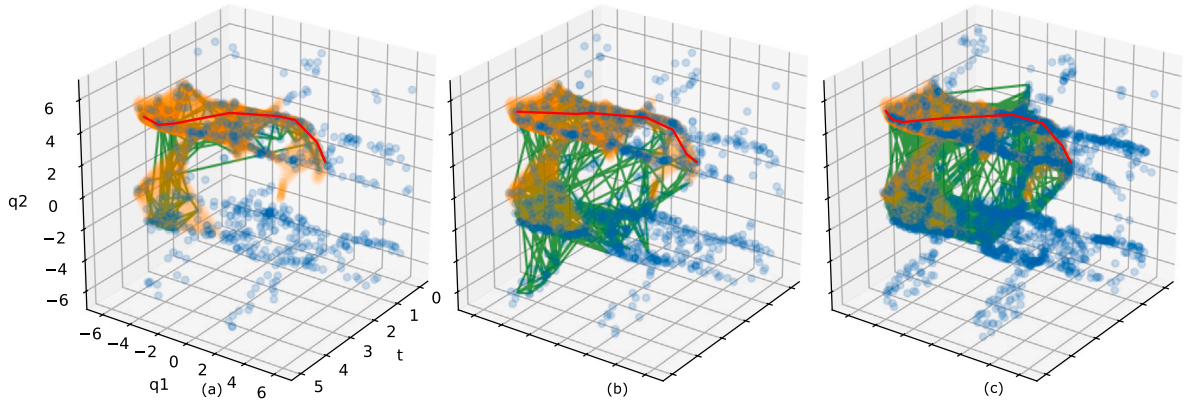


**Fig. 13.** Results of the method of Hauser and Emmons [23] for SMD generation, applied to the scenario of Section 5.1. (a): results of applying [23] taking the same computation time as our method when it parallelizes the optimization of raw trajectories. (b): results of applying [23] taking the same computation time as our method without parallelization. (c): results of applying [23] until obtaining the same optimal cost as our method.

share the same topology. The graph is subsequently traversed to identify preliminary joint trajectories that successfully complete the task. Finally, those joint trajectories are optimized to minimize a cost function.

The proposed framework is flexible, allowing for the selection of different variants for each stage, depending on the application requirements in terms of computational efficiency, optimality, complexity of the kinematic equations, or number of degrees of redundancy. Regarding the first stage of the algorithm, the SMD is advised to be generated using the proposed sampling method when the degree of redundancy is $r = 1$ if there are no collision constraints, or when $r > 1$ in any case (with or without collisions). On the contrary, the presented continuation-based method is more efficient for tracing 1-dimensional SMM when there exist collision constraints that severely reduce the length of the SMM to trace via continuation because some parts of these SMM invade regions of the joint space that produce collisions. The proposed continuation variant to generate the SMD is also suggested when the robot has complex kinematic chains that make Eq. (1) difficult to solve by algebraic elimination as the sampling method requires. The selection for the second stage of the algorithm mainly depends on the desired computational efficiency. If real-time performance is required, the NN-based method is recommended for searching for raw trajectories, as it is the fastest. Otherwise, if the globally optimal solution is desired, the hybrid NN-PA version of the method is advised, since this will produce raw trajectories that are more optimal and the last optimization stage of the proposed framework will converge faster to the globally optimal solution. A similar trade-off is present in the third stage of the algorithm, where the online optimization is recommended for real-time applications where a decent (yet not necessarily globally optimal) feasible trajectory is sought, while the standard optimization is advised when searching for the globally optimal solution.

The proposed method has been evaluated through a series of simulations, which demonstrate the performance of the method in different scenarios, combining different variants of the framework, degrees of redundancy, degrees of freedom, and task constraints. However, for solving problems with a redundancy degree greater than 2, the processing time exceeds the order of magnitude that can be considered acceptable for real-time applications.

Computing SMMs employing the continuation variant we propose in Section 4.4.1 can be computationally expensive, especially for degrees of redundancy higher than 1. In fact, when our framework has to rely on continuation for SMM generation, it is mainly because attempting to solve the kinematic constraints that define the SMMs via algebraic elimination (as in Section 4.1) may become too complicated due to the complexity of the kinematic chains of the robot. If continuation becomes too computationally expensive, and pure algebraic elimination is intractable, we suggest hybrid approaches like the algebraic-numerical method based on constraint curves that we proposed in [43]. Currently, we are working on extensions of these hybrid methods to address limitations of purely algebraic methods and continuation methods.

Additional future work will focus on the development of efficient methods for the generation of self-motion manifolds for degrees of redundancy greater than 2. Also, an efficient method for the spatial analysis of the c-bundles that compose the SMD, which currently is done by the clustering and matching stages, would be beneficial for the computational efficiency of the graph generation stage, as these two stages take an important fraction of the overall time of the method. Real experiments with physical robots could also be conducted to validate the proposed framework in a real-world scenario.

## CRediT authorship contribution statement

**Marc Fabregat-Jaén:** Writing – review & editing, Writing – original draft, Visualization, Software, Resources, Methodology, Investigation, Data curation, Conceptualization. **Adrián Peidró:** Writing – review & editing, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Matteo Colombo:** Methodology, Investigation, Conceptualization. **Paolo Rocco:** Writing – review & editing, Validation, Supervision, Resources, Investigation. **Óscar Reinoso:** Writing – review & editing, Supervision, Resources, Project administration, Investigation, Funding acquisition.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

The authors would like to thank Dr. Josep M. Porta for his insight on the implementation of the higher-dimensional continuation method for the generation of manifolds. This work was supported by the MCIN/AEI/10.13039/501100011033 and the ESF+ [PID2020-116418RB-I00, PRE2021-099226].

## Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.mechmachtheory.2025.106020.

## Data availability

Data will be made available on request.

## References

[1] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, Force Control, Springer, 2009.
[2] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, Int. J. Robot. Res. 5 (1) (1986) 90–98.
[3] D.E. Whitney, Resolved motion rate control of manipulators and human prostheses, IEEE Trans. Man-Mach. Syst. 10 (2) (1969) 47–53.
[4] A. Liegeois, et al., Automatic supervisory control of the configuration and behavior of multibody mechanisms, IEEE Trans. Syst. Man Cybern. 7 (12) (1977) 868–871.
[5] S.B. Slotine, B. Siciliano, A general framework for managing multiple tasks in highly redundant robotic systems, in: Proceeding of 5th International Conference on Advanced Robotics, Vol. 2, 1991, pp. 1211–1216.
[6] F. Flacco, A. De Luca, O. Khatib, Control of redundant robots under hard joint constraints: Saturation in the null space, IEEE Trans. Robot. 31 (3) (2015) 637–654.
[7] A.M. Zanchettin, P. Rocco, Motion planning for robotic manipulators using robust constrained control, Control Eng. Pract. 59 (2017) 127–136.
[8] M. Faroni, M. Beschi, N. Pedrocchi, A. Visioli, Predictive inverse kinematics for redundant manipulators with task scaling and kinematic constraints, IEEE Trans. Robot. 35 (1) (2018) 278–285.
[9] A. Mavrommati, C. Osorio, R.G. Valenti, A. Rajhans, P.J. Mosterman, An application of model predictive control to reactive motion planning of robot manipulators, in: 2021 IEEE 17th International Conference on Automation Science and Engineering, CASE, IEEE, 2021, pp. 915–920.
[10] N. Ratliff, M. Zucker, J.A. Bagnell, S. Srinivasa, CHOMP: Gradient optimization techniques for efficient motion planning, in: 2009 IEEE International Conference on Robotics and Automation, IEEE, 2009, pp. 489–494.
[11] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, S. Schaal, STOMP: Stochastic trajectory optimization for motion planning, in: 2011 IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 4569–4574.
[12] A. Albu-Schäffer, A. Sachtler, Redundancy resolution at position level, IEEE Trans. Robot. (2023).
[13] A. Peidro, O. Reinoso, A. Gil, J.M. Marín, L. Paya, A method based on the vanishing of self-motion manifolds to determine the collision-free workspace of redundant robots, Mech. Mach. Theory 128 (2018) 84–109.
[14] J.W. Burdick, On the inverse kinematics of redundant manipulators: Characterization of the self-motion manifolds, in: Advanced Robotics: 1989: Proceedings of the 4th International Conference on Advanced Robotics Columbus, Ohio, June 13–15, 1989, Springer, 1989, pp. 25–34.
[15] C.L. Lück, Self-motion representation and global path planning optimization for redundant manipulators through topology-based discretization, J. Intell. Robot. Syst. 19 (1997) 23–38.
[16] Z. Zhou, J. Zhao, Z. Zhang, X. Li, Motion planning method of redundant dual-chain manipulator with multiple constraints, J. Intell. Robot. Syst. 108 (4) (2023) 69.
[17] P. Wenger, P. Chedmail, F. Reynier, A global analysis of following trajectories by redundant manipulators in the presence of obstacles, in: [1993] Proceedings IEEE International Conference on Robotics and Automation, IEEE, 1993, pp. 901–906.
[18] J.A. Pámanes G, P. Wenger, J.L. Zapata D, Motion planning of redundant manipulators for specified trajectory tasks, Adv. Robot. Kinematics: Theory Appl. (2002) 203–212.
[19] E. Ferrentino, P. Chiacchio, On the optimal resolution of inverse kinematics for redundant manipulators using a topological analysis, J. Mech. Robot. 12 (3) (2020) 031002.
[20] E.J. Haug, Redundant serial manipulator inverse position kinematics and dynamics, J. Mech. Robot. 16 (8) (2024) 081008.
[21] A. Peidro, E.J. Haug, Obstacle avoidance in operational configuration space kinematic control of redundant serial manipulators, Machines 12 (1) (2023) 10.
[22] M. Fabregat-Jaen, A. Peidro, A. Gil, D. Valiente, O. Reinoso, Exploring feasibility maps for trajectory planning of redundant manipulators using RRT, in: 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation, ETFA, IEEE, 2023, pp. 1–8.
[23] K. Hauser, S. Emmons, Global redundancy resolution via continuous pseudoinversion of the forward kinematic map, IEEE Trans. Autom. Sci. Eng. 15 (3) (2018) 932–944.
[24] S. LaValle, Rapidly-exploring random trees: A new tool for path planning, Res. Rep. 9811 (1998).
[25] D. Berenson, S.S. Srinivasa, D. Ferguson, J.J. Kuffner, Manipulation planning on constraint manifolds, in: 2009 IEEE International Conference on Robotics and Automation, IEEE, 2009, pp. 625–632.
[26] L. Jaillet, J.M. Porta, Path planning under kinematic constraints by rapidly exploring manifolds, IEEE Trans. Robot. 29 (1) (2012) 105–117.
[27] L. Jaillet, J.M. Porta, Asymptotically-Optimal Path Planning on Manifolds, in: Robotics: Science and Systems VIII, The MIT Press, 2013, pp. 145–152.
[28] Y. Yang, Y. Wu, J. Pan, An interval branch-and-bound-based inverse kinemetics algorithm towards global optimal redundancy resolution, 2021, arXiv preprint arXiv:2104.12183.
[29] M.E. Henderson, Multiple parameter continuation: Computing implicitly defined k-manifolds, Int. J. Bifurc. Chaos 12 (03) (2002) 451–476.

