Programa de Doctorado en Tecnologías Industriales y de Telecomunicación

Coordinación robótica multi-brazo para la planificación óptima de trayectorias sin colisiones. Aplicación a la manipulación de objetos estáticos y en movimiento.

Daniel Mateu Gómez

Director de la tesis

Dr. Carlos Pérez Vidal

Universidad Miguel Hernández de Elche
Mayo 2024

La presente Tesis Doctoral, titulada "*Coordinación robótica multi-brazo para la planificación óptima de trayectorias sin colisiones. Aplicación a la manipulación de objetos estáticos y en movimiento*" se presenta bajo la modalidad de tesis por compendio de publicaciones. El cuerpo de dicha tesis queda constituido por el siguiente artículo, cuya referencia bibliográfica completa se indica a continuación:

Mateu-Gomez, D.; Martínez-Peral, F.J.; Perez-Vidal, C. <<Multi-Arm Trajectory Planning for Optimal Collision-Free Pick-and-Place Operations>>. Technologies 2024, 12, 12. https://doi.org/10.3390/technologies12010012

- Título de la revista: Technologies
- Factor de impacto (JCR 2022): 3.6
- Categoría: ENGINEERING, MULTIDISCIPLINARY. Cuartil Q1 (41/178)

Así mismo, la presente Tesis integra el siguiente artículo que actualmente se encuentra en proceso de revisión:

Mateu-Gomez, D.; Martínez-Peral, F.J.; Perez-Vidal, C. <<Dual-Arm Coordination for Mutual-Collision-Avoidance to Implement a Pick-on-the-Fly Application>>

- Título de la revista: Robotics and Computer-Integrated Manufacturing
- Factor de impacto: (JCR 2022): 10.4
- Categoría: COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS.
  Cuartil Q1, Decil D1 (7/110)
- Estado: en revisión

UNIVERSITAS
Miguel Hernández

AUTORIZACIÓN DE PRESENTACIÓN DE TESIS DOCTORAL POR COMPENDIO DE PUBLICACIONES

Director: Carlos Pérez Vidal

Título de la Tesis: **Coordinación robótica multi-brazo para la planificación óptima de trayectorias sin colisiones. Aplicación a la manipulación de objetos estáticos y en movimiento.**

Autor: Daniel Mateu Gómez

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

El director de la tesis reseñada certifica que ha sido realizada bajo su dirección por Daniel Mateu Gómez en el Programa de Doctorado en Tecnologías Industriales y de Telecomunicación de la Universidad Miguel Hernández de Elche, y autoriza su presentación

Elche, 2024

_____
Carlos Pérez Vidal

UNIVERSITAS
Miguel Hernández

PROGRAMA DE DOCTORADO EN TECNOLOGÍAS INDUSTRIALES Y DE TELECOMUNICACIÓN

Dr. Óscar Reinoso García, Coordinador del Programa de Doctorado en Tecnologías Industriales y de Telecomunicación en la Universidad Miguel Hernández de Elche

CERTIFICA

Que el trabajo realizado por D. Daniel Mateu Gómez, titulado **Coordinación robótica multi-brazo para la planificación óptima de trayectorias sin colisiones. Aplicación a la manipulación de objetos estáticos y en movimiento**, ha sido dirigido por el Dr. Carlos Pérez Vidal, y se encuentra en condiciones de ser leído y defendido como Tesis Doctoral ante el correspondiente tribunal en Universidad Miguel Hernández de Elche.

Lo que firmo para los efectos oportunos en

Elche, 2024

_____
Óscar Reinoso García

ABSTRACT

The current industrial environment demands that production lines adapt quickly to meet the demands of customized manufacturing according to customer needs. As robotic technology transitions from rigid, pre-programmed systems to collaborative entities equipped with sensors, these robots integrate into smaller manufacturing environments, working alongside others to increase efficiency and save workspace. However, this collaboration also brings its own challenges, especially when it comes to optimizing object manipulation in a workspace shared by multiple robots.

This thesis proposes an innovative approach based on an architecture that abstracts or decouples the robot from the rest of the system. This division allows for both generating robust and accurate solutions, by delegating physical control to the robot's internal controller, and facilitating the portability of the system to other models of commercial robots. The main objective of the developed system is to automate a solution that minimizes the time required to complete the pick-and-place task. This objective hides various tasks that must be solved in the background, such as generating trajectories for the robots, collision prevention, assignment between robots and pieces, or determining the order of processing the pieces.

This architecture has been progressively developed and refined throughout the course of this research. The initial study establishes the foundations of this architecture, focusing on a pick-and-place operation that integrates a basic robot moving in a plane, with two arms and two degrees of freedom each. The following study extends the capabilities of the system to include a commercial robot and addresses collision issues in a shared workspace. The experiments conducted in this thesis incorporate a bimanual robot known as IRB1400 (YuMi). Finally, the third study introduces specific capabilities for pick-and-place scenarios involving moving pieces on a conveyor belt.

R E S U M E N

El entorno industrial actual requiere que las líneas de producción se adapten rápidamente para satisfacer las demandas de fabricación personalizadas según las necesidades de los clientes. A medida que la tecnología robótica transiciona de sistemas rígidos y preprogramados a entidades colaborativas equipadas con sensores, estos robots se integran en entornos de fabricación más pequeños, trabajando junto con otros para aumentar la eficiencia y ahorrar espacio en el lugar de trabajo. Sin embargo, esta colaboración también trae consigo sus propios desafíos, especialmente cuando se trata de optimizar la manipulación de objetos en un espacio de trabajo compartido por múltiples robots.

Esta tesis propone un enfoque innovador basado en una arquitectura que abstrae o desacopla el robot del resto del sistema. Esta división permite tanto generar soluciones robustas y precisas, al delegar el control físico al propio controlador interno del robot, como facilitar la portabilidad del sistema a otros modelos de robots comerciales. El principal objetivo del sistema desarrollado es automatizar una solución que minimice el tiempo requerido para completar la tarea de pick-and-place. Este objetivo esconde en segundo plano diversas tareas que debe resolver, como la generación de trayectorias para la aproximación y transporte de las piezas, la prevención de colisiones, la asignación entre robots y piezas o determinar el orden de procesamiento de las piezas.

Esta arquitectura se ha desarrollado y extendido gradualmente a lo largo de esta Tesis. El estudio inicial establece las bases de esta arquitectura, centrándose en una operación de pick-and-place que integra un robot básico que se desplaza en un plano, con dos brazos y dos grados de libertad cada uno. El siguiente estudio extiende las capacidades del sistema para incluir un robot comercial y aborda la problemática de colisiones en un espacio de trabajo compartido. Los experimentos realizados en esta tesis incorporan un robot bimanual conocido como IRB1400 (YuMi). Finalmente, el tercer estudio introduce capacidades específicas para escenarios de pick-and-place que involucran piezas en movimiento sobre una cinta transportadora.

A G R A D E C I M I E N T O S

Quisiera expresar mi más sincero agradecimiento a todas las personas que han contribuido de alguna manera a la realización de esta tesis.

En primer lugar, quiero agradecer a mi tutor, Carlos, por su constante apoyo, orientación y estímulo durante todo el proceso de investigación. Su dedicación y sabios consejos fueron fundamentales para alcanzar los objetivos planteados.

También deseo expresar mi gratitud a mis compañeros de laboratorio, Jorge y especialmente a Fran, por su valiosa colaboración y por compartir conmigo su conocimiento y experiencia a lo largo del último año.

Además, quiero dar las gracias a mi pareja y a mis hijos por su amor, paciencia y comprensión durante todo este tiempo. Agradezco su apoyo incondicional y su capacidad para adaptarse a los cambios y las demandas que implicaba mi trabajo en la tesis. ¡Y sobre todo, gracias por no tirarme de casa por desaparecer durante horas frente al ordenador!

Este trabajo ha sido parcialmente financiado por el proyecto CPP2021—008593, MICIU/AEI/10.13039/501100011033 y por la Unión Europea NextGenerationEU/ PRTR.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRÓNIMOS

A3C	Asynchronous Actor-Critic Agent
ACO	Ant Colony Optimization
AI	Artificial Inteligence
APF	Artificial Potential Fields
BFS	Breadth First Search
BUA	Best Uniform Algorithm
Cobot	Collaborative Robot
CSV	Comma Separated Value
DDPG	Deep Deterministic Policy Gradient
DOF	Degress Of Freedom
DT	Digital Twin
EEM	Exhaustive Enumeration Method
GA	Genetic Algorithm
GDL	Grados De Libertad
IFR	International Federation of Robotics
IHLS	Iterated Hybrid Local Search
LSC	Linear Scan Camera
MA-PDDL	Multi-Agent Planning Domain Definition Language
MDP	Markov Decision Process
PCB	Printed Circuit Board
PDDL	Planning Domain Definition Language
PID	Proporcional, Integral y Derivativo
PPDDL	Probabilistic Planning Domain Definition Language
PPO	Proximal Policy Optimization
PSO	Particle Swarm Optimization
QP	Quadratic Programming
RAM	Random Access Memory
RCD	Residuos de Construcción y Demolición
RL	Reinforcement Learning
RNN	Recurrent Neural Network

SAC	Soft Actor-Critic
SEM	Scanning Electron Microscope
SFLA	Shuffled Frog-Leaping Algorithm
VI	Value Iteration

1

INTRODUCCIÓN

1.1 Consideraciones industriales generales

Las líneas de producción están experimentando una transformación significativa debido a la creciente demanda de lotes de fabricación cortos y personalizados adaptados a requisitos específicos de los clientes. Esta tendencia tiene importantes implicaciones para las configuraciones de las fábricas y sus niveles de automatización [1]. Para adaptarse a estos cambios, las fábricas deben mejorar su flexibilidad y adaptabilidad [2][3].

Paralelamente, hay una notable evolución en la robótica. Anteriormente, los robots eran inflexibles, preprogramados y carecían de capacidades sensoriales. Sin embargo, la última generación de robots es colaborativa, fácilmente reprogramable y equipada con múltiples sensores avanzados. Como resultado, estos nuevos robots se emplean cada vez más en líneas de ensamblaje, desmontaje y embalaje, especialmente en pequeñas fábricas o para la producción de lotes reducidos.

La integración de estos robots avanzados en los sistemas de producción permite mejorar la eficiencia y reducir los requisitos de espacio de trabajo al operar junto a humanos en espacios compartidos. Sin embargo, esta configuración cooperativa introduce desafíos en el control de colisiones. La figura 1.1 ilustra un grupo de robots compartiendo el mismo espacio de trabajo y realizando una tarea común de pick-and-place. En este escenario, los robots deben colocar las piezas del mismo color en sus respectivas bandejas sin chocar entre sí, completando la tarea de la manera más eficiente posible.

Esta transición hacia sistemas de producción más flexibles y una robótica avanzada tiene importantes implicaciones económicas. Las empresas que pueden adaptarse rápidamente para producir lotes personalizados de manera eficiente obtienen una ventaja competitiva para satisfacer las demandas de los clientes. Además, la integración de robots colaborativos permite procesos de fabricación más rentables y ágiles, lo que potencialmente conduce a un aumento de la productividad.

Sin embargo, la inversión inicial en la actualización de los sistemas y la adquisición de tecnología robótica avanzada puede plantear desafíos financieros para las empresas más pequeñas. No obstante, los beneficios a largo plazo en términos de eficiencia y competitividad son sustanciales.

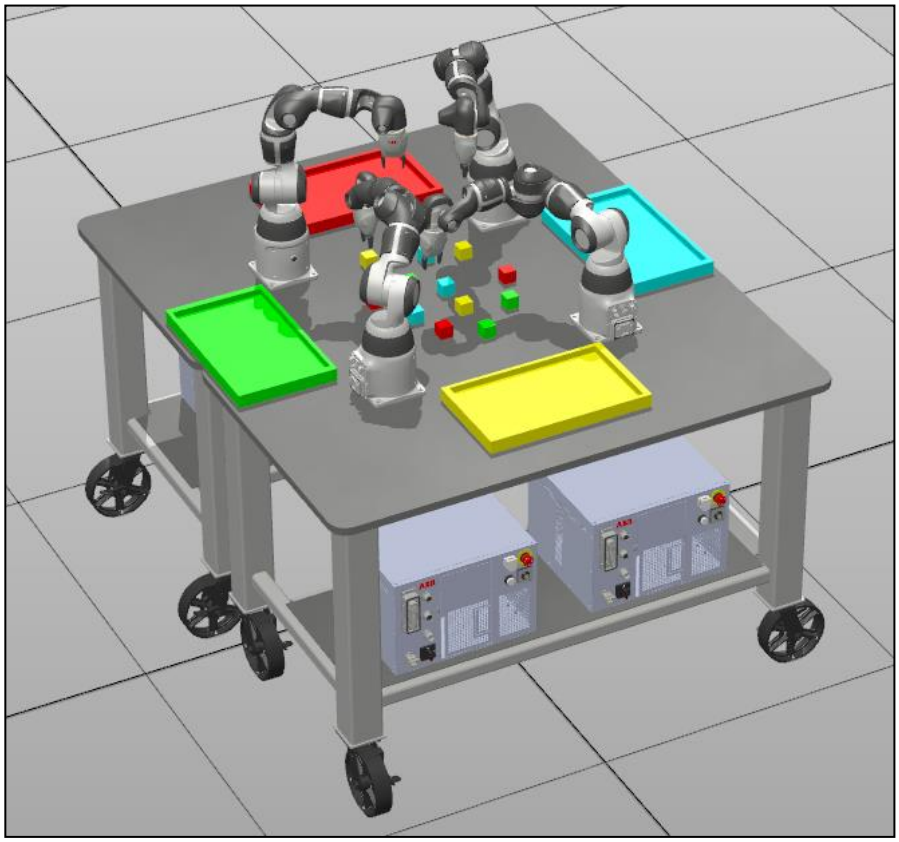


Figura 1.1 Configuración de cuatro robots colaborativos ABB IRB 14050 realizando operaciones de pick-and-place en un espacio de trabajo compartido

1.2 Programación de trayectorias en robótica

Tradicionalmente, las trayectorias de los robots se programan para asegurar que no colisionen con elementos estáticos en su entorno. Esto es relativamente sencillo en tareas repetitivas, ya que la planificación de trayectorias sin colisiones solo necesita hacerse una vez.

Sin embargo, cuando se producen modificaciones en el proceso de producción, como cambios en el diseño o reposicionamiento de elementos de fabricación, se hace necesario replanificar las trayectorias sin colisiones y reprogramar el manipulador correspondiente.

En situaciones donde el entorno es impredecible, como cuando varios robots trabajan juntos de forma autónoma, los operadores están modificando el espacio de trabajo o hay alteraciones en las condiciones del proceso de producción, se necesita un enfoque alternativo.

En ese caso, cada robot puede reaccionar a los cambios en tiempo real. La capacidad de adaptarse de manera óptima a las necesidades de fabricación flexibles sería una ventaja significativa en los nuevos procesos de producción, especialmente desde un punto de vista económico. En primer lugar, es necesario reducir el tiempo de inactividad asociado con la reprogramación de robots o la reconfiguración del espacio de trabajo, aumentando así la productividad general. En segundo lugar, se necesita responder rápidamente a los cambios en la demanda o especificaciones del producto, reduciendo el tiempo de comercialización y potencialmente atendiendo las demandas de producción. Además, los enfoques modulares en robótica pueden ofrecer ahorros de costos al minimizar la necesidad de reprogramación o la compra de sistemas completamente nuevos cuando se requieren modificaciones.

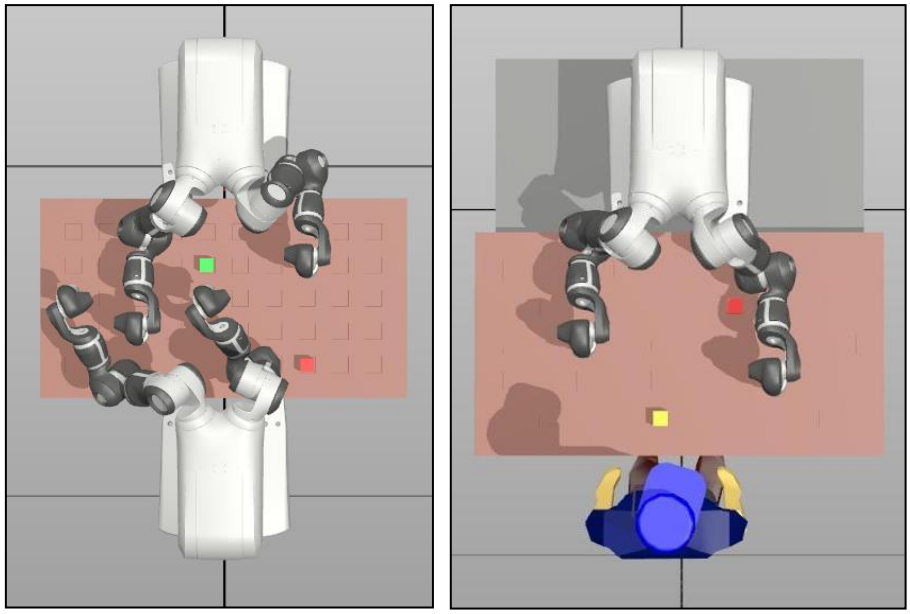
Esta agilidad en los procesos de fabricación puede conducir a una mayor eficiencia, reducción de costos y mayor competitividad en el mercado. En general, invertir en sistemas de robótica modular que puedan adaptarse en tiempo real a los requisitos de producción en evolución puede generar beneficios económicos sustanciales en las líneas de producción.

1.3 Eficiencia en la manipulación robótica colaborativa multirobot

La manipulación concurrente de objetos en la fabricación implica no solo ejecutar tareas evitando colisiones entre robots y el entorno, sino también hacerlo de una manera eficiente (en el menor tiempo posible). El análisis estadístico indica que optimizar el tiempo empleado en las operaciones de pick-and-place es crucial para mejorar la eficiencia general de producción.

Por ejemplo, en el proceso de producción representado en la figura 1.2 se deben realizar una serie de operaciones de pick-and-place en un orden no predefinido.

Esto implica que un cierto número de piezas, como en las operaciones de kitting [4], deben ser manipuladas en cualquier orden estableciendo una secuencia de pick-and-place que minimice el tiempo de operación.



a) b)

Figura 1.2 Configuraciones diferentes de pick-and-place:
a) Dos robots colaborativos ABB YuMi (IRB 14000),
b) Un robot colaborativo YuMi del mismo tipo y un operario compartiendo el mismo espacio de trabajo

En este escenario, se pueden considerar varias configuraciones diferentes. Una opción sería un conjunto de manipuladores, como se muestra en la figura 1.2a, mientras que otra sería una combinación de dos manipuladores trabajando junto a un operador humano, como se muestra en la figura 1.2b. Este último caso es particularmente “problemático” porque el sistema debe adaptarse a la "desaparición" de las piezas procesadas por el operador.

Este cambio repentino en el entorno requiere un recálculo instantáneo de la solución óptima. Desde una perspectiva estadística, optimizar el tiempo de operación de recogida y colocación es crucial para la productividad. Por ejemplo, algunos estudios encontraron que reducir los tiempos de ciclo de pick-and-place en solo un 10% puede conducir a un aumento del 7% en la productividad total.

Además, según datos de la Federación Internacional de Robótica (IFR), la adopción de robots colaborativos (cobots) como los mostrados en la figura 1.2b está aumentando rápidamente, con una tasa de crecimiento anual estimada del 25% entre 2022 y 2026.

Este crecimiento está impulsado por la capacidad de los cobots de trabajar junto a operarios humanos, mejorando la flexibilidad y la productividad en entornos de fabricación. Implementar estrategias para minimizar el tiempo de operación de pick-and-place, especialmente en escenarios con requisitos de orden no predefinido, puede impactar significativamente en la eficiencia de la producción, contribuyendo a una mayor competitividad y rentabilidad en la industria.

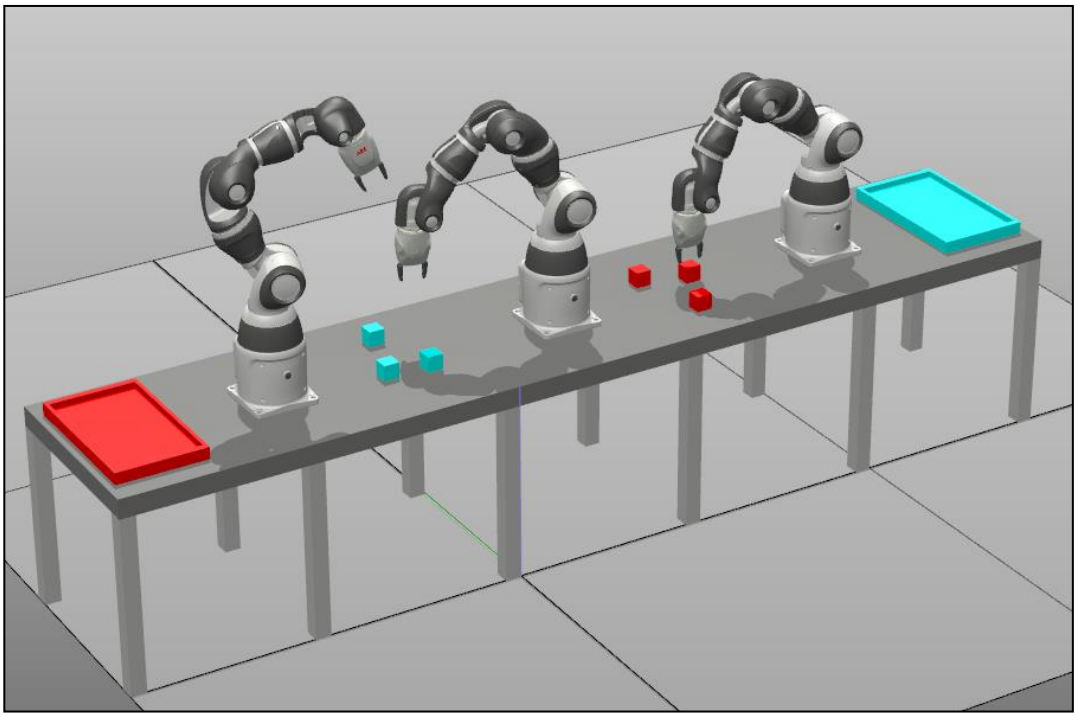


Figura 1.3 Escenario en el que se requiere colaboración entre los robots para completar la tarea

Un caso particular, como el que se ilustra en la figura 1.3, surge cuando un robot (por ejemplo, cualquiera de los extremos) requiere la colaboración de otro (por ejemplo, el robot central) para acceder y colocar una pieza en la ubicación deseada. Esta tarea cooperativa va

más allá de una simple planificación de trayectoria; requiere una planificación estratégica de la tarea, especialmente cuando se busca la eficiencia en la realización de la misma.

En estos casos, coordinar acciones entre robots implica más que solo movimiento; requiere una planificación estratégica para optimizar el tiempo total de finalización de la tarea. Además, factores como la distancia entre los robots, la disponibilidad de recursos y la complejidad de la tarea añaden un desafío adicional. Algunos estudios muestran que las tareas cooperativas entre robots en entornos de fabricación pueden reducir los tiempos de ciclo de producción hasta un 15% en comparación con las tareas realizadas por robots individuales.

Además, varios trabajos de investigación destacan la importancia de la comunicación en tiempo real entre robots para minimizar retrasos y mejorar la productividad general. En situaciones donde los robots necesitan colaborar para lograr un objetivo común, una planificación y coordinación efectiva de la tarea son esenciales para maximizar la eficiencia y la productividad. Esto implica no solo optimizar la planificación de trayectorias, sino también implementar algoritmos avanzados y protocolos de comunicación para garantizar una cooperación sin problemas entre los robots, lo que finalmente conduce a un mejor rendimiento y a reducir los tiempos de producción.

El Capítulo 3 de esta tesis introduce un enfoque innovador para un problema que aún no ha sido resuelto. Presenta una solución diseñada para escenarios cooperativos donde múltiples piezas estáticas y dos manipuladores robots comparten un espacio de trabajo común. Este enfoque pretende automatizar y optimizar el proceso, eliminando la necesidad de intervención humana en la programación de las trayectorias de los robots, y en su lugar se enfoca en minimizar el tiempo de finalización de la tarea. En este capítulo, el enfoque propuesto se aplica a un escenario que incluye múltiples piezas, que van desde 2 hasta 10, y dos manipuladores, en este caso, un robot colaborativo de dos brazos. Al utilizar algoritmos avanzados y técnicas de comunicación en tiempo real, el sistema coordina dinámicamente las acciones de los robots para manejar eficientemente diversas tareas dentro del espacio de trabajo compartido.

1.4 Consideraciones particulares de esta Tesis

En esta tesis, se aborda el problema de pick-and-place con una pose fija para el efector final del robot. Esto significa que la pose del efector final no necesita calcularse para cada punto de

pick o de place, lo que simplifica el cálculo de la trayectoria del robot entre puntos. Este concepto se puede observar en la figura 1.4.

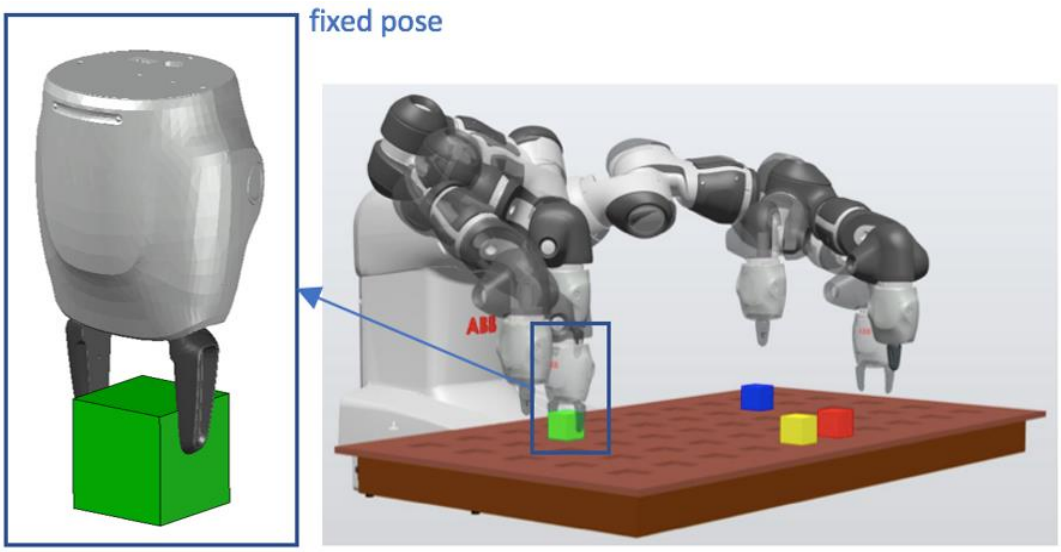


Figura 1.4 Configuración de un robot de doble brazo YuMi: Pose del efector final fija

Este trabajo se enfoca en manipuladores robóticos duales, que están ganando popularidad debido a su capacidad para realizar tareas cooperativas, como trabajar en sistemas automatizados de selección de pedidos [5] y ayudar a personas mayores y discapacitadas a vestirse [6]. Los manipuladores duales ofrecen ventajas en eficiencia y versatilidad, lo que los hace cada vez más comunes en diversas industrias [7][8].

Sin embargo, la programación de manipuladores robóticos duales redundantes presenta varios desafíos:

- Problema de resolución de redundancia: Generar trayectorias en tiempo real es costoso debido a la redundancia del propio manipulador. Existen innumerables soluciones para cada trayectoria, muchas de las cuales son problemáticas por su cercanía a puntos de singularidad.
- Estrategia para evitar colisiones en tiempo real: A medida que los manipuladores robóticos duales ejecutan movimientos, la probabilidad de que se produzcan colisiones

se incrementa exponencialmente. Sin estrategias adecuadas para evitar estas colisiones, las medidas de seguridad encaminadas a evitar daños en los robots, reducen la productividad de la tarea.

1.5 Manipulación multirobot de objetos móviles

Tras la introducción de un nuevo enfoque para manipular piezas estáticas en el Capítulo 3, el Capítulo 4 explora la manipulación de objetos dinámicos. Cuando las posiciones de los elementos de producción cambian, se hace necesario volver a planificar trayectorias libres de colisiones para todos los manipuladores. Esta replanificación es esencial para garantizar un funcionamiento fluido y eficiente en entornos dinámicos [9][10].

Además, las industrias que utilizan cintas transportadoras enfrentan un desafío particular, ya que el movimiento de los objetos introduce una complejidad adicional en el sistema. En tales entornos, cada robot debe seleccionar y manejar un objeto a la vez mientras evita colisiones con otros robots que intentan recoger diferentes objetos en movimiento en el mismo espacio de trabajo.

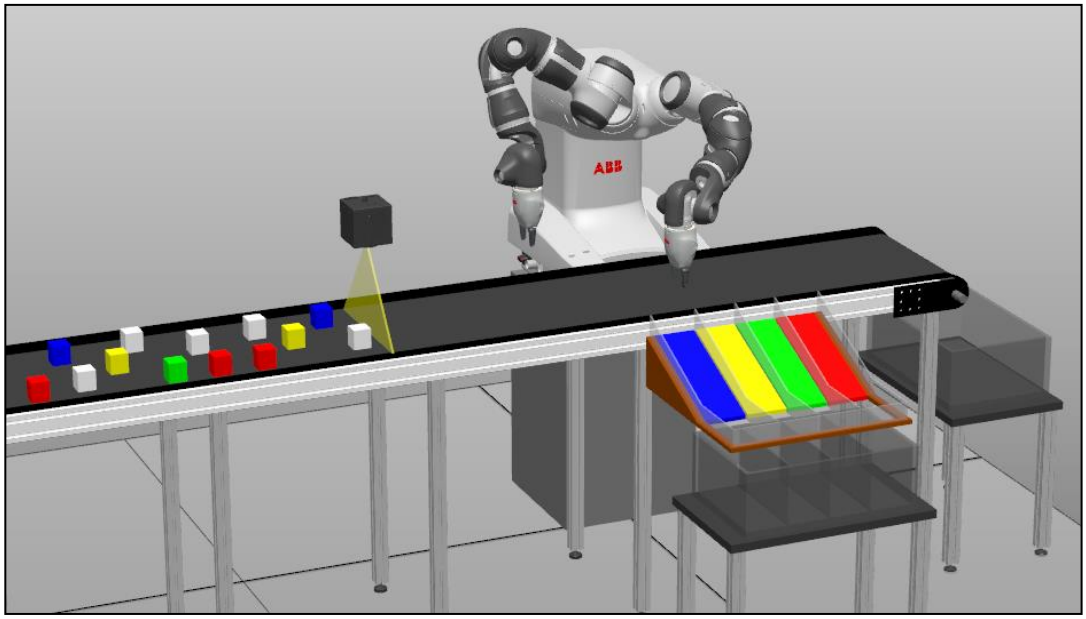


Figura 1.5 Configuración para manipular piezas en movimiento (tarea de clasificación)

Abordar este desafío requiere soluciones innovadoras. A pesar de una extensa revisión de la literatura, no se encontraron referencias sobre el modo de evitar colisiones mutuas para manipular objetos en movimiento, comúnmente conocido como “on-the-fly manipulation”. Debido a la efectividad del enfoque propuesto en el Capítulo 3, se propone una extensión en el Capítulo 4 para manipular objetos en movimiento. Esta tesis proporciona una solución novedosa y robusta al problema representado en la figura 1.5 que muestra una aplicación de clasificación o sorting. La aproximación planteada en esta tesis proporciona una solución novedosa y sub-óptima para la manipulación robótica en entornos dinámicos.

2

ESTADO DEL ARTE

2.1 Operaciones de pick-and-place en la industria

En los últimos años, ha habido diferentes trabajos enfocados en el desarrollo de algoritmos para mejorar la tarea de pick-and-place, especialmente enfocados en reducir el tiempo de operación. Este desafío se ha abordado en diversas aplicaciones, como en la producción de calzado [9], ensamblaje de PCBs [11], y el manejo de nanocables individuales en Microscopía Electrónica de Barrido (SEM) [12]. Estudios anteriores han abordado diferentes aspectos de la optimización de pick-and-place, incluyendo la secuenciación de colocación y la asignación de alimentadores [13].

El esfuerzo por minimizar el tiempo de operación ha sido explorado extensamente utilizando algoritmos metaheurísticos. Debido a la complejidad de estos problemas, se usan comúnmente algoritmos heurísticos y metaheurísticos. Ejemplos de tales algoritmos incluyen la Optimización por Enjambre de Partículas (PSO) [14], Algoritmos Genéticos (GA) [15], y Optimización de Colonias de Hormigas (ACO) [16]. Estos métodos utilizan técnicas de búsqueda estocástica que imitan la evolución biológica y la selección natural. Uno de los beneficios de estos métodos es su tendencia a proporcionar soluciones que a menudo son muy cercanas a lo óptimo en la mayoría de los casos [17].

En este sentido, se han empleado numerosos algoritmos matemáticos para mejorar las operaciones industriales de pick-and-place. En [18], se propone un algoritmo ACO, que supera al Método de Enumeración Exhaustiva (EEM), aunque el estudio no se centra específicamente en procesos de pick-and-place multiobjetivo. Para el ensamblaje de PCBs, los algoritmos GA y ACO proporcionan mejores soluciones en términos de tiempo de ejecución en comparación con los algoritmos PSO y SFLA debido a sus estructuras optimizadas [13]. El algoritmo de Búsqueda Local Iterada Híbrida (IHLS) combina búsqueda local y programación entera para reducir significativamente el tiempo de computación para procesos a gran escala en

comparación con otras heurísticas. En [17], se introduce un nuevo algoritmo metaheurístico llamado Algoritmo de Mejor Uniformidad (BUA), que genera soluciones aleatorias dentro de un espacio de búsqueda para encontrar la secuencia de pick-and-place óptima.

2.2 Minimización del tiempo de operación: Monorobot vs Multirobot

El algoritmo BUA logra resultados satisfactorios en menos tiempo en comparación con el algoritmo GA, aunque para tamaños de problemas más pequeños. Su aplicación se centra en un proceso secuencial de pick-and-place que implicaba un solo efector final moviendo un objeto tras otro de forma secuencial. Un enfoque similar se describe en [9], donde un Algoritmo de Árbol de Decisión optimiza la secuencia de pick-and-place utilizando un robot de dos brazos para posicionar componentes de calzado en un molde. El Algoritmo de Árbol de Decisión se utiliza para reconocer patrones y calcular la mejor secuencia optimizada en tiempo real. Los resultados muestran que la implementación del algoritmo mejora el rendimiento en comparación con la planificación aleatoria.

Aunque [19] tiene un objetivo similar, sigue un enfoque diferente. Esta tesis se centra en la planificación de trayectorias libres de colisiones al priorizar el tiempo para alcanzar el objetivo. Sin embargo, a diferencia del trabajo citado, no integra la preplanificación del orden de la operación para minimizar el tiempo de ejecución de la tarea. Este aspecto en particular se ha resuelto de manera efectiva en el algoritmo propuesto en esta tesis.

En general, los algoritmos diseñados para prevenir colisiones entre manipuladores robóticos dependen de medir la distancia mínima entre ellos. Esta distancia mínima calculada se utiliza en cada intervalo de muestreo para el control en tiempo real y evitar colisiones, lo que requiere del cálculo de la distancia en cada periodo de muestreo. Algunos enfoques calculan la distancia utilizando una simplificación de los eslabones [20], pero a pesar de tratarse de una simplificación, este método sigue incurriendo en un alto coste computacional.

Otros métodos proponen simplificar aún más el proceso al medir la distancia mínima entre dos segmentos de línea que representan los enlaces del robot. Si bien este enfoque permite un cálculo rápido de la distancia, tratar a los robots como un conjunto de líneas conduce a valores de distancia imprecisos, sin tener en cuenta la forma real de los eslabones del robot.