[30] D. DeMers, K. Kreutz-Delgado, Canonically parameterized families of inverse kinematic functions for redundant manipulators, in: Proceedings of the 1994 IEEE International Conference on Robotics and Automation, IEEE, 1994, pp. 1881–1886.
[31] T. Wu, J. Zhao, B. Xie, A novel method for computing self-motion manifolds, Mech. Mach. Theory 179 (2023) 105121.
[32] I. Banfield, H. Rodríguez, Generation of the self-motion manifolds of a functionally redundant robot using multi-objective optimization, in: Robotics for Sustainable Future: CLAWAR 2021 24, Springer, 2022, pp. 438–452.
[33] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise, in: Kdd, Vol. 96, 1996, pp. 226–231.
[34] R. Sedgewick, Algorithms in c, part 5: Graph Algorithms, third ed., Addison-Wesley Professional, 2001.
[35] S. Boyd, L. Vandenberghe, Convex optimization, Cambridge University Press, 2004.
[36] A. Bambade, S. El-Kazdadi, A. Taylor, J. Carpentier, Prox-qp: Yet another quadratic programming solver for robotics and beyond, in: RSS 2022-Robotics: Science and Systems, 2022.
[37] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, S. Boyd, OSQP: An operator splitting solver for quadratic programs, Math. Program. Comput. 12 (4) (2020) 637–672.
[38] J. Schulman, J. Ho, A.X. Lee, I. Awwal, H. Bradlow, P. Abbeel, Finding locally optimal, collision-free trajectories with sequential convex optimization., in: Robotics: Science and Systems, Vol. 9, Berlin, Germany, 2013, pp. 1–10.
[39] J. Pan, S. Chitta, D. Manocha, FCL: A general purpose library for collision and proximity queries, in: 2012 IEEE International Conference on Robotics and Automation, IEEE, 2012, pp. 3859–3866.
[40] A. Peidró, A. Gil, J.M. Marín, Y. Berenguer, L. Payá, O. Reinoso, Monte-Carlo workspace calculation of a serial-parallel biped robot, in: Robot 2015: Second Iberian Robotics Conference: Advances in Robotics, Volume 2, Springer, 2016, pp. 157–169.
[41] A. Peidró, M. Fabregat-Jaén, A. Gil, D. Valiente, Ó. Reinoso, Teaching redundancy resolution in redundant parallel manipulators with an interactive and graphical simulation, in: INTED2024 Proceedings, IATED, 2024, pp. 2677–2686.
[42] E. Ferrentino, P. Chiacchio, A topological approach to globally-optimal redundancy resolution with dynamic programming, in: ROMANSY 22–Robot Design, Dynamics and Control: Proceedings of the 22nd CISM IFToMM Symposium, June 25-28, 2018, Rennes, France, Springer, 2018, pp. 77–85.
[43] A. Peidró, L. Payá, S. Cebollada, V. Román, Ó. Reinoso, Solution of the forward kinematics of parallel robots based on constraint curves, in: International Conference on Informatics in Control, Automation and Robotics, Springer, 2020, pp. 386–409.

Revista Iberoamericana de Automática e Informática Industrial

Revista Iberoamericana de Automática e Informática Industrial 21 (2024) 262-273

www.revista-riai.org

# Planificación jerárquica de movimientos de un robot trepador bípedo en estructuras tridimensionales reticulares

Marc Fabregat-Jaén[a,*], Adrián Peidró[a], Paula Mollá-Santamaría[a], Francisco José Soler[a], Óscar Reinoso[a,b]

[a] *Instituto de Investigación en Ingeniería de Elche (I3E), Universidad Miguel Hernández, Avda. de la Universidad s/n, Edificio Innova, 03202, Elche, España.*
[b] *ValgrAI: Valencian Graduate School and Research Network of Artificial Intelligence, Camí de Vera s/n, Edificio 3Q, 46022 Valencia, España.*

**To cite this article:** Fabregat-Jaén, M., Peidró, A., Mollá-Santamaría, P., Soler, F.J., Reinoso, Ó. 2024. Hierarchical motion planning of a biped climbing robot in three-dimensional truss structures. Revista Iberoamericana de Automática e Informática Industrial 21, 262-273. https://doi.org/10.4995/riai.2024.20779

**Resumen**

Los robots trepadores deben ser capaces de navegar autónomamente estructuras tridimensionales reticulares para evitar que operarios humanos se expongan a riesgos significativos al realizar tareas de mantenimiento en tales escenarios. Este artículo presenta un algoritmo de planificación jerárquica de movimientos para robots trepadores bípedos. A diferencia de las técnicas convencionales, nuestro algoritmo descompone el problema global en varios subproblemas, cada uno dedicado a gestionar aspectos específicos del proceso de generar una secuencia de puntos de adhesión. De forma inicial, se planifica la ruta global, que incluye la secuencia de caras que se atravesarán para alcanzar el punto designado, y qué puntos de transición se emplearán para cambiar de una cara a otra de la secuencia. Posteriormente, se calcula el camino que deberá recorrer el robot a lo largo de cada una de las caras que conforman la ruta global. Para la validación del método presentado, se incluyen imágenes y vídeo en un entorno de simulación.

*Palabras clave:* Planificación de trayectorias, Robots trepadores, Robots redundantes, Estructuras reticulares, Espacio de trabajo

**Hierarchical motion planning of a biped climbing robot in three-dimensional truss structures**

**Abstract**

Climbing robots must be capable of autonomously navigating three-dimensional truss-like structures to prevent human operators from being exposed to significant physical risks when performing maintenance tasks in such environments. This article presents a hierarchical motion planning algorithm for biped climbing robots. Unlike other conventional techniques, our algorithm decomposes the global three-dimensional problem into multiple sub-problems, each one dedicated to managing specific aspects of the process of generating the sequence of footholds. Initially, the global route is planned, which includes the sequence of faces to be traversed to reach the designated point, identifying which transition points will be used for changing from one face to another in the sequence. Subsequently, the path that the robot must follow along each of the faces comprising the global route is calculated. For the validation of the presented method, video and images taken in a simulation environment are included.

*Keywords:* Path planning, Climbing robots, Redundant robots, Truss structures, Workspace

## 1. Introducción

Toda construcción precisa de supervisiones rutinarias y tareas de conservación y mantenimiento. Estas acciones suelen desempeñarlas trabajadores humanos en ubicaciones de considerable altura, poniendo en riesgo su integridad física en el transcurso. La finalidad principal de los robots escaladores es reemplazar la ejecución de dichas labores altamente arriesgadas en ubicaciones que son intrínsecamente peligrosas para los seres humanos, o directamente inaccesibles. En la literatura se han propuesto robots trepadores para diversas aplicaciones, como el ajuste de contenedores de metal (Chen et al., 2019), la inspección de puentes metálicos (Nguyen and La, 2021), de cascos

*Autor para correspondencia: mfabregat@umh.es.

de barcos (Huang et al., 2017), y de grietas (Jang et al., 2020).

Para ejecutar estas tareas, los robots deben tener la capacidad de explorar escenarios, a menudo complejos, en tres dimensiones. Dependiendo de cómo se exploran estas estructuras, se pueden clasificar dos tipos de locomoción (Tavakoli et al., 2011). En primer lugar, los robots escaladores de movimiento continuo utilizan ruedas o carriles para desplazarse por el entorno, resultando en robots sencillos y rápidos (Chen et al., 2019; Nguyen and La, 2021; Huang et al., 2017).

Por otra parte, los robots trepadores de movimiento paso a paso constan de patas, en cuyos extremos se ubican garras, las cuales están unidas mediante cadenas cinemáticas de varios grados de libertad (GDL), que les dota de mayor maniobrabilidad para evitar obstáculos a costa de una mayor lentitud. Este artículo se enfoca en robots trepadores bípedos, que son aquellos que poseen dos garras. Durante su movimiento, a cada paso, uno de los extremos o garras permanece fijo en la estructura; mientras que la otra garra, que está libre, se mueve actuando la mencionada cadena cinemática hasta colocarla en una pose deseada. Al finalizar el movimiento, el extremo libre se pega al plano trepado, liberando la garra previamente fija. Este intercambio de roles permite llevar a cabo un nuevo paso. Algunos ejemplos de robots trepadores de estructuras de movimiento paso a paso son: HyReCRo (Peidró et al., 2019), CROC (Yang et al., 2016), W-Climbot (Zhu et al., 2020), 3DCLIMBER (Tavakoli et al., 2011), y ROMA 2 (Gimenez et al., 2002).

Considerando que los robots paso a paso trepan por estructuras metálicas reticulares, formadas por diversas vigas y barras que tienen una sección transversal considerablemente menor que su longitud, los mecanismos de adhesión que suelen usar estos robots para adherirse a dichas vigas son preferentemente de tres tipos: agarre prensil mecánico (Gimenez et al., 2002; Tavakoli et al., 2011), con el que se usan garras formadas por varios dedos que abrazan dichas vigas; adhesión magnética (Yang et al., 2016; Peidró et al., 2019), con el que se utilizan electroimanes o imanes permanentes que se pegan a las vigas gracias a su ferromagnetismo; o succión neumática (Zhu et al., 2020; Prados et al., 2023), con el que se emplean bombas de vacío o efecto Venturi para adherirse a las superficies. La solución que adopta el robot HyReCRo es la adhesión magnética, que tiene la ventaja de requerir el acceso a las vigas de la estructura solo por una de sus caras, en lugar de requerir abrazar dichas vigas por varias de sus caras de forma simultánea, como el agarre prensil mecánico. El algoritmo de planificación de trayectorias que presentamos en este artículo, por tanto, asume que las garras del robot requieren acceder solo a una de las caras de las vigas de la estructura para pegarse a ellas, quedando fuera del alcance del algoritmo la sujeción mediante garras prensiles.

Para ejecutar trabajos en un escenario reticular tridimensional, los robots trepadores necesitan la capacidad de navegarla autónomamente. El problema de planificación de movimientos consiste en calcular una secuencia de posiciones y orientaciones (poses), que permitan al robot alcanzar un punto deseado en la estructura, a partir de un punto de origen. Las soluciones convencionales al problema, como las propuestas para rovers, coches autónomos o robots aspiradores, están formuladas para planificar movimientos en un entorno bidimensional. No obstante, los robots trepadores añaden una nueva dimensión al trabajar en entornos verticales tridimensionales. Consecuentemente, se deben explorar espacios de búsqueda mayores, y el tiempo de cómputo que conllevan estas técnicas de planificación de trayectorias clásicas se vuelve prohibitivamente alto. Aunque la literatura ha presentado soluciones convincentes para la planificación de movimientos de robots trepadores de movimiento continuo (Breitenmoser and Siegwart, 2012; Stumm et al., 2012; Fang et al., 2020), la planificación de movimientos para robots de movimiento paso a paso sigue siendo una cuestión abierta. Estos robots requieren de una serie discreta de puntos de adhesión en vez de una trayectoria continuo, por lo que las técnicas mencionadas resultan inadecuadas.

Un reducido grupo de investigadores ha presentado soluciones al desafiante problema que supone la planificación de movimientos en robots trepadores paso a paso. El trabajo de Yang et al. (2016), propone obtener una ruta utilizando el método de Levenberg-Marquardt para obtener la solución de un sistema no lineal compuesto de funciones de coste sigmoidales, con el objetivo controlar la pose de la garra libre (efector final), mientras se evitan colisiones y se respetan límites articulares. Por otro lado, en (Chen et al., 2016) se propone una solución, basada en representar el escenario por medio de voxels, para resolver el problema con un algoritmo multicapa basado en A*. Este algoritmo utiliza hasta cuatro capas secuenciales en caso de que la capa anterior falle o caiga en un óptimo local. El enfoque del algoritmo A* también se implementa en el trabajo de Quin et al. (2016), donde se genera un grafo de *footholds* válidos y, mediante una función heurística, se evalúa cada nodo a partir de la información del escenario tomada por sensores. En una línea análoga, en (Pagano and Liu, 2017), se construye un árbol basándose en el entorno captado por los sensores del robot. Aunando los dos enfoques, Zhu et al. (2020) presentan un algoritmo multifase. En la primera fase, se determinan los puntos de transición entre las caras de la estructura mediante la optimización de un sistema no lineal, similar al enfoque de (Yang et al., 2016). En la segunda fase, se calcula la trayectoria en cada cara determinando el próximo punto de adhesión de acuerdo a la distancia a una ruta obtenida previamente con el algoritmo A*.

Este artículo presenta un algoritmo de planificación de movimientos jerárquico de un robot trepador bípedo de movimiento paso a paso para estructuras reticulares. El algoritmo consta de dos planificadores principales: el Planificador de Puntos de Transición (PPT), que, combinando el algoritmo A* con técnicas de inteligencia artificial, determina la secuencia global óptima de caras de la estructura que deben recorrerse, así como los puntos de las caras que permiten al robot pasar de una cara a la siguiente; y el Planificador de Puntos Intermedios (PPI), que determina la secuencia de puntos en los que las garras del robot deben pegarse para recorrer cada una de las caras. El planificador PPI, que opera con los puntos generados por el planificador PPT, agrega un nivel adicional a la jerarquía de la solución presentada, ya que planifica primero, de forma aproximada, un camino continuo (PPIC) dentro de cada cara; y, a continuación, refina dicho camino continuo mediante un algoritmo novel que determina el camino discreto (PPID): es decir, la secuencia de puntos discretos a los que deben ir pegándose las garras del robot para recorrer la cara. La solución presentada bebe de la idea general de fragmentar el problema global tridimensional en subproblemas en superficies bidimensionales, expuesta en (Zhu et al., 2020) . Sin embargo, los métodos utilizados para

abordar el problema presente en cada nivel de la jerarquía del planificador son distintos a los utilizados en el citado artículo. Gracias a ello, nuestra solución es aplicable a robots con una cadena cinemática más compleja, como es el caso del robot HyReCRo estudiado en este artículo.

La estructura del artículo restante está organizada de la siguiente manera. El robot HyReCRo se presenta en la Sección 2, así como el modelado y tratamiento previo de la estructura por donde se moverá el robot. Posteriormente, la Sección 3 presenta el planificador de movimientos, con los diferentes niveles de la jerarquía del algoritmo. Una simulación que valida el correcto funcionamiento del método propuesto se muestra en la Sección 4. Por último, en la Sección 5, se extraen las conclusiones del artículo y se plantean futuras líneas de investigación.

## 2. Un robot trepador de estructuras bípedo

### 2.1. Robot HyReCRo

El algoritmo propuesto a lo largo del artículo ha sido estudiado utilizando el prototipo del robot HyReCRo mostrado en la Figura 1(a). Este es un robot bípedo trepador de movimiento paso a paso, con el fin de realizar tareas de conservación e inspección en estructuras tridimensionales reticulares metálicas. El robot se compone de dos patas conectadas en serie mediante una cadera, utilizando articulaciones rotacionales ($\theta_j$ en la Figura 1(b)). La cinemática de la pata genérica $j$ se muestra en la Figura 1(b). Cada pata está configurada por dos mecanismos paralelos de 2 GDL idénticos, cuyas longitudes se modifican por medio de 4 actuadores lineales ($l_{1j}, r_{1j}, l_{2j}$ y $r_{2j}$ en la Figura 1(b)), y cada pareja de mecanismos comparte un eslabón central común (cuerpo central).

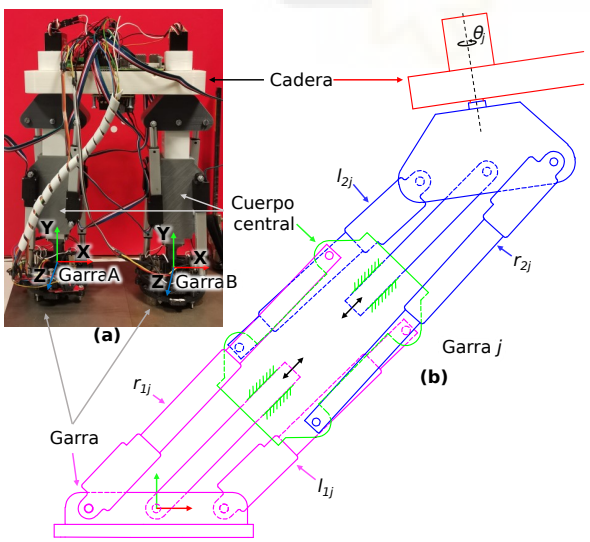


Figura 1: (a) Prototipo del robot HyReCRo. (b) Cinemática de la pata $j$.

Como resultado, el robot es de arquitectura híbrida serie-paralela, y posee 10 GDL, lo que conlleva una alta redundancia cinemática. El robot utiliza dos garras magnéticas ubicadas en ambos extremos de la cadena cinemática para trepar por estructuras tridimensionales reticulares metálicas (Peidró et al., 2019). Para automatizar el proceso de pegado y dotar al robot de la autonomía que precisa, en la garra se encuentran instalados tres sensores de distancia. En (Fabregat-Jaén et al., 2022), se presentó un algoritmo para el control cinemático del pegado, que utiliza la información de los sensores para identificar la superficie trepada y controlar la aproximación a la misma.

### 2.2. Notación y modelado de la estructura

Los robots trepadores deben desplazarse a través de escenarios tridimensionales, tales como los resultantes de estructuras reticulares presentes en los esqueletos de numerosas naves industriales, puentes o torres de distribución eléctrica. Son escenarios sumamente estructurados, compuestos de vigas interconectadas. Cada viga se puede describir como el conjunto de las caras planas que la forman. Cada una de estas caras $F$ se modela con un polígono $\mathcal{P}$, el cual se describe por la posición de sus vértices en el espacio tridimensional. Además, cada cara lleva asociada un vector unitario $\mathbf{n}$, normal al plano de dicha cara, que apunta hacia el lado que es accesible por el robot (Zhu et al., 2020):

$$F = (\mathcal{P}, \mathbf{n}) \tag{1}$$

donde

$$\mathcal{P} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n_v} \mid n_v \geq 3\} \tag{2}$$

y $\mathbf{v}_n$ es cada uno de los $n_v$ vértices coplanares del polígono.

Por lo tanto, una estructura reticular tridimensional $\mathcal{E}$ se modela como una colección de caras, que representa los elementos que componen la estructura:

$$\mathcal{E} = \{F_1, F_2, \ldots, F_{n_f}\} \tag{3}$$

donde $n_f$ es el número de caras que forman la estructura.

Todos los vectores están formulados relativos a un sistema de referencia inercial, que se encuentra fijado en un punto de la estructura, y que referenciaremos como "sistema de referencia del mundo" a lo largo del artículo.

### 2.3. Preprocesamiento de la estructura

Para planificar de manera eficiente una trayectoria utilizando el método que proponemos, es esencial obtener los puntos de transición posibles en todas las caras de la estructura. Una pareja de puntos de transición se define como dos *footholds* (o puntos de pegado), cada uno perteneciente a distintas caras, donde el robot fijaría cada uno de sus extremos para llevar a cabo la transición entre ellas. Como paso previo, cada polígono $\mathcal{P}$, que define cada cara de la estructura, se encoge como se ilustra en la Figura 2c, generando una versión escalada del polígono $\mathcal{P}$ distanciada de éste una distancia $d$. De este modo, para cada cara se genera un primer polígono o perímetro encogido tomando $d = d_{cncv}$ (polígonos azules a trazos en la Figura 2), y un segundo polígono encogido tomando $d = d_{cnvx}$ (polígonos verdes a trazos en la Figura 2). $d_{cncv}$ y $d_{cnvx}$ son distancias, medidas desde el borde de la cara, a las que se puede colocar el centro de la garra del robot para permitir que éste realice transiciones cóncavas (entre caras de distintas vigas) y convexas (entre caras de una misma viga), respectivamente, a la vez que se garantiza que las garras apoyan completamente en las caras sin quedar parcialmente al aire. La elección de estas dos distancias $d_{cnvx}$ y $d_{cncv}$ se ha realizado calculando el espacio de trabajo del robot a orientación constante como se describe en (Peidró et al., 2016), con las orientaciones necesarias para realizar transiciones con-

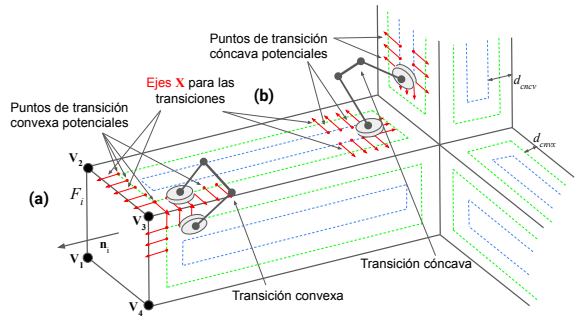


Figura 2: (a) Representación de la cara $F_i$ del entorno, formada por el polígono $\mathcal{P}_i$, compuesto por los vértices $\mathbf{v}_i$ y el vector $\mathbf{n}_i$, normal a la cara. (b) Puntos de transición potenciales, obtenidos encogiendo los perímetros de las caras para establecer las transiciones cóncavas (puntos en azul) y convexas (puntos en verde). (c) Preceso de encogimiento de los polígonos.

vexas y cóncavas respectivamente, y tomando distancias que coloquen a la garra libre en puntos de dichos espacios de trabajo que, además, estén sobre la nueva cara.

Ambos perímetros reducidos se discretizan mediante un muestreo con una resolución especificada (cuanto más alta la resolución, mayor cantidad de puntos de transición se evaluarán posteriormente), proporcionando las posiciones potenciales de transición. Finalmente, a cada una de las posiciones se le asigna un vector (que define la orientación), el cual supone la dirección en la que el eje **X** de la garra del robot apuntaría al pegarse para iniciar la transición. Este vector de orientación, dibujado en rojo en la Figura 2, se dirigirá hacia el borde más próximo, considerando el borde como la intersección de los dos planos de las caras entre las cuales el robot llevaría a cabo la transición.

Un punto de pegado (o *foothold*) es la posición y orientación a las cuales se pega cada una de las garras del robot. Para efectuar un desplazamiento, el robot debe llevar a cabo un paso, durante el cual desplaza su extremo libre hasta la pose indicada (*foothold* deseado). Estas poses se pueden definir con una matriz de transformación homogénea $4 \times 4$, la cual implícitamente incorpora los valores de una traslación y una rotación respecto a un sistema de coordenadas:

$$\mathbf{FH} = \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

donde **x**, **y** y **z** son los ejes del sistema de referencia de la garra en el punto de pegado (**X**, **Y**, **Z** en la Figura 1); y **p** es la posición del *foothold* respecto al mundo.

Cada matriz determina una pose de pegado sobre una de las caras de la estructura, la cual siempre llevará un vector normal asociado (1). Es por ello que uno de los ejes que describen la orientación del *foothold* se conocerá en todo momento, debido a que estará asociado, por definición, a una cara, cuyo vector normal es conocido. En el caso del robot HyReCRo, cuando se encuentra pegado a un plano, el eje **Y** del extremo pegado coincidirá el vector normal al plano **n**. En consecuencia, mientras se expresen respecto al mismo sistema de referencia, el vector **y** del *foothold* coincidirá con el vector **n**, normal a la cara.

Así pues, los *footholds* potenciales de la estructura estarán completamente definidos, puesto que se conoce su posición **p**, resultado del proceso de discretización del perímetro encogido de cada cara; el vector **x**, que señala el borde de la transición; **y**, que corresponde al vector normal **n**; y el vector **z**, resultado del producto vectorial entre **x** y **y**.

## 3. Algoritmo de planificación de movimientos jerárquico

En esta sección, se propone un algoritmo jerárquico para la resolución del problema de planificación de movimientos tridimensional para un robot trepador de estructuras bípedo de movimiento paso a paso. Dados un punto de pegado inicial $\mathbf{FH}_i$ y un punto de pegado final $\mathbf{FH}_f$, el algoritmo genera una secuencia de *footholds* $\mathcal{FH}$ que posibilitarán al robot trepar desde el origen hasta el punto final, a través de la estructura. La jerarquía del método y sus niveles están ilustrados en el esquema de la Figura 3.