Por el contrario, un enfoque alternativo considera los eslabones como elipsoides, como se demostró en [19], logrando un equilibrio entre precisión y coste computacional. Sin embargo, persisten desafíos en cuanto a la optimización del tiempo de operación y la resolución de redundancias.

El uso de Campos de Potencial Artificial (APF) ha demostrado ser un método efectivo para evitar colisiones entre robots [21][22][23]. Khatib et al. demuestran en [24] que un enfoque de evasión de obstáculos en tiempo real basado en APF, es adecuado tanto para manipuladores como para robots móviles, lo que permite operaciones en entornos complejos. Por otro lado, Volpe et al. [25] introducen una función potencial supercuadrática capaz de generar fuerzas repulsivas para evitar obstáculos.

A pesar de sus beneficios, los métodos APF tienen limitaciones en el manejo de varios tipos de obstáculos y requieren cálculos intensivos, especialmente en escenarios con obstáculos tridimensionales. En consecuencia, estos enfoques son más adecuados para robots móviles y manipuladores no redundantes que para los redundantes.

Zhang et al. [20] proponen una estrategia basada en programación cuadrática (QP) que integra una norma de velocidad mínima, seguimiento de trayectoria deseada y evasión de obstáculos. Resuelven el problema QP utilizando una red neuronal recurrente (RNN) para reducir el coste computacional asociado con el cálculo de la pseudoinversa. Guo et al. [26] mejoran aún más el enfoque de Zhang et al. ajustando ciertos parámetros en la restricción de evasión de obstáculos para minimizar cambios de velocidad discontinuos.

Otros investigadores [27][28][29] también han propuesto métodos para prevenir colisiones. Sun et al. [27] introducen una técnica para calcular la probabilidad de colisión de un planificador de movimiento de un manipulador, considerando suposiciones de movimiento gaussiano y teniendo en cuenta la incertidumbre de detección.

Oh et al. [28] introducen una solución analítica de cinemática inversa considerando límites de articulaciones y evasión de auto-colisiones para un manipulador de 7 grados de libertad (DOF) con articulaciones esféricas en hombro y muñeca. Sin embargo, su estudio se enfoca únicamente en un manipulador redundante individual. Aunque este enfoque podría potencialmente extenderse a dos brazos de un manipulador bimanual, el coste computacional sería extremadamente alto.

Nicolis et al. [30], entre otros, proponen una red neuronal recurrente (RNN) para lograr un movimiento sincronizado de un robot de múltiples brazos para manipulación cooperativa. En este escenario, todos los manipuladores agarran el mismo objeto, por lo que el movimiento de cada uno está limitado por los otros y viceversa. Este problema cooperativo es común en la literatura, existiendo numerosas referencias que lo abordan [31].

Sin embargo, esta Tesis presenta un problema cooperativo donde los efectores finales no están conectados por la pieza que se está manipulando, sino que cada manipulador maneja una pieza diferente, similar a [32] y [33]. En estos casos, las piezas son estáticas, y la principal diferencia entre ellos es que [33] proporciona una solución continua, mientras que [32] ofrece una solución óptima (discretizada) en términos de tiempo de operación.

2.3 Manipulación de piezas móviles en entornos multirobot

Respecto a piezas móviles, no hay referencias en la literatura que aborden este tema más allá de los conocidos sistemas de seguimiento utilizados en la industria para manejar piezas en cintas transportadoras (conveyor tracking).

En este contexto, Yang et al. [34] desarrollan un sistema para realizar tareas de recoger y colocar en una cinta transportadora de aeropuerto para clasificar el equipaje según su destino. En [35], Anton et al. realizaron un análisis comparativo de tres métodos para ejecutar aplicaciones de recogida en movimiento: Utilizando solo una cámara, utilizando una cámara y un encoder, y utilizando una cámara y un sensor basado en láser. Los experimentos realizados muestran que el uso de encoders proporciona un buen rendimiento a bajo costo, pero no abordan el problema del espacio compartido por robots.

Por otro lado, en [36], Dong Ku et al. diseñan un sistema para mejorar la eficiencia de la clasificación de Residuos de Construcción y Demolición (RCD), logrando una alta productividad, pero de nuevo, sin superponer los espacios de trabajo de los robots.

3

OPERACIÓN DE PICK-AND-PLACE CON ROBOT PLANAR 2GDL Y PIEZAS ESTÁTICAS

3.1 Introducción

En este capítulo se presenta una arquitectura de control aplicada a un proceso de pick-and-place dinámico donde múltiples robots operan en el mismo espacio de trabajo, y las piezas están situadas en posiciones arbitrarias en cada iteración del proceso. El objetivo principal es automatizar el control de los robots (o brazos del robot en este caso) en un entorno variante, y minimizar el tiempo de procesado de cada operación de pick-and-place.

Reunir todos estos elementos simultáneamente en un proceso de pick-and-place supone un desafío que requiere resolver otros problemas derivados, como evitar colisiones entre los robots que operan en un espacio común, determinar el orden apropiado de procesado de las piezas, determinar la asignación más ventajosa entre piezas y brazos y generar las trayectorias de los robots para aproximar y transportar las piezas. Comúnmente los procesos de pick-and-place en la industria no incorporan todos estos componentes. O bien solo opera un único robot en el proceso, u operan múltiples robots, pero no comparten un espacio de trabajo, o si comparten el espacio de trabajo se trata de un proceso repetitivo, donde la posición inicial y final de las piezas es la misma en cada iteración. En cualquiera de los casos anteriores, las trayectorias pueden ser diseñadas y predefinidas de antemano.

La arquitectura de control presentada en este capítulo sienta las bases de la empleada en los próximos capítulos. En este caso, la arquitectura se somete a prueba en un escenario simple que incorpora un robot de dos grados de libertad, conocido en adelante como robot planar 2GDL. La viabilidad del sistema planteado en este estudio depende de varios factores, tales como la robustez de la solución, la precisión en los movimientos del robot, y los recursos necesarios (cómputo y memoria) necesarios para aplicar este sistema en tiempo real.

3.2 Materiales y métodos

3.2.1 *Materiales*

En esta configuración inicial, el sistema está formado por un robot bimanual y un conjunto de piezas estáticas presentes en la zona de trabajo del robot. El objeto de esta configuración básica es desarrollar un método funcional para la operación de pick-and-place, extensible posteriormente a robots comerciales. El robot empleado para esta etapa es un modelo simplificado de un robot que se mueve en el plano horizontal. Está formado por un par de brazos de 2 grados de libertad cada uno. y cada brazo está modelado por segmentos lineales, como se puede observar en la figura 3.1. Las piezas se encuentran en el mismo plano que el robot, y están modeladas por puntos.

El objetivo de la tarea es transportar un determinado número de piezas a sus respectivos destinos. La posición inicial de las piezas se conoce al iniciar la tarea, mientras que la posición final está prefijada de antemano. Este comportamiento simboliza una situación hipotética donde un conjunto de piezas, inicialmente presente en posiciones arbitrarias, debe ser transportado a sus respectivos depósitos.

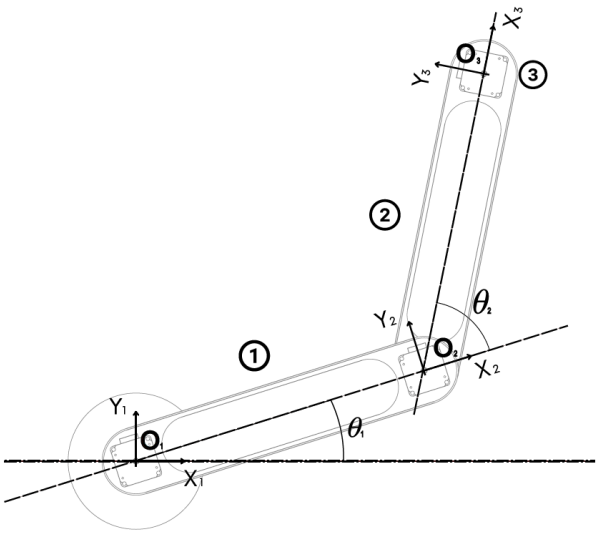


Figura 3.1 Descripción del robot planar 2GDL

3.2.2 *Diseño de la arquitectura basada en la discretización del espacio de trabajo*

El objeto principal de este estudio es introducir un método que sea escalable a procesos que cuenten con un robot comercial. En particular, este método debe soportar configuraciones que incluyan más de un robot operando en la misma área de trabajo. La arquitectura planteada en este capítulo es la base de la empleada en posteriores capítulos donde el robot YuMi de ABB es introducido en el setup.

En este capítulo, el robot presentado es un robot planar de dos brazos, con dos grados de libertad cada uno. Es una simplificación respecto al tipo de robots que se pueden encontrar en la industria, con el propósito de centrar el esfuerzo en el diseño de la arquitectura del sistema. Esta arquitectura pone el foco en los siguientes factores:

1. **Portable a robot comercial**. La arquitectura desarrollada para el robot planar debe ser escalable a un caso de uso con un robot comercial.

2. **Orientado a aplicación práctica**. El sistema debe ser aplicable en un proceso en tiempo real

3. **Robusto y preciso**. Es esencial que el posicionamiento de los robots sea exacto para garantizar resultados óptimos durante la operación de pick-and-place. Así mismo, el control del robot debe ser robusto para garantizar una consistencia en la operación.

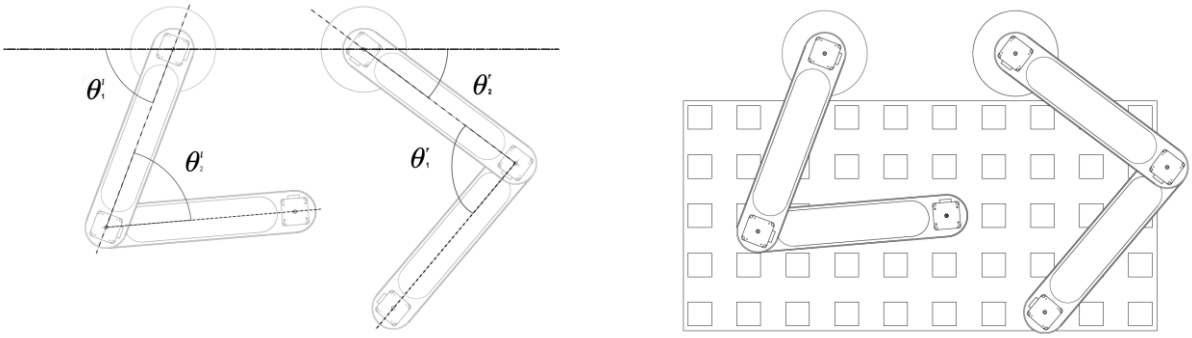
Comúnmente, los robots comerciales son más complejos, y cuentan con un mayor número de grados de libertad. En concreto, el robot YuMi empleado en esta Tesis consta de 14 GDL y cada uno de estos grados de libertad presume de una alta resolución angular. El coste computacional de un enfoque basado en el control directo de los servos que regulan la posición angular de cada grado de libertad es enorme, lo cual no favorece a una potencial aplicación práctica del sistema. Del mismo modo, un control directo de las articulaciones del robot introduciría numerosos problemas a tratar, algunos de ellos ya resueltos por el fabricante del robot, tales como la precisión y robustez en el movimiento.

Para el caso particular del robot planar 2GDL, asumiendo una resolución angular baja (ej.: 5 grados), y un rango angular de 180 grados, el número de potenciales configuraciones del robot bimanual sería:

$$\text{Num\_configuraciones} = ((\Psi/\ \Theta)^{L})^{N} = ((180/5)^{2})^{2} = 1.679.616$$

donde Ψ es el rango angular medio de los enlaces del robot, Θ la resolución articular del robot, L el número de enlaces que conforma cada robot y N el número de robots.

El número resultante es enorme, incluso en estas condiciones simplificadas del problema. Por esta razón el enfoque empleado en este estudio es diferente, La arquitectura está basada en la discretización del espacio de trabajo de los robots. En este caso, tratándose de un robot planar, la discretización da lugar a una rejilla en dos dimensiones.



a) b)

Figura 3.2: Estrategia de control del robot, a) basada en configuración articular, b) basada en discretización del espacio de trabajo

La constelación de puntos que conforma esta cuadrícula 2D se emplea como puntos de referencia que la trayectoria del robot debe seguir. Específicamente, por trayectoria se entiende el camino que recorre el efector final del robot. En este enfoque, el número de configuraciones potenciales definidas para los robots (o robot bimanual) depende directamente del tamaño y resolución de la cuadrícula definidas. Asumiendo una cuadrícula de 10x5, como se observa en la figura 3.2, este valor sería tremendamente inferior al obtenido anteriormente:

$$\text{Num\_configuraciones} = (X \cdot Y)^{N} = (10 \cdot 5)^{2} = 250$$

Este enfoque reduce drásticamente el coste computacional del método empleado para la operación de pick-and-place, aunque ignora una serie de factores críticos, tales como el espacio que el robot ocupa en la zona de trabajo, la consideración de posiciones entre puntos de la rejilla, la operación de pick o place en sí, etc… Estos factores no pueden ignorarse y deben ser introducidos de una manera alternativa, que mantenga la dimensión del problema contenida.

El espacio que los robots ocupan en la zona de trabajo determina si existe colisión entre ellos dada una determinada configuración articular de éstos. En este estudio inicial, el cuerpo de los robots se representa mediante segmentos lineales de grosor infinitesimal, uno por cada enlace del robot. En los capítulos posteriores, el robot empleado en la investigación sí ocupa un volumen complejo en el espacio 3D, y esta situación se aborda de una forma diferente.

En el caso del robot planar, la detección de colisiones se reduce a la comprobación de colisión entre los segmentos que conforman cada brazo del robot.

De igual modo, existen limitaciones en la zona de influencia de los robots. Es una configuración multi-robot, además de la zona común de trabajo, es muy probable que existan algunas zonas exclusivamente alcanzables por un subconjunto de los robots. Esto depende de muchos factores, como la definición de la zona de trabajo, la colocación de los robots, y el área de influencia de cada uno. En este sentido, alguna de las piezas, dependiendo de su localización, podrían ser procesadas solo por un subconjunto de los robots.

En el caso del robot planar, la determinación de factibilidad para un robot en una posición determinada de la cuadrícula está basada en el cómputo de la cinemática inversa. Si no se encuentra una configuración articular que satisfaga dicha posición del efector final, la posición se considera inalcanzable. Esta solución también podría ser aplicable en los capítulos posteriores para el caso del robot YuMi, aunque se emplea una alternativa diferente equivalente.

Otro de los factores ignorados en el modelo de rejilla son las posiciones existentes entre puntos de la cuadrícula. Es probable que alguna pieza esté situada en una de estas posiciones. Esta situación afectaría exclusivamente a la operación de pick o place, donde el efector del robot debe posicionarse con precisión para realizar la operación. Para afrontar esta situación, el sistema mantiene una relación entre los puntos de la cuadrícula y las coordenadas XY en el plano. Esta relación es directa para el caso general, donde cada punto de la cuadrícula está separado 100mm de sus adyacentes, pero no es obligatorio que la rejilla 2D se corresponda

con una constelación de puntos equidistantes en el plano. Si la posición inicial o final de una pieza no se corresponde con ningún punto de la cuadrícula, el punto de la cuadrícula más cercano se asocia a esta nueva posición XY.

Las trayectorias generadas para cada robot vienes determinadas por una secuencia ordenada de puntos de la cuadrícula. El paso obligado por dichos puntos del grid puede resultar en una trayectoria no necesariamente óptima entre dos puntos A y B. pero habilita esta arquitectura basada en la discretización del espacio de trabajo.

Otro de los factores ignorados en este modelo de rejilla es la operación de pick o place. En un proceso real, el robot probablemente se moverá por un plano Z superior al que se sitúan las piezas. Durante esta operación, el robot descenderá para agarrar o soltar la pieza, y posteriormente volverá a ascender al plano Z inicial. Esta secuencia de acciones está integrada en la arquitectura del setup que incluye el Robot YuMi, pero en este caso, con el robot planar, la operación de pick o place se reduce a situarse en las coordenadas XY apropiadas y ejecutar la acción de pick o place.

Un factor final que no pasa por alto en este modelo de rejilla es que una determinada posición del efector final de un robot puede ser obtenida mediante diversas configuraciones articulares. Esta redundancia puede complicar la planificación del movimiento del robot. En el caso del robot YuMi, introducido en el próximo capítulo, la cantidad de grados de libertad por brazo es 7. Los grados de libertad necesarios para posicionar y orientar un punto (efector final del robot) en el espacio XYZ son 6. El grado de libertad adicional permite que existan múltiples configuraciones articulares que representan la misma posición del efector final. La cantidad precisa de configuraciones articulares que pueden lograr la misma posición del efector final se ve influenciada por diversos elementos, como el diseño particular del robot, las restricciones de las articulaciones y la ubicación del efector final. Esta situación está

Para un robot planar que cuenta con solo dos grados de libertad, existen dos configuraciones articulares que pueden establecer la misma posición. Dado que el setup incluye dos brazos o robots de estas características, la elección de la configuración correcta es sencilla. Es preferible que el codo que forma cada robot apunte al extremo opuesto del otro robot, como indica la figura 3.3b. Esta configuración reduce las probabilidades de colisión entre brazos.

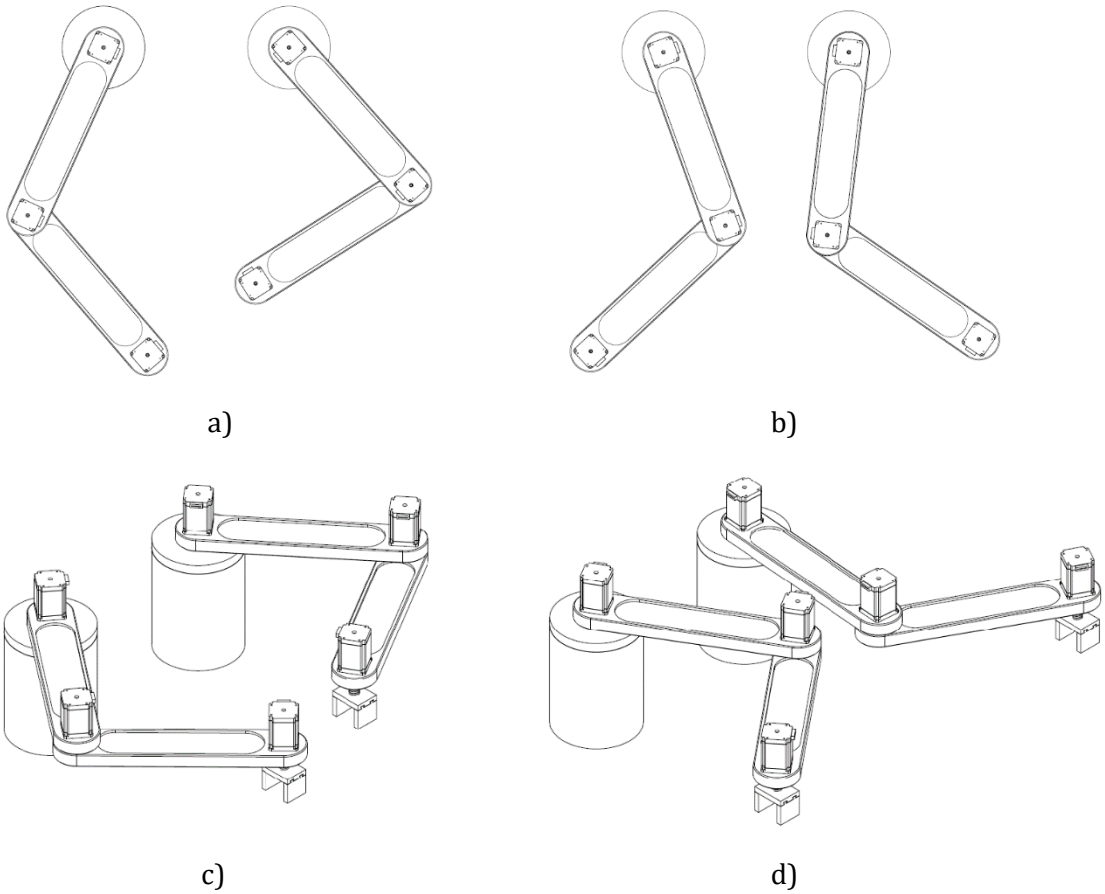


Figura 3.3 Configuraciones articulares en el robot planar de 2GDL por brazo,
a) codos hacia el exterior, b) codos hacia el interior, c) codos hacia el exterior (vista 3D),
d) codos hacia el interior (vista 3D)

3.2.2.1 Sincronización temporal de los robots

En una tarea de pick-and-place donde múltiples robots operan en una zona de trabajo compartida, es necesario asegurar que no se producen colisiones entre éstos. Para ello, los robots deben actuar colaborativamente, cediendo el paso a otro robot si es necesario para conseguir un objetivo global.

En este trabajo, la lógica de control está centralizada. El controlador tiene visibilidad completa del estado de cada uno de los robots y es responsable de la toma decisiones para cada uno de ellos. Estas decisiones se aplican de forma síncrona a todos los robots y la cadencia con la que se ejecutan se le asigna el término de tiempo de paso. Esta nomenclatura será la usada para representar este concepto en el resto del documento.

3.2.2.2 Movimientos soportados

La sincronización de las acciones de los robots implica la existencia de algunas limitaciones en el tipo de movimientos que cada robot puede realizar en cada paso de tiempo. Si en un paso de tiempo, un robot tiene que recorrer una distancia mucho mayor que otro, la única manera de mantener esta sincronización es usando velocidades dispares para cada robot, o añadiendo tiempos de espera para los robots que terminan la acción primero. Ninguna de estas opciones es aceptable. En determinadas situaciones, un robot se movería extremadamente lento, o rápido, o incluso variando significativamente su velocidad a lo largo de su trayectoria, o realizando mini etapas con pausas a lo largo de su trayectoria.

Para evitar esta situación, la arquitectura define que todos los robots deben realizar un recorrido similar en cada paso de tiempo. Es más, que todos los robots deben realizar un recorrido similar en cualquier paso de tiempo. Esta condición se implementa limitando las acciones soportadas por el robot a transiciones entre puntos adyacentes de la rejilla 2D resultante de la discretización del espacio de trabajo de los robots. Estos puntos son equidistantes, siendo la distancia entre ellos de 100mm para la configuración seleccionada en este trabajo.

En base a esta decisión, los movimientos soportados por un robot son los cuatro movimientos ortogonales en el plano (este, oeste, norte y sur). En cada paso de tiempo, el destino del movimiento es el punto adyacente de la cuadrícula encontrado en la dirección tomada.

En el siguiente capítulo, con la introducción del robot YuMi, la cantidad de posibles movimientos aumenta, soportando movimientos del efector final en las tres dimensiones. Los movimientos diagonales en el plano realizan un recorrido ligeramente superior al de los ortogonales. A su vez, el recorrido de las diagonales en las tres dimensiones es superior al de

las diagonales en el plano. El sistema es consciente de ello y aplica un factor corrector a la velocidad del robot con el fin de mantener un tiempo similar para todos los desplazamientos.

3.2.2.3 Concepto de trayectoria

La trayectoria entre dos puntos A y B de la cuadrícula está representada mediante una secuencia ordenada de puntos adyacentes de la misma cuadrícula. Estos puntos determinan la zona de paso de la trayectoria. Cada tiempo de paso, el robot se dirige hacia el siguiente punto de referencia en la trayectoria, hasta completarla. Esta operación se realiza en cada robot simultáneamente. Dado que las distancias recorridas por cada robot en cada paso de tiempo son iguales, el tiempo invertido en esta operación es el mismo, y las trayectorias obtenidas son suaves y continuas.

Las trayectorias generadas para cada robot son independientes, y la duración y el momento en que se inician no es necesario que estén coordinados. Mientras un robot está llegando a una pieza, otro puede estar empezando su trayectoria de aproximación a otra pieza, o incluso estar realizando la operación de descenso para coger la pieza. La sincronización entre robots sucede nivel de tiempo de paso. En cada tiempo de paso, cada robot avanza hacia su siguiente posición objetivo dentro de su trayectoria.

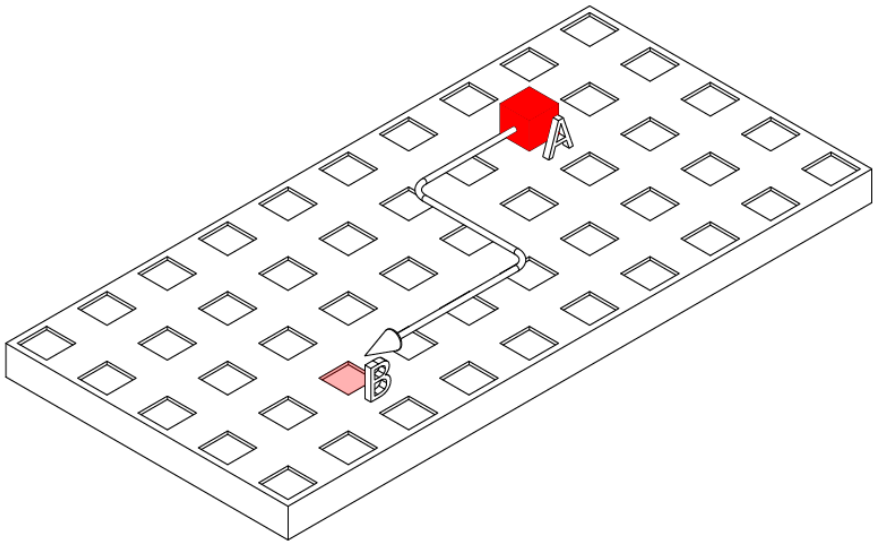


Figura 3.4 Representación de la trayectoria del robot guiada por un conjunto ordenado de puntos adyacentes en el espacio de trabajo discretizado.

En el trabajo expuesto en el capítulo siguiente, el número de movimientos soportados en esta arquitectura por el robot comercial YuMi aumenta para incluir desplazamientos por el espacio 3D. Asimismo, se soportan movimientos diagonales en las tres direcciones del espacio.

3.2.2.4 Arquitectura de doble capa

La arquitectura diseñada para la operación de pick-and-place está basada en la discretización del espacio de trabajo del robot. Este enfoque permite reducir la dimensión del problema, habilitando una metodología capaz de proporcionar una solución en tiempo real. Sin embargo, para realizar la transición entre dos puntos de la cuadrícula debe posicionarse en multitud de posiciones intermedias, las cuales no están contempladas en el modelo de rejilla.

Este problema se resuelve delegando el movimiento entre puntos de la rejilla al robot., usando instrucciones estándar que cualquier robot comercial suele soportar, tales como dirigirse a determinada posición cartesiana en 3D, o realizar una transición a una nueva configuración articular. Para el caso del robot planar simulado, los movimientos consisten en transiciones directas al siguiente punto de referencia en la trayectoria. La representación de un movimiento entre puntos no aporta nada al sistema planteado. En el estudio del próximo capítulo, la simulación con el robot YuMi ya incluye las transiciones completas entre puntos del grid, y dichas transiciones son ejecutadas por el propio robot. La arquitectura planteada no realiza un control a bajo nivel de los servos del robot, sino que opera encima de las capacidades del propio robot

3.2.2.5 *Robustez y precisión en los movimientos*

En esta arquitectura, la trayectoria de un robot se representa como una secuencia ordenada de puntos de la rejilla, y siguiendo esta filosofía de doble capa, los movimientos entre puntos son gestionados por el controlador interno del robot. La robustez y precisión en los

movimientos viene dictaminada por las capacidades que el robot ofrece. Este enfoque de doble capa, donde la capa superior toma decisiones, y las transforma en instrucciones entendibles por el robot, genera soluciones muy robustas y precisas.

En el caso del robot planar, al tratarse de un robot básico simulado, estos matices no aplican. La precisión y repetibilidad en los movimientos son perfectas. En el caso del robot YuMi, u otro robot comercial, este factor es diferencial respecto a otros enfoques basados en el control directo de los servos del motor. En este contexto, realizar un control a tan bajo nivel implicaría prescindir de toda la capa de control que el propio fabricante aporta.

3.2.2.6 Operación de pick-and-place

La operación de pick-and-place no viene representada en el modelo de rejilla. Este modelo se focaliza en el movimiento realizado en el plano para aproximar o transportar una pieza. La operación de pick o place, normalmente requiere un movimiento descendente, una apertura o cierre de pinza y una ascensión para volver al plano de movimientos.

En el robot planar, la operación de pick-and-place está simplificada. Cada brazo del robot es capaz de recoger, transportar y depositar piezas, pero el modelo del brazo per se, no incluye una pinza para realizar dicha operación. El único requisito es estar posicionado en las mismas coordenadas XY que la pieza. Tras ejecutar la acción de pick, se considera que el robot ha recogido la pieza, y tras ejecutar la operación de drop, se considera que la pieza ha sido depositada.

En el caso del robot YuMi, expuesto en el siguiente capítulo, los movimientos para aproximar o transportar piezas se realizan en el espacio tridimensional, y las operaciones de pick-and-place están debidamente modeladas (descenso de la pinza, apertura o cierre y ascenso de la pinza). No obstante, al igual que en este capítulo, las operaciones de pick-and-place no forman parte del modelo de rejilla tridimensional. El plano de las piezas no forma parte del área de movimiento, por lo tanto, el ascenso o descenso de la pinza no se realiza mediante los movimientos estándar (subir y bajar). La operación completa de pick-and-place se modela de un modo alternativo. La razón detrás de esta decisión es evitar añadir en el área de movimientos un nuevo nivel de altura con la posibilidad de recalar en cualquiera de sus ubicaciones. Al fin y al cabo, la operación de pick-and-place solo se puede realizar de determinadas posiciones.

3.2.2.7 Arquitectura de doble capa frente a control de las articulaciones del robot

Una estrategia basada en el control directo de los servos que controlan la posición articular del robot elevaría la complejidad del problema. Por un lado, la cantidad de diferentes combinaciones articulares para N robots es inmensa, incluso para el caso del robot planar 2GDL. Por otro lado, este enfoque no aprovecharía las capacidades del robot comercial, donde la problemática de robustez y precisión está resuelta por el propio fabricante. Uno de los propósitos de la arquitectura diseñada en este estudio es la potencial aplicabilidad en un proceso real. Para ello, el coste computacional durante la operación de pick-and-place debe estar contenido, y la precisión y robustez en los movimientos debe estar asegurada.

Una estrategia de dos capas basada en la configuración articular de los robots, aprovecharía las capacidades del robot. La capa superior sería la encargada de resolver la lógica de más alto nivel, como el orden correcto de procesado de las piezas, la asignación entre piezas y robots, las trayectorias de los robots, etc… La capa inferior consistiría en la capa de software proporcionada por el fabricante. Este enfoque todavía padece de una complejidad elevada debido a la gran cantidad de combinaciones articulares entre robots. Esto no solo tiene unimpacto en el estado actual del sistema, que se representa en parte por la configuración articular de los robots, sino también en las acciones potenciales que se pueden llevar a cabo en él, que también pueden ser cualquier combinación deseada de las articulaciones de los robots. Es decir, el tamaño del estado del sistema crece exponencialmente con el número de robots. Aún en configuraciones con dos robots, la cantidad de estados diferentes puede ser enorme. Para el caso simple del robot planar 2GDL, asumiendo una resolución articular de 1 grado, y un rango articular de 180 grados, la cantidad de diferentes combinaciones articulares con ambos brazos sería:

$$\text{Tamaño\_estado} = ((\text{num\_configuraciones})^{L})^{N} = (180/5)^{2})^{2} = 1.679.616$$

Adicionalmente, el estado del sistema no solo cuenta con la configuración articular de los robots, sino que debe modelar otras características del proceso, tales como el estado actual de la operación (posición inicial y destino de las piezas, qué piezas han sido ya procesadas, cuales están siendo transportadas, qué brazo está transportando cada pieza, etc…).

El tamaño total del estado del sistema todavía incrementaría más tras considerar el total de información modelada por éste.

De igual manera, la cantidad de acciones posibles sería equivalente al tamaño del estado del sistema, ya que un robot podría pasar de una configuración articular específica a cualquier otra. Mantener una relación tabular entre estados del sistema y acciones deseables en cada de uno estos es apenas posible en el caso del robot planar 2GDL y no es viable en configuraciones que incluyan robots más complejos, como es el caso del robot YuMi con 14 grados de libertad.

Emplear técnicas de aproximación de funciones para modelar esta relación ayudaría a contener el coste de almacenamiento a expensas de un mayor coste computacional, tanto en la fase de entrenamiento como en la de inferencia. Obtener esta relación estado-acción, es decir, seleccionar la acción más apropiada según el estado actual, es el propósito de cualquier método empleado para resolver el problema. El coste computacional para obtener esta relación mediante aproximadores de funciones, tales como redes neuronales, es enorme y no puede realizarse en tiempo real. La alternativa sería generar una solución de antemano, en el laboratorio, para lo cual deberíamos tener en cuenta cualquier combinación de posiciones iniciales de las piezas, lo cual aumentaría, si cabe aún más la complejidad del problema. En última instancia, está relación no dejaría de ser una aproximación a la relación óptima estado-acción, por lo que medidas adicionales externas deberían introducirse para vigilar resultados inesperados, como colisiones.

Otra alternativa aplicable al enfoque basado en configuraciones articulares, consistiría en reducir la cantidad de configuraciones disponibles en cada robot. La dificultad en este caso reside en definir un buen criterio para descartar un gran número de ellas. A modo de referencia, una estación de trabajo donde operase un robot de 14GDL, como YuMi, y asumiendo una resolución angular de 1 grado y un rango angular de 180 grados de media, elevaría el número de posibles configuraciones del robot a:

$$((\text{num\_configuraciones})^{L})^{N} = (180/1)^{14} \simeq 4*10^{31}$$

Basar la reducción exclusivamente en degradar la resolución angular de cada articulación no sería una medida aceptable por varias razones. La primera es que incluso con resoluciones

angulares pobres, como por ejemplo 10 grados, la dimensión del problema sigue siendo considerable ($18^{14} \simeq 4*10^{17}$). Además, la operación de pick-and-place requiere cierta precisión en la configuración del robot, lo cual no se conseguiría con una resolución articular pobre. Por último, discretizar uniformemente a lo largo de todas las articulaciones no parece una estrategia inteligente. Dentro de esta discretización existirán muchas configuraciones poco útiles (por ejemplo, el robot con la pinza apuntando hacia arriba).

No obstante, encontrar un criterio válido para descartar configuraciones articulares, reducir la complejidad del sistema y habilitar la generación en tiempo real de una solución aplicable a robots comerciales, es el objetivo de la arquitectura definida en este estudio.

Como se ha descrito a lo largo del capítulo, este criterio está basado en numerosas técnicas empleadas simultáneamente. La idea principal es discretizar el espacio de trabajo de los robots. En los estudios realizados en esta Tesis, la discretización empleada es 10x5x3, y en el caso particular del robot planar 2GDL es 10x5. La constelación de puntos resultante de esta discretización hace referencia a la posición cartesiana del efector final del robot, en particular la pinza en el problema de pick-and-place planteado. Las posibles configuraciones del robot desde este punto de vista son drásticamente inferiores. Aún así, un robot con 7GDL de libertad por brazo puede disponer de múltiples configuraciones que cumplen el criterio de posicionar el robot en cualquiera de los puntos definidos en la rejilla resultante de la discretización. Considerando que la orientación del efector final no es relevante durante la fase de aproximación a las piezas o transporte de éstas, se ha definido por diseño que el efector final siempre apunte hacia la mesa de trabajo donde están colocadas las piezas. Esta decisión reduce en mayor manera la cantidad de configuraciones articulares disponibles, aunque todavía existen muchas de ellas que siguen cumpliendo este requisito. Dado que tampoco es relevante una configuración concreta que cumpla este requisito de posicionamiento y orientación de la pinza, se ha definido una única configuración articular por cada posición de la rejilla, para cada robot. El criterio seguido para seleccionar estas configuraciones está basado en asegurar una trayectoria suave del robot entre dos puntos A y B, y en disminuir las posibilidades de colisión entre brazos.