El nivel superior divide el problema en dos planificadores diferentes. Primero, el Planificador de Puntos de Transición (PPT), presentado en la Sección 3.1, obtiene la trayectoria global tridimensional; es decir, los *footholds* necesarios para alcanzar el objetivo (moverse desde $\mathbf{FH}_i$ hasta $\mathbf{FH}_f$). Cada una de estas transiciones necesarias entre caras viene definida por dos puntos. Por ejemplo: ${}^a\mathbf{FH}_{dep}$ y ${}^b\mathbf{FH}_{arr}$ indican los puntos de pegado de la garra fija y la garra libre, respectivamente, que permiten al robot cambiar desde la cara $F_a$ hasta la cara $F_b$. Para ejecutar este planificador es necesario el procesar previamente la estructura de la forma que se ha mostrado en la Sección 2.3, con el fin de calcular los puntos de transición potenciales en todas las caras.

El segundo planificador (PPI, Sección 3.2), a partir de cada par de puntos devueltos por el planificador global PPT, obtiene los puntos de pegado intermedios necesarios para moverse a lo largo de la cara. Esto significa que, dados un punto de entrada $\mathbf{FH}_{arr}$ y otro de salida $\mathbf{FH}_{dep}$ de una cara, el planificador PPI calculará la serie de *footholds* que permitirá al robot llegar desde $\mathbf{FH}_{arr}$ hasta $\mathbf{FH}_{dep}$ a través de la cara. En el esquema mostrado, se puede observar que serán necesarios $n$ planificadores PPI, donde $n$ es el número de caras por las que pasa el camino global devuelto por el primer planificador PPT.

El planificador PPI añade otro nivel a la jerarquía al estar dividido en dos subniveles. En primer lugar, el planificador PPIC (Sección 3.2.1), obtiene la trayectoria continua más corta que une la pareja de puntos de transición devuelta por el primer planificador del primer nivel (PPT). Seguidamente, el planificador PPID, presentado en la Sección 3.2.2, utiliza dicha trayectoria continua para calcular la secuencia discreta de puntos de pegado que usará el robot para atravesar la cara en cuestión (desde $\mathbf{FH}_{arr}$ hasta $\mathbf{FH}_{dep}$). La unión de las salidas de cada uno de estos $n$ bloques resulta en la secuencia de puntos de pegado $\mathcal{FH}$ que conectan el punto inicial $\mathbf{FH}_i$ con el final $\mathbf{FH}_f$ a través de la estructura.

### 3.1. *Planificador de Puntos de Transición (PPT)*

La finalidad de este planificador es producir la serie de puntos de transición que posibilite al robot llegar desde el punto inicial hasta el destino. Ambos puntos (origen y destino) representan poses, que incluyen posiciones y orientaciones, ubicadas generalmente en caras de diferentes elementos de la estructura. De este modo, se aborda el problema de manera integral, obteniendo en primer lugar las transiciones necesarias entre caras,

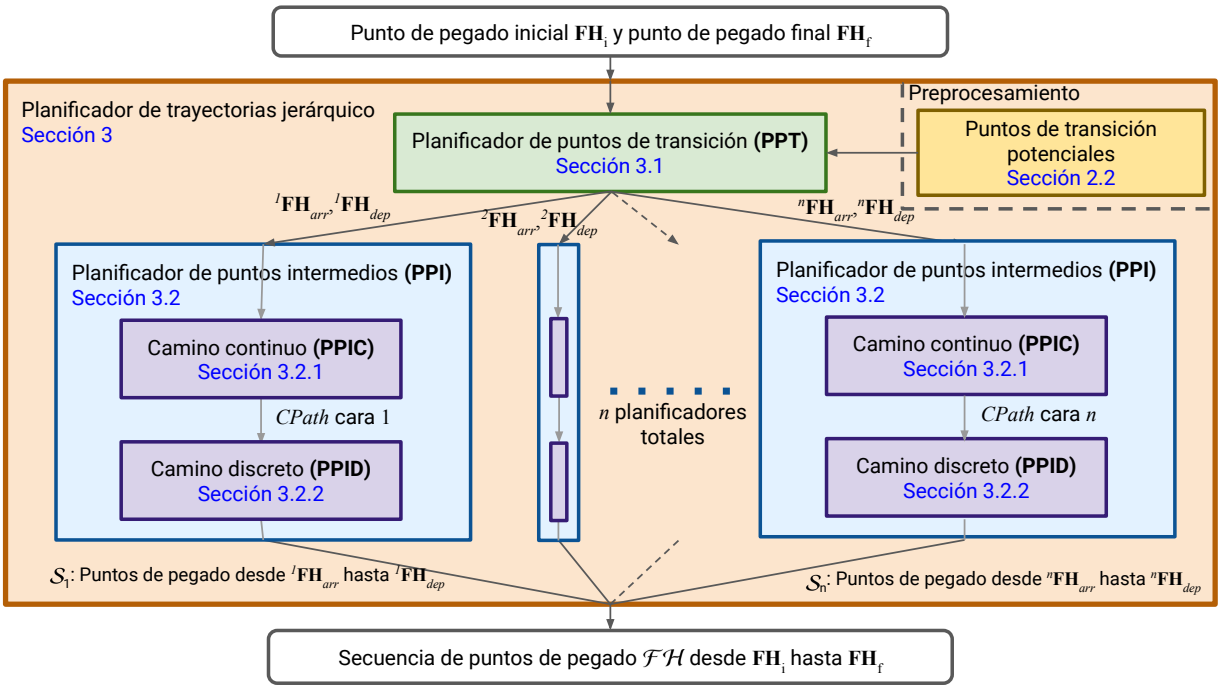


Figura 3: Esquema del algoritmo jerárquico de planificación de movimientos.

para, posteriormente, planificar el movimiento a lo largo de cada una en planificadores siguientes.

Para generar la trayectoria global (la serie de puntos de transición entre caras), se ha empleado un algoritmo fundamentado en el A*. Para implementarlo, el entorno se organiza en forma de grafo. El algoritmo A* se desplazará por el grafo en busca del camino óptimo, utilizando una función heurística para orientar la exploración del grafo (Hart et al., 1968). En nuestro caso, hemos seleccionado la distancia euclídea entre puntos como función heurística. El uso del algoritmo A* nos permite obtener el camino global óptimo de un solo paso, en contraste con la solución propuesta por Zhu et al. (2020), que para determinar las caras y puntos de transición emplean tres etapas: análisis de transitabilidad entre cada par de caras, búsqueda de aquellas caras que permiten unir punto inicial y el punto final, y optimización de los puntos de transición en dichas caras.

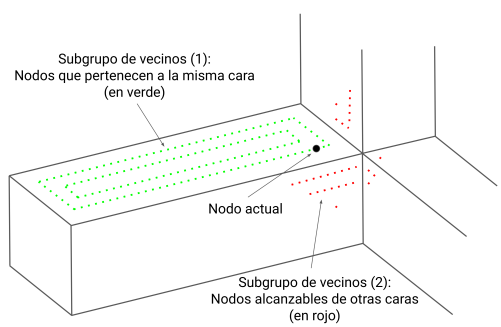


Figura 4: Vecinos de un nodo en cuestión y diferenciación de los subgrupos.

Los nodos que componen el grafo que se explora son los puntos de transición calculados con anterioridad (ver Sección 2.3 y Figura 2). En este punto de la planificación global, se destaca un aspecto clave que distingue al planificador propuesto de los métodos A* convencionales: la determinación de los vecinos de un nodo específico del grafo. Para un nodo determinado, la totalidad de sus vecinos está formada por (1) los demás puntos de pegado (nodos) pertenecientes a la misma cara, y (2) los *footholds* en otras caras que son accesibles desde el nodo en cuestión (ver Figura 4). La identificación de los nodos del subgrupo (2), es decir, aquellos puntos de transición alcanzables en otras caras, se lleva a cabo mediante un test de alcanzabilidad.

La capacidad de que una pose específica sea alcanzada por la garra libre se determina mediante un test de alcanzabilidad, partiendo de la premisa de que la garra fija está adherida al nodo actual, del cual se busca determinar los vecinos. En el caso de robots no redundantes, es sencillo de determinar, verificando si la solución de la cinemática inversa es real, evita colisiones y respeta los límites articulares. No obstante, los robots que presentan redundancia cinemática, como es el caso del robot HyReCRo, presentan infinitas posibles configuraciones como solución de su cinemática inversa. Por ende, no es posible verificar directamente la existencia de una solución válida, por lo que se recurre al test de alcanzabilidad para determinar si se puede alcanzar una posición y orientación determinadas. El rendimiento del test de alcanzabilidad es crucial para una planificación de movimientos veloz, puesto que constituye el elemento fundamental del planificador PPT. Durante el desarrollo del algoritmo, se han barajado varios métodos:

- **Iterativo**: se generan soluciones de la cinemática inversa para la pose sometida al test. Si se logra generar una

solución real, que cumpla con los límites articulares del robot, y esté libre de colisiones, la pose se identifica como alcanzable. Para robots redundantes (con $n$ articulaciones actuadas y una pose definida por $m$ coordenadas independientes), se seleccionan valores aleatorios para $(n-m)$ de las coordenadas articulares actuadas y se resuelven las otras $m$ a partir de las ecuaciones de la cinemática inversa analítica (Peidro et al., 2015). En el caso de que no se obtenga una solución válida (real, que cumpla con los límites articulares y no produzca colisiones), se generan aleatoriamente nuevos valores y se comprueba que la solución a la cinemática inversa sea válida, o se haya alcanzado un número concreto de intentos (por ejemplo: 300). Este método se ha revelado como no repetible y lento. La variabilidad en la cantidad de iteraciones es considerable debido a la naturaleza aleatoria inherente al método, puesto que en situaciones en las que no existe una solución factible, se requiere esperar a que se realice el número de intentos preestablecidos, lo cual introduce una significativa fluctuación del tiempo de cálculo.

- **Analítico**: consiste en analizar la cinemática específica de cada robot con el fin de determinar de forma analítica la alcanzabilidad de una pose. Por ejemplo, se verifica si la posición deseada se encuentra a una distancia superior a la suma de las longitudes de los eslabones del manipulador cuando este está completamente extendido. Este es el enfoque seguido, por ejemplo, por Zhu et al. (2020), gracias a que su robot trepador tiene una cinemática muy sencilla y carece de redundancia, de modo que el espacio de trabajo cuando una garra está fija es una esfera, y se puede determinar fácilmente un test analítico de este tipo para determinar si el punto evaluado está al alcance. Sin embargo, en robots de cinemática más compleja o redundantes, como es el caso del robot HyReCRo estudiado en este artículo, tal criterio analítico no es fácil de formular, ya que el espacio de trabajo de este robot no es una esfera ni otro tipo de forma de geometría sencilla. Por lo tanto, el test analítico tiene un fuerte carácter *ad-hoc*, hecho a medida para cada robot, y difícil de generalizar.

- **Mediante redes neuronales**: se desarrolla una red neuronal de clasificación binaria para la determinación la alcanzabilidad de una pose, ya sea alcanzable o no alcanzable. Si bien la obtención de resultados es muy rápida una vez que la red está entrenada, se debe tener en cuenta que el proceso de entrenamiento y la generación de los datos utilizados para el entrenamiento conllevan un tiempo que puede ser considerable. Además, se debe tener en cuenta que la precisión de la red nunca puede ser perfecta, por lo que siempre existe la posibilidad de que una pose sea clasificada erróneamente.

Dada la complejidad cinemática del robot HyReCRo, el método analítico no es factible (Peidro et al., 2015). Por otro lado, el método iterativo presenta una variabilidad considerable en cuanto al tiempo de ejecución y el resultado obtenido. Ante estas consideraciones, se ha optado por el uso de redes neuronales para la determinación de la alcanzabilidad de una pose.