La posibilidad de colisión entre brazos se reduce seleccionando configuraciones articulares más ventajosas, donde el codo de ambos brazos apunta hacia el exterior, en dirección opuesta al otro brazo.

Conseguir una trayectoria suave implica no llegar a situaciones de límite angular en ninguna de las articulaciones del robot. Si esto ocurre, el robot tendría que realizar una transición a una configuración articular completamente diferente, que cumpla con el posicionamiento deseado del efector final del robot, y el movimiento realizado durante esta transición rompería la suavidad en la trayectoria realizada.

Este modelo de rejilla ignora factores críticos como el volumen que ocupa el robot en el espacio de trabajo, el cual determina si existe colisión en una configuración dada, o el diseño del robot, que determina si es capaz de alcanzar determinadas posiciones. Toda esta información (colisiones, configuración articular y alcanzabilidad) está contenida en una base de datos que se genera en el laboratorio, y que se consulta en tiempo real. Esta base de datos sería específica para cada robot comercial.

Otro de los factores que impactan en la complejidad del problema es la cantidad de acciones disponibles para cada robot. En el contexto de modelo de rejilla, se reduce a transiciones entre posiciones de ésta. Por diseño, se ha definido que las acciones disponibles son las que realizan una transición a alguna de las posiciones adyacentes en la rejilla. En el caso del robot YuMi, incluye movimientos ortogonales y diagonales en el plano y también entre planos adyacentes. En el caso del robot planar de este capítulo, se permite solo movimientos ortogonales en el plano. Bajo esta decisión, el concepto de trayectoria se define como un conjunto ordenado de posiciones por donde el efector final del robot debe pasar. Esta decisión también simplifica la sincronización entre robots y el cómputo de colisiones entre éstos. Todos los robots realizan un recorrido equivalente en cada transición entre puntos de la rejilla. La resolución de la rejilla se considera suficientemente alta como para asegurar que no hay colisión entre robots durante la transición entre dos puntos de ésta si no existe colisión en sus posiciones iniciales y finales.

Para asegurar una buena gestión de las colisiones en el sistema, las acciones indicadas al robot se presentan en forma de configuración articular, en lugar de posición cartesiana del efector final del robot. Esto permite tener un control total de la configuración adquirida por el robot en cada posición de la rejilla, lo cual no podría conseguirse indicando una determinada posición 3D del robot, ya que el software interno del robot tendría libertad para elegir cualquier configuración articular que cumpliera con el requisito.

3.2.3 *Diseño del modelo del sistema*

Los objetivos principales establecidos para este sistema son automatizar la tarea de pick-and-place y minimizar el tiempo que se tarda en realizarla. Existe una serie de factores inherentes a las tareas de pick-and-place que el método planteado debe tener en cuenta:

1. **Secuencia de pick-and-place**. En un proceso con múltiples piezas, el sistema debe determinar el orden óptimo de procesado de las piezas con el objetivo de minimizar el tiempo de la tarea.
2. **Asignación de piezas**. La posición de los robots y de las piezas tienen un impacto importante en el reparto de las piezas entre los brazos.
3. **Prevención de colisión entre robots.** En un sistema con múltiples robots en la misma zona de trabajo, coordinar el movimiento de todos ellos es necesario para evitar colisiones.
4. **Generación de trayectorias.** El recorrido que realiza cada brazo para aproximarse a una pieza o transportarla a su destino influye significativamente en el tiempo requerido para la tarea.

El orden de procesado de las piezas impacta en el conjunto de trayectorias que los robots deben realizar para completar la tarea. Se trata de una tarea colaborativa donde el objetivo global es transportar una serie de piezas. En algunas situaciones, un robot deberá ceder el paso a otro con este fin global. El orden óptimo de procesado depende de los siguientes factores: posición inicial de las piezas, destino de éstas, localización de la base de los robots, posición inicial del efector final de los robots y diseño o implementación particular del robot en cuestión, que determina las situaciones de colisión o área de influencia.

La asignación o reparto de piezas entre los robots también tiene una gran influencia en las trayectorias resultantes. Minimizar globalmente el recorrido realizado por los robots es equivalente a minimizar el tiempo necesario para completar la tarea. En algunas situaciones, puede ser óptimo asignar dos piezas al mismo robot en lugar de repartirlas entre ambos robots. La asignación óptima de piezas depende de los mismos factores que el orden de procesado de éstas.

En una estación de trabajo donde múltiples robots operan compartiendo el espacio, es importante poder evaluar las situaciones de colisión entre robots para poder generar trayectorias libres de éstas.

El modelo del sistema es una simplificación del proceso real. Debe contener toda la información relevante del proceso y excluir la innecesaria a la hora de obtener una solución al problema. Características como el color o peso del robot o la temperatura ambiente de la sala son datos del mundo real irrelevantes para valorar una solución en este caso. Otros datos, como la posición de las piezas, son cruciales. En el contexto de este estudio, las piezas están modeladas como piezas cúbicas, las cuales son agarradas por el centro por la pinza que el robot integra, En un contexto diferente, la forma de las piezas podría ser más compleja, y esta información podría requerirse como parte del modelo del sistema.

3.2.3.1 Espacio de estados

El concepto de espacio de estados hace referencia al conjunto de diferentes estados en los que se puede encontrar el sistema (específicamente el modelo del sistema). El estado no solo consiste en la posición de los robots, sino en el conjunto de todas las variables necesarias para modelar la esencia del problema. En el caso de estudio de esta Tesis el sistema es totalmente observable, lo cual significa que todas las variables del sistema son medibles y accesibles por el algoritmo. Considerando que en este capítulo se describe un sistema de pick-and-place que integra un robot planar 2GDL y 4 piezas, las variables necesarias para modelar el estado son la posición de los dos brazos del robot, y el estado de cada uno de los brazos y piezas.

La posición del robot se representa mediante coordenadas XY dentro de la rejilla, que para este estudio es de 10x5. La relación entre la posición en la rejilla y la posición en la mesa de trabajo es directa. Todos los puntos de la rejilla son equidistantes, y la distancia entre ellos es de 100mm. Esta posición hace referencia al efector final del robot.

El estado del robot hace referencia a su situación respecto a las piezas. Un robot puede estar transportando una pieza o no. En caso afirmativo, esta variable indica la pieza que está transportando. En caso negativo, no hay necesidad de distinguir entre situación de reposo o de aproximación a una pieza, simplemente indica que está disponible para procesar una nueva pieza.

Tabla 3.1 Variables internas del modelo del sistema formado por el robot planar 2GDL y 4 piezas

Variable	Rango	Descripción
Posición del robot 1	[0-9], [0-4]	Coordenadas XY de la posición del efector final del robot 1 en la rejilla 2D, para
Posición del robot 2	[0-9], [0-4]	Coordenadas XY de la posición del efector final del robot 2 en la rejilla 2D, para
Estado del robot 1	0-K	Indica la pieza que el robot 1 está transportando (1-K) o 0 si no tiene agarrada ninguna pieza
Estado del robot 2	0-K	Indica la pieza que el robot 2 está transportando (1-K) o 0 si no tiene agarrada ninguna pieza
Estado de la pieza 1	0-1	Indica si la pieza 1 todavía reposa en la mesa de trabajo (1) o ha sido recogida por el robot (0)
Estado de la pieza 2	0-1	Indica si la pieza 2 todavía reposa en la mesa de trabajo (1) o ha sido recogida por el robot (0)
Estado de la pieza 3	0-1	Indica si la pieza 3 todavía reposa en la mesa de trabajo (1) o ha sido recogida por el robot (0)
Estado de la pieza 4	0-1	Indica si la pieza 4 todavía reposa en la mesa de trabajo (1) o ha sido recogida por el robot (0)

El estado de una pieza indica si la pieza todavía está en la mesa de trabajo o ya ha sido recogida. Por otro lado, no es necesario diferenciar explícitamente en esta variable si la pieza está siendo transportada o ya ha sido depositada en su destino. Esta información se puede inferir del conjunto de las variables estado del robot y estado de las piezas.

Inferencia del estado de las piezas

```
if estado pieza == 0 then
    pieza ← en posición inicial (no recogida)
else if ningún brazo está transportando la pieza then
    pieza ← en posición destino (depositada)
else
    pieza ← pieza en transporte
end
```

Figura 3.5 Lógica empleada para determinar el estado completo de las piezas

En la tabla 3.1, se omite la posición inicial de cada una de las piezas. Esta información determina las posiciones donde el robot puede realizar la operación de pick. De igual manera, se omite la posición final de las piezas que determina la situación de los depósitos donde la operación de place se ejecuta. Esta información, junto a la posición de los robots, es necesaria en el modelo ya que impacta en la viabilidad de las operaciones de pick-and-place en una determinada posición.

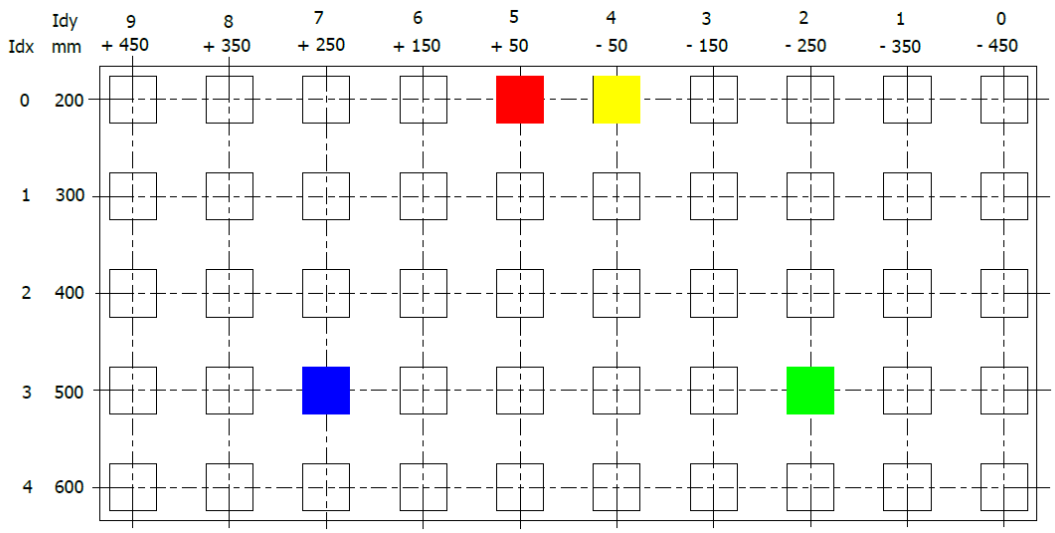


Figura 3.6 Relación entre la discretización del espacio de trabajo y el indexado empleado internamente por el algoritmo.

En este trabajo, por definición. la posición destino de las piezas está prefijada de antemano. Esta situación representa un proceso repetitivo donde cada nuevo conjunto de piezas debe ser transportado a la misma posición, o simplemente hace referencia a la localización fija de los depósitos donde las piezas deben ser almacenadas. Cada pieza tiene asignada una posición de destino independiente, que puede coincidir o no con la de otras piezas.

Tabla 3.2 Constantes internas del modelo del sistema formado por el robot planar 2GDL y 4 piezas

Variable	**Rango**	**Descripción**
Posición inicial de la pieza 1	X: [0-9], Y: [0-4]	Coordenadas XY de la posición inicial del efector final del robot 1
Posición inicial de la pieza 2	X: [0-9], Y: [0-4]	Coordenadas XY de la posición inicial del efector final del robot 2
Posición inicial de la pieza 3	X: [0-9], Y: [0-4]	Coordenadas XY de la posición inicial del efector final del robot 3
Posición inicial de la pieza 4	X: [0-9], Y: [0-4]	Coordenadas XY de la posición inicial del efector final del robot 4
Posición final de la pieza 1	X: [0-9], Y: [0-4]	Coordenadas XY de la posición final del efector final del robot 1
Posición final de la pieza 2	X: [0-9], Y: [0-4]	Coordenadas XY de la posición final del efector final del robot 2
Posición final de la pieza 3	X: [0-9], Y: [0-4]	Coordenadas XY de la posición final del efector final del robot 3
Posición final de la pieza 4	X: [0-9], Y: [0-4]	Coordenadas XY de la posición final del efector final del robot 4
N	2	Número de robots
K	4	Número de piezas

Por otro lado, la posición inicial de las piezas es arbitraria al inicio de cada operación de pick-and-place. Esta situación representa un proceso repetitivo donde en cada iteración las piezas podrían encontrarse en una posición diferente.

Considerando que la posición de destino de las piezas está predeterminada, esta información pasa a ser un elemento constante del sistema, al igual que el número de robots o número de piezas en el proceso. Por otro lado, la posición inicial de las piezas es arbitraria en cada iteración del proceso, por lo que esta información debería formar parte de las variables internas del modelo del sistema en lugar de considerarse una constante. La desventaja de considerar esta información como parte de las variables internas del sistema es que incrementa en gran manera la dimensión del problema. Para el caso particular de este trabajo, dado que el proceso de pick-and-place incluye 4 piezas, y las dimensiones de la rejilla son 10x5, el factor de incremento del espacio de estados sería:

$$\text{Factor Incremento} = \text{num\_posiciones\_pieza}^{K} = (10x5)^{4} = 6.250.000$$

La cantidad de estados en el sistema incrementaría drásticamente si introducimos este factor de incremento. Para evitar esta situación, la posición inicial de las piezas se considera constante, lo cual implica que cada iteración del proceso requiere renovar el modelo del sistema. La ventaja de esta técnica es que evita una explosión inmanejable de la cantidad de estados diferentes en el sistema. La desventaja es que no permite generar una solución general del problema en el laboratorio, sino que esta solución debe ser calculada en tiempo real, ya que, en cada iteración del proceso, el modelo del sistema puede ser diferente (en particular debido a la posición inicial de las piezas).

El tiempo de cómputo requerido para generar una solución depende de las dimensiones del problema (espacio de estados y espacio de acciones). Este tiempo debe estar contenido, ya que el objetivo es desarrollar un método capaz de operar en tiempo real en un proceso. En el contexto de este capítulo con el robot planar 2GDL, se asume que este tiempo es nulo. En los próximos capítulos, donde se introduce el robot YuMi en la estación de trabajo, este factor se tiene en cuenta y se introduce una nueva técnica para dividir este cómputo en dos partes, una de mayor entidad realizada en el laboratorio y otra parte más liviana ejecutada en tiempo real durante la operación de pick-and-place.

3.2.3.1.1 Estado inicial y estados finales

El estado inicial del sistema es único y diferente en cada iteración del proceso. Está representado por el valor específico que cada una de las variables del sistema posee en este momento inicial.

Tabla 3.3 Estado inicial del sistema

Posición robot 1	Posición robot 2	Estado robot X	Estado pieza X
Específica	Específica	0	1

El estado de los robots es inicialmente siempre 0, indicando que no está transportando ninguna pieza. El estado de las piezas es 1 al inicio, indicando que todas las piezas están todavía en su posición inicial. La posición de los robots es específica para cada iteración del proceso.

El objetivo es evolucionar desde este estado inicial al estado final. En realidad, existen múltiples estados finales.

Tabla 3.4 Estado final del sistema

Posición robot 1	Posición robot 2	Estado robot X	Estado pieza X
Arbitraria	Arbitraria	0	0

Dado que todas las piezas han sido depositadas en su lugar de destino, el estado de las piezas es 0, indicando que las piezas no están en su posición de inicio. Igualmente, el estado de los robots es 0, indicando que ningún robot está transportando una pieza en estos momentos. Las potenciales posiciones de los robots son múltiples. En el caso de que ambos brazos depositen las últimas piezas simultáneamente, la posición de cada robot se corresponderá con la posición destino de cada uno de los robots. No obstante, si un brazo completa su tarea antes que el otro, la posición de éste puede ser variable al final del proceso. La tarea es de carácter colaborativo, por lo cual, en determinadas situaciones, un brazo deberá

apartarse para ceder el paso al otro. Cuando el brazo que ha completado su tarea necesita colaborar con el otro brazo, su posición final es arbitraria. Bajo estos parámetros, la condición de tarea completada depende exclusivamente del valor de las variables estado de los robots y estado de las piezas.

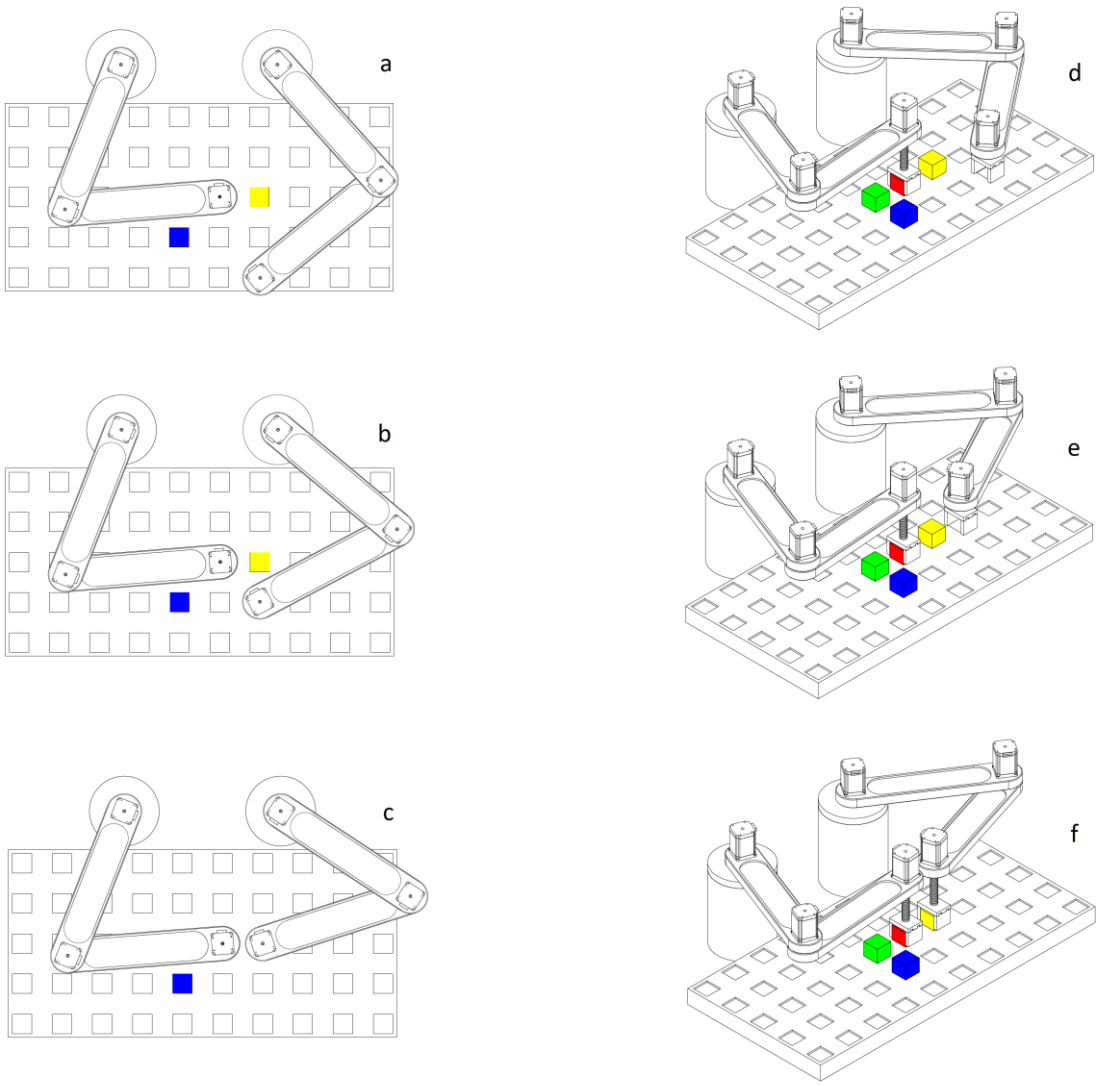


Figura 3.7 Múltiples estados finales del sistema. Vista 2D (a, b, y c) y vista 3D (d, e, y f)

Encontrar la secuencia óptima de estados existente entre el estado inicial y el final es el objetivo de este sistema. El concepto de óptima hace referencia a la secuencia de estados más corta que une el estado inicial y el final. En realidad, existen múltiples secuencias que cumplen este criterio. En este estudio inicial, cualquiera de las soluciones óptimas se considera suficientemente buena. En los próximos capítulos, donde se introduce el robot YuMi, se introduce alguna consideración adicional que influye en la selección de entre las múltiples soluciones óptimas.

3.2.3.1.2 Tamaño del espacio de estados

El tamaño del espacio de estados hace referencia a la cantidad de diferentes estados posibles en el modelo del sistema. Este número resulta de la combinación de los posibles valores de cada una de las variables internas del sistema. Para el caso particular de la configuración con el robot planar 2GDL (N = 2), 4 piezas (K = 4) y una discretización del espacio de trabajo de (X·Y = 10·5), el tamaño del espacio de estados es:

$$\text{Tamaño espacio estados} = (X \cdot Y)^N + (K + 1)^N \cdot 2^K = (10 \cdot 5)^2 \cdot (4 + 1)^2 + 2^4 = 1.000.000$$

El primer término (X · Y) modela el número de posibles configuraciones para cada uno de los (N) robots. El segundo término (K + 1) representa los posibles valores del estado de cada uno de los (N) robots. Finalmente, el último término (2) indica la cantidad de posibles valores que puede adoptar el estado de cada una de las (K) piezas.

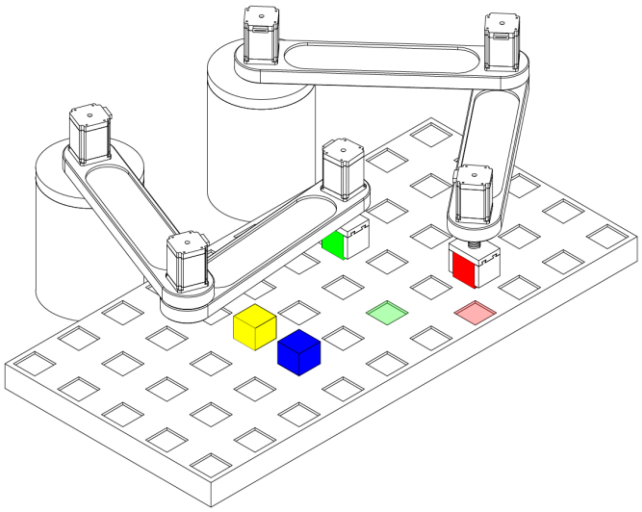
En realidad, este valor es un número en bruto. Existen múltiples estados del sistema que son inválidos. Por ejemplo, es incompatible que el estado indique que un brazo está transportando una de las piezas, y el estado de esa pieza indique que todavía está en su posición de reposo. También sería inválido la situación donde el estado de ambos brazos indica que están transportando la misma pieza.

El algoritmo desarrollado en este trabajo realiza un filtrado inicial de todos los estados inválidos del sistema, y con ello se reduce la dimensión del problema. Este filtrado reduce el número total de estados aproximadamente a un 15% del número indicado por la fórmula.

En el próximo capítulo, se introducen algunos cambios en el sistema que influyen en el modelo generado para éste. En particular, el espacio se discretiza en tres dimensiones en lugar de en el plano, y la operación de pick-and-place se modela correctamente (descenso, apertura o cierre de la pinza y ascenso). Forzosamente, la nueva información incrementa el tamaño del espacio de estados (aproximadamente 10 veces).

3.2.3.1.3 Correspondencia visual del estado interno del sistema

El modelo del sistema solo captura las características esenciales del proceso, e ignora el resto de información del proceso real. La descripción del modelo se puede consultar en la sección 3.2.3. El algoritmo desarrollado para resolver el problema de pick-and-place trabaja sobre el modelo del sistema, luego solo opera en base a las variables internas definidas en este modelo. La siguiente figura representa varias escenas de una simulación del problema acompañadas del valor de las variables internas del modelo correspondientes.



Variable	Valor
Posición robot 1 (izq)	[3, 4]
Posición robot 2 (der)	[4, 3]
Estado robot 1	3
Estado robot 2	1
Estado pieza 1 (verde)	0
Estado pieza 2 (azul)	1
Estado pieza 3 (roja)	0
Estado pieza 4 (amarilla)	1

Figura 3.8 Correspondencia interna del modelo respecto a un instante en el proceso de pick-and-place

3.2.3.2 *Espacio de acciones*

El espacio de acciones engloba el conjunto de acciones que cada uno de los robots puede realizar. Las acciones disponibles para cada robot incluyen el movimiento a través del plano, las operaciones de pick-and-place y permanecer en reposo.

El tipo de movimientos soportados por cada robot consiste en desplazamientos ortogonales hacia alguna de las casillas adyacentes en la rejilla 2D. Estos movimientos se pueden representar mediante las siglas N-S-E-O (norte, sur, este y oeste).

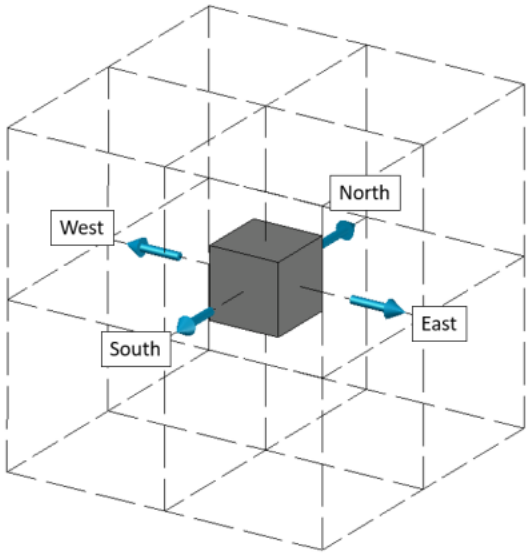


Figura 3.9 Movimientos soportados en el plano

La acción de pick o place engloba todo el conjunto de operaciones necesarias para llevar a cabo ésta. En el caso del robot planar, las acciones de pick-and-place son inmediatas. Este robot simple no dispone de pinza para agarrar la pieza. El único requisito es estar posicionado en las mismas coordenadas XY que la pieza. Tras ejecutar la acción de pick, se considera que el robot ha recogido la pieza, y tras ejecutar la operación de drop, se considera que la pieza ha sido depositada. En el próximo capítulo, con el robot YuMi, la acción de pick-and-place se modela con la siguiente secuencia de operaciones: descenso hacia el plano de las piezas, apertura o cierre de la pinza y ascenso de nuevo al plano de movimientos.

Las acciones de pick-and-place solo pueden ejecutarse bajo determinadas condiciones:

Tabla 3.5 Viabilidad de las acciones de pick-and-place

Acción	Condición
Pick	Robot posicionado en las coordenadas XY de la posición de inicio de una de las piezas AND pieza en su posición inicial
Place	Robot posicionado en las coordenadas XY de la posición de destino de una de las piezas AND robot transportando dicha pieza

Finalmente, en algunas situaciones es ventajoso que un robot permanezca en reposo en lugar de encontrarse realizando movimientos innecesarios (por ejemplo, cuando un robot ha procesado todas sus piezas, pero el otro todavía no ha completado su tarea).

El número de acciones soportadas por cada robot (o brazo del robot) es 7:

- Movimientos para aproximar o transportar una pieza (N, S, E y O)
- Pick-and-place: coger o dejar pieza
- Reposo: el robot no se mueve

El número de acciones en el sistema incrementa exponencialmente con el número de robots. Para el caso del robot planar 2GDL

$$\text{Num\_acciones\_sistema} = \text{num\_acciones\_robot}^{N} = 7^2 = 49$$

En la práctica, la acción de sistema donde ambos robots permanecen en reposo se descarta, al no aportar nada, dando un total útil de 48 acciones útiles. En el estudio con el robot YuMi, se consideran movimientos en el espacio 3D, incluyendo movimientos diagonales en las tres coordenadas, lo cual incrementa significativamente el número de acciones del sistema.

3.2.3.3 Espacio de recompensas

Dentro de las múltiples acciones que los robots pueden tomar en una determinada situación, algunas serán más favorables que otras, y en particular algunas cumplirán con el objetivo de minimizar el tiempo de la tarea.

El concepto de recompensa hace referencia a la calidad de la acción tomada. Si la acción es inapropiada, el sistema debería recibir una penalización, o recompensa negativa. Si la acción es favorable, la recompensa debería ser positiva. A esta recompensa se le conoce también como recompensa inmediata. Es decir, no tiene en cuenta el objetivo global, únicamente valora esta acción aisladamente. Es común que no haya elementos para determinar si una acción individual es favorable o no, o si lo es en mayor o menor medida que el resto, por lo cual la recompensa en muchas ocasiones es nula. Si todas las acciones presentan un mismo valor (por ejemplo, cero), significa que no hay una preferencia clara entre todas las acciones. En determinadas ocasiones, existen determinados estados del sistema donde tomar una acción es claramente más favorable que otras (por ej. en el videojuego de Mario Bros, si en un determinado estado, hay opciones de coger una moneda, la acción preferible es la que consigue coger la moneda).

En el caso de la operación de pick-and-place, no es fácil determinar si una acción es mejor que otra de antemano. No existen objetivos locales. Solo hay un objetivo global que consiste en procesar todas las piezas en el menor tiempo posible. El orden en el que se procesan las piezas, la asignación entre robots y piezas y las trayectorias podría verse afectado por la definición de las recompensas. Por otro lado, no es deseable determinar manualmente objetivos locales de antemano (por ej: pieza 1 asignada a robot 2) pues lo deseable es que el algoritmo encuentre una solución óptima a este problema. Si manualmente se fuerzan determinados comportamientos, es probable que la solución obtenida sea de peor calidad.

Tabla *3.6* Tipos de recompensas

Recompensa	Comentario
+100	Tarea completada (todas las piezas están depositadas)
-20	Acción inválida en un determinado estado
-1	Resto de acciones

En este caso se ha definido tres tipos de recompensa. Una muy positiva indicando que el objetivo se ha logrado. Esta recompensa está asociada exclusivamente a la acción o acciones que en un determinado estado lleva a completar la tarea. Otro tipo de recompensa muy negativa está asociado a las acciones que claramente no deben tomarse. En particular, a las todas situaciones que se conocen de antemano. Por ejemplo, si un brazo se mueve hacia el este E, y esto provoca salirse de los límites de la rejilla, esta acción se considera inválida, Igualmente, existen otras razones para asignar esta penalización en otras situaciones, como acciones que provocan una colisión entre robots, o que pretenden alcanzar una posición imposible. Para el resto de las acciones, no existe información de antemano acerca de la calidad de éstas. No obstante, se le asigna un pequeño valor negativo, en lugar de cero. El propósito de esta pequeña penalización general es influenciar al algoritmo para que encuentre una solución que implique el menor número de acciones. Dado que el algoritmo trata de maximizar la recompensa acumulada durante toda la operación, esta pequeña penalización ayuda a elegir la solución óptima (o una de ellas) de entre todas las soluciones posibles.

3.2.4 *Solución*

La tarea de pick-and-place se ha planteado como un problema de toma decisiones. En este caso decisiones hace referencia a las acciones que debe tomar para cumplir con el objetivo. El aprendizaje por refuerzo es una de las áreas que se dedican a este dominio. Está inspirado en la psicología conductista, es decir, que acciones se deben tomar con el fin de maximizar alguna noción de recompensa.

En este estudio inicial, se ha optado por modelar el sistema como un proceso de decisión de Markov (o en inglés, Markov Decision Process, MDP). En capítulos posteriores, este enfoque se compara con otros, tales como grafos o Planning Description Domain Language (PDDL).

3.2.4.1 *Proceso de decisión de Markov* (MDP)

El proceso de decisión de Markov es un marco matemático para modelar problemas de toma de decisión. Abarca multitud de casuísticas, incluyendo sistemas mono-agente o multi-agente, entornos deterministas o estocásticos, modelos parcial o totalmente observables, espacio de estados y acciones discretos o continuos, etc …

En este estudio, el modelo del sistema presenta las siguientes características:

Tabla *3.7* Propiedades del modelo del sistema

Característica	**Tipo**
Espacio de estados	Discreto y finito
Espacio de acciones	Discreto y finito
Incertidumbre	Entorno determinista
Visibilidad	Estado totalmente observable
Modelo MDP	Conocido
Número de agentes	1

Por definición, el modelo del sistema es conocido. Las variables internas que lo forman (posición de los robots, de las piezas y estado de robots y piezas) han sido seleccionadas por diseño para este tipo de problema de pick-and-place multi-robot. Cualquier problema modelado como un MDP está compuesto de 4 componentes: función de transición $P(s' | s, a)$, función de recompensa $R(s, a, s')$, espacio de estados S y espacio de acciones A. Para que un problema de toma de decisiones pueda ser modelado como un MDP, éste ha de cumplir con la propiedad de Markov, es decir, los efectos de una acción tomada en un determinado estado dependen exclusivamente de ese estado.

El problema de pick-and-place cumple dicha propiedad de Markov. Considerando que el sistema está en el estado s_t, y el agente toma una acción a_t, el sistema pasa al estado s_{t+1} independientemente del histórico previo de acciones (s_{t-i} para i = 0 … t-1) y estados (a_{t-i} para i = 0 … t-1), como muestra la figura 3.10. Por ejemplo, dado un específico estado del sistema, donde uno de los robots está en una determinada posición, si el robot toma la acción de moverse hacia el este, el sistema pasará a un nuevo estado donde el robot está posicionado en la casilla adjunta en dirección este, y no tiene ninguna relevancia como se ha llegado a la posición inicial antes de empezar dicho movimiento.

El espacio de estados se ha descrito en la sección 2.1.3.1, y el espacio de acciones en la 2.1.3.2. La función de transición indica el estado al que pasa el sistema tras tomar una acción en un determinado estado. Es decir, hace referencia a la dinámica del sistema o a su comportamiento. La función de recompensa indica la valoración de la acción tomada en un determinado estado

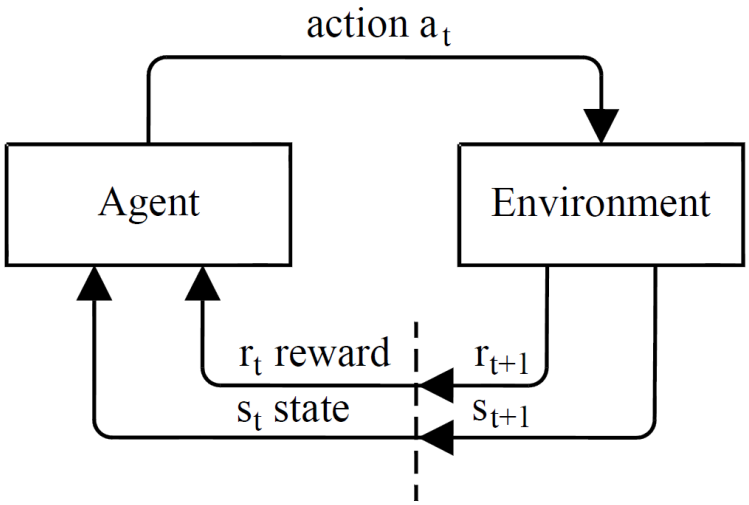


Figura 3.10 Relación agente-entorno

Tanto el espacio de estados como el espacio de acciones son discretos y finitos. El tamaño de éstos aumenta exponencialmente con el número de robots y de piezas. En este estudio, el número de robots se limita a dos y el número de piezas puede ser variable. El espacio de estados es totalmente observable, lo cual significa que, para cualquier instante de tiempo, se puede obtener el valor de todas las variables internas que forman el modelo del sistema. Una última característica del sistema es la ausencia de incertidumbre. El sistema es determinista en todas sus facetas:

- Entorno determinista: tomar una acción a en un estado s, lleva al sistema siempre al mismo estado s'. De entre todos los estados posibles del sistema, s' es único. Hay un 100% de probabilidad de pasar a este estado y 0% al resto
- Acción determinista: cada estado tiene una única acción asociada. Esta acción se toma con un 100% de probabilidad y el esto con el 0%

Aunque el sistema es multi-agente por naturaleza, ya que existe más de un robot (o brazo robot) operando en el mismo espacio de trabajo, el modelo del sistema es mono-agente. Es decir, todas las decisiones están centralizadas en una única entidad. Esta entidad es capaz de observar el estado completo del sistema y determinar las acciones que debe tomar cada robot en cada momento. La ventaja principal de este enfoque es que una única entidad posee toda

la información necesaria para la toma de decisiones, y no necesita comunicación ni coordinación con otras entidades, como ocurre en un sistema distribuido multi-agente. De igual manera los algoritmos asociados son más sencillos. La principal desventaja es la escalabilidad. El tamaño del espacio de estados incrementa exponencialmente con el número de robots.

3.2.4.2 Algoritmos

Una vez un problema de cualquier índole (física, logística, robótica, videojuego, etc…) está modelado bajo el marco del Proceso de Decisión de Markov, el comportamiento del modelo es estándar. El agente observa el estado actual del sistema, y en base a éste toma una determinada acción. Esta acción sobre el sistema, a su vez, provoca una transición en el estado de éste y genera una recompensa que el agente emplea durante la fase de entrenamiento.

Aprendizaje por refuerzo es un área enfocada en problemas de toma de decisión modelados bajo el marco del Proceso de Decisión de Markov. El propósito de esta clase de algoritmos es encontrar la acción o acciones óptimas para cada estado del sistema. Este concepto se conoce como política del MDP. En un caso general, indica la probabilidad de seleccionar una determinada acción estando en un determinado estado. En el caso particular de este estudio, considerando que el sistema es determinista, la política indica la acción o acciones óptimas, ya que puede no ser única, a tomar en cada estado.

Existen multitud de algoritmos dentro de esta área. Una de las formas más comunes de clasificarlos es en base a la manera de aprender el modelo del sistema. Esta clasificación divide los algoritmos en dos bloques. En el primero se agrupan los que se denominan libre de modelos (en inglés, Model-Free). Libre de modelos significa que no aprenden explícitamente el modelo del sistema, es decir, no generan una representación de la función de transición y de la función de recompensa. Obviamente, aprenden esta información de manera implícita, ya que es necesario para poder encontrar una buena política para el MDP. A su vez, dentro de este bloque, se suele dividir en otros dos grupos, los basados en optimizar la función de valor (en inglés, Value function o Q-Value function) y los basados en optimizar la política del sistema directamente, sin representación explícita de la función de Valor.

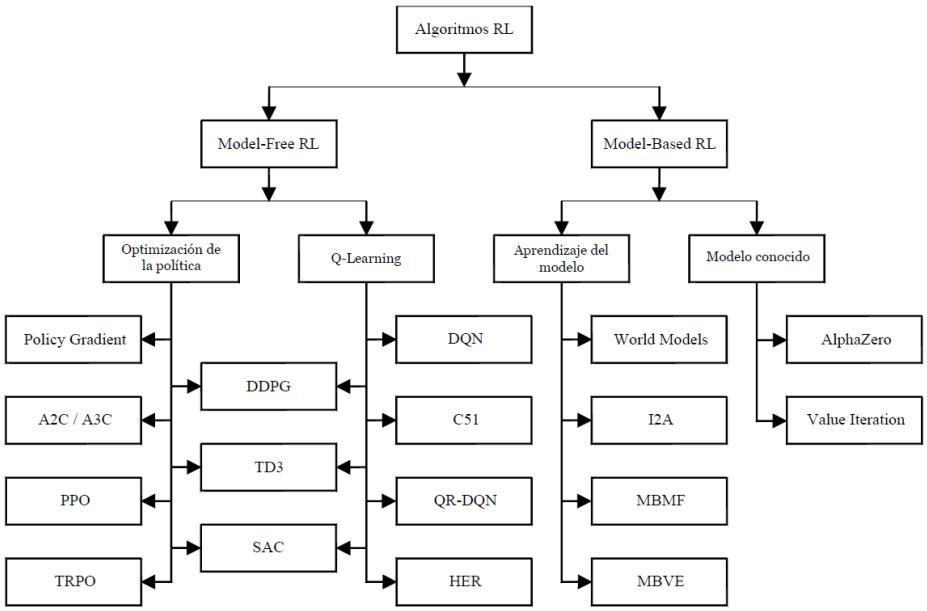


Figura 3.11 Clasificación de algoritmos RL

Algoritmos como DDPG (Deep Deterministic Policy Gradient), o SAC (Soft actor-critic) pertenecen al grupo de optimización de la política. Dentro de los basados en la optimización de la función de valor se encuentran algoritmos como A3C (Asynchronous Actor-Critic Agent) o PPO (Proximal Policy Optimization). El otro gran bloque de algoritmos agrupa los algoritmos basados en el modelo del sistema (en inglés, Model-Based). Este tipo de algoritmos está compuesto de dos fases: la primera consiste en aprender el modelo del sistema (función de transición y de recompensa), y la segunda en encontrar una buena política en base a disponer del modelo del sistema. Este bloque también se divide en dos. Por un lado, están los algoritmos que realizan ambas fases, y por otro lado, los algoritmos que solo necesitan realizar la segunda fase, ya que el modelo del sistema viene dado.

Los algoritmos más eficientes computacionalmente son los que pertenecen al grupo de Model-Based con el modelo del sistema conocido, ya que la fase de aprendizaje, bien explícita o implícitamente de este modelo no es necesaria. En este estudio, el modelo del sistema viene dado, es decir, está definido por diseño para la tarea multi-robot de pick-and-place. Esto implica que el algoritmo adecuado para resolver este problema se encuentra en la subgrupo de algoritmos Model-based con el modelo dado. Dentro de este grupo, se encuentra Value Iteration. Es un algoritmo de programación dinámica, de carácter iterativo, que siempre converge a una solución óptima. Es el algoritmo más liviano de los presentes en este grupo y es la opción seleccionada.

3.2.4.2.1 Value Iteration

Value Iteration es un algoritmo que se ajusta perfectamente a las características de este sistema: discreto, finito, tamaño contenido, determinista y totalmente observable. Estas características permiten una representación tabular del problema, tanto para la función de transición y de recompensa, como para la función de Valor y la política del MDP.

El componente principal de este algoritmo es la función de Valor (V). Esta función representa la recompensa acumulada desde el estado actual hasta llegar al estado final siguiendo una política π determinada (V^{π}). El valor de esta función es dependiente de la política empleada, ya que el camino que conecta el estado actual con el estado final es también diferente y por ende, las recompensas obtenidas en cada uno de estos caminos.

$$V^{\pi}(s) = E_{\pi}\left[\sum_{k=0}^{H} \gamma^{\,k} R_{k+1} \middle| s_{t=s}\right] \tag{1}$$

El objetivo de todo algoritmo de aprendizaje por refuerzo (en inglés, Reinforcement Learning) es encontrar la política que maximiza la recompensa acumulada desde cualquier estado hasta el estado el final. A esta política se le conoce como política óptima. Pueden existir una o múltiples políticas óptimas en un MDP. La función de valor que resulta de aplicar la política óptima se conoce como función de Valor óptima (V^{*}). Esta función es única para el MDP. Todas las políticas óptimas, si hay más de una resultan en la misma función de valor óptima.

$$V^{*}(s) = \max_{\pi}\left[\sum_{k=0}^{H} \gamma^{\,k} R_{k+1} \middle| \pi, s_{t=s}\right] \tag{2}$$

Value Iteration es un algoritmo de carácter iterativo. Inicialmente, se parte de una política arbitraria. La función de Valor resultante de aplicar esta política es una estimación bastante mala del valor de la recompensa acumulada entre cualquiera de los estados y el estado final, comparado con la función de Valor óptima. Es coherente, ya que las acciones aplicadas en el sistema son arbitrarias en este punto.

En cada iteración se actualiza la función de Valor en base a la ecuación de Bellman:

$$\mathrm{V}(s) = \max_a \sum_{s'} P_a(s'|s)[R(s,a,s') + \gamma \times V(s')] = \max_a [R(s,a) + V(s')] \quad (3)$$

La ecuación de Bellman puede adoptar una forma más simplificada cuando se aplica a modelos deterministas, y se usa descuento aditivo (en inglés, additive discounting). En particular, se pueden eliminar el término probabilístico $P_a(s'|s)$ y el factor de descuento γ (en inglés, discount factor). Dado que el modelo es determinista, solo existe un estado *s'* resultante de aplicar la acción *a* en el estado *s*. Es decir, la probabilidad de transición a un determinado *s'* es 100%, mientras que la probabilidad de transición al resto de estados es nula. En estas condiciones, el sumatorio para todo *s'* que incluye la probabilidad de transición a ese estado *s'* es reemplazado por el valor unidad. Por otro lado, en términos generales, el factor de descuento γ se emplea para degradar el Valor de recompensas futuras. Existen tres razones para emplear este término:

- En problemas de horizonte infinito, es decir, problemas que no terminan nunca, se emplea como un truco matemático para convertir una suma infinita de recompensas en un valor finito. Esto ayuda a probar la convergencia de varios algoritmos.
- Si existe incertidumbre en el sistema, o la política del MDP es probabilística, no se puede determinar con certeza a que estado se realiza la siguiente transición tras tomar una determinada acción *a*. Esto implica que no se puede conocer con certeza la recompensa recibida tras tomar dicha acción. Tras tomar N acciones, la incertidumbre de conocer el estado donde el agente estará siguiendo una determinada política es enorme, por lo tanto, la estimación de recompensas acumuladas hasta ese punto es muy pobre. Dado que esta estimación es peor conforme aumenta la profundidad de la secuencia de acciones, se aplica un efecto penalizador en estas para restarles influencia.

$$G = R_i + \gamma R_{i+1} + \gamma^2 + \gamma^3 R_{i+3} + \cdots \quad (4)$$

- En problemas donde existe escasa información útil de recompensa (en inglés, sparse reward), es una manera de reducir el valor de éstas proporcionalmente al instante futuro en el que se obtienen. Es una técnica indirecta empleada para optimizar el número de pasos requeridos para obtener una solución.

El factor de descuento es un valor entre 0 y 1. En un extremo, si su valor es 1, es equivalente a prescindir de este factor y pesar por igual todas las recompensas recibidas a futuro.

Si su valor es 0, es equivalente a prescindir totalmente de cualquier estimación de recompensa recibida a futuro y confiar exclusivamente en la recompensa inmediata recibida tras realizar la siguiente acción. Comúnmente se emplea un valor inferior pero cercano a uno, para tener en cuenta la estimación de recompensas recibidas a futuro, pero con menos peso cuanto más alejadas están. En la siguiente gráfica se puede apreciar el peso que se le otorga a recompensas futuras para diferentes valores del factor de descuento.

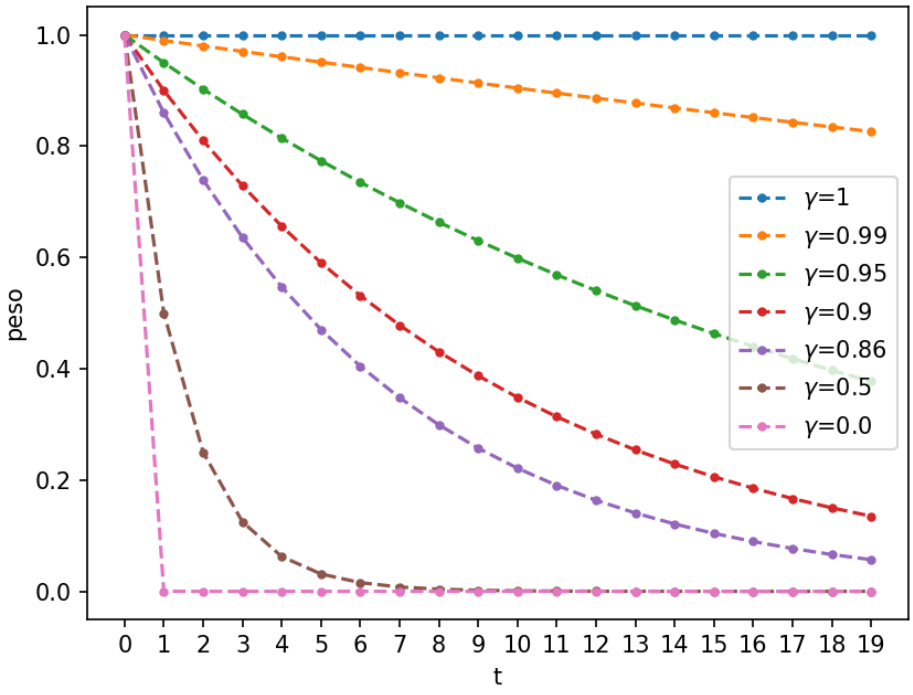


Figura 3.12 Peso asignado a las recompensas futuras en base al valor seleccionado para el factor de descuento

En el caso de este estudio, el horizonte del problema es finito, es decir, la operación de pick-and-place se completa en un determinado número de pasos. Por otro lado, el sistema es determinista, tanto el entorno como las acciones, luego el valor de las recompensas futuras no es una estimación, sino un valor real exacto conocido. Finalmente, la información útil de recompensa en el modelo del sistema es realmente bastante escasa.

Por un lado, existe un tipo de recompensa muy positiva que se obtiene al alcanzar el estado terminal, que indica que la tarea se ha completado. Por otro lado, existe un tipo de recompensa muy negativa para indicar acciones que se deben evitar en determinadas situaciones, que puede provocar un escenario de colisión entre robots, requerir que un robot alcance una posición imposible, etc… Para el resto de las situaciones, no existe información de recompensa útil que ayude a guiar al algoritmo hacia una solución.

Un factor de descuento inferior a la unidad sería útil para reducir el valor de la recompensa acumulada a lo largo del camino. En particular, la solución que alcanza el estado terminal en un menor número de pasos obtiene una recompensa mayor, ya que el peso que se le aplica a esta recompensa obtenida en el estado terminal es mayor (γ^M, donde M es el número de pasos). No obstante, introducir un factor de descuento inferior a la unidad implica el uso de coma flotante en el algoritmo, lo cual tiene un impacto relevante tanto en los recursos de memoria consumidos, como en el tiempo de cómputo.

En este estudio se ha utilizado la técnica del descuento aditivo. Esta técnica consiste en asignar una pequeña recompensa negativa (penalización) en cada estado donde no existe información útil de recompensa. El efecto de esta técnica para el sistema determinista tratado en este estudio es similar al de usar un factor de descuento inferior a la unidad, con la ventaja de que se puede mantener un uso exclusivo de números enteros en el algoritmo.

Como se ha descrito previamente, el algoritmo es de naturaleza iterativa. Comienza con una función de Valor V^{π} arbitraria, y en cada iteración se actualiza usando la ecuación de Bellman. Está teóricamente demostrado que la estimación de V^{π} mejora con cada actualización y que converge a la función de Valor óptima V^{*} en un número finito de iteraciones. A partir de este punto, $V^{\pi} = V^{*}$, la función de Valor ya no cambia en sucesivas iteraciones. En la práctica, se define un umbral ε de valor bastante pequeño, y el algoritmo se detiene cuando la función de Valor varía menos de este umbral entre dos iteraciones, lo cual ocurre antes de la

convergencia pura. Bajo este escenario, la función de Valor es casi óptima, y en esta situación la política derivada de ésta es la misma o casi la misma que se obtendría de la función de Valor óptima.

Una vez converge la función de Valor, la política π se deriva de V^*mediante la siguiente ecuación:

$$\pi^*(s) = \underset{a}{argmax}\left[R(s,a) + \gamma \times \sum_{s'} T(s,a,s') \times V^*(s')\right] = \underset{a}{argmax}[R(s,a) + V^*(s')] \quad (5)$$

Considerando que el modelo del sistema es determinista, finito y de tamaño contenido, es factible representar todos los componentes del MDP en forma tabular o matricial. En este estudio, el algoritmo de Value Iteration está implementado en Python, un lenguaje de programación muy extendido en el área de aprendizaje automático. Es un leguaje muy amigable pero relativamente lento. No obstante, Python permite conectar con librerías implementadas en otro lenguaje más eficiente, como C. En particular, el algoritmo hace uso de una librería de cálculo numérico llamada numpy. En este contexto, una operación matricial (ie.: A + B) consiste en una línea de Python, y toda la operación en sí está implementada en un lenguaje de bajo nivel (C) altamente optimizado., lo cual confiere una eficiencia al programa similar a la de un programa en C puro, sobre todo en escenarios como el de este estudio, donde las matrices son enormes.

A continuación, se puede consultar el pseudo-código de la versión matricial de Value Iteration, que incluye las simplificaciones aplicadas en las ecuaciones 4 y 5. Cabe resaltar que, entre las múltiples acciones óptimas posible en cada estado, *argmax* selecciona una. En particular la primera de ellas, basándose en el orden en que están codificadas las acciones.

```
Datos: ε: valor pequeño
Resultado: π: política óptima
Function ValueIteration
    /* Inicialización */
    Inicializar V(s) con valores arbitrarios, excepto V(terminal)
    V(terminal) = 0
    /* Bucle hasta converger */
    Δ ← 0
    while Δ < ε do
        for each s ∈ S do
            v ← V(s)
            V(s) ← max_a [R(s,a) + V(s')]
            Δ ← max(Δ, |v – V(s)|)
        end
    end
    /* Retornar la política óptima */
    return π /* π(s) ← argmax_a [R(s,a) + V(s')] */
end
```

Figura 3.13 Pseudo-código para versión matricial del algoritmo Value Iteration

4

OPERACIÓN DE PICK-AND-PLACE CON ROBOT YUMI Y PIEZAS ESTÁTICAS

4.1 Introducción

En este capítulo, la operación de pick-and-place representa una situación de proceso real, donde el robot planar de 4 grados de libertad es sustituido por un robot bimanual comercial de 14 grados de libertad. Así mismo, las piezas residen en un plano Z diferente al de los movimientos del robot, lo cual requiere una correcta modelización de la operación de pick-and-place (descender al plano de las piezas, agarrar o soltar la pieza y volver al plano de movimientos).

Por otro lado, el volumen que ocupa en el espacio cada brazo del robot dificulta la operación de aproximación y transporte de las piezas cuando se realiza en el mismo plano para ambos brazos. En este capítulo el área de movimientos de los robots se escala del plano definido en el anterior capítulo a un volumen 3D. La navegación en un espacio tridimensional expande las opciones de trayectorias para los robots, incluyendo la posibilidad de que se crucen a diferentes niveles de altura.

La arquitectura de control empleada en el capítulo anterior es la base de la definida en el actual. En este caso, debido a la forma y diseño complejos del robot, es necesario introducir una nueva técnica para valorar las situaciones de colisión entre robots y la posibilidad de alcanzar ciertas localizaciones, Considerando que es un robot redundante, a diferencia del estudio del capítulo anterior, existen múltiples configuraciones de los brazos que cumplen con el criterio de localizar la pinza en una determinada posición y orientación, por lo cual el control articular de los brazos del robot es otro de los elementos nuevos a introducir en la nueva arquitectura.

4.2 Materiales y métodos

4.2.1 Materiales

La configuración completa de la estación de trabajo incluye un robot YuMi, una mesa de trabajo, y un conjunto de piezas que reposan sobre ésta.

YuMi es un robot comercial del fabricante ABB, que está diseñado para aplicaciones de pick-and-place de pequeñas piezas, y tareas de inspección. Al tratarse de un robot colaborativo (CoBot) también puede cooperar con humanos en el mismo espacio de trabajo sin necesidad de jaulas de protección. Se trata de un robot que posee 7 grados de libertad en cada uno de sus brazos, y cada brazo está equipado con una pinza que permite manipular las piezas.

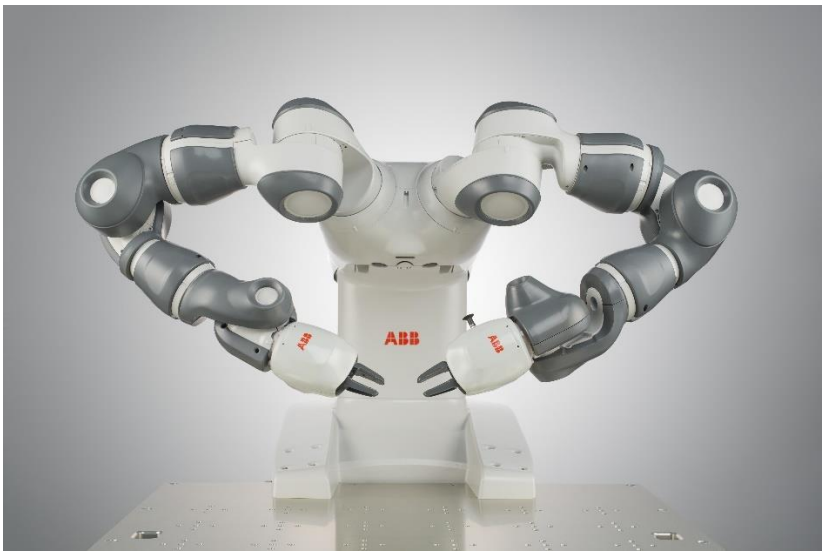


Figura 4.1 IRB14000 robot (Dual-arm YuMi) (www.ABB.com)

ABB ofrece múltiples opciones como efector final para YuMi. En este estudio, cada brazo integra una pinza (en inglés gripper), como se observa en la figura 4.1. Otras opciones disponibles incorporan una ventosa e incluso un sistema de visión embebido. En la tabla 4.1 se pueden consultar las características principales de este robot.

Las piezas están modeladas con forma cúbica, de dimensiones 40x40x40 mm. A cada una de las piezas se le asigna un color diferente que se emplea para identificar qué pieza es, y cuál

es su posición de destino. La posición inicial de las piezas es arbitraria en cada iteración de la operación de pick-and-place. Por otro lado, la posición final está predefinida para cada una de las piezas, emulando la localización de un depósito donde un determinado tipo de pieza debe almacenarse.

Tabla *4.1* Características principales del robot IRB 14000 (YuMi)

Característica	**Valor**
Carga	0.5 kg por brazo
Alcance	559 mm
Precisión	0.02 mm
Peso	38 kg
Protección IP	IP30
Número de ejes	14
Controlador	Integrado
Montaje	Mesa de trabajo

La mesa de trabajo hace a la vez de soporte para el robot y para las piezas. Todos estos elementos se han incluido en un software de simulación que proporciona el fabricante ABB, que se conoce como RobotStudio®.

RobotStudio® es un software que permite programar y simular offline aplicaciones robóticas. En palabras del propio fabricante, "con RoboStudio puedes tener plena confianza de que lo que ves en la pantalla coincide con cómo se moverá el robot en la vida real". Este tipo de entornos donde los elementos simulados son una fiel representación de los dispositivos reales se conoce como gemelo digital (en inglés, digital twin). Disponer de un gemelo digital (DT) permite desarrollar con plena confianza una aplicación robótica antes de ponerla en producción.

Respecto a la programación del robot, ABB ofrece diversas alternativas entre las que se encuentra el lenguaje de programación RAPID, comúnmente empleado para los robots de este fabricante. Toda la algoritmia de este estudio ha sido implementada en Python. Adicionalmente, una capa de software en RAPID permite conectar la solución generada con el robot.

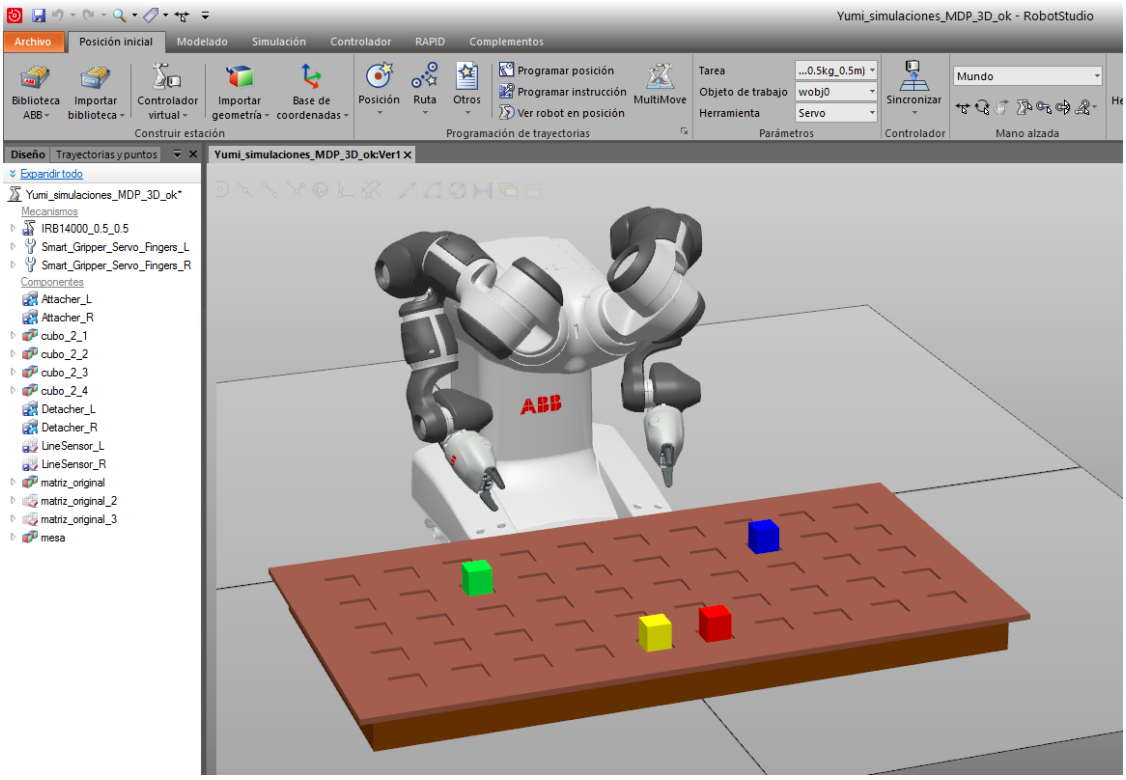


Figura 4.2 Versión de escritorio del software RobotStudio mostrando el proyecto de pick-and-place con el robot YuMi

El objetivo de la tarea es transportar un determinado número de piezas a los lugares designados. En este estudio, se llevaron a cabo pruebas con un rango de 2 a 10 piezas en total.

4.2.2 *Diseño de la arquitectura basada en la discretización del espacio de trabajo*

La arquitectura de control empleada en este estudio parte de la base de la definida en el capítulo anterior. El propósito de esta arquitectura es generar una solución que pueda implementarse en un proceso real. En este estudio, el sistema de pick-and-place simulado integra un robot YuMi, pero la solución generada debe ser fácilmente portable a otros modelos de robot de este o diferente fabricante. De igual manera que en el anterior estudio, la solución generada debe minimizar el tiempo de la operación de pick-and-place, para lo cual debe encontrar solución a las mismas preguntas: en qué orden se han de procesar las piezas,

qué robot debe procesar cada pieza, qué trayectorias debe seguir cada robot para aproximar o transportar una determinada pieza, y cómo evitar colisiones entre los robots. Considerando que se trata de una tarea colaborativa, el objetivo de cada brazo no es completar el procesado de sus piezas lo antes posible, sino que el conjunto de la tarea se complete en el menor tiempo posible, para lo cual, en determinadas situaciones un robot deberá entorpecer su trayectoria para permitir el avance del otro.

El problema de pick-and-place se plantea de nuevo como un problema de toma de decisiones, aunque en este estudio se proponen tres diferentes alternativas para el modelado del sistema, basadas respectivamente en Modelos de Decisión de Markov, grafos y PDDL (en inglés, Planning Domain Definition Language). Al adoptar un enfoque de toma de decisiones, es esencial definir el conjunto de acciones que se pueden tomar dentro del sistema y comprender el concepto del estado del sistema.

En consonancia con la arquitectura definida en el capítulo anterior, el enfoque se basa en las coordenadas cartesianas del efector final de cada brazo del robot YuMi. En este caso, el espacio de trabajo se discretiza en una constelación tridimensional de puntos, en lugar de una rejilla bidimensional.

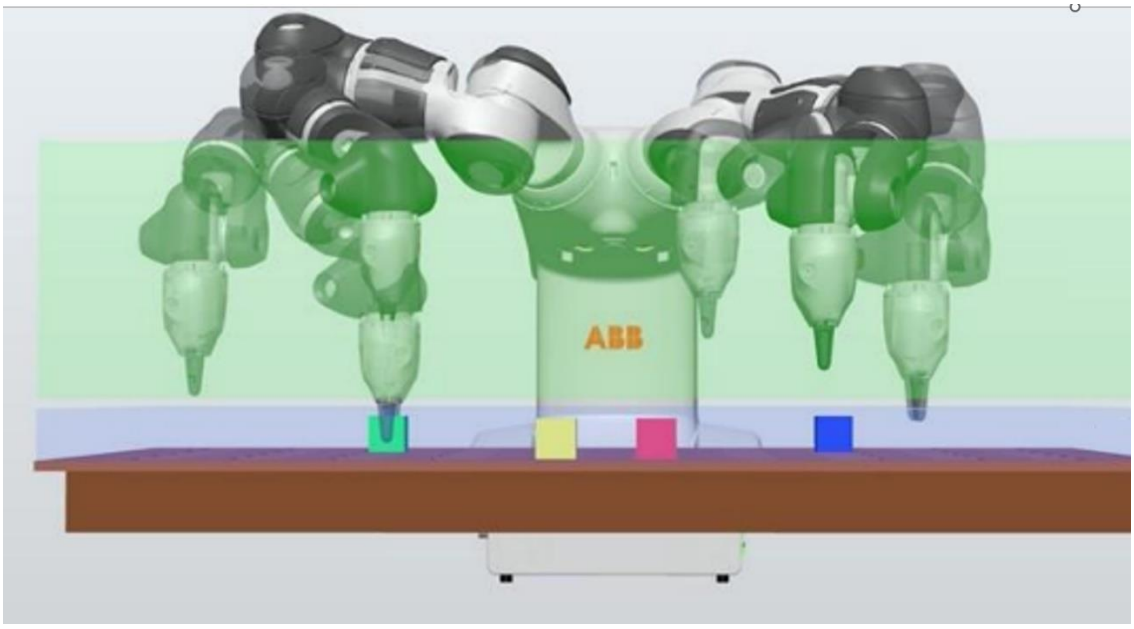


Figura 4.3 Representación de la zona de movimientos del robot y la zona de localización de las piezas

La zona coloreada de verde en la figura 4.3 representa el volumen por el que el robot se desplaza para aproximar o transportar una pieza. En adelante, esta zona se llamará zona, área o espacio de movimientos. La zona coloreada de morado representa al plano (mesa de trabajo) donde están situadas las piezas, y el volumen de espacio por el que robot se desplaza para coger o dejar una pieza. En adelante, esta zona se referirá como zona de pick-and-place.

El nivel de discretización de la zona de movimientos es configurable en la implementación desarrollada para esta arquitectura. No obstante, en este estudio se ha experimentado con estas tres opciones: 10x5x1, 10x5x2 y 10x5x3. La primera de ellas coincide con la configuración del estudio anterior (rejilla 2D). Las otras dos permiten moverse al robot (al efector final del robot concretamente) en diferentes niveles de altura. La distancia entre los diferentes planos de altura es tal que la pinza de un brazo no colisiona con la pinza del otro si ambas estuvieran en las mismas coordenadas del plano XY, pero a diferente nivel de altura. En particular, la distancia entre los diferentes puntos en cualquiera de los planos es de 100 mm, y la distancia entre planos es de 120mm.

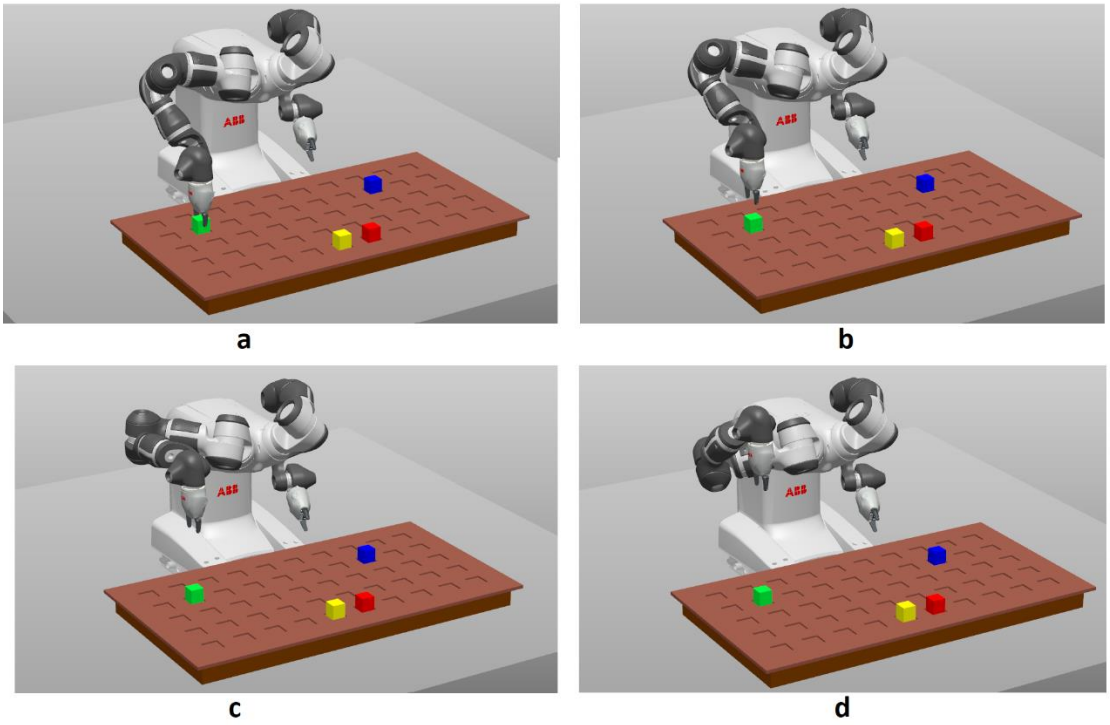


Figura 4.4 Brazo del robot a diferente nivel de altura: a) 110mm, b) 180mm, c) 310mm y d) 440mm

La arquitectura de control en este capítulo está estructurada en dos capas. La capa de alto nivel realiza la toma de decisiones necesaria para cumplir con todos los objetivos de alto nivel expuestos anteriormente, tales como la generación de trayectorias, la asignación de piezas, etc.. La capa de bajo nivel consiste en el control directo de los movimientos del robot. Esta capa está gestionada por el controlador interno del robot YuMi. El interfaz entre ambas capas es el lenguaje de programación suministrado por el fabricante. En este caso particular es el lenguaje RAPID (ABB). Otros fabricantes ofrecen un lenguaje de programación diferente, pero todos ellos soportan las directrices básicas necesarios en esta arquitectura, tales como "realizar una transición a una nueva configuración articular". Dado que el control a bajo nivel está gestionado por el controlador interno del robot, la solución generada aúna robustez y precisión en los movimientos.

Las acciones disponibles para los brazos del robot en el área de movimiento implican moverse a cualquiera de los puntos adyacentes en la constelación de puntos resultante de la discretización del espacio de trabajo. Esta restricción de movimientos a las celdas adyacentes reduce la cantidad de posibles movimientos a 27 en lugar de 150, que es el total de puntos existentes en la versión de rejilla 3D 10x5x3. Se trata de una reducción considerable, aunque en número es algo superior al caso del robot planar del anterior capítulo, donde solo se permitía el movimiento en las 4 direcciones del plano. Por otro lado, permite sincronizar el movimiento del robot tras cada movimiento, ya que la distancia recorrida por cada brazo es similar.