La resolución de la cinemática inversa del robot HyReCRo debe hacerse diferenciando entre dos situaciones (Peidro et al., 2015): una situación regular, y otra singular, en la que los ejes **Z** de las dos garras (véase la Figura 1) quedan paralelos o antiparalelos. Las redes neuronales de clasificación de la alcanzabilidad también se ven afectadas por esta distinción. Por ende, se han generado dos redes neuronales distintas: una destinada a poses regulares, contenidas en el plano de la cara a la que está pegada la garra fija; y otra específica para transiciones, que asumen posturas singulares en las que los ejes **Z** de ambas garras del robot son paralelos o antiparalelos, puesto que estas posturas son las más favorables para realizar este tipo de transiciones (Peidró et al., 2019).

La arquitectura de las redes se muestra en la Figura 5 y corresponde a un perceptrón multicapa. La capa de entrada está compuesta por 6 neuronas, equivalentes a los 6 parámetros que describen una pose en un espacio tridimensional (3 de posición $(p_x, p_y, p_z)$ y 3 de rotación $(r_x, r_y, r_z)$, correspondientes a ángulos de Euler **ZYX**). Las dos capas ocultas son lineales y están formadas por 32 y 64 neuronas, y ambas usan la función ReLU como función de activación. La capa de salida es de una única neurona, con función de activación sigmoide, que devuelve un valor entre 0 y 1, donde 0 indica que la pose no es alcanzable y 1 que sí lo es.

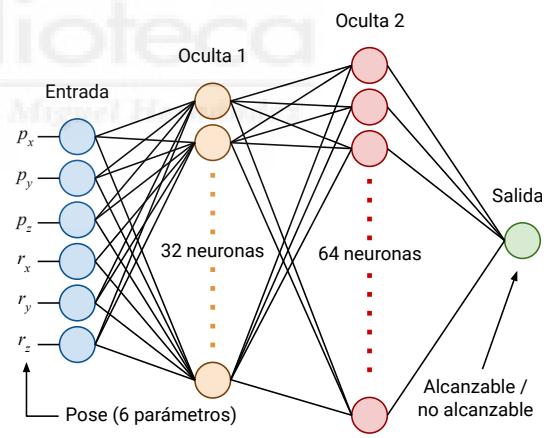


Figura 5: Arquitectura de la red neuronal de clasificación de alcanzabilidad.

Para entrenar ambas redes, se han obtenido datos de entrenamiento mediante la generación de poses aleatorias dentro de una aproximación del espacio de trabajo del robot en forma de esfera, y se han clasificado mediante el test de alcanzabilidad iterativo, almacenando los resultados (es decir, si existe solución o no). Se han generado $2 \times 10^7$ poses diferentes para el entrenamiento de cada red. Tras sintonizar los hiperparámetros y llegar a los seleccionados[1], se ha obtenido una precisión de 97,85% para transiciones entre caras y 98,32% para el resto de movimientos dentro de una cara; precisiones que resultan suficientes para los fines de este estudio. Con las redes entrenadas, formular la determinación de los vecinos es sencillo. En primer

[1]Tamaño lotes: 10, épocas: 200, tasa aprendizaje: 0.01, optimizador: SGD (Descenso Gradiente Estocástico), función pérdida: BCE (Entropía Cruzada Binaria).

lugar, se utilizan *k*-d trees (Bentley, 1975) como estructura de datos para realizar una búsqueda eficiente en una esfera, la cual se centra en la garra fija. El radio de la esfera necesita ser lo suficientemente amplio como para abarcar el espacio de trabajo completo del robot, y su elección requiere un análisis del espacio de trabajo, similar al enfoque seguido en (Peidró et al., 2017). Este paso permite descartar de manera instantánea la gran mayoría de las poses no alcanzables por el robot, evitando así la necesidad de aplicar el test de alcanzabilidad a estos puntos. Aunque esta aproximación como una esfera del espacio de trabajo es ciertamente imprecisa, únicamente se utiliza para descartar puntos que indudablemente no son alcanzables. Aquellos puntos que se incluyan en la aproximación (pertenezcan a la esfera), pero no formen parte realmente del espacio de trabajo, serán posteriormente descartados. Por último, todos los *footholds* seleccionados (que caen dentro de la esfera) se someten al test de alcanzabilidad, y se añaden a la colección de vecinos si se determinan como alcanzables y la configuración adquirida para alcanzar dicho punto no produce colisiones.

Como resultado, tal y como se ilustra en la Figura 6, el planificador PPT proporcionará la serie de puntos de transición que posibilitarán al robot alcanzar eficientemente el destino. Si una pareja de puntos de transición consecutivos se encuentra en una misma cara $F_i$, se designarán como punto de llegada ${}^{i}\mathbf{FH}_{arr}$ y de salida ${}^{i}\mathbf{FH}_{dep}$ de dicha cara. En el otro planificador de este nivel jerárquico, que se describe en la subsección siguiente 3.2, se calcularán los *footholds* intermedios dentro de una misma cara para cada pareja de puntos de llegada y de salida.

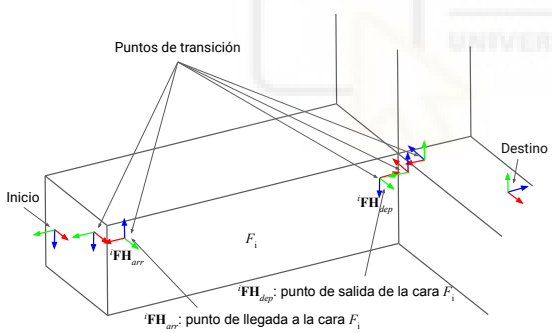


Figura 6: Puntos de transición devueltos por el planificador PPT.

### 3.2. *Planificador de Puntos Intermedios (PPI)*

En la ruta global generada por el planificador PPT, es posible que una pareja de puntos de transición consecutivos pertenezcan a una misma cara, tal y como se muestra con los puntos ${}^{i}\mathbf{FH}_{arr}$ y ${}^{i}\mathbf{FH}_{dep}$ en la Figura 6. En consecuencia, el robot necesita ser capaz de moverse a lo largo de la cara y trazar una ruta efectiva dentro de ella. El Planificador de Puntos Intermedios (PPI) entra en acción para determinar la ruta más eficiente en términos de longitud y número de pasos para recorrer la cara desde el punto de entrada hasta el punto de salida.

Con el planificador PPI se añade un nivel adicional de profundidad al planificador jerárquico, ya que se divide en dos planificadores. En primer lugar, se establece la trayectoria continua más corta mediante la construcción de un grafo de visibilidad. Posteriormente, se obtiene la serie de puntos de pegado discretos, adaptándolos en la medida de lo posible al camino continuo y maximizando la distancia cubierta en cada movimiento.

#### 3.2.1. *Planificador de Puntos Intermedios Continuo (PPIC)*

El objetivo del primer planificador PPIC de este nivel es dar con la ruta continua *CPath* más corta que recorra la cara, uniendo los puntos de llegada y de salida de la cara. Para ello, en primer lugar, se reduce el perímetro del polígono que define la cara para formar un polígono ”seguro”, el cual asegura que las garras del robot se apoyen dentro de la cara completamente. A continuación, se procede a construir un grafo de visibilidad (Lee and Preparata, 1984), donde cada vértice del polígono es un nodo y la conexión entre cada par de nodos está indicada por las aristas del polígono.

Para resolver esta fase, Zhu et al. (2020) emplean un algoritmo A* para evadir o atravesar posibles obstáculos presentes en las caras. En nuestro caso, tales obstáculos no existen, ya que, en el caso de que los hubiera, se tratarían como nuevas caras que ya habrían sido procesadas por el planificador PPT. Por lo tanto, determinar el camino continuo mediante un grafo de visibilidad será la forma más computacionalmente eficiente de obtener el camino continuo más corto en cada cara.

En la Figura 7 (ignórense por el momento los puntos de pegado intermedios), se visualiza el camino continuo *CPath* calculado en una cara, junto con el perímetro “seguro” utilizado para estructurar el grafo.

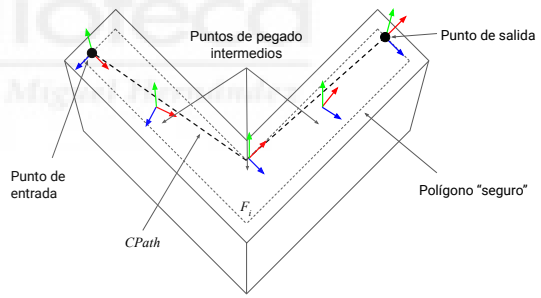


Figura 7: Salida del planificador PPI.

#### 3.2.2. *Planificador de Puntos Intermedios Discreto (PPID)*

Una vez que se ha calculado el camino continuo, se procederá a determinar el camino discreto, compuesto por la secuencia de puntos de pegado en los que se adherirán las garras. De manera iterativa, el algoritmo calculará el siguiente mejor *foothold*, empleando como guía el camino continuo generado en el planificador PPIC. De forma intuitiva, se podría tomar el punto en el que la frontera del espacio de trabajo del robot interseca con el camino continuo devuelto por el planificador PPIC con el fin de maximizar la distancia recorrida en cada paso. Esta es la idea utilizada en (Zhu et al., 2020), y es apropiada cuando la cinemática del robot permite alcanzar cualquier posición de su espacio de trabajo con cualquier orientación. Sin embargo, en robots como el HyReCRo, donde algunas posiciones de su espacio de trabajo permiten alcanzar todas las orientaciones, pero otras solo un subconjunto de ellas, es necesario realizar una planificación mediante un algoritmo más sofisticado, que es el que se describe en esta sección.

Con el fin de acelerar la ejecución del algoritmo para ser empleado en tiempo real, se propone realizar un cálculo previo *offline* del espacio de trabajo plano para no tener que resolver la cinemática inversa para cada punto evaluado. Se entiende como espacio de trabajo plano a la intersección entre el espacio de trabajo alcanzable del robot y el plano que define la cara explorada, que coincidirá con el plano formado por los ejes **X** y **Z** de la garra fija en cada punto de pegado de la cara. En la Figura 8 se muestra el robot y su espacio de trabajo plano en perspectiva.

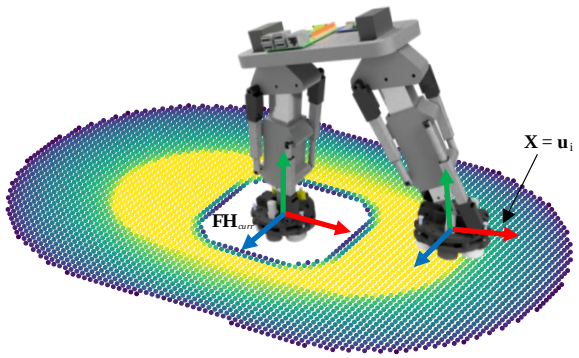


Figura 8: Vista en perspectiva del espacio de trabajo plano del robot HyReCRo.