En el caso de rejilla 10x5x3, la distancia recorrida por el robot es ligeramente diferente en función del tipo de movimiento: ortogonal (100 mm), diagonal en el plano (141 mm) y diagonal en las 3 dimensiones 185 mm). Para mantener la sincronización entre robots, cada uno de estos tipos de movimientos se realiza a una velocidad ligeramente diferente, compensando con ello el efecto de la diferencia de distancia recorrida.

Las trayectorias descritas por cada brazo del robot se representan mediante un conjunto ordenado de puntos adyacentes en la rejilla 3D. Dado que el tiempo invertido en recorrer cada uno de estos desplazamientos es muy similar, la trayectoria resultante es aparentemente suave, continua, de velocidad similar en ambos brazos. Si ambos brazos recorrieran distancias dispares, para mantener la sincronización entre ellos, habría que introducir una diferencia de velocidad notable entre ellos, o un tiempo de espera en el que termina primero.

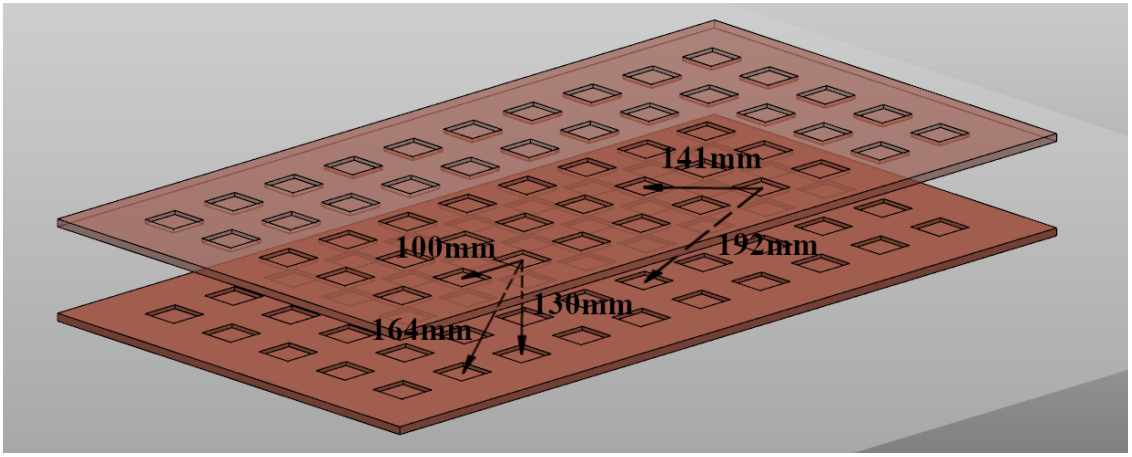


Figura 4.5 Discretización del espacio de trabajo (distancia entre puntos de la rejilla resultante)

La arquitectura definida en este estudio considera a los robots y las piezas como puntos en el espacio de trabajo discretizado. Estos puntos hacen referencia al efector final de cada brazo del robot y al centro de las piezas. Es decir, cuando el robot está en las mismas coordenadas XY en las que se encuentra una pieza, y en el nivel de altura adecuado, la operación de agarre se produce en el centro de la pieza.

No obstante, tanto las piezas como el robot ocupan un cierto volumen en el espacio y es un factor que no se puede ignorar. La forma y diseño del robot es complejo, y valorar las situaciones de colisión en este sistema basado en puntos es crucial. Por otro lado, el área de influencia del robot es limitada. Dado que existe una cierta distancia entre la base de ambos brazos, habrá zonas del espacio de trabajo alcanzables únicamente por uno de los robots. En este modelo de puntos, es también necesario conocer a qué puntos de la rejilla 3D puede desplazarse cada brazo robot.

Por otro lado, para mantener una trayectoria suave entre dos puntos en el espacio, es necesario asegurar que no se alcanzan discontinuidades en las articulaciones del robot. El rango angular de cada eje del robot está limitado a un cierto valor. Si durante la trayectoria lineal del efector final una de las articulaciones llega a su límite articular, el robot debe adoptar una configuración nueva para alcanzar la posición objetivo, y se produce una discontinuidad en el movimiento,

Para tener en cuenta todos estos factores, el sistema recurre a la información almacenada en una base de datos.

4.2.3 *Base de datos específica para cada modelo de robot*

Se ha llevado a cabo un análisis previo para recopilar información sobre el área de influencia de cada brazo robot, las posibles situaciones de colisión y las configuraciones articulares más ventajosas para cada posición en la rejilla 3D. Esta información ha sido almacenada en una base de datos. Dado que el acceso a la base de datos se realiza exclusivamente en base a la posición de los robots, y no es necesario realizar búsquedas complejas, cualquier medio de almacenamiento es suficiente para emular dicha base de datos. En este estudio en particular, la información se ha guardado en un archivo CSV (en inglés, Comma Separated Values).

Brazo izquierdo			Brazo derecho			
X	Y	Z	X	Y	Z	Distancia(mm)
200	350	180	400	-450	310	97
200	350	180	400	-350	310	97
200	350	180	400	-250	310	97
200	350	180	400	-150	310	97
200	350	180	400	-50	310	97
200	350	180	400	50	310	94
200	350	180	400	150	310	69
200	350	180	400	250	310	64
200	350	180	400	350	310	0
200	350	180	400	450	310	0
200	350	180	500	-450	310	97
200	350	180	500	-350	310	97
200	350	180	500	-250	310	97
200	350	180	500	-150	310	97
200	350	180	500	-50	310	97
200	350	180	500	50	310	88
200	350	180	500	150	310	67
200	350	180	500	250	310	64
200	350	180	500	350	310	0
200	350	180	500	450	310	0
200	350	180	600	-450	310	0
200	350	180	600	-350	310	97
200	350	180	600	-250	310	97
200	350	180	600	-150	310	97
200	350	180	600	-50	310	92
200	350	180	600	50	310	73

Figura 4.6 Base de datos – Detección de colisión

La figura 4.6 muestra un extracto de la información relativa a colisiones en la base de datos. Para cada combinación de posición entre ambos brazos se almacena la distancia mínima entre ellos. La posición mínima se calcula comparando la distancia existente entre cada uno de los enlaces de un brazo (α) y cada uno de los enlaces del otro brazo (β). La menor de las distancias

es la distancia mínima entre ambos brazos.

$$\delta_{ij} = \left\| \lambda i_{\alpha} - \lambda j_{\beta} \right\| \quad (6)$$

$$\delta_{min} = \min(\delta_{ij}) \quad (7)$$

RobotStudio® permite detectar situación de colisión real dada una determinada configuración de los brazos del robot. No obstante disponer de la distancia mínima entre brazos es más útil, ya que permite establecer un umbral configurable que determine si ambos brazos están en situación de colisión o no. Por debajo de este umbral, el sistema considera colisión aun no existiendo una colisión real. De este modo, no solo se consigue evitar colisiones, sino mantener una zona de seguridad entre ambos robots.

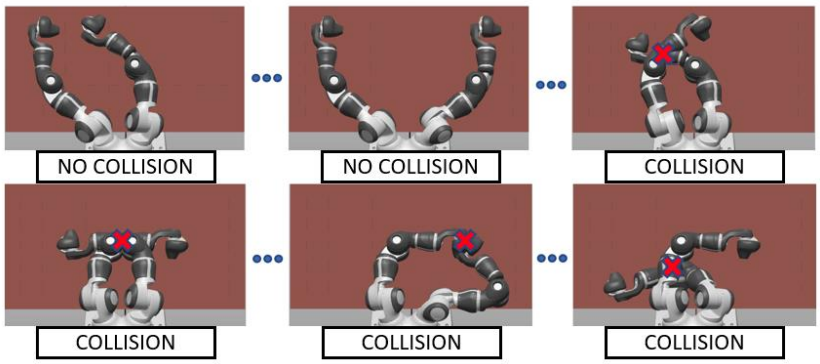


Figura 4.7 Representación visual de varias situaciones de colisión

Las posiciones XYZ en las figuras 3.6 y 3.8 están expresadas en milímetros. El robot se encuentra centrado en el eje Y (0 mm). El rango de movimiento en cada uno de los ejes es:

- X: entre 200 mm y 600 mm, con pasos de 100 mm
- Y: entre -450 mm y 450 mm, con pasos de 100 mm
- Z: entre 180 mm y 420 mm, con pasos de 120 mm

Esta configuración obedece a la rejilla de tamaño 10x5x3. La base de datos contiene un total de $(10x5x3)^2 = 22500$ de entradas con información de colisión.

La figura 4.8 muestra un extracto de la información relativa a la configuración articular del robot en cada una de las posiciones de la rejilla 3D. En las ubicaciones que el robot no puede alcanzar, se almacena guarda una configuración articular con todos los valores en ceros. La base de datos contiene un total de 150 entradas para cada brazo robot.

Gird Size	10	5																		
Coordenada											Coordenada									
X	Y	Z	J1	J2	J3	J4	J5	J6	J7		X	Y	Z	J1	J2	J3	J4	J5	J6	J7
200	-450	110	28.3071	-67.9543	13.5739	287.712	16.4658	-200.076	-97.1765		200	-450	110	0	0	0	0	0	0	0
200	-350	110	26.2926	-67.5528	36.1995	225.059	21.5618	-134.67	-96.9764		200	-350	110	0	0	0	0	0	0	0
200	-250	110	0	0	0	0	0	0	0		200	-250	110	0	0	0	0	0	0	0
200	-150	110	0	0	0	0	0	0	0		200	-150	110	0	0	0	0	0	0	0
200	-50	110	0	0	0	0	0	0	0		200	-50	110	0	0	0	0	0	0	0
200	50	110	0	0	0	0	0	0	0		200	50	110	0	0	0	0	0	0	0
200	150	110	0	0	0	0	0	0	0		200	150	110	0	0	0	0	0	0	0
200	250	110	0	0	0	0	0	0	0		200	250	110	0	0	0	0	0	0	0
200	350	110	0	0	0	0	0	0	0		200	350	110	0	0	0	0	0	0	0
200	450	110	0	0	0	0	0	0	0		200	450	110	-27.9943	-69.5378	13.8357	75.4214	17.901	198.134	95.8717
300	-450	110	35.8889	-73.0294	8.5283	274.292	37.3437	-186.671	-78.7733		300	-450	110	0	0	0	0	0	0	0
300	-350	110	34.7914	-73.0329	30.2329	248.312	40.1562	-153.641	-78.826		300	-350	110	0	0	0	0	0	0	0
300	-250	110	40.9132	-71.3243	45.774	230.153	51.075	-127.32	-80.7262		300	-250	110	-146.921	-84.3602	10.2473	140.797	102.519	80.1183	110.687
300	-150	110	56.0015	-68.5803	54.5268	221.159	67.2145	-107.825	-85.2304		300	-150	110	-127.004	-77.2845	31.6796	140.345	103.173	80.0409	106.975
300	-50	110	79.3665	-67.3488	54.8739	219.04	84.1614	-93.2776	-92.8298		300	-50	110	-104.547	-71.8193	46.8572	141.003	97.7453	84.7443	100.676
300	50	110	104.926	-69.9212	46.6444	220.072	96.4356	-83.5004	-101.353		300	50	110	-79.2979	-69.3164	55.0652	141.65	85.4799	94.581	92.4113
300	150	110	127.588	-75.5539	31.4312	221.014	101.917	-78.8839	-107.959		300	150	110	-56.1863	-70.5287	54.721	139.108	68.5758	109.187	84.8624
300	250	110	147.566	-82.8915	9.9529	220.776	101.396	-79.1169	-111.969		300	250	110	-41.1886	-73.2027	45.9988	129.903	52.5704	128.625	80.1727
300	350	110	0	0	0	0	0	0	0		300	350	110	-35.0401	-74.7907	30.5097	112.184	41.7887	154.384	77.9657
300	450	110	0	0	0	0	0	0	0		300	450	110	-36.0484	-74.5796	8.8641	87.3762	38.8144	186.254	77.5539
400	-450	110	49.5616	-69.7269	-5.6954	273.934	51.4743	-186.306	-68.9572		400	-450	110	0	0	0	0	0	0	0
400	-350	110	48.3002	-70.0127	16.2287	257.818	52.8975	-160.055	-68.6045		400	-350	110	0	0	0	0	0	0	0
400	-250	110	53.8106	-67.6265	30.4277	245.532	59.3102	-137.352	-70.3253		400	-250	110	-153.365	-71.7122	-7.0247	127.832	90.5261	90.7902	112.776
400	-150	110	66.3361	-63.462	37.6143	238.105	68.7932	-118.595	-74.7927		400	-150	110	-131.563	-64.3613	15.3182	127.316	91.33	90.312	104.13
400	-50	110	85.5973	-59.7037	37.338	234.918	78.6865	-104.098	-83.0648		400	-50	110	-108.616	-60.8749	29.9231	126.928	87.7101	95.1364	93.298
400	50	110	108.843	-59.1008	29.639	234.125	86.2402	-94.3073	-94.3204		400	50	110	-85.7753	-61.5418	37.6112	125.911	80.216	104.964	82.3721
400	150	110	132.135	-62.8009	15.0092	233.973	89.9686	-89.564	-105.535		400	150	110	-66.7742	-65.2599	37.8927	122.573	70.3684	119.431	74.1924
400	250	110	154.068	-70.5181	-7.3798	233.702	89.3645	-90.1939	-114.472		400	250	110	-54.3267	-69.3376	30.7301	115.199	60.9228	138.001	69.5872

Figura 4.8 Base de datos – Configuración articular y viabilidad de la posición

Esta base de datos es específica para un modelo de robot en particular, y unas dimensiones de la rejilla 3D determinadas. Portar esta solución a otro modelo de robot implica repetir este estudio previo.

Dada una determinada posición del efector final del robot, existen múltiples configuraciones articulares que satisfacen esa posición. Una de las restricciones en este estudio consiste en fijar la orientación de la pinza hacia la mesa de trabajo. Esta medida reduce significativamente las opciones, y simplifica la posterior detección de colisión. Este tipo de configuración se mantiene durante todo el proceso, incluyendo los desplazamientos en el área de movimientos y las operaciones de pick-and-place. Durante la operación de pick-and-place, la pinta desciende, y dado que su orientación constantemente apunta hacia la pieza, únicamente necesita cerrar la pinza para agarrarla. No obstante, entre las múltiples configuraciones posibles que cumplen esta orientación de la pinza, se selecciona una de entre

todas, siguiendo un criterio basado en asegurar la suavidad y continuidad de la trayectoria del robot.

Este criterio favorece configuraciones donde los codos de los brazos del robot apuntan hacia el extremo opuesto del otro. Este escenario reduce las probabilidades de colisión entre robots. Por otro lado, se comprueba que cada brazo robot es capaz de realizar una transición entre cada uno de los puntos de la rejilla 3D sin llegar a una limitación articular. Bajo estas condiciones, es seguro afirmar que cualquier trayectoria del robot basada en transiciones entre puntos de la rejilla es continua.

4.2.3.1 Generación de la base de datos

El análisis de las situaciones de colisión, viabilidad de posición y configuración articular se ha llevado a cabo en RobotStudio®. Este análisis se ha automatizado por medio de una aplicación desarrollada en RAPID. Esta aplicación es responsable de generar las configuraciones articulares asociadas a cada posición de la rejilla 3D, verificar si existen situaciones de colisión y comprobar si un brazo no puede alcanzar una determinada posición. Al mismo tiempo genera un informe con los resultados obtenidos, que posteriormente es transformado en la base de datos que el algoritmo utiliza. La figura 4.9 indica la lógica que sigue esta aplicación para obtener la información de configuración articular, siempre que una determinada posición sea alcanzable por la pinza del brazo robot.

Análisis de la configuración articular y viabilidad de la posición

```
Mover el robot a la posición inicial
for i from 1   (XxYxZ) do
    Obtener coordenadas del algoritmo
    Chequear si la posición es alcanzable
    if posición no alcanzable then
        configuración articular ← todo ceros
    else
        Determinar configuración articular
    end
    Anotar viabilidad de la posición, configuración articular y
     coordenadas cartesianas en el informe de salida
end
```

Figura 4.9 Pseudo-código para generación de base de datos – Análisis de configuración articular

La base de datos es común para todos los experimentos realizados con las diferentes discretizaciones del espacio de trabajo (10x5x1, 10x5x2 y 10x5x3). El algoritmo indexa la información en base al plnao por los que puede moverse el brazo robot e ignora el resto de información.

La figura 4.10, por su parte muestra la técnica empleada mediante RobotStudio para determinar si existe colisión en una determinada configuración de los robots. Se ha optado por calcular la distancia mínima entre brazos, permitiendo al algoritmo configurar un cierto umbral por debajo del cual se considera colisión.

```
Análisis de colisiones
    for each altura rejilla then
    |   Obtener distancia entre brazos
    end
    Mover los brazos del robot α & β a su posición inicial
    for each valor XYZ do
    |   for each valor XYZ do
    |   |   Obtener configuración articular de ambos brazos α & β
    |   |   Mover α and β a dicha configuración
    |   |   Chequear si ambas posiciones son alcanzables
    |   |   if ambas son alcanzables then
    |   |   |   Medir distancia entre los enlaces de cada brazo
    |   |   |   Distancia mínima ← la Obtener distancia mínima entre brazos
    |   |   else
    |   |   |   Distancia mínima ← 0
    |   |   end
    |   |   Anotar coordenadas y distancia mínima en el informe
    |   end
    end
```

Figura 4.10 Pseudo-código para generación de base de datos – Análisis de colisiones

4.2.4 *Diseño del modelo del sistema*

El modelo del sistema representa el problema del mundo real ignorando los detalles irrelevantes y solo conservando los esenciales para la solución. En este estudio se ha introducido un par de modificaciones respecto al sistema empleado en el capítulo anterior. Por un lado, se introduce un robot comercial en lugar del robot planar simple, y por otro se añade una nueva característica al problema de pick-and-place: colaborar en el transporte de una pieza por parte de ambos brazos.

La presencia de un robot comercial requiere de unos métodos diferentes para evaluar las situaciones de colisión entre robots, la viabilidad del posicionado del efector final en una determinada ubicación y la selección de una configuración articular apropiada para cada ubicación en el espacio de trabajo. Estos métodos están descritos en la sección anterior. Aunque en este estudio toda la información está obtenida desde RobotStudio®, la idea es portable a otros modelos de robot de otros fabricantes.

Por otro lado, en esta configuración con el robot YuMi, las operaciones de pick-and-place son reales, es decir, la pinza desciende hacia la zona de las piezas, agarra o deja la pieza y vuelve a subir al área de movimiento. Esta operación de pick-and-place está modelada de una manera alternativa a los movimientos regulares en el área de movimientos. Incluir el plano de las piezas como parte del área de movimientos incrementaría notablemente el tamaño del espacio de estados del sistema.

Existen situaciones donde una pieza no es alcanzable por uno de los brazos en su posición de inicio, y no es alcanzable por el otro brazo en su posición de destino. En este escenario, ninguno de los brazos es capaz de procesar dicha pieza. En este estudio se incorpora la posibilidad de que ambos brazos colaboren para el transporte de dicha pieza. Uno de los brazos es el encargado de recoger la pieza en su posición de inicio, transportarla hasta un punto intermedio alcanzable por ambos brazos, y depositarla en ese punto. El otro brazo, recoge la pieza de nuevo y la transporta a la posición de destino.

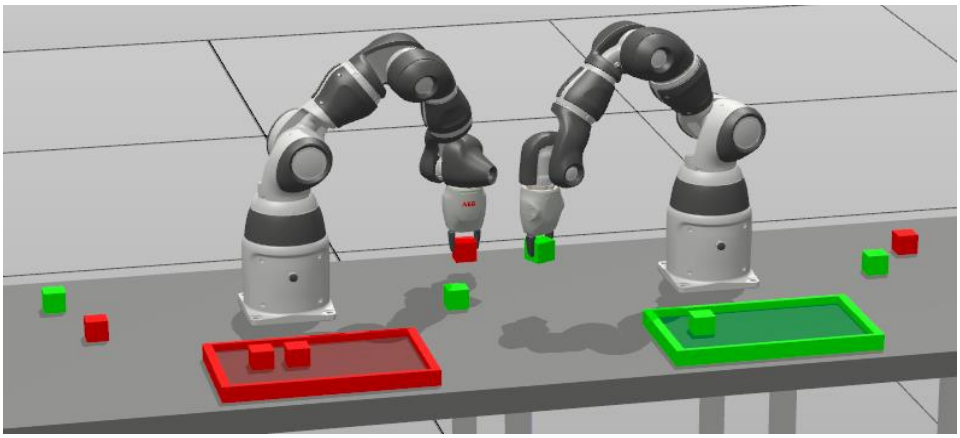


Figura 4.11 Representación de operación colaborativa entre brazos para transportar una pieza

4.2.4.1 *Espacio de estados*

El estado del sistema refleja la situación actual del proceso. Está formado por la mínima y necesaria información que se necesita para modelar la esencia del proceso. Al igual que en el estudio anterior, es necesario conocer la posición de los brazos del robot, la posición de las piezas y el estado de ambos. Adicionalmente, es necesario introducir nueva información para soportar las nuevas características de este estudio: robot comercial con operación completa de pick-and-place y colaboración para transportar una pieza.

Tabla 4.2 Variables internas del modelo del sistema formado por el robot YuMi y K piezas

Variable	**Rango**	**Descripción**
Posición del robot i	[0- XYZ]	Coordenadas XY de la posición del efector final del robot i en la rejilla 3D
Estado del robot i	0-K	Indica la pieza que el robot i está transportando (1-K) o 0 si no tiene agarrada ninguna pieza, para todo i =1..N
Estado de la pieza j	0-2	Indica si la pieza todavía está en su posición de inicio (0), ha sido transportada hasta un punto intermedio (1) o ya no reposa sobre la mesa de trabajo (2), para todo j = 1…K
Estado de la pinza i	[0 – P]	Estado de la operación de pick-and-place para cada robot i

La operación de agarre o recogida de la pieza puede modelarse de múltiples formas. Incluir una nueva variable interna (estado de la pinza) para reflejar esta situación es la opción más directa, pero incrementaría las dimensiones del espacio de trabajo significativamente. En concreto, la operación de pick o place requiere de (P + 1) pasos de tiempo. Un paso de tiempo para el cierre o la apertura de la pinza, P/2 pasos de tiempo para el descenso de la pinza y otros P/2 pasos de tiempo para volver al área de movimiento.

$$\Upsilon = (P+1)^N = 3^2 = 9 \tag{8}$$

donde Υ sería el factor de incremento del espacio de estados. Para una configuración con dos robots (N = 2) y únicamente un paso de tiempo para el descenso o subida de la pinza (P = 2), el espacio de estados se multiplicaría por nueve.

Con el fin de reducir las dimensiones del problema, se ha optado por modelar la información de la operación de pick-and-place de un modo alternativo. El rango de valores válidos para la posición del robot se ha extendido P·K valores. El rango [0-XYZ] está dedicado a la posición del efector final, mientras que los valores a partir de XYZ, que en la configuración 10x5x3 sería 150, hacen referencia al estado de la operación de pick-and-place. Dado que se trata de un escalar que solo puede contener un valor, ambas informaciones son excluyentes. O bien recoge la información de la posición del robot o bien el estado de la operación de pick-and-place. Esta solución permite contener el aumento del espacio de estados:

$$\Upsilon = \left(\frac{X \cdot Y \cdot Z + P \cdot K}{X \cdot Y \cdot Z}\right)^N = \left(\frac{10 \cdot 5 \cdot 3 + 2 \cdot 4}{10 \cdot 5 \cdot 3}\right)^2 = 1.11 \quad (9)$$

En las mismas condiciones, el aumento del espacio de estados es residual (11%) frente al uso de una nueva variable independiente de estado (900%).

No obstante, es crucial conocer la posición del robot en todo momento, dado que un problema modelado bajo el Proceso de Decisión de Markov no puede depender de información previa. Es decir, durante la operación de agarre o depositado de la pieza, la información de la posición del brazo robot debe estar disponible igualmente. En esta situación el término añadido a las posibles ubicaciones del brazo robot (X·Y·Z) es:

$$T = P \cdot i + p_i \quad (10)$$

donde P es un valor constante (P=2 en este estudio), i es la pieza (0 ..K-1) y p_i es el estado de la pinza (0, 1 o 2) del robot i.

Esta forma de codificar el estado de la pinza es posible gracias a que la posición del robot se puede inferir del resto de variables internas del estado cuando está realizando la operación de pick-and-place.

Tabla 4.3 Valores para el estado de la pinza en el modelo del sistema

Estado de la pinza	Descripción
0	Robot en movimiento normal en la rejilla 3D
1	Robot en el plano de las piezas
2	Robot ha agarrado o soltado la pieza

Durante la operación de pick-and-place, el término añadido a la posición del robot permite conocer la pieza sobre la que se está actuando (pi). Esta información junto al estado de la pieza permite conocer si el brazo está agarrando la pieza o soltándola. Esta información a su vez, se emplea para recuperar la posición de inicio o destino de dicha pieza, que es la posición donde el robot se encuentra en estos momentos.

```
Inferencia de la posición del robot
if valor < XYZ then
    posición ← valor
else
    pieza = (valor – XYZ / K)
    if estado pieza == 1 then
        posición <- posición inicial de la pieza
    else if estado pieza == 2 then
        posición ←posición intermedia de la pieza
    else
        posición ←posición final de la pieza
    end
end
```

Figura 4.12 Lógica empleada para determinar la posición de un brazo durante la operación de pick-and-place

Las constantes internas del modelo del sistema incluyen las del modelo del capítulo anterior (robot planar), además de las nuevas en referencia a la operación de pick-and-place (P) y la posibilidad de incluir estados intermedios (T) para colaborar en el transporte de una pieza. La posición inicial y final de las piezas en este caso es tridimensional. El valor de T es 0 para indicar que no se soporta la colaboración entre brazos para transportar una pieza. Si T > 1, indica la cantidad de posiciones intermedias dedicadas para una posible maniobra de transporte colaborativo. La constante P/2 indica la cantidad de pasos de tiempo necesarios

para descender a recoger o depositar la pieza, o ascender para volver al plano de movimientos. Este valor dependerá de la distancia entre el área de movimientos y el área de las piezas. En este estudio, el valor de esta constante se ha fijado a 2, es decir, solo es necesario un paso de tiempo para descender o ascender entre ambas áreas de trabajo.

Tabla *4.4* Constantes internas del modelo del sistema formado por el robot YuMi y K piezas

Variable	Rango	Descripción
Posición inicial de la pieza j	X: [0-9], Y: [0-4], Z: [0-2]	Coordenadas XYZ de la posición inicial de la pieza j, para todo j = 1..K
Posición final de la pieza j	X: [0-9], Y: [0-4] , Z: [0-2]	Coordenadas XYZ de la posición final de la pieza j, para todo j = 1..K
N	2	Número de robots
K	4	Número de piezas
T	Arbitrary	Número de estados intermedios (>=0)
P	Arbitrary	Número de pasos de tiempo necesarios para la operación de pick-and-place

Dado que la arquitectura de control en este estudio es una extensión de la del capítulo anterior, el modelo del sistema hereda las propiedades de éste: discreto, finito, y mono-agente. De igual modo, el control está basado en una arquitectura de dos capas. Desde la perspectiva de la capa superior, el brazo robot se desplaza discretamente entre puntos adyacentes de la rejilla 3D. Desde la perspectiva de la capa inferior, el brazo robot realiza trayectorias continuas durante la aproximación o transporte de las piezas.

4.2.4.1.1 Estado inicial y estados finales

Las condiciones iniciales determinan el estado inicial del sistema, que es único. Por el contrario, existen múltiples estados finales. Todos los estados que indican que todas las piezas han sido depositadas se consideran estados finales o estados terminales.

Tabla *4.5* Estado inicial del sistema

Posición robot 1	Posición robot 2	Estado robot X	Estado pieza X
Específica	Específica	0	1

En este caso, las características nuevas añadidas al modelo (navegación tridimensional, posiciones intermedias, operación completa de pick-and-place) no tienen ninguna implicación en el estado inicial del modelo, que coincide con el estado descrito en el estudio anterior con el robot planar.

Tabla *4.6* Estado final del sistema

Posición robot 1	Posición robot 2	Estado robot X	Estado pieza X
Arbitraria	Arbitraria	0	0

De igual manera, tampoco tienen ninguna influencia en la definición de los estados finales, ya que este momento no se encuentra realizando una operación de pick-and-place ni las piezas están en una posición intermedia.

4.2.4.1.2 Tamaño del espacio de estados

El concepto de espacio de estados captura todas las posibles configuraciones del estado del modelo del sistema. El tamaño del espacio de estados obedece a la siguiente fórmula:

$$Num\ states\ = (X \cdot Y \cdot Z + K \cdot P)^N \cdot (K+1)^N \cdot (2+T)^K \quad (11)$$

Para el caso particular de la configuración 10x5x3, dos brazos robots, 4 piezas, un paso de tiempo para el ascenso o descenso hacia la pieza, y una posición intermedia, la cantidad de estados brutos posibles en el sistema es:

$$Num\ states\ = (10 \cdot 5 \cdot 3 + 4 \cdot 2)^2 \cdot (4+1)^2 \cdot (2+1)^4 \quad = 50.552.100 \quad (12)$$

Este valor es significativamente mayor que el calculado en el capítulo anterior, pero es la diferencia que permite añadir navegación tridimensional, posiciones intermedias para transporte colaborativo y modelar correctamente la operación de pick-and-place.

Cabe resaltar que T=1 indica que cada pieza puede disponer de una posición intermedia para el transporte, pero no necesariamente la misma en todas las piezas.

4.2.4.1.3 Transporte colaborativo de las piezas

Una de las nuevas características en este estudio es el transporte colaborativo de las piezas como se puede apreciar en la figura 4.13.

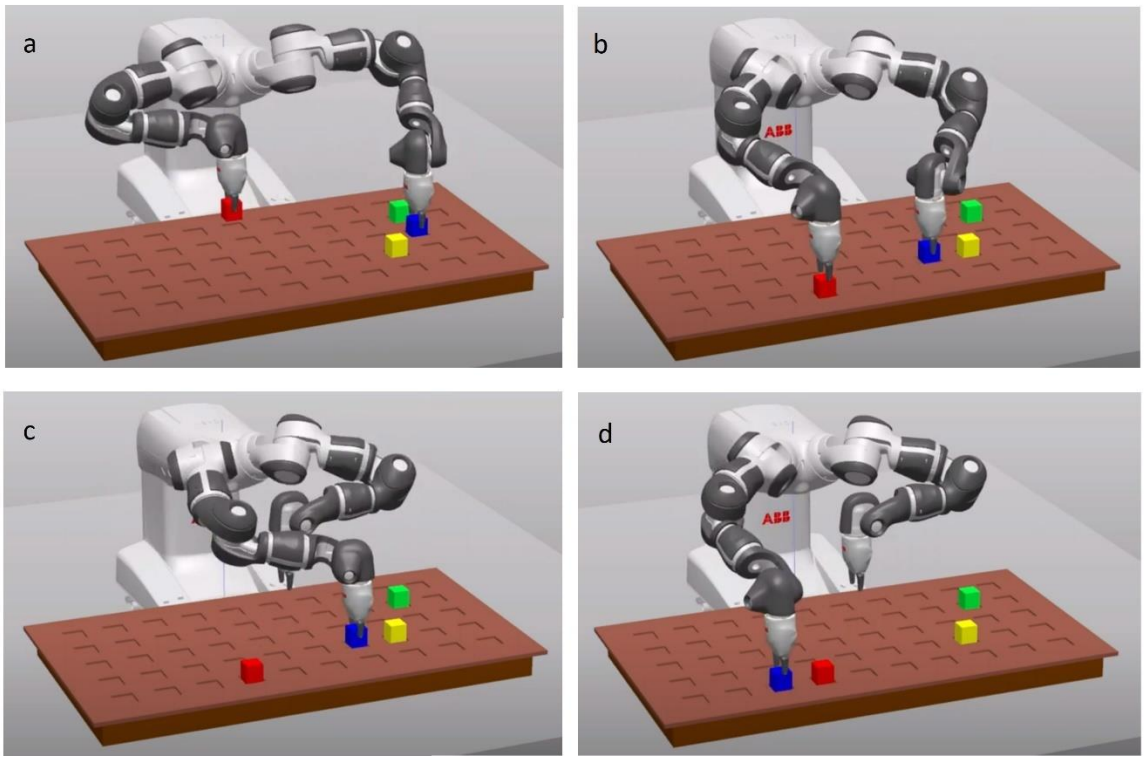
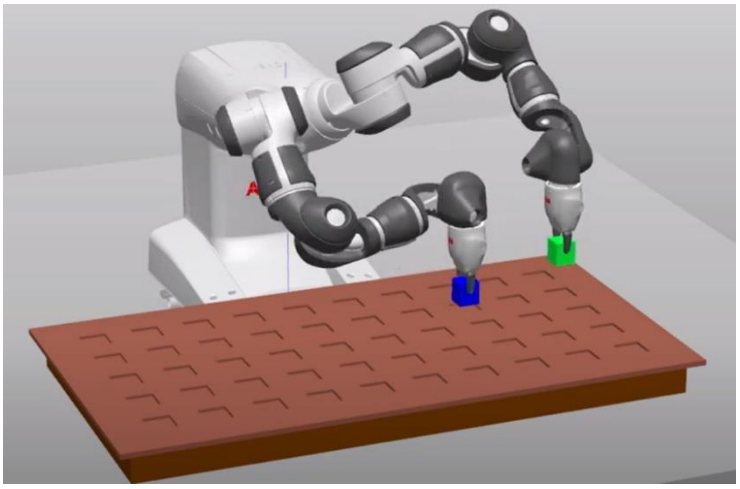


Figura 4.13 Transporte colaborativo de una pieza (color azul)

En la primera instantánea (a) se observa al brazo izquierdo cogiendo la pieza azul que está situada en su posición de inicio. Unos instantes después (b), deposita la pieza en una posición intermedia, ya que el brazo no es capaz de alcanzar la posición de destino de la pieza. En (c), el brazo derecho recoge la pieza y la transporta hasta (d) su posición de destino.

4.2.4.1.4 Correspondencia visual del estado interno del sistema

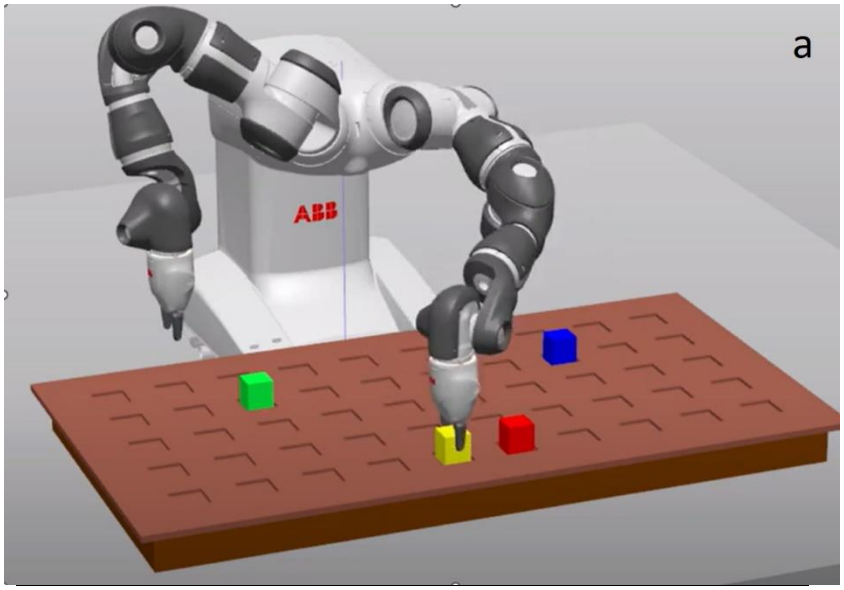
El modelo del sistema solo captura las características esenciales del proceso, e ignora el resto de información del proceso real. A continuación, se muestran algunas instantáneas de la simulación junto a la representación interna equivalente del estado.



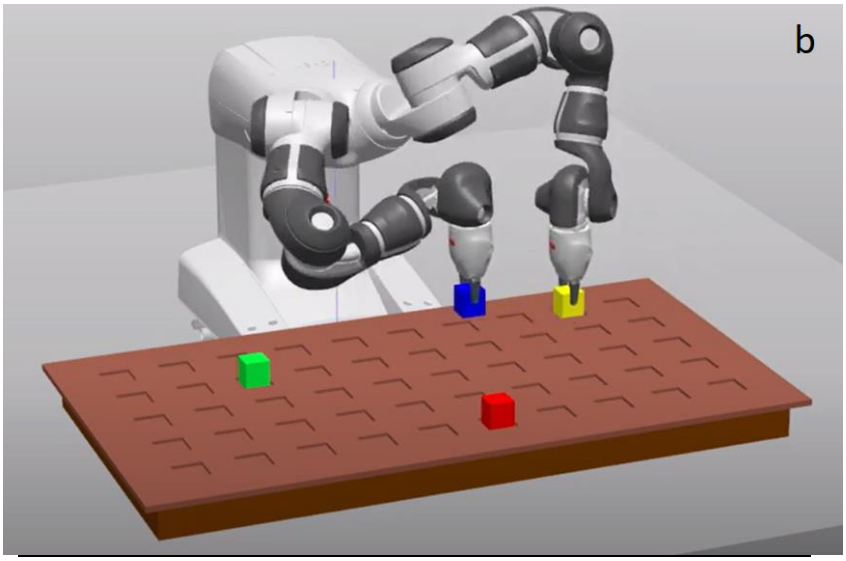
Variable	Valor
Posición robot 1 (izq)	[1, 0 ,0]
Posición robot 2 (der)	[-, -, -]
Estado robot 1	1
Estado robot 2	0
Estado pieza 1 (verde)	0
Estado pieza 2 (azul)	1
Estado pinza 1	0
Estado pinza 2	2

Figura 4.14 Relación entre proceso real y modelado interno - configuración con 2 piezas

En esta configuración de dos piezas, el brazo izquierdo (robot 1) está transportando la pieza 1 (verde), y el brazo derecho acaba de coger la pieza 2 (azul), pero todavía se encuentra situado en el plano de la pieza. El estado del robot 2 no cambia a valor 2 (pieza azul) hasta que el brazo asciende de nuevo al área de movimientos. La pieza uno está siendo transportada, por lo cual su estado es 0, mientras que la pieza 2 todavía se encuentra en la operación de pick, y su valor es todavía 1. Respecto a la pinza, el robot 1 está transportando la pieza, luego no está realizando una operación de pick ni de place y su valor es 0. Por el contrario, el robot 2 está en el proceso de la operación de pick de la pieza 2, justo después de cerrar la pinza, por lo que el valor del estado de la pinza es 2.



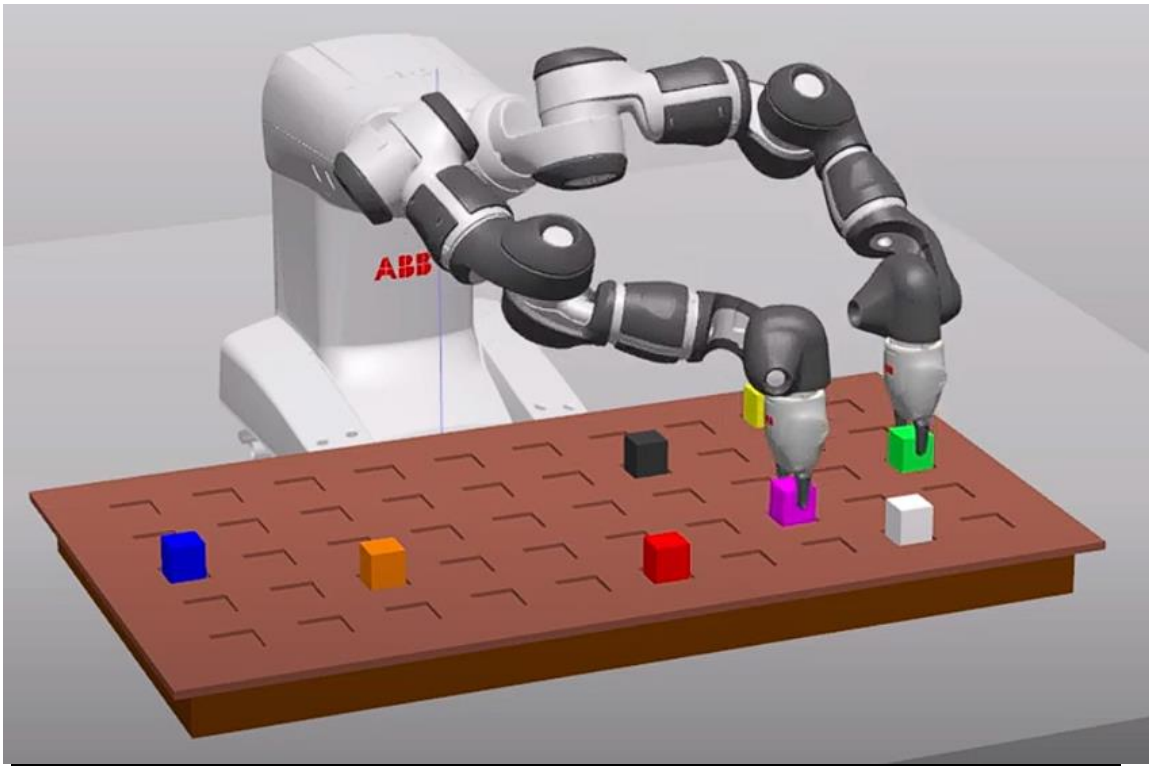
Pos R1	Pos R2	Estado R1	Estado R 2	Estado Pinza 1
[-, -, -]	[8, 0, 0]	0	0	2
Estado P1 (verde)	**Estado P2 (azul)**	**Estado P3 (amarilla)**	**Estado P4 (roja)**	**Estado Pinza 2**
0	1	1	0	0



Pos R1	Pos R2	Estado R1	Estado R 2	Estado Pinza 1
[-, -, -]	[3, 1, 0]	2	1	1
Estado P1	**Estado P2**	**Estado P3**	**Estado P4**	**Estado Pinza 2**
0	0	1	0	0

Figura 4.15 Relación entre proceso real y modelado interno - configuración con 4 piezas

La figura 4.15 muestra un par de instantáneas correspondientes a una simulación con cuatro piezas. En la foto de arriba (a), la situación es la siguiente: la piezas verde y roja ya han sido procesadas y se encuentran en su posición de destino. La pieza azul todavía se encuentra en su posición de inicio, mientras que la amarilla acaba de ser recogida por el brazo 1 (izquierda). Dado que el brazo 1 se encuentra en medio de una operación de pick-and-place, su posición no está codificada explícitamente en la variable de la posición del robot, sino que se deriva de la información del resto de variables del estado. En la foto de abajo (b), el brazo 1 está completando el procesado de la pieza amarilla, mientras que el brazo 2 (derecho) acaba recoger la pieza azul. El brazo 2 se encuentra en la zona de las piezas, pero todavía no ha soltado la pieza amarilla, por lo tanto, el estado de la pinza es 1.



Pos R1	Pos R2	Estado R1	Estado R 2	Estado Pinza 1	Estado Pinza 2	Estado P1 (azul)
[-,-,-]	[-,-,-]	4	7	1	1	0
Estado P2 (naranja)	**Estado P3 (rojo)**	**Estado P4 (negro)**	**Estado P5 (rosa)**	**Estado P6 (amarillo)**	**Estado P7 (blanco)**	**Estado P8 (verde)**
0	1	0	0	0	1	0

Figura 4.16 Relación entre proceso real y modelado interno - configuración con 8 piezas

La figura 4.16 muestra una instantánea extraída de una simulación con 8 piezas. En este caso, el procesado de las piezas de color azul, naranja, y amarillo han sido completado, por lo tanto, el estado de estas piezas es 0. Las piezas rosa y verde están a punto de ser depositadas en su zona de destino, por lo cual su estado es igualmente 0. Dado que ambos brazos se encuentran en la operación de place, sus posiciones en la rejilla son desconocidas explícitamente. Por otro lado, ambos brazos se encuentran en la misma situación; han descendido al área de las piezas, pero no han soltado todavía la pieza. En esta situación, el estado de la pinza en ambos brazos es 1. Finalmente, el estado de los brazos indica la pieza que está transportando cada uno. El brazo 1 (izq) está depositando la pieza verde (7), mientras que el brazo 2 hace lo propio con la pieza rosa (4).

4.2.4.2 Espacio de acciones

De acuerdo con la arquitectura definida, cada brazo del robot puede moverse hacia cualquiera de los puntos adyacentes en el espacio de trabajo. Considerando que el espacio de trabajo ha sido discretizado tridimensionalmente, y que el robot soporta movimientos ortogonales y diagonales en todos los ejes, el número de posibles movimientos asciende a 26 (9 posiciones debajo de la posición actual, 9 arriba, y 8 en el mismo plano). En determinadas situaciones, la acción más favorable es mantenerse en reposo, ya sea porque no hay ninguna pieza pendiente de procesar para un brazo o porque detenerse permite avanzar al otro brazo hacia su objetivo. En este sentido, mantenerse en reposo es una acción adicional en el sistema. Finalmente, la operación de pick-and-place se modela de una manera alternativa a los movimientos regulares del robot. Las acciones de pick y place representan el conjunto de movimientos necesarios para completar la operación (descenso de la pinza, agarre o depositado de la pieza y ascenso de la pinza). El número de acciones posibles en cada brazo es:

$$Num\_acciones\_robot = \left(N_{movimientos} + N_{reposo} + N_{pick} + N_{place}\right) = 26 + 1 + 1 + 1 = 29$$

y el número de acciones en el sistema incrementa exponencialmente con el número de

robots (o brazos robot), ya que cada robot se mueve independientemente del resto. En el caso particular del robot YuMi con dos brazas resulta en 841 acciones posibles

$$Num\_acciones\_sistema = Num\_acciones\_robot^{N} = 29^2 = 841$$

La complejidad del problema, desde la perspectiva de recursos necesarios de memoria y de cómputo, ha aumentado ostensiblemente respecto al robot planar 2GDL. Para aliviar esta situación, en este estudio se han considerado diferentes modos de operación donde un el robot soporta un subconjunto de las posibles acciones. El documento se refiere a estos modos en adelante, como modos de navegación.

4.2.4.2.1 Modos de navegación

Se han definido cuatro modos de navegación, desde el más liviano donde el robot solo puede moverse ortogonalmente en el plano hasta el más completo donde puede moverse en cualquiera de las 26 direcciones descritas en el apartado anterior:

- Modo 1: Permite movimientos ortogonales (N, S, E y O) en el plano, como muestra la figura 4.17a. Este modo no permite navegar en el espacio tridimensional, solo en el plano inferior de la rejilla 10x5x3 definida en este estudio.
- Modo 2: Permite los movimientos del modo 1, y añade movimientos diagonales (Norte-Este, Norte-Oeste, …) en el plano, como muestra la figura 4.17b.
- Modo 3: Permite los movimientos del modo 2, y añade movimientos verticales hacia arriba y abajo, como se observa en la figura 4.17c. Este es el primero de los modos en permitir la navegación tridimensional del robot
- Modo 4: Permite los movimientos del modo 3, y añade movimientos diagonales en las tres dimensiones, como se indica en la figura 4.17d. Este modo permite todos los movimientos (26) soportados en esta arquitectura.

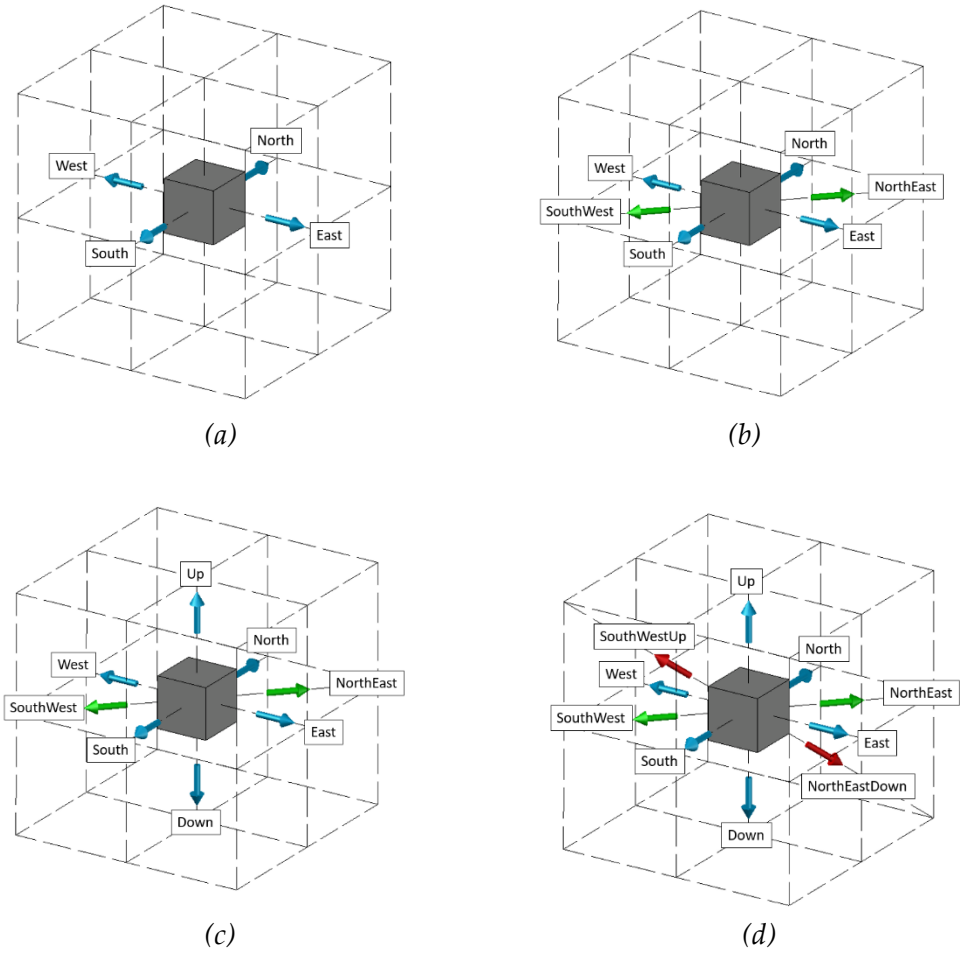


Figura 4.17 Modos de navegación: a) Movimientos ortogonales en el plano, b) Añadido movimientos diagonales en el plano, c) Añadido movimiento de subir y bajar, d) Añadido movimientos diagonales en todas las direcciones

El modo de navegación 1 se corresponde con los movimientos soportados por el robot planar 2GDL del capítulo anterior, y eleva el número de acciones del sistema a 49. El resto de los modos incrementa este número hasta alcanzar las 841 posibles acciones del modo 4. El modo de navegación es un parámetro adicional del sistema que puede configurarse, y que tiene un efecto directo en los recursos computacionales y de memoria requeridos. Por ejemplo, si aumenta el número de piezas, una manera de combatir la complejidad del sistema es reducir el modo de navegación.

4.2.4.3 *Comportamiento del modelo del sistema*

Dado que la tarea está modelada como un problema de toma de decisiones modelado bajo el marco del Proceso de Decisión de Markov, como muestra la figura 3.10, el comportamiento del sistema (o entorno) es reactivo a la acción realizada por los robots (o agente). Dada una determinada acción *a* realizada en un estado *s*, genera una transición del sistema a un estado *s'* y el sistema proporciona una valoración (recompensa) *r* de la acción tomada en dicho estado *s*. La figura 4.18 muestra el pseudo=código correspondiente al comportamiento del modelo del proceso de pick-and-place:

```
Datos: estado s: tupla que contiene el valor de las variables internas del modelo de sistema
       acción a: tupla que contiene las acciones a realizar por cada uno de los robots
Resultado: siguiente estado s' y recompense r
Function ModelBehaviour (estado, acción)
    Chequear si la acción es válida en este estado
    if acción is not válida then
        return tupla(estado s, PENALIZACIÓN ALTA
    end
    Determinar el siguiente estado s' y la recompense obtenida tras realizar la acción a en el estado s
    Chequear que no se excede los límites de la rejilla por parte de ninguno de los robots
    Chequear colisión entre robots
    for i from 1 to N do
        Chequear viabilidad de la posición del robot en el nuevo estado
    end
    if alguna comprobación falla then
        # El estado no cambia
        recompensa r ← PENALIZACIÓN ALTA
    else
        if tarea completada then
            recompensa r ← RECOMPENSA ALTA
        else
            recompensa r ← PENALIZACIÓN PEQUEÑA
        estado s ← siguiente estado s'
        end
    end
    return tupla(esado s, recompense r)
end
```

Figura 4.18 Pseudo-código del comportamiento del modelo del sistema

El comportamiento del modelo del sistema está implementado en Python. La implementación realiza una serie de verificaciones tanto de la acción *a* ejecutada en el estado *s*, como de la integridad del estado resultante *s'*. Respecto a las acciones, realiza verificaciones tales como asegurar que la pinza del brazo se encuentra en las coordenadas iniciales o finales de alguna de las piezas, y que la pieza se encuentra en su posición de inicio o de destino respectivamente. En cuando al nuevo estado, se realizan verificaciones tales como asegurar que no deriva en una situación de colisión entre robots, que ambos brazos son capaces de posicionarse en la nueva ubicación y que ninguno de los brazos excede los límites de la rejilla definida en el problema.

4.3 Solución

La tarea de pick-and-place está modelada como un problema de toma de decisiones. El modelo creado para esta tarea en este estudio es determinista, mono-agente y discreto, como indica la tabla 3.7 del capítulo anterior. Dadas estas propiedades, existen múltiples técnicas que pueden ser empleadas para resolver el problema, En este estudio se proponen tres métodos diferentes: Value Iteration (MDP), BFS (grafos) y PDDL.

El proceso para generar una solución consiste en múltiples tareas clasificadas en 3 fases, como se observa en la figura 4.19. La primera fase se centra en las tareas a nivel de proyecto que deben realizarse una única vez. Existen elementos en el problema que son comunes para todos los experimentos realizados en este estudio, tales como el robot YuMi, o la resolución de la rejilla (10x5x3) en la discretización del espacio de estados. Si alguno de estos elementos cambia, se considera un proyecto diferente (por ejemplo, introducir un modelo de robot diferente en la tarea de pick-and-place), y esta fase debe ejecutarse de nuevo. La tarea principal de esta fase es generar la base de datos que incorpora la información de situaciones de colisión, de viabilidad de ubicación para cada brazo, y la configuración articular asociada a cada posición en el espacio. En este estudio, esta fase se apoya en la herramienta RobotStudio®, ya que el robot empleado pertenece al fabricante ABB. En el caso de introducir un robot de otro fabricante, habría que utilizar las herramientas que proporcione éste.

La base de datos generada en la fase de proyectos es usada como entrada en las siguientes fases, tanto para generar el modelo del sistema, como para generar una solución para éste.

La fase a nivel de tarea consiste en todo el trabajo restante que se puede realizar antes de poner la solución en producción. Esta fase depende de elementos que pueden variar en el proceso dentro de un mismo proyecto, como, por ejemplo, el número de piezas a procesar. Otro de los elementos que puede variar es el número de acciones disponibles, en función del modo de navegación empleado. Si ninguno de estos elementos cambia en el proyecto, esta fase es suficiente con ejecutarla una vez antes de poner el sistema en producción. Esta fase únicamente aplica a los métodos de solución MDP y grafos.

La fase a nivel de ejecución se divide en tres partes, dependiendo de la técnica empleada para generar una solución. Una vez generada la solución, aparece la figura del planificador de tareas (en inglés, Task Plan) que es responsable de coordinarse con el robot para ejecutar cada una de las acciones en el momento adecuado. En este estudio, los experimentos se han realizado en RobotStudio®, por lo cual el planificador de tareas se comunica con el simulador en lugar de con un robot real.

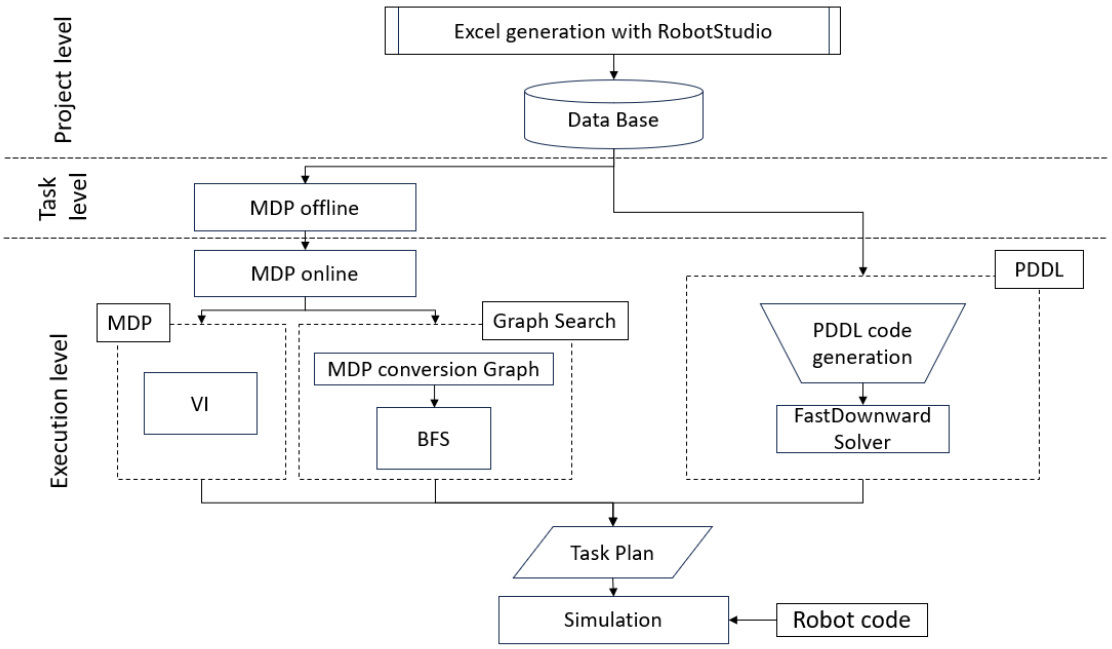


Figura 4.19 Flujo del proceso para generar una solución basado en tres técnicas diferentes: MDP, grafos y PDDL

La comunicación se realiza por medio de un archivo, en el cual se describe el conjunto de trayectorias y operaciones de pick-and-place que cada robot debe realizar. En un proceso real, esta comunicación debería ser continua y en tiempo real, observando en cada paso de tiempo el estado actual del sistema y comunicando la acción apropiada para cada robot en base a ello.

En el caso de emplear la técnica de MDP, el primer paso es completar el modelo del sistema (MDP). Generar el modelo es un proceso bastante costoso (depende de las dimensiones del espacio de estados, y del espacio de acciones). Este modelo se genera en su mayoría en la fase a nivel de tarea. No obstante, existe una parte de él que no puede completarse hasta la fase de ejecución. Se trata de la parte relacionada con la operación de pick-and-place. Dado que se desconoce la posición de las piezas de antemano, no se puede valorar si la acción de pick o place es válida en una determinada situación. Al inicio de cada iteración en la tarea de pick-and-place, el sistema es informado de la posición inicial y final de las piezas. Con esta información, se completa el modelo del sistema. Este enfoque basado en dividir la generación del modelo en dos fases permite reducir significativamente el coste computacional necesario en tiempo real. Tras generar el modelo del sistema, el siguiente paso es encontrar una solución para éste. Para ello, en este estudio se emplea el algoritmo Value Iteration.

Otro de los métodos empleados es el basado en grafos. Dado que el modelo es discreto, finito, determinista y mono-agente, el modelo MDP puede ser fácilmente transformado en un grafo. Una vez convertido en un grafo, existen múltiples algoritmos de búsqueda que pueden emplearse para encontrar una solución. En este caso, a diferencia del MDP, la solución es en forma de plan en bucle abierto, es decir, una secuencia predefinida de acciones. Si se produjera alguna situación inesperada (por ejemplo, un operador retira una pieza), esta fase sería necesario ejecutarla de nuevo para encontrar una nueva solución.

El último método comparado en este estudio es PDDL. Este método está basado en una descripción del modelo del sistema en dicho lenguaje (PDDL), y una descripción de las condiciones iniciales y los objetivos. Tanto la generación de estas descripciones, como la obtención de una solución pertenecen a las tareas a nivel de ejecución. La descripción del modelo del sistema es muy extensa, y está generada automáticamente mediante un script una vez se conocen las posiciones iniciales y finales de las piezas. Para generar una solución se utiliza un software planificador estándar. En este estudio, se ha empleado un planificador de libre uso, llamado FastDownward.

4.3.1 *MDP – Value Iteration*

El algoritmo de Value Iteration implementado en el anterior estudio es agnóstico a la complejidad del problema. En este estudio, tanto el espacio de estados como el espacio de acciones es significativamente mayor en el caso del robot planar. Además, se introducen nuevas características, tales como el modelo completo de la operación de pick-and-place y la posibilidad de colaborar en el transporte de una pieza. Independientemente de estas características, e incluso de la naturaleza del problema, cualquier tarea modelada bajo el Marco de Decisión de Markov en forma matricial puede resolverse usando el algoritmo implementado en el estudio anterior.

Todo lo descrito en el apartado 3.2.4.2.1 del capítulo anterior aplica en esta sección.

4.3.2 *Grafo - BFS*

En este enfoque, el modelo del sistema está representado mediante un grafo. Este grafo está construido en base al modelo MDP empleado en el método anterior. El primer paso es común al otro método, generar el modelo MDP del sistema. A continuación, aplicando una serie de transformaciones, se deriva un grafo equivalente al MDP.

Un grafo está formado básicamente por tres componentes: nodos, transiciones y costes asociados a estas transiciones, Estos elementos tienen una relación directa con los componentes del MDP. Los nodos tienen correspondencia con los estados del modelo MDP. Las transiciones tienen relación directa con las acciones del MDP, y los costes del grafo se corresponden con las recompensas en un MDP.

El grafo derivado del MDP, no obstante, es un modelo reducido del original, ya que descarta todas las transiciones que reciben una alta penalización, indicando situaciones indeseables, como colisión entre robots, o posición no alcanzable para uno de los brazos. En el caso del MDP, no era posible eliminar pares estado-acción, pues la versión matricial del algoritmo requiere de matrices completas. En este caso, el algoritmo empleado no opera en base a matrices, sino que recorre los nodos secuencialmente, por lo cual reducir el tamaño del grafo es una ventaja adicional.

Tabla *4.7* Transformaciones aplicadas entre MDP y grafo

MDP (recompensa)	Grafo (coste)	Significado
-20	---	Acción a no válida en estado s
+100	+1	Tarea completada
-1	+1	Resto de transciones

Todas las transiciones *s* → *s'* que reciben una alta penalización (-20) en el MDP son ignoradas en el momento de generar el grafo. Por otro lado, a diferencia del método basado en MDP donde el objetivo es maximizar la recompensa recibida a largo plazo, en un grafo, el objetivo de un algoritmo de búsqueda es minimizar el coste obtenido entre dos nodos del grafo. En este contexto, la pequeña penalización (-1) del MDP se transforma en una pequeña penalización (+1) en el grafo. Respecto a la recompensa obtenida en el estado final, o múltiples estados finales, es indiferente desde el punto de vista del grafo. Los algoritmos de búsqueda se basan en recorrer los nodos del grafo, y es suficiente con reconocer qué estados se consideran estados finales para determinar una condición de parada. El hecho de otorgar una recompensa grande en la última transición de cada una de las posibles soluciones no aporta información adicional. Por esta razón, se ha decidido unificar todos los costes del grafo (+1).

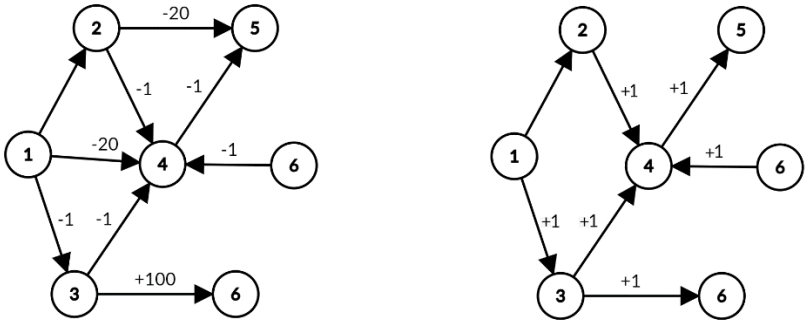


Figura 4.20 Representación de transformación entre modelo MDP y grafo

Dado que todos los costes del grafo son iguales, es posible emplear cualquier algoritmo de búsqueda para resolverlo. En este caso, se ha seleccionado Breadth First Search (BFS) como algoritmo de búsqueda. Este tipo de algoritmo recorre los nodos del grafo por niveles de ramificación. Primero visita todos los nodos de nivel 1, que son los alcanzables desde el nodo inicial. Una vez visitados todos los nodos del nivel 1, recorre todos los nodos del nivel 2, que son los que pueden alcanzarse desde los nodos del nivel 1, y así sucesivamente. El algoritmo se detiene cuando se completa el primer nivel donde se encuentra un nodo final. Dado que todos los costes del grafo son iguales, el algoritmo puede detener su búsqueda nada más encontrar el primer nodo final. No es necesario completar todos los nodos del nivel al que pertenece.

Otros algoritmos han sido considerados, tales como Dijkstra o A*, pero dado que todos los costes del grafo son iguales, no aportan ninguna ventaja, al no disponer de información adicional, en forma de costes diferentes, que ayude a guiar el recorrido del grafo de una manera más eficiente.

La figura 4.21 muestra el pseudo-código asociado al algoritmo de búsqueda implementado en este estudio, que se detiene nada más encontrar el primer estado final.

```
Datos: s0: nodo inicial
       Q: cola FIFO inicialmente vacía
Resultado: secuencia ordenada de estados entre s0 y el estado final
function BFS
    Insertar el nodo s0 en la cola Q
    Marcar nodo s0 como visitado
    // Bucle hasta encontrar el estado final
    while Q is not vacío do
        s ←extraer primer nodo de la cola Q
        foreach nodo s' alcanzable desde s do
            c ← coste de la transición s – s'
            if c == +100 then
                // Solución encontrada
                Retornar camino entre nodo inicial s0 y s'
            else if s' is not visitado and is not in Q then
                Insertar s' en Q
            End
        end
    end
    // Solución no encontrada
    retornar camino vacío
end
```

Figura 4.21 Pseudo-código de algoritmo BFS

4.3.3 *PDDL*

Planning Description Domain Language (PDDL) es un intento de estandarización de lenguajes de planificación para inteligencia artificial (en inglés, AI). Existen múltiples versiones, algunas de ellas permiten modelar incertidumbre (PPDDL), entornos multi-agente (MA-PDDL), etc…

En este estudio, el problema es determinista y mono-agente, luego se ha empleado el lenguaje PDDL oficial. El modelado del problema consiste en dos archivos. Uno modela la dinámica del sistema, lo cual es equivalente al modelo MDP. El otro recoge la configuración inicial del sistema y el/los objetivos a cumplir. El primer tipo de archivo se conoce como descripción del dominio, y está constituido por predicados y acciones. Los predicados son hechos (por ejemplo: el robot 1 está en la posición XYZ, o existe colisión en el estado actual). Estos predicados se evalúan como verdadero o falso. Las acciones son el equivalente a la función de transición en el MDP; dada una determinada acción en el estado actual, qué cambio de estado se produce en el sistema. El otro tipo de archivo se conoce como descripción del problema, y es el equivalente a la definición del estado inicial y los estados finales en el MDP, y la definición de las variables internas que lo forman (posición de cada robot, etc...).

Este método se reduce a la descripción del problema mediante estos dos archivos. El análisis del contenido de estos archivos y la generación de una solución al problema es responsabilidad de una herramienta externa. Existen múltiples planificadores que soportan el lenguaje PDDL. En este estudio se ha seleccionado uno de los más populares, y de libre uso, llamado FastDownward.

Los archivos de descripción del problema son muy extensos. En el caso del archivo de descripción del dominio, existe una sección del problema dedicada a cada una de las acciones del sistema, como puede observarse en la figura 4.22. El extracto de esa figura hace referencia a la situación donde un robot realiza la operación de pick y el otro realiza un movimiento en el plano (Norte). Este archivo contiene unas 30 mil líneas. La estructura de dicho archivo es repetitiva para cada acción del sistema, lo cual facilita la generación automática de éste mediante un script.

```
;;;  Movimientos ARM1 : down_piece - ARM2 : back  ;;;

(:action move+down_piece+back
    :parameters (?omf1 ?omf2 - agent ?x1 ?y1 ?z1 ?x2 ?y2 ?z2 ?z1n ?x2n - agent_pos )
    :precondition (and
                        (not (= ?omf1 ?omf2))
                        ;arm1
                        (at ?omf1 ?x1 ?y1 ?z1)
                        (dec ?z1 ?z1n)
                        (= z0 ?z1)
                        (= z3 ?z1n)
                        ;arm2
                        (at ?omf2 ?x2 ?y2 ?z2)
                        (dec ?x2 ?x2n)
                        (not (= z3 ?z2))
                        ;arms collision/reachability
                        (not (unreachable ?omf1 ?x1 ?y1 ?z1n))
                        (not (unreachable ?omf2 ?x2n ?y2 ?z2))
                        (not (collision ?omf1 ?omf2 ?x1 ?y1 ?z1n ?x2n ?y2 ?z2))
                  )
    :effect       (and
                        ;arm1
```

Figura 4.22 Extracto del archivo de descripción del domino

El archivo de descripción del problema es igualmente extenso. En la figura 4.23, se puede observar un extracto resumen de las partes que componen este archivo. Las zonas donde aparece la marca '…' indica que dicha zona del archivo ha sido eliminada en la figura con el fin de poder mostrar en una sola imagen un esquema de todas las partes. La primera sección consiste en las condiciones iniciales del problema. Esto incluye las posiciones de los brazos del robot, las posiciones de inicio y destino de las piezas, el estado de las piezas y el estado de los brazos del robot. A continuación, aparece un listado de las posiciones que cada brazo del robot no puede alcanzar en el espacio de trabajo, y otro listado de las posiciones donde los robots colisionan. Esta última lista es la más larga y representa el 99% del tamaño del archivo. Finalmente, se define el objetivo del problema. Como se aprecia en el extracto, el objetivo es que el estado de cada una de las piezas indique que ya ha sido depositada en su destino, independientemente del valor del resto de variables internas del sistema.