Para generar el espacio de trabajo planar del robot, se discretiza en forma de malla, usando una resolución lo suficientemente alta como para obtener una discretización precisa y densa. Para cada punto de la malla, también se discretizan y calculan las orientaciones alcanzables. A partir de los puntos obtenidos, el modelo del espacio de trabajo se estructura como un conjunto de posiciones alcanzables (2D). A cada punto se le vincula una serie de vectores $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$, como se ilustra en la Figura 9(a), que son las orientaciones alcanzables de entre las discretizadas (1D). Cada elemento de este grupo de orientaciones es un vector unitario $\mathbf{u}_i$ paralelo a la cara, el cual sería el equivalente al eje **X** del extremo libre al pegarse en dicho punto. De modo que el espacio de trabajo planar se configura como un grupo de datos tridimensional, que recoge las poses alcanzables por el robot. De igual modo, para cada una de las poses alcanzables, se almacena una de las infinitas posibles soluciones de la cinemática inversa que permiten alcanzar dicha pose, que para ser conservadores a

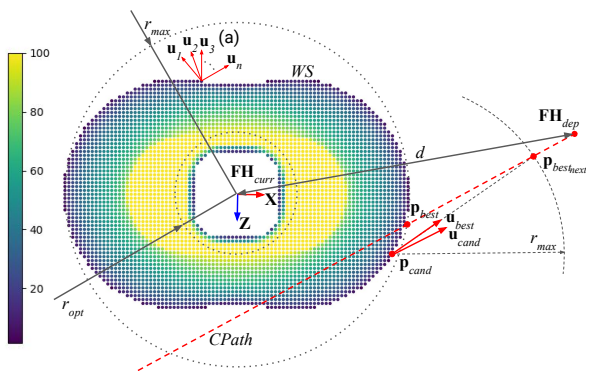


Figura 9: Espacio de trabajo plano del robot y planificador PPID (con $r = r_{max}$).

la hora de evitar colisiones con elementos el entorno, se elige como la configuración articular que maximiza el volumen ocupado por el robot El espacio de trabajo planar del robot HyReCRo se muestra en la Figura 9, donde se codifica el número de orientaciones alcanzables, de entre las discretizadas para cada punto, mediante un mapa de colores donde un valor del 100% (puntos amarillos) indica que se puede alcanzar cualquier orientación en esa posición; es decir, que el eje $\mathbf{u}_i$ mostrado en la Figura 9(a) podría apuntar en cualquier dirección del plano entre 0° y 360°.

El planificador PPID calculará iterativamente, desde el punto de llegada $\mathbf{FH}_{arr}$, el siguiente mejor punto de pegado $\mathbf{FH}_{best}$, siguiendo el camino continuo *CPath* devuelto por el planificador PPIC, hasta que se alcance el punto de salida $\mathbf{FH}_{dep}$, tal y como se muestra en el Algoritmo 1.

**Algorithm 1** Planificador PPID

1: $\mathcal{S} \leftarrow \mathbf{FH}_{arr}$
2: $\mathbf{FH}_{curr} \leftarrow \mathbf{FH}_{arr}$
3: **while** $\mathbf{FH}_{curr} \neq \mathbf{FH}_{dep}$ **do**
4: $\quad d \leftarrow$ distancia entre $\mathbf{FH}_{curr}$ y $\mathbf{FH}_{dep}$
5: $\quad r \leftarrow \text{mín}(r_{max}, d - r_{opt})$
6: $\quad \mathbf{FH}_{curr} \leftarrow$ NextBestFoothold($\mathbf{FH}_{curr}, r$)
7: $\quad$ Añadir $\mathbf{FH}_{curr}$ a la secuencia $\mathcal{S}$
8: **return** $\mathcal{S}$ ▷ puntos de pegado desde $\mathbf{FH}_{arr}$ hasta $\mathbf{FH}_{dep}$

En cada iteración del planificador PPID, se calcula la distancia $d$ (línea 4 del Algoritmo 1), que corresponde a la distancia entre el *foothold* de la garra fija en la iteración actual $\mathbf{FH}_{curr}$ y el destino en la cara $\mathbf{FH}_{dep}$. El alcance deseado $r$ se obtiene a partir de la distancia $d$ (línea 5). El alcance es la distancia que se desea cubrir con el paso de la iteración en cuestión (es decir, al pegar la garra libre estando la otra garra fijada en $\mathbf{FH}_{curr}$): $r = \text{mín}(r_{max}, d - r_{opt})$, donde $r_{max}$ es el alcance máximo del robot, la cual es posible aproximar a partir de su espacio de trabajo plano (Figura 9); y $r_{opt}$ se refiere al alcance óptimo, que corresponde al radio de la circunferencia formada por los puntos en el espacio de trabajo planar en los que se maximiza la cantidad de orientaciones alcanzables, como se muestra en la Figura 9. La segunda elección ($d - r_{opt}$) es la que permitiría preparar la pose de la garra fija de modo que, en el último paso, la pose deseada ($\mathbf{FH}_{dep}$) pertenezca a la circunferencia de radio $r_{opt}$, y que así la garra libre pueda ser pegada en dicha pose con cualquier orientación. El triedro $\mathbf{FH}_{dep}$ está impuesto por el planificador PPT, y es necesario alcanzar la orientación exigida por el triedro dado; esta elección de $r$ asegura que el origen de este triedro caiga en la circunferencia $r_{opt}$, lo cual asegura la alcanzabilidad del punto con cualquier orientación.

Como muestra la línea 6 del Algoritmo 1, en cada iteración es necesario calcular el siguiente mejor punto de pegado $\mathbf{FH}_{best}$ mediante la rutina NextBestFoothold, que se muestra en el Algoritmo 2, y cuyos pasos se ilustran en la Figura 9 y se describen a continuación.

En primer lugar, se calcula la mejor posición de pegado $\mathbf{p}_{best}$ como la intersección del camino continuo *CPath* (previamente calculado por el planificador PPIC) y un círculo, centrado en el *foothold* actual del robot $\mathbf{FH}_{curr}$, y de radio igual al alcance deseado $r$ (línea 2 del Algoritmo 2).

**Algorithm 2** Cálculo del siguiente mejor punto de pegado

1: **function** NEXTBESTFOOTHOLD($\mathbf{FH}_{curr}$, $r$)
2: $\mathbf{p}_{best} \leftarrow$ intersección de $CPath$ y círculo($\mathbf{FH}_{curr}$, $r$)
3: $WS_{sort} \leftarrow WS$ contenido en $\mathcal{P}$, ordenado respecto a $\mathbf{p}_{best}$
4: **for** $\mathbf{p}_{cand}$ **in** $WS_{sort}$ **do**
5: $\mathbf{p}_{best_{next}} \leftarrow$ intersección de $CPath$ y círculo($\mathbf{p}_{cand}$, $r$)
6: $\mathbf{u}_{best} \leftarrow$ vector desde $\mathbf{p}_{cand}$ que apunta hasta $\mathbf{p}_{best_{next}}$
7: $\mathbf{u}_{cand} \leftarrow$ vector en $\mathbf{p}_{cand}$ más cercano a $\mathbf{u}_{best}$
8: **if** $\mathbf{u}_{cand} \cdot \mathbf{u}_{best} \leq \varepsilon$ **and** libre de colisiones **then**
9: $\mathbf{FH}_{best} \leftarrow \{\mathbf{p}_{cand}, \mathbf{u}_{cand}\}$
10: **return** $\mathbf{FH}_{best}$

Seguidamente, el polígono $\mathcal{P}$, que define la cara de la que se está obteniendo el camino, se interseca con el espacio de trabajo planar $WS$ (el mostrado en la Figura 9), descartando así los puntos del espacio de trabajo que caen fuera de la cara. Los puntos restantes se ordenan según su distancia a $\mathbf{p}_{best}$ (línea 3).

A continuación, se recorre cada punto del espacio de trabajo ya ordenado $WS_{sort}$, desde el más cercano a $\mathbf{p}_{best}$, hasta el más alejado (línea 4). Para cada posición de pegado candidata $\mathbf{p}_{cand}$, se calcula $\mathbf{u}_{best}$ como el vector unitario que señala $\mathbf{p}_{best_{next}}$ desde $\mathbf{p}_{cand}$ (línea 6). $\mathbf{p}_{best_{next}}$ es la predicción del mejor punto de pegado de la siguiente iteración, y se obtiene como la intersección de una circunferencia centrada en $\mathbf{p}_{cand}$ con un radio igual al alcance deseado $r$ y del camino continuo $CPath$ (línea 5). De este modo $\mathbf{u}_{best}$ sería el eje $\mathbf{X}$ de la garra libre al fijarla en $\mathbf{p}_{cand}$, y, dado que el alcance del robot presenta su máximo en la dirección de este eje (ver Figura 9), se lograría maximizar el paso recorrido en la dirección de $\mathbf{u}_{best}$ en la siguiente iteración (cuando la garra libre se pegase en $\mathbf{p}_{cand}$ y pasase a ser la garra fija en la siguiente iteración del Algoritmo 1).

Seguidamente, se selecciona el vector $\mathbf{u}_{cand}$ más cercano a $\mathbf{u}_{best}$ (línea 7), a partir todos los vectores $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n\}$ que definen las orientaciones alcanzables en $\mathbf{p}_{cand}$, calculados durante el preprocesamiento del espacio de trabajo plano (Figura 9).

Para validar la elección, se debe comprobar que el vector elegido $\mathbf{u}_{cand}$ sea lo suficientemente próximo a $\mathbf{u}_{best}$. La cercanía se cuantifica mediante el producto escalar entre ambos vectores, cuyo valor indicará que serán cercanos cuando es próximo a 1. Si el resultado del producto escalar alcanza cierto valor umbral $\varepsilon$ (e.g., $\varepsilon = 0{,}998$, que corresponde a un rango de aproximadamente $\pm 4°$) y la configuración articular adoptada para alcanzar $\mathbf{p}_{cand}$, con orientación $\mathbf{u}_{cand}$, está libre de colisiones (línea 8), se devuelve el punto de pegado $\mathbf{FH}_{best}$, formado por la posición $\mathbf{p}_{cand}$, $\mathbf{u}_{cand}$ como vector que define el eje $\mathbf{X}$, $\mathbf{Y}$ perpendicular a la cara, y el $\mathbf{Z}$ obtenido como el producto vectorial de $\mathbf{X}$ e $\mathbf{Y}$ (líneas 9-10).

Si no se cumplen las condiciones establecidas en la línea 8, el algoritmo seguirá recorriendo el espacio de trabajo plano ordenado ($WS_{sort}$, línea 4) hasta dar con una combinación de posición y orientación que sí lo haga.

Cabe destacar que, tanto la determinación del orden del espacio de trabajo, como la búsqueda de la posición y orientación más cercanas, hacen uso de $k$-d trees como estructura de datos para reducir los tiempos de búsqueda.

Finalmente, y volviendo al Algoritmo 1, tras determinar $\mathbf{FH}_{best}$, se añade a la secuencia $\mathcal{S}$ de puntos de pegado (línea 7), y todo el proceso se repite hasta que el punto de pegado actual coincide con el punto de salida de la cara $\mathbf{FH}_{dep}$ (linea 3). Tras esto, el Algoritmo 1 devuelve la secuencia $\mathcal{S}$ (línea 8) de puntos de pegado que une el punto de entrada $\mathbf{FH}_{arr}$ y el punto de salida de la cara $\mathbf{FH}_{dep}$, como se muestra en la Figura 7.

## 4. Simulación

Para validar el desempeño del algoritmo propuesto en este artículo, se ha llevado a cabo una simulación con el objetivo de planificar los movimientos del robot en una estructura reticular tridimensional. Los experimentos se han realizado en Python, con un procesador AMD Ryzen 7 5800U.

El entorno simulado consiste en dos vigas perpendiculares cruzadas. Estas vigas tienen longitudes de 1 y 2 metros, respectivamente, con perfiles cuadrados de lado igual a 0.15 metros. De esta manera, la solución propuesta puede ser evaluada en una estructura compleja que requiere de múltiples transiciones y movimientos a través de las caras exploradas.

El tratamiento previo de la estructura, detallado en la Sección 2.3, se ha realizado empleando un paso de 0.05 metros en la discretización de los *footholds* potenciales. Se han utilizado unas distancias de 0.25 y 0.07 metros para el cálculo de las transiciones cóncavas y convexas ($d_{cncv}$ y $d_{cnvx}$ en la Figura 2), respectivamente. En el planificador PPIC, se han generado los polígonos "seguros" reduciendo los perímetros de las caras una distancia equivalente a la utilizada para obtener los puntos de las transiciones convexas (0.07 metros). En lo que respecta al espacio de trabajo plano, utilizado en el planificador PPID, se ha generado con una resolución de 0.01 metros en las dimensiones de posición y 5 grados para la orientación, como se muestra en la Figura 9. Las resoluciones han sido seleccionadas tras un análisis, el cual se discute en la Sección 4.1.