La estructura de este archivo también está muy definida, y es fácil automatizar su contenido mediante un script. Para simplificar la implementación del problema de pick-and-place, se ha generado un par de archivos (descripción del dominio y descripción del problema) para cada configuración del sistema (todas las combinaciones resultantes de los cuatro modos de navegación y un número variable de piezas, entre 2 y 10).

```
    ; Two arms (arm1 is right arm, arm2 is left arm)
    (at arm1 x0 y8 z0)
    (at arm2 x4 y6 z0)

    ; Four pieces
    ; ... initial location
    (piece_init piece1 x1 y7 z3)
    (piece_init piece2 x1 y2 z3)
    (piece_init piece3 x4 y5 z3)
    (piece_init piece4 x4 y4 z3)

    ; ... target location
    (piece_end piece1 x2 y0 z3)
    (piece_end piece2 x2 y9 z3)
    (piece_end piece3 x0 y1 z3)
    (piece_end piece4 x0 y8 z3)

    ; ... piece status
    (piece_st piece1 p0)
    (piece_st piece2 p0)
    (piece_st piece3 p0)
    (piece_st piece4 p0)

    ; ... arm carrying the piece
    (not (carry arm1)) ; ARM1 NOT CARRYING ANY PIECE
    (not (carry arm2)) ; ARM2 NOT CARRYING ANY PIECE

    ; PIECES REACHABILITY
    ; INIT
    (unreachable arm1 x0 y9 z3)
    ; (unreachable arm1 x0 y8 z3) 4
    (unreachable arm1 x0 y7 z3)
    (unreachable arm1 x0 y6 z3)
    (unreachable arm1 x0 y1 z3)
    ...
    (unreachable arm1 x0 y0 z3)
    (unreachable arm2 x4 y0 z0)

; COLISION ARM1 - ARM2 Z3
    (collision arm1 arm2 x0 y9 z3 x0 y9 z3)
    (collision arm1 arm2 x0 y9 z3 x0 y8 z3)
    (collision arm1 arm2 x0 y9 z3 x0 y7 z3)
    (collision arm1 arm2 x0 y9 z3 x0 y6 z3)
    (collision arm1 arm2 x0 y9 z3 x0 y5 z3)
    (collision arm1 arm2 x0 y9 z3 x0 y4 z3)
    (collision arm1 arm2 x0 y9 z3 x0 y3 z3)
    (collision arm1 arm2 x0 y9 z3 x0 y2 z3)
    (collision arm2 arm1 x4 y0 z0 x3 y2 z0)
    (collision arm2 arm1 x4 y0 z0 x3 y1 z0)
    (collision arm2 arm1 x4 y0 z0 x3 y0 z0)
    (collision arm2 arm1 x4 y0 z0 x4 y9 z0)
    (collision arm2 arm1 x4 y0 z0 x4 y8 z0)
    ...
    (collision arm2 arm1 x4 y0 z0 x4 y2 z0)
    (collision arm2 arm1 x4 y0 z0 x4 y1 z0)
    (collision arm2 arm1 x4 y0 z0 x4 y0 z0)

  )
; goal
  (:goal
     ; All pieces placed in their location
     (and (piece_st piece1 p2)
          (piece_st piece2 p2)
          (piece_st piece3 p2)
          (piece_st piece4 p2)
     )
  )
```

Figura 4.23 Extracto del archivo de descripción del problema para la configuración con 4 piezas

4.4 Resultados

4.4.1 *Viabilidad del sistema*

Los experimentos realizados en este estudio se han llevado a cabo dentro del simulador RobotStudio ®. Este simulador está basado en el motor de físicas de Algoryx, lo cual confiere fidelidad, rapidez y realismo a las simulaciones. El propio fabricante ABB indica que lo que se observa en el simulador es lo que se espera en el mundo real, por lo cual la información generada a través del simulador se considera correcta.

Robotstudio® se ha utilizado tanto para generar la base de datos que contiene información acerca de colisiones y configuraciones articulares, como para simular los experimentos.

4.4.2 *Comparación de la solución obtenida con los diferentes métodos: MDP, grafo y PDDL*

La arquitectura definida en este estudio determina que todos los brazos se mueven sincronizadamente. En cada paso de tiempo, cada brazo del robot realiza una transición hacia la nueva ubicación dentro de la trayectoria que debe seguir, El número de pasos necesarios para completar la tarea es directamente proporcional al tiempo empleado para ello.

Se han realizado experimentos con múltiples configuraciones, donde se ha variado el modo de navegación, el número de piezas en la tarea, y el tipo de método para resolver el problema. Las condiciones iniciales son las mismas para todos los métodos (posición inicial y final de las piezas y posición inicial de los robots). Los resultados se pueden observar en la tabla 4.8. Cuando una entrada está marcada con "--" significa que no se ha obtenido una solución para dicho experimento.

En dicha tabla se observa que todos los métodos convergen a una solución de igual calidad (mismo número de pasos), y ello es debido que todos ellos obtienen una solución óptima. La solución obtenida por cada método no es necesariamente la misma, ya que existen múltiples soluciones óptimas. De igual manera, como parece intuitivo, el número de pasos necesarios para completar la tarea aumenta con el número de piezas presentes en la configuración del experimento.

Tabla *4.8* Número de pasos necesarios para completar la tarea

		Número de pasos		
Modo navegación	**Número piezas**	**Value Iteration**	**BFS**	**PDDL**
1	2	28	28	28
	4	46	46	46
	6	55	55	55
	8	68	68	68
	9	77	77	77
	10	--	--	88
2	2	25	25	25
	4	37	37	37
	6	46	46	46
	8	56	56	56
	9	--	--	67
	10	--	--	75
3	2	25	25	25
	4	37	37	37
	6	46	46	46
	8	--	--	56
	9	--	--	67
	10	--	--	75
4	2	24	24	24
	4	35	35	35
	6	--	--	44
	8	--	--	56
	9	--	--	66
	10	--	--	73

PDDL es el único método capaz de obtener una solución en todos los experimentos. Los experimentos sin solución basados en MDP y grafo hace referencia a situaciones que requieren un coste computacional o de recursos de memoria muy alto. Por último, se observa que la solución es ligeramente mejor en determinados experimentos al usar un modo de navegación más complejo. La mejora más notable se produce al pasar del modo de navegación 1 al 2, debido a la introducción de los movimientos diagonales en el plano. La posibilidad de

desplazarse en las tres dimensiones (modos de navegación 3 y 4) tiene un efecto muy pequeño en la calidad de la solución.

4.4.3 *Coste computacional en tiempo real*

De acuerdo con el flujo presentado en la figura 4.19, existen tres fases. La fase a nivel de proyecto y la fase a nivel de tarea se realizan en el laboratorio, antes de implementar la solución en el proceso real. La fase de ejecución hace referencia a todas las tareas realizadas en tiempo real durante la operación de pick-and-place. Esta fase incluye, en el caso de los métodos basados en MDP y grafo, el completado del modelo del sistema (MDP) y la generación de una solución. En el caso de PDDL, incluye la generación automática de los archivos de descripción del domino y el problema, y la generación de una solución.

En general, independientemente del método empleado para generar una solución, cada iteración del proceso comienza con el procesado de imagen de las piezas situadas en la mesa de trabajo. De este procesado se obtiene la posición y tipo de cada una de las piezas que es notificado al sistema. El procesado de imagen no forma parte de este estudio, el cual recibe directamente una notificación con la información relativa a las piezas.

La tabla 4.9, recoge el tiempo empleado en cada experimento durante la fase de ejecución. Específicamente, hace referencia exclusivamente al tiempo empleado para obtener la solución, ignorando el resto de las tareas, tales como el completado del MDP o la generación de archivos de descripción en PDDL.

El algoritmo que requiere menos tiempo para encontrar una solución es claramente BFS. La comparación entre Value Iteration y PDDL depende del número de piezas en la configuración del experimento. Para cuatro o menos piezas, Value Iteration obtiene una solución en menos tiempo. Para más de cuatro piezas, PDDL es más eficiente. La implementación realizada en este estudio está hecha en Python. No obstante, Python es capaz de importar módulos implementados en un lenguaje más eficiente. En este estudio, el algoritmo BFS ha sido implementado en C, e importado como un módulo de Python. El algoritmo Value Iteration está basado en la librería numpy que también está implementada en C. Finalmente, el planificador empleado en PDDL es una herramienta externa, implementada en C++

Tabla *4.9* Tiempo de cómputo de la solución durante la operación

Tiempo de cómputo de la solución				
Modo navegación	**Número piezas**	**Value Iteration**	**BFS**	**PDDL**
1	2	0.4s	0,001s	15s
	4	4.6s	0,016s	16s
	6	39s	0,125s	19s
	8	4m 37s	0,973s	36s
	9	5m 57s	2,918s	1m 15s
	10	--	--	2m 56s
2	2	0.8s	0,016s	39s
	4	10s	0,047s	41s
	6	1m 26s	0,267s	47s
	8	9m 45s	1,851s	1m 20s
	9	--	--	2m 38s
	10	--	--	5m 49s
3	2	8.2s	0,032s	3m 39s
	4	1m 26s	0,376s	4m 02s
	6	11m 18s	2,907s	5m 53s
	8	--	--	16m 08s
	9	--	--	26m 54s
	10	--	--	1h 29m 24s
4	2	37s	0,110s	14m 30s
	4	6m 27s	1,146s	15m 37s
	6	--		22m 19s
	8	--		1h 27m 17s
	9	--		2h 14m 46s
	10	--		5h 24m 10s

4.4.4 *Recursos de cómputo y memoria necesarios en laboratorio*

Esta sección hace referencia exclusivamente a los métodos basados en MDP y grafo. En estos métodos, la generación del modelo del sistema se divide entre la fase de tarea y la fase de ejecución. La tabla 4.10, expone los recursos de memoria y tiempo necesarios para generar el MDP durante la fase de tarea.

El coste de generación del MDP es directamente proporcional al tamaño del espacio de estados y al tamaño del espacio de acciones. Los experimentos realizados incluyen todos los posibles modos de navegación y configuraciones que integran entre 2 y 10 piezas. El modo de navegación impacta en el tamaño del espacio de acciones, mientras que el número de piezas hace lo propio en el tamaño del espacio de estados.

El número de estados brutos (todas las posibles combinaciones de las variables internas del modelo) está calculado por medio de la ecuación (11). La primera operación durante

Tabla 4.10 *Recursos de tiempo y memoria necesarios para generar el modelo del sistema (MDP)*

Cómputo del modelo del sistema en laboratorio						
Modo de navegación (# acciones)	# Piezas	Tiempo	Disco (GB)	RAM (MB)	# estados MDP	# estados brutos
1 (49)	2	12s	0,004	8	13.568	104.976
	4	2m 06s	0,043	81	138.240	1.345.600
	6	15m 57s	0,328	619	1.052.672	12.054.784
	8	1h 43m 58s	2,142	4.037	6.864.896	90.326.016
	9	4h 28m 04s	5,265	9.923	16.875.520	236.748.800
	10	-	-	-	-	607.129.600
2 (121)	2	36s	0,010	20	13.568	104.976
	4	6m 04s	0,103	201	138.240	1.345.600
	6	45m 46s	0,783	1.523	1.052.672	12.054.784
	8	4h 49m 30s	5,107	9.968	6.864.896	90.326.016
	9	-	-	-	-	236.748.800
	10	-	-	-	-	607.129.600
3 (169)	2	05m 42s	0,090	28	13.568	853.776
	4	54m 58s	0,863	280	138.240	9.985.600
	6	6h 47m 04s	6,383	2.135	1.052.672	82.301.184
	8	-	-	-	-	571.401.216
	9	-	-	-	-	1.445.068.800
	10	-	-	-	-	3.580.825.600
4 (841)	2	28m 40s	0,444	137	13.568	853.776
	4	4h 39m 42s	4,235	1.395	138.240	9.985.600
	6	-	-	-	-	82.301.184
	8	-	-	-	-	571.401.216
	9	-	-	-	-	1.445.068.800
	10	-	-	-	-	3.580.825.600

la generación del MDP, es filtrar los estados inválidos o incoherentes. Estados incoherentes incluyen situaciones tales como ambos brazos transportando la misma pieza, o un brazo transportando una pieza que todavía no ha sido recogida. Estados inválidos incluyen situaciones tales como colisión entre brazos. El número de estados válidos resultante conforma el modelo del sistema. Se trata de un porcentaje en el rango 10-15% del número de estados brutos.

Una vez generado el modelo del sistema, se almacena en disco. El tamaño en disco es directamente proporcional al tamaño del MDP. En particular, cada estado se codifica como un entero de 32-bit y la recompensa como un entero de 16-bit, lo cual suma un total de 6 bytes por cada entrada estado-acción del MDP. Los requisitos de memoria RAM son mayores que en disco, ya que el algoritmo mantiene una copia del MDP en memoria durante su ejecución, además del espacio.

La implementación de la generación del MDP está realizada en Python. Consiste en aplicar una serie de chequeos, y cierta lógica para determinar el siguiente estado y la recompensa recibida para cada par estado-acción. Dado que es un proceso repetitivo para cada entrada del MDP, el tiempo requerido para generar el MDP es proporcional a su tamaño

4.5 Discusión

En este estudio, el espacio de trabajo se ha discretizado de modo que la constelación de puntos resultante se emplea como puntos de referencia para la generación de las trayectorias del robot. Este enfoque tiene una clara ventaja frente al de otros estudios basados en el control de la configuración articular del robot, la reducción de la complejidad del modelo del sistema. Esta reducción de la complejidad permite generar soluciones en tiempo real para la operación de pick-and-place.

Otro aspecto clave de este enfoque es la división del problema en dos capas. La capa superior aborda problemas de alto nivel, como determinar la asignación entre piezas y brazos, el orden de procesamiento de las piezas y las trayectorias de los robots. La capa inferior hace referencia al controlador interno del propio robot, es decir, a sus capacidades inherentes. Está separación de funciones permite generar trayectorias precisas y robustas, respaldadas por las garantías del fabricante.

El interfaz entre ambas capas es el lenguaje de programación que el fabricante proporciona, que en el caso de ABB es RAPID. En particular, el tipo de instrucciones empleado en este interfaz es común a cualquier robot en general. Por ejemplo, mover el brazo izquierdo a una determinada configuración articular. Dado que este tipo de instrucciones son estándar en todo tipo de robots, el sistema planteado es portable a otros modelos de robot, incluso de otros fabricantes.

La unión de ambas características, discretización del espacio de trabajo y descomposición en dos capas, permite generar trayectorias guiadas por los puntos de dicha discretización. La capa superior modela la trayectoria como una secuencia ordenada de puntos de referencia por donde ésta debe pasar y la capa inferior ejecuta la transición entre cada uno de estos puntos. Además, el enfoque actual trata la tarea de pick-and-place desde la perspectiva de un problema de toma de decisiones. En este caso, el modelo del sistema es mono-agente, es decir, existe una única entidad centralizada determinando las acciones de los brazos del robot y además estas acciones están forzosamente sincronizadas en el tiempo. Esta decisión de diseño afecta al tipo de movimientos soportados por el robot en el sistema. Dado que las trayectorias están representadas como una secuencia ordenada de puntos, la distancia recorrida por cada brazo en cada paso de esta secuencia debe ser similar. De lo contrario, un brazo debería realizar una pausa, esperando a que le otro brazo complete su movimiento antes de proceder al siguiente paso dentro de la secuencia. Para mantener una trayectoria suave y continua, se ha limitado los posibles movimientos de cada brazo a los puntos contiguos al que se encuentran. Este conjunto de decisiones limita el tipo de trayectorias generadas, pero permite simplificar el modelo del sistema y con ello reducir el cómputo necesario para obtener una solución.

Todos los experimentos realizados en este estudio se han llevado a cabo en RobotStudio. Este software está basado en el motor de físicas AGX Dynamics de la empresa Agoryx, y el propio fabricante (ABB) asegura que lo que se observa en el simulador se corresponde con lo que ocurre en la realidad. No obstante, pruebas adicionales en una configuración real permitirían confirmar los resultados obtenidos en el simulador.

El modelado del problema en este estudio representa, tanto a las piezas como a los robots, como puntos en el espacio tridimensional. Esta decisión permite mantener contenido el tamaño del modelo del sistema, que incluye información sobre la posición y estado de los robots y las piezas, pero ignora otros factores críticos, tales como el espacio real que ocupan

los brazos en el espacio de trabajo, la capacidad de cada robot para alcanzar una determinada posición, o la configuración articular apropiada en cada ubicación del espacio de trabajo. Esta información se introduce en el sistema de un modo alternativo. Se ha construido una base de datos con esta información, la cual es específica para el robot YuMi en las condiciones de los experimentos realizados. Esta base de datos sería necesario regenerarla si el robot YuMi es reemplazado por otro modelo de robot o si la resolución en la discretización del espacio de trabajo se modifica. La base de datos está generada mediante RobotStudio. Si se reemplaza el modelo de robot por el de otro fabricante, sería necesario informarse de las capacidades del simulador que éste ofrezca, y de la fidelidad de la información que aporta respecto al robot real.

Aún introduciendo una serie de técnicas para limitar el tamaño del problema, el sistema adolece de un problema de escalabilidad. No escala bien con el número de robots, ni con el número de piezas. En este estudio, el número de robots se ha fijado en dos, y el número de piezas en diez. En concreto, se han realizado experimentos en el rango entre 2 y 10 piezas. Se ha observado que los recursos de memoria y tiempo necesarios para resolver el problema con más de 4 piezas es enorme, por lo cual se ha definido un nuevo concepto: modos de navegación. Con el fin de reducir todavía más el coste computacional, se ha definido una serie de subconjuntos de acciones. El subconjunto de acciones más pequeño, que incluye solo movimientos ortogonales en el plano, es el de menor coste computacional, pero por otro lado es menos flexible en sus movimientos, lo cual afecta al tiempo requerido para completar la tarea. Por ejemplo, para realizar un movimiento diagonal entre dos puntos requiere realizar dos movimientos ortogonales. De igual manera, el modo de navegación más completo permite a los brazos navegar en las tres dimensiones y con ello posibilita navegar en diferentes niveles de altura y realizar cruces entre brazos. No obstante, se ha observado que la ganancia obtenida no compensa el cómputo adicional requerido. En concreto, el modo de navegación 2, que soporta movimientos ortogonales y diagonales en el plano, es el que obtiene mejores resultados desde un punto de vista computacional y de calidad de la solución.

En este estudio se han comparado tres métodos: MDP, grafo y PDDL. Se ha tratado de realizar una comparación justa desde la perspectiva de coste computacional. Para ello, todos ellos están implementados en un lenguaje de bajo nivel (C o C++). Se ha observado que BFS resulta ser el más rápido, con diferencia, en generar una solución. Por un lado, este tipo de algoritmos de búsqueda recorre cada nodo del grafo una única vez, a diferencia de Value Iteration que actualiza todos los estados en cada iteración. Por otro lado, el grafo es una

versión reducida del MDP que descarta todos los estados y transiciones no factibles en el sistema (por ejemplo, la acción conlleva exceder los límites del grid definido, o provoca una situación de colisión). Esta situación justifica la gran eficiencia del algoritmo de búsqueda. Por el contrario, el resultado de un algoritmo de búsqueda es un plan en bucle abierto, a diferencia de la solución generada para un MDP, que indica la acción apropiada en cada posible estado del sistema. Es decir, si el estado del sistema varía inesperadamente (por ejemplo, un operario retira una pieza), la solución basada en grafo debería regenerarse de nuevo en este punto.

Una de las características añadidas en este estudio respecto al capítulo anterior, donde se sientan las bases de la arquitectura de control empleada, es la posibilidad de transportar colaborativamente una pieza cuando ninguno de los brazos puede alcanzar tanto la posición inicial como la de destino de la pieza. El sistema permite definir hasta T puntos intermedios donde un brazo deposita la pieza y el otro completa el transporte. Estos puntos deben ser predefinidos de antemano, ya que permitir al algoritmo encontrar la mejor posición en función de la situación aumentaría la complejidad del sistema en exceso. La ventaja es que permite encontrar una solución independientemente de la situación inicial y final de las piezas y de las capacidades del robot.

Finalmente, hay que comentar que es común en problemas de toma de decisiones guiar al proceso de obtención de la solución en base a la definición de objetivos locales, ya sea en forma de recompensas con mayor valor en el MDP o de costes de menor valor en el grafo. En este estudio no se han definido objetivos locales para no contaminar la solución con decisiones predefinidas por parte del diseñador que resulten en una solución global subóptima. El objetivo único y global es completar el procesado de todas las piezas en el menor tiempo posible.

4.6 Conclusiones

En este estudio se ha definido una arquitectura de control que permite automatizar la generación de una solución para tareas de pick-and-place donde múltiples robots operan en un espacio de trabajo común. Este sistema se ha evaluado para un modelo de robot concreto, IRB 14000 (conocido como YuMi). No obstante, el sistema es portable a cualquier modelo de robot, incluso de otro fabricante. La solución generada es óptima desde respecto al número

de pasos requeridos para completar la tarea. Esta solución incluye encontrar la asignación adecuada de piezas, el orden apropiado de procesado de éstas, y las trayectorias recorridas por cada brazo del robot asegurando que no existe colisión entre ellos durante la operación.

En este estudio se han explorado tres métodos diferentes, basados en MDP, grafos y PDDL. La arquitectura de control planteada tiene un buen desempeño con un número bajo de robots y piezas, pero no escala bien debido a los recursos de memoria y cómputo necesarios. El método basado en PDDL ha sido capaz de encontrar una solución en todos los experimentos realizados, mientras que los otros dos métodos se han visto limitados principalmente por la memoria requerida en alguno de los experimentos. BFS es el método, con diferencia, que ha sido capaz de encontrar una solución en el menor tiempo. En concreto, es el único de los tres métodos capaz de generar una solución en tiempo real dado un proceso real.

Respecto a los diferentes modos de navegación introducidos en la arquitectura, se ha observado que la flexibilidad de movimientos que aporta la navegación tridimensional no compensa el coste computacional adicional que ésta implica. En los experimentos realizados, la capacidad de moverse cada brazo a diferente nivel de altura, y posibilitar cruces entre ellos, ha supuesto una mejora residual sobre la solución encontrada para el sistema cuando soporta solo movimientos en el plano. Por otro lado, la introducción de movimientos diagonales en el plano sí ha supuesto una mejora significativa en varios experimentos, lo cual compensa el cómputo adicional requerido respecto al sistema cuando soporta solo movimientos ortogonales en el plano.

5

OPERACIÓN DE PICK-AND-PLACE CON ROBOT YUMI Y PIEZAS EN MOVIMIENTO

5.1 Introducción

En este capítulo, la operación de pick-and-place introduce piezas en movimiento. La mesa de trabajo, donde residen un determinado número de piezas, ha sido reemplazada por un flujo continuo de piezas. Estas piezas se clasifican en varios tipos, y cada tipo de pieza debe ser almacenado en un contenedor específico. Esta configuración representa procesos comúnmente encontrados en la industria que cuentan con una cinta transportadora que mueve elementos que deben ser clasificados. Generalmente, en estos procesos suele operar un único robot, o en caso de existir más de uno, cada robot tiene su propio espacio de trabajo. Esta situación permite generar soluciones independientes para cada robot, sin necesidad de lidiar con potenciales problemas de colisión entre ellos. En este estudio, se trata el caso donde múltiples robots han de compartir el espacio de trabajo en esta operación.

En consonancia con los capítulos anteriores, este problema ha sido enfocado desde la perspectiva de un problema de toma de decisiones. En este caso, la complejidad aumenta al introducir piezas en movimiento, ya que implica determinar en qué momento y posición debe ser cada una recogida. En este estudio, el sistema se ha modelado bajo el marco de Proceso de Decisión de Markov. En base a las conclusiones obtenidas en el anterior estudio, modelar el sistema como un grafo resultaría en soluciones computacionalmente más eficientes. Por otro lado, la solución basada en MDP es más genérica, es válida incluso ante cambios inesperados del estado del sistema (por ejemplo, un operario retira una pieza), y permite extender en un futuro, el problema a entornos que incluyan incertidumbre. Dado que la arquitectura del sistema es modular, es relativamente sencillo introducir el modelado como grafo al igual que en el estudio anterior.

5.2 Métodos y Materiales

5.2.1 *Materiales*

La configuración completa de la estación de trabajo incluye un robot bimanual, una cinta transportadora, una cámara lineal para la detección de las piezas, un encoder para determinar el avance de la cinta transportadora, un flujo continuo de piezas y un conjunto de depósitos donde se almacenan las piezas.

El robot empleado en este estudio es el mismo que en el capítulo anterior, el IRB 14000, conocido como YuMi. Es un robot colaborativo, bimanual, y con 14 grados de libertad, enfocado en tareas de manipulación de piezas pequeñas, ya que el peso máximo que soporta por brazo es de medio kilo.

Las piezas se modelan como cubos de 40x40x40 mm, y el tipo de pieza viene representado por el color asociado a ella. El conjunto de depósitos sigue el mismo código de colores que las piezas. Existe un caso especial de pieza, coloreada de blanco, que indica que ese tipo de pieza no debe ser procesado por el robot, y terminará almacenándose en el depósito ubicado en el final de la cinta de transporte.

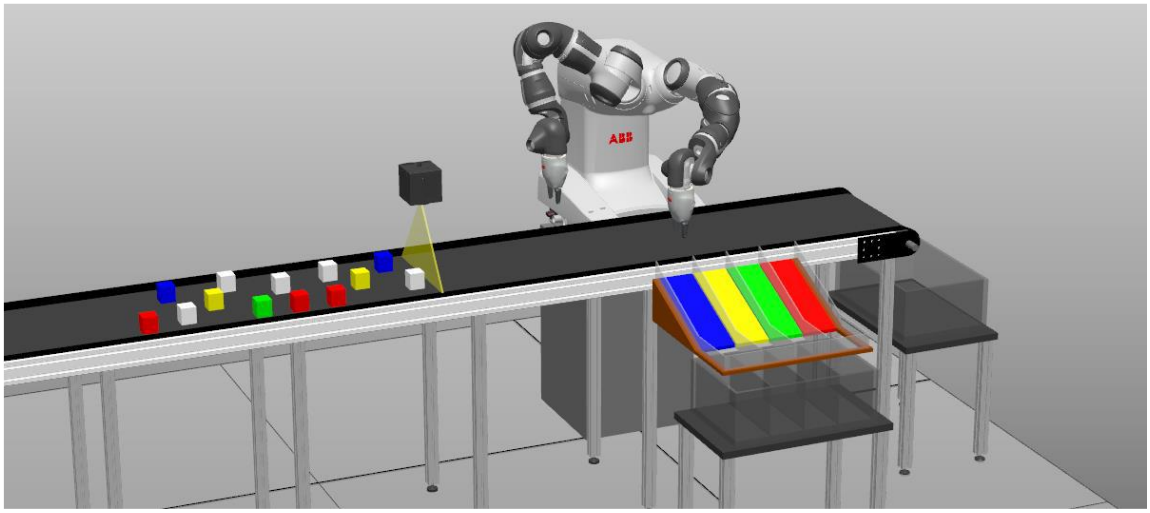


Figura 5.1 Configuración de operación de pick-and-place con robot YuMi y piezas en movimiento

La cámara representa el conjunto de cámara más procesado de imagen, cuyo objetivo es determinar, en que posición están las piezas y su tipo. El procesado de imagen no forma parte de este trabajo. En este estudio, el sistema recibe una notificación simulada de la entidad cámara cuando una pieza pasa por debajo de su posición.

5.2.2 *Diseño de la arquitectura del sistema*

5.2.2.1 Bloque de componentes del sistema

La tarea bajo estudio consiste en clasificar un flujo continuo de piezas transportadas por una cinta transportadora. Estas piezas deben recogerse en movimiento (en inglés, pick-on-the-fly) y depositarse en sus respectivos depósitos. La arquitectura de control empleada en este estudio está basada en la arquitectura del capítulo anterior, pero debe resolver dos limitaciones principales que dicha arquitectura presenta: no escalar bien con el número de piezas, y no soportar piezas en movimiento.

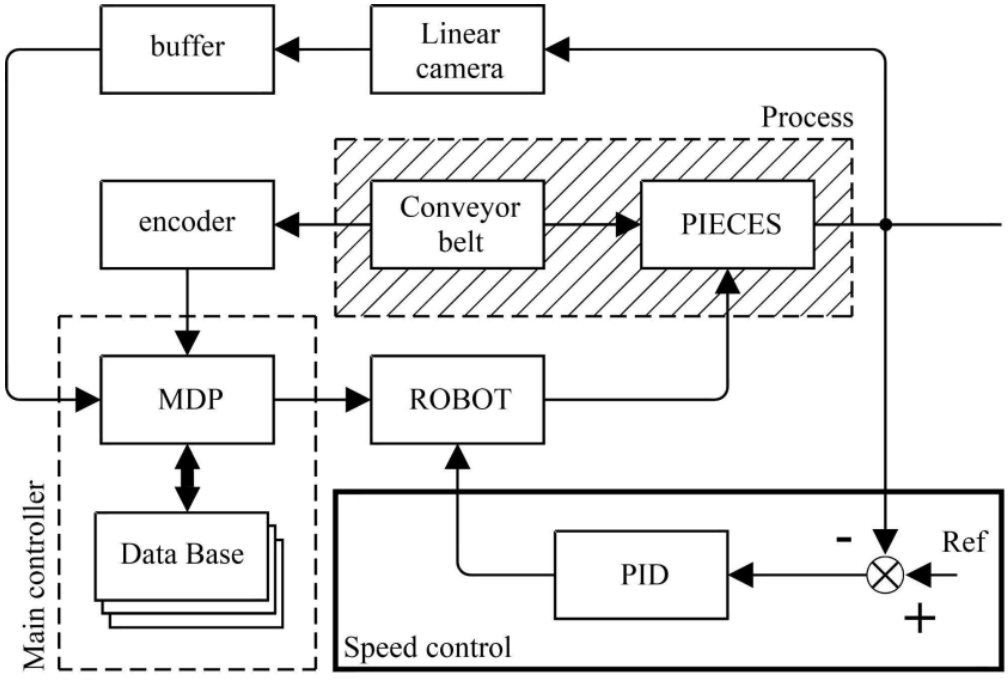


Figura 5.2 Bloque de componentes del sistema

La figura 5.2 es una representación de los bloques que componen la tarea de pick-and-place en este estudio. Existe una cinta transportadora que avanza a una velocidad determinada, moviendo un flujo continuo de piezas. Estas piezas deben clasificarse y almacenarse en sus respectivos depósitos. Para este estudio, se han definido cinco tipos de piezas. A cada tipo de pieza se le ha asignado un color. El color blanco tiene un significado especial. Las piezas de este color no deben ser procesadas por el robot, sino que deben progresar por la cinta hasta caer en un depósito ubicado al final de ésta. El resto de las piezas deben ser recogidas por el robot y depositadas en el depósito identificado con su mismo color. Dado que el robot es bimanual, éste puede procesar un máximo de dos piezas al mismo tiempo. El orden y momento en que cada pieza es procesada es determinado por el controlador principal del sistema.

Este controlador recibe una notificación de la cámara cada vez que ésta detecta una pieza. Junto con esta información, se entrega la información relativa a la pieza (posición y tipo). En un proceso real, se define que se emplearía una cámara lineal LSC (Linear Scan Camera). Esta cámara tomaría imágenes lineales sincronizadas con el avance de la cinta, y las acumularía en un buffer. Este buffer contendría imágenes estándar 2D que serían procesadas para determinar si una pieza ha sido detectada. El resultado de este procesado consistiría en las coordenadas XY de la posición de la pieza, y el momento en el que cruzó la zona de detección. Cada vez que una pieza fuera detectada, se enviaría una notificación al controlador principal junto con esta información sobre la pieza. El proceso de imagen no es objeto de este estudio, y la entidad cámara es emulada de modo que cada vez que una pieza pasa por debajo de la zona de detección, el controlador principal del sistema recibe dicha notificación con la información de la pieza. Dada esta descomposición del problema, es posible integrar este procesado a posteriori en un proceso real. Por otro lado, las piezas permanecen en constante movimiento, a la velocidad marcada por la cinta transportadora. Ésta integra un encoder que permite al controlador principal determinar el progreso de dicha cinta, y con ello inferir la posición actual de cada pieza.

El último bloque representado es el controlador de velocidad del robot. Dado que las piezas están en constante movimiento, la sincronización entre el robot y la pieza durante la operación de pick-and-place debe ser precisa. En el capítulo anterior, las piezas permanecen estáticas, luego independientemente de cuando realice la operación de recogida de pieza el robot, la pieza estará en la posición esperada.

En un proceso real, tanto la velocidad de la cinta transportadora, como la velocidad de los robots pueden sufrir ligeras fluctuaciones, que podrían afectar negativamente a la operación de pick-and-place. Por lo tanto, es necesario integrar un controlador de velocidad para reducir al mínimo estás potenciales situaciones.

5.2.2.2 Arquitectura de control

La arquitectura de control en este estudio está basada, al igual que en el estudio anterior, en la discretización del espacio de trabajo, y en la consideración de los robots y las piezas como puntos en el espacio. En este caso, las piezas son puntos que se mueven en el plano. Considerando que introducir información adicional en el modelo del sistema para conocer la posición actual de cada pieza va a incrementar significativamente el tamaño del espacio de estados, y dado que del estudio anterior se concluyó que el movimiento en el espacio tridimensional del efector final del robot no aporta una diferencia significativa en la solución, en este estudio, la discretización del espacio de trabajo consiste en una constelación de puntos 2D, al igual que en el caso del robot planar (rejilla 10x5x1). De igual manera, los modos de navegación 3 y 4 del anterior estudio no aplican, ya que el robot no se puede mover por diferentes niveles de altura. Esta decisión de diseño busca compensar en cierta manera la complejidad del problema al introducir las piezas en movimiento sin afectar significativamente a la calidad de la solución.

Esta discretización del estado mantiene el mismo objetivo, definir la trayectoria de los robots (o brazos del robot en el caso del robot YuMi) como una secuencia ordenada de puntos de la rejilla 2D definida. En cada paso de tiempo, cada robot realiza una transición a la siguiente posición dentro de su respectiva trayectoria. El movimiento realizado entre dos puntos de dicha rejilla está controlado por el controlador interno que integra el robot YuMi. El sistema planteado en este estudio genera acciones de alto nivel (por ejemplo, realizar transición a una nueva configuración articular) y el controlador interno del robot ejecuta dichas peticiones. Este enfoque permite conseguir una solución robusta y precisa.

5.2.2.3 Descomposición del problema en subproblemas solapados

La principal limitación de la arquitectura en el estudio anterior era la escalabilidad respecto al número de robots y el número de piezas. En este estudio, la cantidad de robots es fija, dos (robot bimanual) mientras que la cantidad de piezas es indefinida. Para afrontar esta limitación se ha empleado la técnica de dividir el problema en subproblemas y resolver independientemente cada uno de ellos.

No obstante, si el número de piezas se divide en grupos independientes (por ejemplo, grupos de dos piezas), la solución resultante no sería muy eficiente. Por lo general, en una situación donde cada brazo procesa una pieza, es muy probable que un brazo termine su tarea antes que el otro. En esta situación, el brazo que termina primero estaría desaprovechando un tiempo valioso hasta que el otro brazo complete su tarea, y en este punto ambos brazos repetirían el proceso de nuevo con un par nuevo de piezas.

Para aprovechar el tiempo disponible de cada brazo del robot al máximo, la división del problema se realiza en subproblemas solapados. La figura 5.3 es una representación esquematizada de cuatro situaciones extraídas de uno de los experimentos realizados, que permite valorar esta descomposición en subproblemas solapados. Por diseño, cada subproblema contiene hasta dos piezas, y las piezas seleccionadas para cada subproblema en un determinado momento son las piezas más avanzadas en la cinta transportadora, dado que estas piezas serían las primeras en caer en el depósito dedicado a las piezas blancas. Cada instantánea de la simulación en la figura 5.3 está formada por los siguientes elementos: una regla que indica la posición en milímetros dentro de la zona de trabajo del robot, una cámara LSC (C) situada en la posición -850mm, un conjunto de piezas desplazándose de izquierda a derecha, dos brazos robot (el izquierdo de blanco y el derecho de negro) y cuatro depósitos con el mismo código de colores que las piezas. El depósito situado al final de la cinta transportadora no está representado en este esquema.