En el vídeo adjunto (también disponible en: https://youtu.be/fUznNYWVfgQ) se puede observar la simulación completa, donde se muestra el camino generado por el algoritmo, así como el movimiento del robot a lo largo del mismo. En la Figura 10 se muestra el camino completo devuelto por el planificador jerárquico (Figura 10(a)) y varios instantes de la simulación (Figura 10(b-f)). Nótese que, tanto en la Figura 10, como en el vídeo adjunto (ambos corresponden al mismo experimento), el robot siempre se mueve libre de colisiones, a pesar de que, por la coloración de la representación, pueda parecer que existe alguna interferencia u oclusión. Esto se debe a cómo trata Matplotlib los gráficos 3D: aunque no haya interferencia, cuando 2 objetos se solapan desde el punto de vista de la cámara, su coloración se combina (el gris de la estructura y el azul del robot) y puede dar lugar a confusiones.

La trayectoria devuelta está compuesta de los triedros de la figura, que equivalen a los puntos de pegado donde las garras se irán adhiriendo. Es visible que, en una primera instancia, el Planificador de Puntos de Transición (PPT) determina la necesidad realizar 2 transiciones entre las caras, utilizando la serie de puntos de transición formada por los *footholds* $\{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$ (Figura 10(a)) para llegar al punto deseado, partiendo desde el origen. En las Figuras 10(c) y 10(e) se representa al robot realizando las transiciones demandadas por el planificador PPT.

Para cada cara que se debe atravesar, el Planificador de Puntos Intermedios (PPI) determina los *footholds* que se emplearán para recorrer cada cara. El *foothold* intermedio ${}^{f}\mathbf{FH}_i$ representa el elemento número $i$ en la secuencia de puntos de

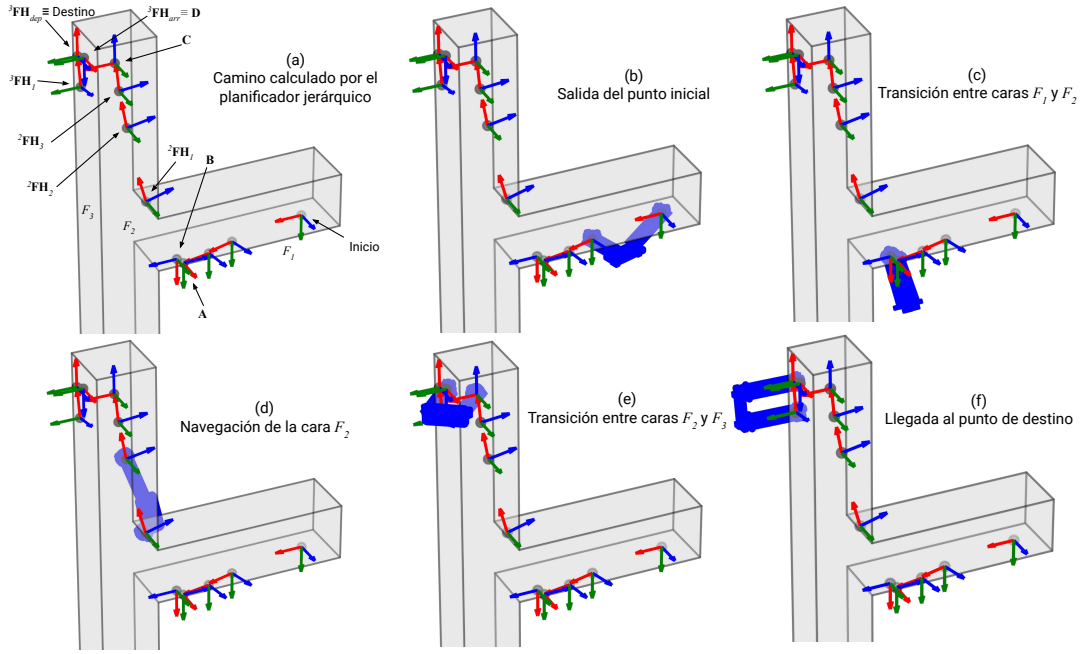


Figura 10: Camino obtenido por el planificador jerárquico (a) e instantes de la simulación (b-f).

pegado necesarios para atravesar la cara $F_f$. Como ejemplo, en la Figura 10(a), se observa la secuencia de puntos de pegado $\{^2\mathbf{FH}_1, ^2\mathbf{FH}_2, ^2\mathbf{FH}_3\}$ que el robot deberá seguir para recorrer la cara $F_2$, y en la Figura 10(d) se muestra un instante de la navegación de la misma (movimiento entre $^2\mathbf{FH}_1$ y $^2\mathbf{FH}_2$).

Es visible que, en la última cara $F_3$ del camino, el punto de salida $^3\mathbf{FH}_{dep}$ de la cara, que corresponde al punto de destino en la estructura, no es alcanzable directamente desde el punto de entrada $^3\mathbf{FH}_{arr}$, a pesar de que se encuentra dentro de una circunferencia con centro en la garra fija y de radio equivalente al alcance máximo (Figura 9). Esto se debe a que el punto no pertenece al espacio de trabajo, ya que se encuentra demasiado próximo a la garra que actúa como base, y ambas garras (la fija y la libre) colisionarían (ver Figura 9). El planificador PPID detecta de manera inteligente que se requiere un punto de pegado intermedio adicional $^3\mathbf{FH}_1$, y lo posiciona a una distancia que equivale al alcance óptimo $r_{opt}$. De esta manera, a partir del *foothold* $^3\mathbf{FH}_1$, se podrá alcanzar el punto de destino con cualquier orientación, resolviendo el problema y pudiendo llegar al destino, tal y como se representa en la Figura 10(f).

Se han obtenido las estadísticas mostradas en la Tabla 1, calculando el tiempo promedio de ejecución de cada uno de los planificadores a lo largo de 100 repeticiones del experimento.

El preprocesamiento del entorno se realiza de forma previa a la ejecución del algoritmo, lo que permite llevar a cabo esta tarea antes de que el robot sea desplegado en la estructura, sin afectar al rendimiento en tiempo real del algoritmo. El tiempo de ejecución medio del algoritmo completo es de 700,743 milisegundos, siendo 688,478 milisegundos atribuibles al planificador PPT y 12,265 milisegundos a la suma del tiempo empleado por los 3 planificadores PPI. Al tratarse de un tiempo por debajo del segundo, consideramos que el algoritmo es lo suficientemente rápido como para ser empleado en tiempo real. El planificador PPI demuestra ser considerablemente eficiente, generando los caminos en un tiempo muy corto, lo que sugiere que hay poco margen para mejoras significativas en su rendimiento. Por otro lado, el planificador PPT presenta una oportunidad para mejoras, ya que representa el 98,25% del tiempo de ejecución completo. No obstante, puede resultar complicado mejorar esta estadística si se desea mantener la exhaustividad en la búsqueda para garantizar un camino global óptimo o cercano al óptimo.

Tabla 1: Tiempo promedio de ejecución de cada planificador del algoritmo.

| Sección | Nivel de la jerarquía | Tiempo (ms) |
|---|---|---|
| 2.3 | Preprocesamiento de la estructura | 136,999 |
| **3** | **Algoritmo completo *(online)*** | **700,743** |
| 3.1 | Planificador de Puntos de Transición | 688,478 |
| 3.2 | Planificador de Puntos Intermedios ($F_1$) | 3,160 |
| | Planificador de Puntos Intermedios ($F_2$) | 7,792 |
| | Planificador de Puntos Intermedios ($F_3$) | 1,313 |

### 4.1. Discusión

Con el fin de ofrecer datos estadísticos que permitan comparar el rendimiento del algoritmo cuando se modifican ciertos parámetros, se han realizado experimentos adicionales, cuyos resultados se muestran en la Tabla 2, en los que se modifica un solo parámetro del algoritmo en cada experimento. Se ha mantenido el escenario de la Figura 10, y se ha variado la resolución a la hora obtener los puntos de transición potenciales descritos en la Sección 2.3, así como las resoluciones, tanto de posición como de orientación, del espacio de trabajo plano del robot, que

se emplea en el planificador PPID (Sección 3.2.2). En los experimentos comparativos se ha evaluado el tiempo de ejecución del algoritmo y el número de pasos que da el robot (es decir, el número de pegados que debe hacer la garra del robot a lo largo del camino) para llegar al destino. Se ha escogido como función de mérito dicho número de pasos, que debería reducirse en la medida de lo posible, ya que es una operación donde los robots paso a paso invierten una gran fracción del tiempo total de su locomoción (Peidró et al., 2019), en contraste con los robots de movimiento continuo que no precisan de esta operación.

Según los resultados de la Tabla 2, se observa que, como es lógico, el tiempo de ejecución del algoritmo se ve afectado por la resolución de la discretización empleada. Ya sea para la discretización de los puntos potenciales, o para la discretización del espacio de trabajo plano, una discretización más fina conlleva un incremento en el tiempo de ejecución. Por el contrario, cuando se incrementa el paso de las discretizaciones, el tiempo de ejecución disminuye, pero la calidad de la solución puede verse afectada, i.e., el número de pasos necesarios para recorrer el camino puede aumentar, lo que conllevaría un incremento considerable en el tiempo empleado para que el robot recorra el camino completo. Dado que la calidad de la solución (el número de pasos) no seguirá mejorando por mucho que se afine la resolución (ya que el robot está limitado por su zancada máxima), mientras que el tiempo de cómputo sí aumentará, existen valores óptimos de los parámetros que permiten alcanzar un compromiso entre dicho tiempo y calidad, habiéndose escogido los parámetros originales de la Tabla 2 para lograr dicho compromiso.

Por otra parte, también se han comparado diferentes fases de nuestro algoritmo con el que puede considerarse como el algoritmo referencia en el estado del arte (Zhu et al., 2020). De los dos planificadores principales, únicamente el planificador PPI es comparable, puesto que el planificador PPT de Zhu et al. (2020) conlleva disponer de un test de alcanzabilidad analítico del robot, lo cual no es factible en nuestro caso, tal y como se ha mencionado en la Sección 3.1.

En cuanto al planificador PPI, se ha comparado tanto el tiempo de ejecución como la calidad de la solución obtenida (es decir, el número de pegados que realiza de las garras), para cada uno de los planificadores por los que está compuesto (PPIC y PPID). Se han realizado los experimentos en un nuevo escenario mostrado en la Figura 11.

El enfoque empleado en el planificador PPIC de Zhu et al. (2020) consiste en un algoritmo A*, frente al grafo de visibilidad empleado en nuestro caso. Promediando 100 simulaciones, el tiempo de ejecución del algoritmo A* (con vecindad a 0,1 metros) es de 8,975 milisegundos, con una longitud de camino de 2,7 metros; mientras que el grafo de visibilidad presenta un tiempo de ejecución de 0,944 milisegundos, con una longitud de 2,631 metros. Por lo tanto, nuestro enfoque resulta en tiempos de ejecución menores y en un camino continuo más corto, como se muestra en la Figura 11(a).

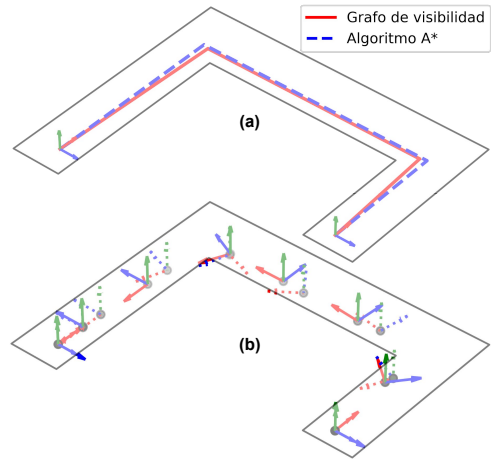


Figura 11: Comparativa planificador PPIC y PPID con diferentes enfoques.

En relación al planificador PPID, el algoritmo que proponemos presenta una mayor elaboración y generalidad para calcular el camino discreto, puesto que resulta más apropiado para robots que, como el robot HyReCRo, no tienen por qué alcanzar todas las orientaciones posibles en todos los puntos de su espacio de trabajo, mientras que el algoritmo empleado por Zhu et al. (2020) asume que el robot puede alcanzar cualquier punto con cualquier orientación. En la Figura 11(b) se muestran las secuencias de puntos de pegado obtenidas por ambos algoritmos, donde los triedros de trazo continuo representan los puntos de pegado obtenidos por nuestro algoritmo, y los triedros a trazos los obtenidos al proceder como Zhu et al. (2020), es decir, tomando simplemente el punto del espacio de trabajo más cercano al destino que corta al camino continuo, sin tener en cuenta la orientación con la que queda la garra al pegarla en dicho punto (que, por lo general, quedará orientada de forma menos favorable para ejecutar ulteriores pasos). El camino obtenido por nuestro algoritmo presenta un menor número de pasos (7 frente a 8), pero un tiempo de cómputo mayor (63,643 frente a 8,299 milisegundos). Este ligero aumento de tiempo de cómputo es preferible si nos ahorramos un paso del robot,

Tabla 2: Comparativa de diferentes parámetros de configuración, para el escenario de la figura 10. Valores medios de 100 repeticiones.

| Modificaciones | | | | Tiempos | | | | | | | Calidad | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Parámetro** | **Valor original** | **Incremento** | **Nuevo valor** | **Preprocesamiento** | **PPT** | **PPI ($F_1$)** | **PPI ($F_2$)** | **PPI ($F_3$)** | **TOTAL** | **Mejora** | **Pasos** | **Incremento** |
| Parámetros originales | | | | 136,999 ms | 688,478 ms | 3,160 ms | 7,792 ms | 1,313 ms | 837,742 ms | - | 11 | 0 |
| Discretización puntos potenciales | 0,050 m | 0,100 m | 0,150 m | 84,498 ms | 153,899 ms | 2,932 ms | 989,954 ms | 1,121 ms | 1.232,404 ms | -32,02 % | 13 | 2 |
| | | 0,050 m | 0,100 m | 105,473 ms | 294,708 ms | 2,910 ms | 7,403 ms | 1,299 ms | 411,793 ms | 103,44 % | 11 | 0 |
| | | 0,020 m | 0,070 m | 128,936 ms | 527,592 ms | 2,946 ms | 7,539 ms | 1,318 ms | 668,331 ms | 25,35 % | 11 | 0 |
| | | -0,020 m | 0,030 m | 271,241 ms | 1.686,992 ms | 8,653 ms | 13,307 ms | 6,081 ms | 1.986,274 ms | -57,82 % | 11 | 0 |
| Discretización posición del $WS$ | 0,010 m | 0,040 m | 0,050 m | *No converge* | | | | | | | | |
| | | 0,020 m | 0,030 m | 136,981 ms | 678,223 ms | 2,194 ms | 7,003 ms | 1,106 ms | 825,507 ms | 1,48 % | 12 | 1 |
| | | -0,005 m | 0,005 m | 135,894 ms | 690,109 ms | 9,606 ms | 27,750 ms | 7,992 ms | 871,351 ms | -3,86 % | 11 | 0 |
| Discretización orientación del $WS$ | 5º | 15º | 20º | 134,583 ms | 682,909 ms | 1,601 ms | 6,125 ms | 1,198 ms | 826,416 ms | 1,37 % | 12 | 1 |
| | | 5º | 10º | 135,104 ms | 688,911 ms | 2,912 ms | 7,329 ms | 1,002 ms | 835,258 ms | 0,30 % | 11 | 0 |
| | | -3º | 2º | 136,004 ms | 694,291 ms | 4,928 ms | 11,583 ms | 3,136 ms | 849,942 ms | -1,44 % | 11 | 0 |

que requiere un tiempo comparativamente mayor para pegar la garra libre en el nuevo punto y despegar la garra anterior, que supone un tiempo de unos 3 segundos (Peidró et al., 2019). Como demuestra este ejemplo, el algoritmo propuesto requiere un menor número de pasos cuando el robot se mueve en caras que, como en la Figura 11(b), tienen varios cambios de dirección. Cabe destacar que, si repetimos esta comparación en caras más simples (e.g., sin cambios de dirección), ambos algoritmos pueden generar el mismo número de pasos, porque, en ese caso, el robot no debe variar mucho su orientación y resulta menos relevante el tener en cuenta la orientación de la garra al pegarse.

## 5. Conclusiones

A lo largo del artículo se ha presentado un algoritmo jerárquico para la planificación de movimientos de un robot bípedo diseñado para trepar estructuras tridimensionales reticulares. Los distintos niveles de la jerarquía abordan el problema de manera estructurada y secuencial, descomponiendo la complejidad de trabajar en un entorno tridimensional en subproblemas planos. En el proceso de calcular el camino, el primer planificador (PPT) determina la secuencia de caras que el robot atravesará, así como los puntos de entrada y salida de cada una de ellas. Seguidamente, para cada cara atravesada, el planificador PPI calcula el camino para recorrerla. El planificador PPI vuelve a dividir el problema en dos: el planificador PPIC calcula el camino continuo más corto que une el punto de entrada y de salida de la cara, y el planificador PPID calcula el camino discreto (secuencia de pasos) que recorra la cara, usando como guía el camino continuo devuelto por el planificador PPIC.

Se ha validado el correcto funcionamiento del algoritmo en simulación, y se han mostrado estadísticas, imágenes y vídeo de los resultados y rendimientos obtenidos. Se ha estudidado cómo afecta la variación de varios parámetros del algoritmo en el tiempo de ejecución y en la calidad de la solución obtenida, encontrando un compromiso; y se ha comparado el rendimiento de los planificadores PPIC y PPID con los propuestos por Zhu et al. (2020), obteniendo resultados favorables en ambos casos.

En lo que se refiere a futuras líneas de investigación, se planea llevar a cabo experimentos reales con el robot HyReCRo a corto plazo. Además, planeamos desarrollar redes neuronales que proporcionen una solución a la cinemática inversa del robot redundante, en contraste con las propuestas en este artículo, que se enfocan en determinar la alcanzabilidad de un punto, lo que mejoraría el rendimiento y aceleraría el método propuesto. Asimismo, la planificación de movimientos como manipulador, es decir, la determinación de los valores de las variables articulares en cada instante cuando el robot debe mover la garra de un *foothold* al siguiente, es un problema que se pretende abordar en estudios futuros, ya que en el ejemplo de la simulación adjunta se ha usado una interpolación lineal por simplicidad, para animar el movimiento de garras entre los *footholds* calculados.

## Agradecimientos

Este trabajo es parte del proyecto PID2020-116418RB-I00, financiado por MCIN/AEI/10.13039/501100011033; y parte de la ayuda PRE2021-099226, financiada por MCIN/AEI/10.13039/501100011033 y por el FSE+.

## Referencias

Bentley, J. L., 1975. Multidimensional binary search trees used for associative searching. Communications of the ACM 18 (9), 509–517.

Breitenmoser, A., Siegwart, R., 2012. Surface reconstruction and path planning for industrial inspection with a climbing robot. In: 2012 2nd International Conference on Applied Robotics for the Power Industry (CARPI). IEEE, pp. 22–27.

Chen, W., Gu, S., Guan, Y., Zhang, H., Liu, G., Tang, H., 2016. A multi-layered path planning algorithm for truss climbing with a biped robot. In: 2016 IEEE International Conference on Information and Automation (ICIA). IEEE, pp. 1200–1205.

Chen, X., Wu, Y., Hao, H., Shi, H., Huang, H., 2019. Tracked wall-climbing robot for calibration of large vertical metal tanks. Applied Sciences 9 (13), 2671.

Fabregat-Jaén, M., Peidró, A., Reinoso, Ó., Soler, F.-J., Jiménez, L. M., May 2022. Automatización del pegado de las garras magnéticas de un robot trepador bípedo. In: Jornadas de Robótica, Educación y Bioingeniería - JREB22. Vol. 1. pp. 57–63.

Fang, L., Yang, Q., Yang, T., 2020. Research on path planning algorithm of two-machine cooperative wall climbing and sanding robot based on ant colony algorithm. In: Proceedings of the Seventh Asia International Symposium on Mechatronics: Volume I. Springer, pp. 501–511.

Gimenez, A., Abderrahim, M., Padron, V., Balaguer, C., 2002. Adaptive control strategy of climbing robot for inspection applications in construction industry. IFAC Proceedings Volumes 35 (1), 19–24.

Hart, P. E., Nilsson, N. J., Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics 4 (2), 100–107.

Huang, H., Li, D., Xue, Z., Chen, X., Liu, S., Leng, J., Wei, Y., 2017. Design and performance analysis of a tracked wall-climbing robot for ship inspection in shipbuilding. Ocean Engineering 131, 224–230.

Jang, K., An, Y.-K., Kim, B., Cho, S., 2020. Automated crack evaluation of a high-rise bridge pier using a ring-type climbing robot. Computer-Aided Civil and Infrastructure Engineering 36, 14 – 29.

Lee, D.-T., Preparata, F. P., 1984. Euclidean shortest paths in the presence of rectilinear barriers. Networks 14 (3), 393–410.

Nguyen, S. T., La, H. M., 2021. A climbing robot for steel bridge inspection. Journal of Intelligent & Robotic Systems 102, 1–21.

Pagano, D., Liu, D., 2017. An approach for real-time motion planning of an inchworm robot in complex steel bridge environments. Robotica 35 (6), 1280–1309.

Peidró, A., Gil, A., Marín, J. M., Berenguer, Y., Payá, L., Reinoso, O., 2016. Monte-carlo workspace calculation of a serial-parallel biped robot. In: Robot 2015: Second Iberian Robotics Conference: Advances in Robotics, Volume 2. Springer, pp. 157–169.

Peidro, A., Gil, A., Marin, J. M., Reinoso, O., 2015. Inverse kinematic analysis of a redundant hybrid climbing robot. International Journal of Advanced Robotic Systems 12 (11), 163.

Peidró, A., Reinoso, Ó., Gil, A., Marín, J. M., Payá, L., 2017. An improved monte carlo method based on gaussian growth to calculate the workspace of robots. Engineering Applications of Artificial Intelligence 64, 197–207.

Peidró, A., Tavakoli, M., Marín, J. M., Reinoso, Ó., 2019. Design of compact switchable magnetic grippers for the hyrecro structure-climbing robot. Mechatronics 59, 199–212.

Prados, C., Hernando, M., Gambao, E., Brunete, A., 2023. Romerin: Organismo robótico escalador basado en patas modulares con ventosas activas. Revista Iberoamericana de Automática e Informática industrial 20 (2), 175–186.

Quin, P., Paul, G., Alempijevic, A., Liu, D., 2016. Exploring in 3d with a climbing robot: Selecting the next best base position on arbitrarily-oriented surfaces. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 5770–5775.

Stumm, E., Breitenmoser, A., Pomerleau, F., Pradalier, C., Siegwart, R., 2012. Tensor-voting-based navigation for robotic inspection of 3d surfaces using lidar point clouds. The International Journal of Robotics Research 31 (12), 1465–1488.

Tavakoli, M., Marques, L., de Almeida, A. T., 2011. 3dclimber: Climbing and manipulation over 3d structures. Mechatronics 21 (1), 48–62.

Yang, C.-h. J., Paul, G., Ward, P., Liu, D., 2016. A path planning approach via task-objective pose selection with application to an inchworm-inspired climbing robot. In: 2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM). IEEE, pp. 401–406.

Zhu, H., Lu, J., Gu, S., Wei, S., Guan, Y., 2020. Planning three-dimensional collision-free optimized climbing path for biped wall-climbing robots. IEEE/ASME Transactions on Mechatronics 26 (5), 2712–2723.