La figura 5.3a representa el momento en el que la primera pieza es detectada por la cámara. En este punto, todavía no se ha generado ningún plan para los brazos y ambos permanecen en reposo. La cámara notifica la aparición de la primera pieza al controlador principal y éste genera un plan para el procesado futuro de dicha pieza cuando alcance el espacio de trabajo del robot. En la figura 5.3b, la segunda pieza (roja) es detectada, y el controlador principal es notificado. Éste descarta el plan anterior y genera un nuevo plan que incluye ambas piezas

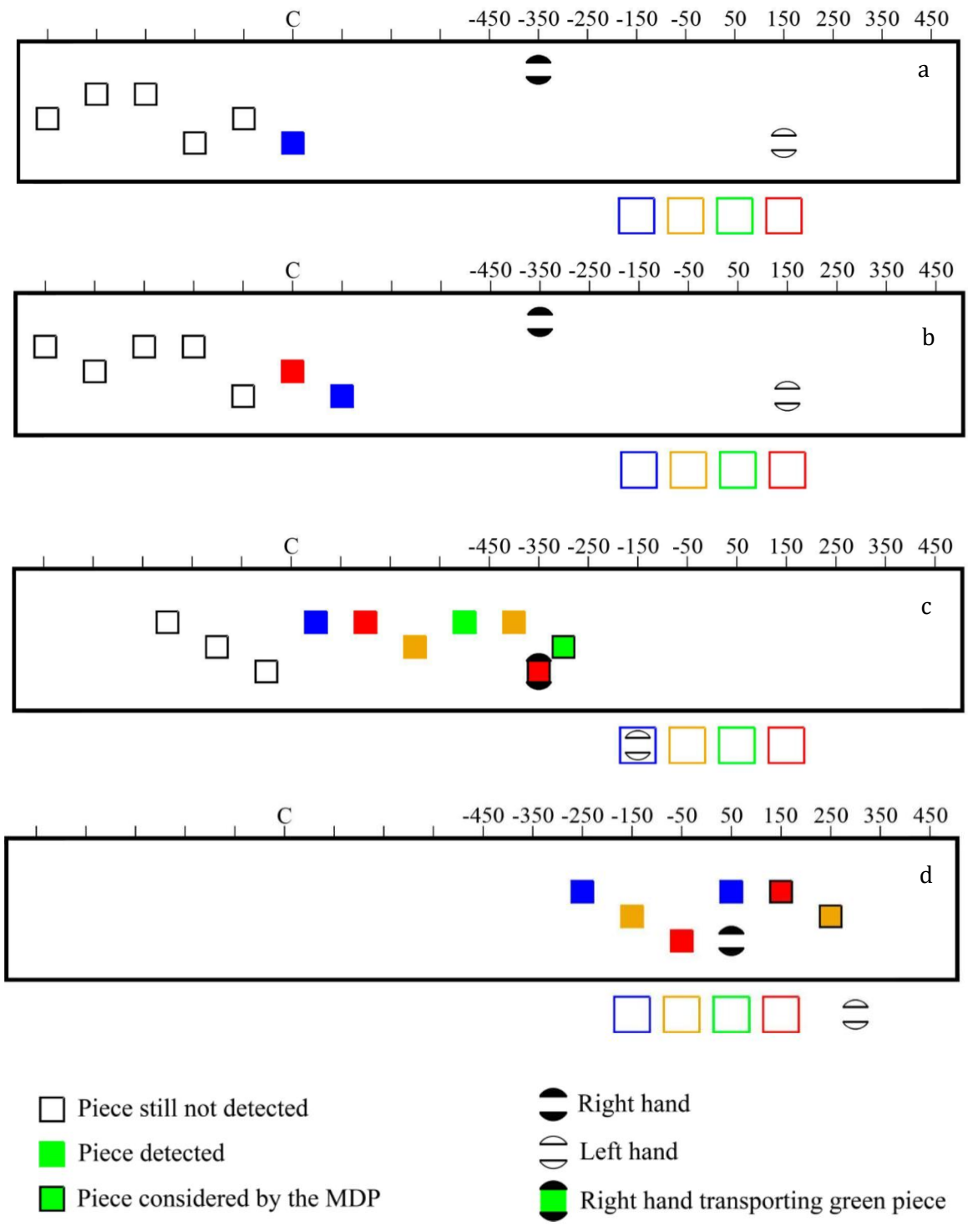


Figura 5.3 Representación de la división del problema en subproblemas solapados

(roja y azul). Ambos brazos todavía continúan en reposo, pues las piezas están lejos aún de alcanzar la zona de trabajo de los robots. La figura 5.3c representa la situación unos pasos de tiempo después, donde el brazo izquierdo acaba de depositar la pieza azul, y el brazo derecho todavía está transportando la pieza roja. En estos momentos, el brazo izquierdo queda liberado y está preparado para procesar una nueva pieza. Aun encontrándose el brazo derecho en proceso de completar su tarea, el plan actual se descarta, y se genera un nuevo modelo MDP que incluye también la siguiente pieza (verde) más adelantada en la cinta transportadora. Las condiciones iniciales en el nuevo MDP reflejan el mismo estado interno del sistema en el que se encontraba el anterior MDP antes de ser descartado (por ejemplo, el estado del brazo derecho indica que estaba transportando la pieza roja). Finalmente, la figura 5.3d representa un momento ya posterior donde el MDP incluye las dos piezas más avanzadas en la cinta en ese momento (roja y naranja). En ese caso, ambos brazos se encuentran libres y preparados para aproximarse a dichas piezas.

Este proceso de solapamiento de MDPs se repite continuamente cada vez que una pieza es depositada, siempre que existan piezas detectadas pendientes de ser procesadas. De lo contrario, el plan no es descartado hasta que ambos brazos completan su tarea.

```
// Inicialización
time_step ← 0
plan ← None
plan_max_piezas ← N  /* N = 2 para los experimentos realizados */
while true
    // Mantener un registro de las piezas detectadas
    if nueva pieza es detectada then
        lista_piezas_detectadas ← añadir nueva pieza
        foreach nodo s' alcanzable desde s do
    end
    if lista_piezas_detectadas no está vacía then
        if plan == None or piezas_en subproblema < plan_max_piezas then
            subproblema ← crear_nuevo_subproblema()
            plan ← generar_nuevo_plan(subproblema)
            time_step ← 0    /* comienza un nuevo plan */
        end
    end
    // Desactivar plan cuando esta completado
    if plan está completado then
        plan ← None
    // Ejecutar siguiente acción del plan
    if plan != None then
        execute plan step
    end
end
```

Figura 5.4 Pseudo-código para emular la gestión de subproblemas

La figura 5.4 representa esquemáticamente como se gestiona la creación de nuevos subproblemas en tiempo real. Inicialmente no existe ningún plan creado, y por lo tanto los brazos del robot permanecen en reposo. El controlador principal mantiene una lista con información sobre las piezas detectadas. Cuando la cámara notifica que ha detectado una nueva pieza, ésta es añadida en dicha lista. Cada subproblema soporta hasta N piezas (en este estudio, el valor se ha fijado en dos). En el momento en que hay piezas detectadas sin procesar, si no hay ningún subproblema en marcha, o si hay uno pero dispone de hueco para nuevas piezas (por ej., se acaba de depositar una pieza), se crea un nuevo subproblema, el cual contiene las piezas pendientes de procesar en el subproblema actual y otra u otras nuevas. En este caso en particular (N=2), pueden entrar 1 o 2 piezas nuevas en el nuevo subproblema. Una vez creado el nuevo subproblema, se genera un plan para resolverlo, y el nuevo plan sustituye al actual. Cuando un plan está en marcha, el controlador principal sirve una acción al robot en cada paso de tiempo. Si no existe ningún plan en marcha, no hay comunicación con el robot.

5.2.3 *Diseño del modelo del sistema*

Dado que el problema principal se descompone en subproblemas solapados, el modelo general del sistema consiste en una combinación de submodelos relativos a cada uno de los subproblemas. Cada uno de los subproblemas soporta hasta N piezas (N=2 en este estudio). En aras de simplificar la estructura del modelo general, todos los submodelos (MDP) incluyen N piezas, independientemente del número concreto de piezas que integra cada uno de los subproblemas. Si un subproblema contiene un número inferior a N de piezas, el resto de las piezas en el correspondiente submodelo se consideran ya procesadas, resultando irrelevante la posición inicial o final asignadas a éstas. Un caso común en el que se produce esta situación es con la detección de la primera pieza, con la cual se forma el primer subproblema.

Gracias a que todos los submodelos MDP presentan una misma estructura, es posible generar en el laboratorio un MDP común para todos los subproblemas. Posteriormente, en función de las condiciones iniciales de cada subproblema (posición de las piezas), el modelo MDP es actualizado.

5.2.3.1 *Espacio de estados*

Cada subproblema consiste en dos robots (específicamente el efector final de cada robot) desplazándose por una rejilla 2D resultante de la discretización del espacio de trabajo. Los robots deben recoger, transportar y depositar hasta un máximo de N piezas. Las piezas están en movimiento, reposando encima de la cinta transportadora que se mueve a una velocidad constante. Esto implica que la operación de pick-and-place debe realizarse en un paso de tiempo apropiado de modo que el robot y la pieza se encuentren en las mismas coordenadas XY. La elección del paso de tiempo apropiado para cada una de las piezas es una incógnita adicional al resto de las presentadas en el estudio anterior (asignación entre piezas y brazos, orden de procesado de las piezas, y generación de trayectorias). El conjunto de todas estas incógnitas determina la calidad de la solución. Toda esta información debe estar recogida en el modelado de cada subproblema. La tabla 5.1 muestra las variables que representan el estado interno de cada submodelo del sistema.

Tabla *5.1* Variables internas del cada submodelo del sistema formado por el robot YuMi y dos piezas

Variable	**Rango**	**Descripción**
Posición del robot i	[0- XY]	Coordenadas XY de la posición del efector final del robot i en la rejilla 3D
Estado del robot i	0-K	Indica la pieza que el robot i está transportando (1-K) o 0 si no tiene agarrada ninguna pieza, para todo i =1..N
Posición de la pieza j	[0- WY]	Coordenadas XY de la posición de la pieza j, para todo j = 1…K
Estado de la pieza j	0-2	Indica si la pieza todavía está en su posición de inicio (0), ha sido transportada hasta un punto intermedio (1) o ya no reposa sobre la mesa de trabajo (2), para todo j = 1…K
Estado de la pinza i	[0 – P]	Estado de la operación de pick-and-place para cada robot i

Dado que las piezas están en movimiento, la posición actual de cada una de las piezas que pertenecen a un determinado subproblema debe formar parte también del estado del modelo. Desde que la cámara notifica la presencia de una nueva pieza, esta pieza puede incorporarse potencialmente a un nuevo subproblema. Esto implica que la posición de la pieza debe poder monitorizarse desde su paso por la zona de detección de cámara hasta el final de la zona de trabajo del robot.

A diferencia de los robots que se pueden desplazar en una rejilla de tamaño XY, las piezas pueden encontrarse en cualquier posición de una rejilla de tamaño WY, donde W > X. Por un lado, la cámara está situada en una posición anterior a la zona de trabajo de los robots. Por otro lado, la velocidad de la cinta transportadora es menor que la velocidad de los brazos del robot. Por definición, existe una relación entre la velocidad de ambos:

$$\varsigma_{\rho} = \varsigma_{\pi} \cdot R \tag{13}$$

donde ς_{π} representa la velocidad de la cinta, ς_{ρ} representa la velocidad del robot, y R es la relación entre ambas velocidades. Dado que las piezas se mueven solidarias a la cinta transportadora, esto significa que cada pieza necesita R pasos de tiempo para recorrer la distancia equivalente entre dos puntos consecutivos de la rejilla 2D. Considerando esta información, el valor W de la rejilla 2D de las piezas depende la posición de la cámara y de la relación R entre velocidades de la cinta y el robot.

$$W = ((X_{camara} - X_{min}) \cdot R) + 1 \tag{14}$$

En el caso particular de este estudio, R=5, Xcamara = 13, y Xmin = 0, lo cual implica que el valor de W es 66, o lo que es lo mismo, una pieza necesitaría 66 pasos de tiempo para recorrer la distancia equivalente entre el punto de detección de la cámara y la ubicación del depósito ubicado al final de la cinta transportadora.

Introducir en el modelo una variable de estado por cada pieza para monitorizar su posición actual aumentaría el tamaño de estado significativamente. En el caso particular de este estudio:

` factor_de_incremento = (WY)K = 105.625

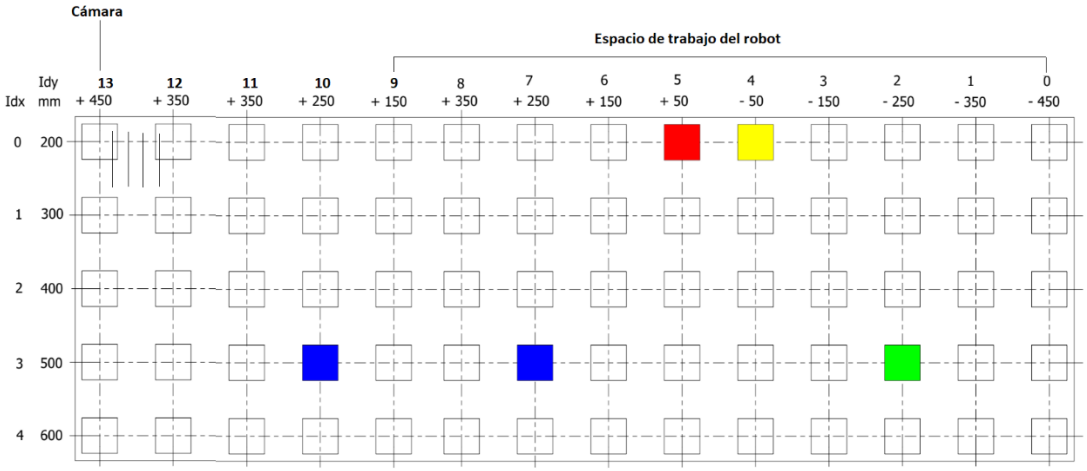


Figura 5.4 Relación entre zona de movimiento de las piezas y los robots

Por otro lado, dado que las piezas se mueven solidariamente con la cinta transportadora y que únicamente varía la coordenada X de su posición (dirección de avance de la cinta), es suficiente con introducir una variable que indique el paso de tiempo (en inglés, time step) actual en el sistema. Conociendo la posición inicial de las piezas, y el paso de tiempo actual es sencillo inferir la posición actual de las piezas:

$$posición_{actual} = posición_{inicial} + time_{step} \cdot velocidad_{cinta} \tag{15}$$

En estas condiciones, el incremento del espacio de estados es igual al rango del eje X de las piezas (x66), que es un valor notablemente inferior al obtenido mediante variables de posición independientes para cada pieza.

La información que permanece constante dentro de cada submodelo es la misma que en el estudio anterior. La diferencia estriba en los valores específicos empleados en este estudio.

Tabla *5.2* Constantes internas del modelo del sistema formado por el robot YuMi y dos piezas

Variable	Rango	Descripción
Posición inicial de la pieza j	X: [0-65], Y: [0-4]	Coordenadas XY de la posición inicial de la pieza j, para todo j = 1..K
Posición final de la pieza j	X: [0-65], Y: [0-4]	Coordenadas XY de la posición final de la pieza j, para todo j = 1..K
N	2	Número de robots
K	2	Número de piezas
T	0	Número de estados intermedios (>=0)
P	2	Número de pasos de tiempo necesarios para la operación de pick-and-place

5.2.3.1.1 Estado inicial y estados finales

Dado que la tarea de pick-and-place se descompone en un flujo continuo de subproblemas solapados, cada subproblema hereda el estado del subproblema anterior en el momento de la transición entre ambos. En otras palabras, el estado inicial de un subproblema coincide con el estado del sistema en el momento en el que el subproblema anterior es descartado. Durante esta transición, el paso de tiempo es siempre inicializado a cero, y la posición inicial de cada pieza se actualiza al valor de su posición actual. La posición inicial de cada pieza es una información que permanece constante durante el resto del subproblema.

Tabla *5.3* Estado inicial del sistema

Posición robot 1	Posición robot 2	Estado robot X	Estado pieza X	Paso de tiempo
Específica	Específica	0	1	0

El estado o estados finales de cada subproblema raramente es alcanzable mientras el flujo de piezas sea continuo. Esto es debido a que en el momento en que una pieza es depositada, un nuevo subproblema es generado y el actual es descartado antes de completarse. Existen dos situaciones en las que el estado final es alcanzado:

- Todas las piezas detectadas en la cinta transportadora están actualmente incorporadas en el subproblema. Dado que no hay nuevas piezas que procesar, el subproblema se completará hasta depositar todas las piezas.
- Si el subproblema soporta máximo 2 piezas (K=2), y el plan generado para este subproblema determina que ambas piezas deben depositarse simultáneamente.

`

Tabla *5.4* Estado final del sistema

Posición robot 1	Posición robot 2	Estado robot X	Estado pieza X	Paso de tiempo
Arbitraria	Arbitraria	0	0	Arbitrario

5.2.3.1.2 *Tamaño del espacio de estados*

El tamaño del espacio de estados hace referencia a la cantidad de diferentes estados posibles en el modelo del sistema. Este número resulta de la combinación de los posibles valores de cada una de las variables internas del sistema.

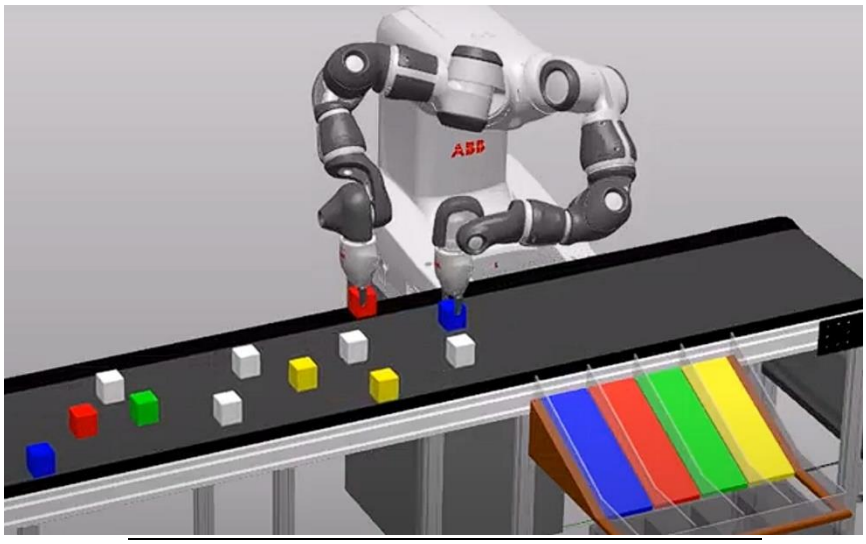
$$Num\ states\ =\ (X \cdot Y + K \cdot P)^N \cdot (K+1)^N \cdot 2^K \cdot N_{\max\_ps} \tag{16}$$

donde las constantes Z y T has sido eliminadas de la ecuación 16 respecto al estudio anterior, al discretizar el espacio de trabajo en un en un plano XY y no soportar posiciones intermedias para un transporte colaborativo de una pieza. Por otro lado, se ha incluido un nuevo término N_{max_ps}, que hace referencia al máximo número de tiempos de pasos soportado en el MDP. Anteriormente se ha deducido el número de pasos que necesitaría una pieza para recorrer la distancia entre el punto de detección de cámara y el depósito ubicado al final de la cinta transportadora. Si un subproblema no es capaz de procesar las piezas que contiene en un numero de pasos inferior a este valor, algunas piezas terminarán en el depósito incorrecto. Por esta razón. N_{max_ps} se definido con este mismo valor (66). Si el sistema no es capaz de encontrar una solución en estas condiciones, significa que algunas piezas del subproblema no pueden ser procesadas.

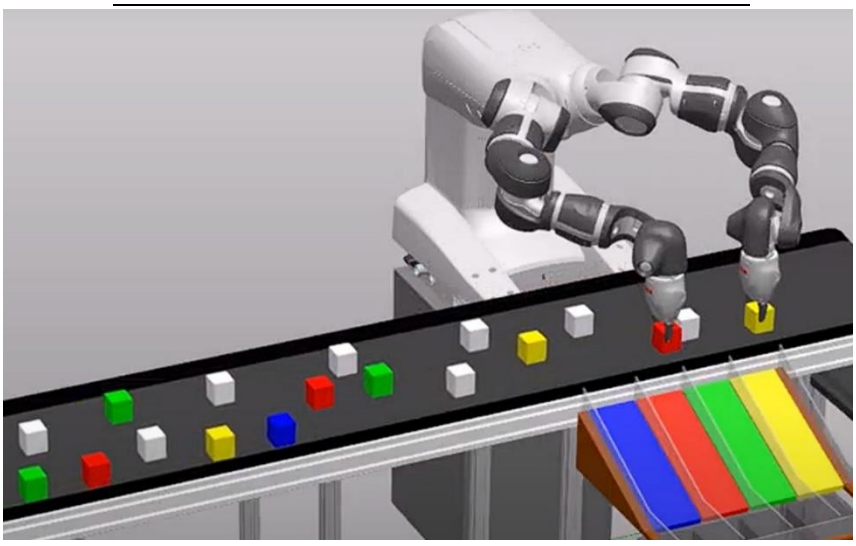
En los experimentos realizados en este estudio, el tamaño del espacio de estados es:

$$Num\ states\ =\ (10 \cdot 5 + 2 \cdot 2)^N \cdot (2+1)^2 \cdot 2^2 \cdot 66 =\ 104.796$$

5.2.3.1.3 Correspondencia visual del estado interno del 112sistema



Pos R1	Pos R2	Estado R1	Estado R 2
[8, 1]	[-, -]	1	2
Estado P1	**Estado P2**	**Estado Pinza 1**	**Estado Pinza 2**
1	0	2	0



Pos R1	Pos R2	Estado R1	Estado R 2
[-, -]	[-, -]	1	2
Estado P1	**Estado P2**	**Estado Pinza 1**	**Estado Pinza 2**
0	0	1	1

Figura 5.5 Correspondencia visual del estado interno del sistema

En la primera escena de la figura 5.5, el brazo izquierdo acaba de cerrar la pinza para agarrar la pieza azul, mientras que el brazo derecho está transportando ya la pieza roja. El subproblema actual incluye únicamente estas dos piezas, ya que, en los experimentos de este estudio, se ha definido N=2. El estado de cada robot indica la pieza que tiene agarrada, mientras que el estado de cada pieza indica si ésta ha sido ya recogida de la cinta transportadora. La pieza se considera recogida cuando el brazo robot asciende con ella. Por esta razón, en este instante el estado de la pieza azul es todavía 1. Por otro lado, el estado de la pinza 1 indica que la pieza acaba de ser agarrada, y el estado de la pinza 2 indica que no se encuentra en proceso de pick-and-place. Cabe resaltar, como se describe en la sección de arquitectura de diseño, que tanto la posición del brazo como el estado de la pinza se almacena en una variable de estado común. Esto implica que es un momento determinado, el valor de esta variable representa la posición del brazo o el estado de la pinza, pero no ambos a la vez. Dado que en este instante el brazo izquierdo se encuentra realizando la operación de recogida de la pieza, la información de su posición no es conocida explícitamente.

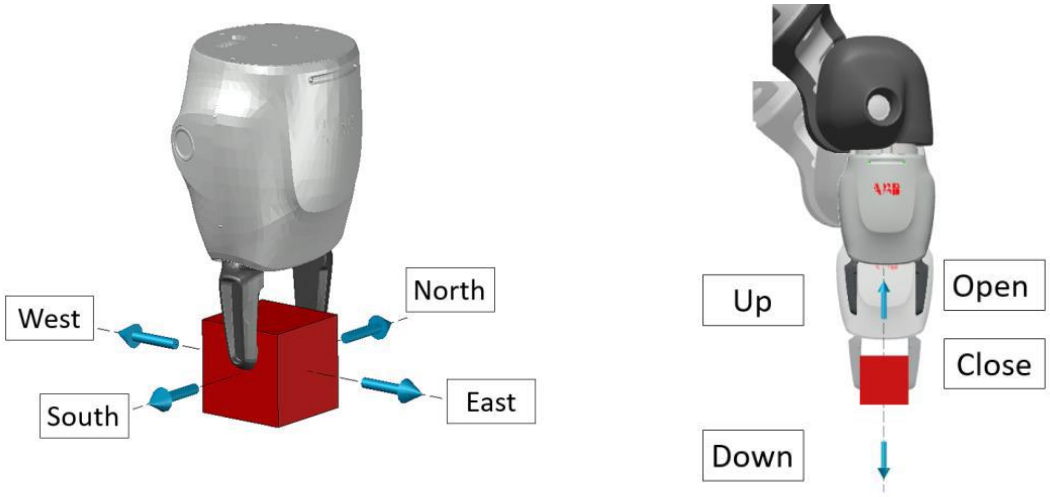
Unos instantes después, el brazo izquierdo deposita la pieza azul, y en estos momentos se genera un nuevo subproblema que incluye a la pieza pendiente de procesar (roja) y la siguiente pieza detectada en la cinta transportadora (amarilla). La segunda escena de la figura muestra el instante en el que se depositan las piezas de este nuevo subproblema. Casualmente en este caso, ambos brazos depositan la pieza simultáneamente. En esta escena en particular, la pinza no se ha abierto todavía, por lo cual el estado de la pinza es 1. El estado de las piezas es 0 desde que fueron recogidas, y el estado de cada brazo indica la pieza que está transportando y apunto de depositar. Como ambos se encuentran realizando la operación de place, la posición de los brazos no se conoce explícitamente. No obstante, esta información se puede inferir del resto de variables de estado.

Tras depositar ambas piezas, el sistema genera un nuevo subproblema que incluye dos piezas nuevas (amarilla y verde). Este proceso se repite continuamente mientras existan piezas en la cinta transportadora.

5.2.3.2 Espacio de acciones

En este estudio, el robot soporta movimientos ortogonales en el plano (modo de navegación 1 definido en el estudio anterior). Considerando que cada robot puede moverse en cualquiera

de las cuatro direcciones, realizar operaciones de pick y de place, o permanecer en reposo, que en este estudio operan dos robots (un robot bimanual), el número de acciones disponibles en el sistema es:



$$Num\ acciones\ =\ (N + S + E + O + reposo + pick + place)^N\ =\ 7^2 = 49$$

Figura 5.6 Acciones soportadas por cada brazo del robot

5.2.4 *Control de velocidad del robot*

En un proceso real, la velocidad de la cinta transportadora puede sufrir pequeñas variaciones respecto a la velocidad configurada. Estas variaciones pueden ser en forma de fluctuaciones sobre el punto de trabajo, o directamente un error constante sobre la velocidad media esperada de la cinta. De igual manera, este error de velocidad puede estar presente en los brazos del robot.

El sistema planteado en este estudio asume una determinada velocidad de la cinta y de los robots. Parte de la solución es determinar en qué momento debe realizar cada brazo la operación de pick sobre cada una de las piezas asignadas. Para que esta operación sea un éxito, las coordenadas XY de la pieza y el brazo deben coincidir en el momento del cierre de la pinza.

Si existe una desviación en la velocidad de la cinta transportadora o de los robots, puede desincronizar el sistema. Si esta desincronización es suficientemente grande, es posible que la pinza se cierre antes o después del paso de la pieza.

La simulación en RobotStudio introduce error tanto en la velocidad del robot como de la cinta. Esto permite analizar esta problemática durante los experimentos y verificar la efectividad de incluir un controlador de velocidad. La velocidad configurada para la cinta transportadora es 40 mm/s, mientras que la velocidad del robot es 200 mm/s.

Un enfoque basado en introducir un controlador de velocidad para la cinta independiente de un controlador de velocidad para el robot ayudaría a mantener la velocidad de cada uno alrededor de los valores configurados, con ello asegurando la sincronización entre piezas y robot. No obstante, mantener una cierta velocidad específica no es uno de los objetivos del sistema, sino simplemente mantener sincronizados las piezas y los robots. En este estudio, se emplea un enfoque diferente. Se introduce un único controlador que actúa sobre la velocidad del robot, pero tiene en cuenta el error de velocidad cometido tanto en las piezas como en el robot.

Dado que la velocidad esperada del robot es 200 mm/s, y que la distancia recorrida en cada paso de tiempo es de 100 mm, el tiempo estimado que se requiere en cada paso de tiempo se de 0.5s. Así mismo, durante este lapso (0.5s), dado que la cinta avanza a 40 mm/s, la expectativa es que cada pieza avance 20 mm. Esta información es clave para el algoritmo, ya que le permite inferir la posición de cada pieza en un determinado paso de tiempo.

El enfoque del controlador de velocidad está basado en determinar la velocidad adecuada del robot, de modo que, durante el tiempo invertido en realizar su movimiento, la cinta haya recorrido exactamente 20 mm. Si el controlador asegura que las piezas avanzan 20 mm cada paso de tiempo, las piezas se encontrarán en la posición apropiada en el momento de la operación de pick. Este control puede realizarse directamente sobre la velocidad del robot, o sobre el tiempo que le cuesta realizar el movimiento (paso de tiempo). El robot soporta ambos escenarios: indicar la velocidad a la que se debe mover, o indicar el tiempo en el que debe realizar el movimiento (de modo que el robot deduce la velocidad a la que debe moverse en función de la distancia a recorrer). En este estudio, el control se ha realizado sobre el paso de tiempo.

En condiciones ideales, cada paso de tiempo sería un intervalo de 0.5s, y las piezas habrían avanzado 20mm. En un proceso real, el intervalo de tiempo (paso de tiempo) indicado al robot variará en torno a 0.5s para compensar los errores de velocidad de la cinta y del propio robot. Por ejemplo, si la cinta se desplaza un poco más lento de lo esperado, el robot reducirá su velocidad para mantener el avance de 20mm por parte de las piezas en cada paso de tiempo.

El controlador de velocidad integrado en este estudio es un controlador PID. La señal de error empleada en este controlador es el error de posición de las piezas respecto a su posición ideal. En cada paso de tiempo, la posición de las piezas puede calcularse en base a su posición inicial. Por ejemplo, si una pieza comienza en la posición X=850mm, en el paso de tiempo t=17 se encontraría en la posición X = (850 - 20·17) = 510mm. La señal de salida del PID es directamente el valor del siguiente paso de tiempo. Siguiendo con el anterior ejemplo, si la pieza se encontrara en realidad en X = 504mm, es decir 6mm más adelantada de lo esperado, el controlador indicaría un paso de tiempo menor para que la pieza disponga de menos tiempo para avanzar en el siguiente paso de tiempo, y con ello termine lo más cerca posible de 490mm, que sería su posición ideal. La ecuación que describe el comportamiento del controlador es:

$$a_k = \tau + 1\psi \cdot [K_p \cdot e_k + K_i \cdot \Sigma(e_i \cdot t_i)_{k-ni=k-1} + K_d \cdot e_k - e_{k-m} \Sigma t_{k-imi=0}] \quad (5)$$

$$e_k = \chi_{\varpi k} - \chi_{\epsilon k} \quad (6)$$

donde $\chi^{\varpi}{}_k$ es la posición medida de la pieza, $\chi^{\epsilon}{}_k$ es la posición ideal de dicha pieza, r es el tiempo de referencia (0.5s) y ψ es la velocidad esperada de la cinta (40 mm/s).

En el mundo real, la posición de la pieza se inferiría de la lectura del encoder conectado a la cinta transportadora. En RobotStudio, es posible medir directamente la posición de la pieza en el simulador. Dado que todas las piezas se mueven solidarias a la cinta transportadora, y por lo tanto se desplazan a la misma velocidad, cualquiera de las piezas puede utilizarse para calcular el error de posición. Por otro lado, como las piezas desaparecen de la cinta transportadora en el momento en que son recogidas por un brazo, se ha creado una pieza virtual, no mostrada en las simulaciones, que se desplaza infinitamente a la velocidad de la cinta transportadora. Esta pieza virtual es la seleccionada para calcular el error del sistema.

5.3 Solución

En este estudio se ha experimentado solo con uno de los algoritmos empleados en el estudio anterior para generar una solución para el problema. El foco de este estudio reside no tanto en el algoritmo empleado sino en los cambios introducidos en la arquitectura y el modelo del sistema para soportar las nuevas características: piezas en movimiento, y flujo continuo de piezas (escalabilidad con el número de piezas). En la figura 5.7 se puede observar el flujo del proceso definido en este estudio para generar una solución.

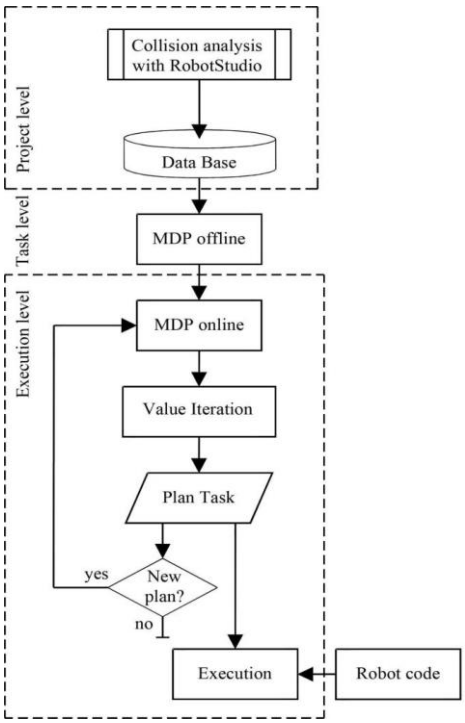


Figura 5.7 Flujo del proceso de generación y aplicación de la solución

Las fases a nivel de proyecto y a nivel de tareas son exactamente las mismas que en el estudio anterior. El objetivo es generar la base de datos con toda la información relativa al robot. Esta información incluye la configuración articular de cada robot para cada posición en el espacio de trabajo, las situaciones de colisión entre robots y las posiciones no alcanzables por cada uno de ellos. El otro gran objetivo es generar el modelo del sistema (MDP).

La fase de ejecución introduce diferencias para adaptar el proceso a un entorno con un flujo continuo de piezas en movimiento. Como se ha descrito en la sección 5.2.2.3, el problema general se divide en un conjunto de subproblemas solapados. El controlador principal genera un nuevo subproblema cada vez que existe la posibilidad de introducir una nueva pieza en éste (por ejemplo, cuando una pieza se ha depositado, crea un hueco libre en el modelo para operar una nueva pieza). Cada vez que se genera un nuevo subproblema, el controlador ejecuta los bloques MDP online y Value Iteration. El primero se encarga de actualizar el modelo del sistema una vez se conoce la posición de las piezas que integran el subproblema. El segundo genera una solución a dicho modelo. Esta solución se pasa al planificador de tareas (Task plan) que es el encargado de aplicarla en el robot. En particular, en cada paso de tiempo indica la acción a realizar por cada robot hasta completar la tarea. Si durante este tiempo se genera un nuevo subproblema, la solución a éste se pasa de nuevo al planificador de tareas, y éste aborta el plan actual y pone en marcha el nuevo.

El algoritmo empleado para obtener una solución del MDP generado es Value Iteration. Este algoritmo es ajeno a las dimensiones del modelo del sistema. Dado que las características del modelo del sistema en este estudio con iguales que las del anterior (discreto, finito, determinista), el algoritmo implementado en el estudio anterior es portable para el estudio actual.

5.4 Resultados

5.4.1 *Validación de los datos*

Los experimentos llevados a cabo en este estudio se han realizado en RobotStudio®. El simulador que integra este software está basado en simulador de físicas AGX Dynamics de la empresa Algoryx. El proyecto de RobotStudio integra un robot YuMi, una cinta transportadora, múltiples piezas cúbicas, un conjunto de depósitos y una cámara. La función de la cámara es únicamente indicar el punto de detección de las piezas. Cada tipo de pieza debe ser clasificado en un determinado depósito. Se ha definido un código de colores aplicado a las piezas y depósitos para identificar el tipo.

Como se ha descrito previamente en este capítulo, el problema de pick-and-place se descompone, en tiempo real, en un conjunto de subproblemas solapados, cada uno de ellos integrando un máximo de N piezas. La figura 5.8 muestra una de estas situaciones de solapamiento en los experimentos realizados. Esta figura muestra cuatro instantáneas de la simulación junto a la solución (o plan) generada para cada uno de los subproblemas involucrados.

La escena de la figura 5.8a se sitúa en t=40, es decir, 40 pasos de tiempo desde el inicio de la operación de pick-and-place. En ese momento, el brazo izquierdo acaba de depositar la pieza azul y el brazo derecho se encuentra transportando la pieza amarilla.

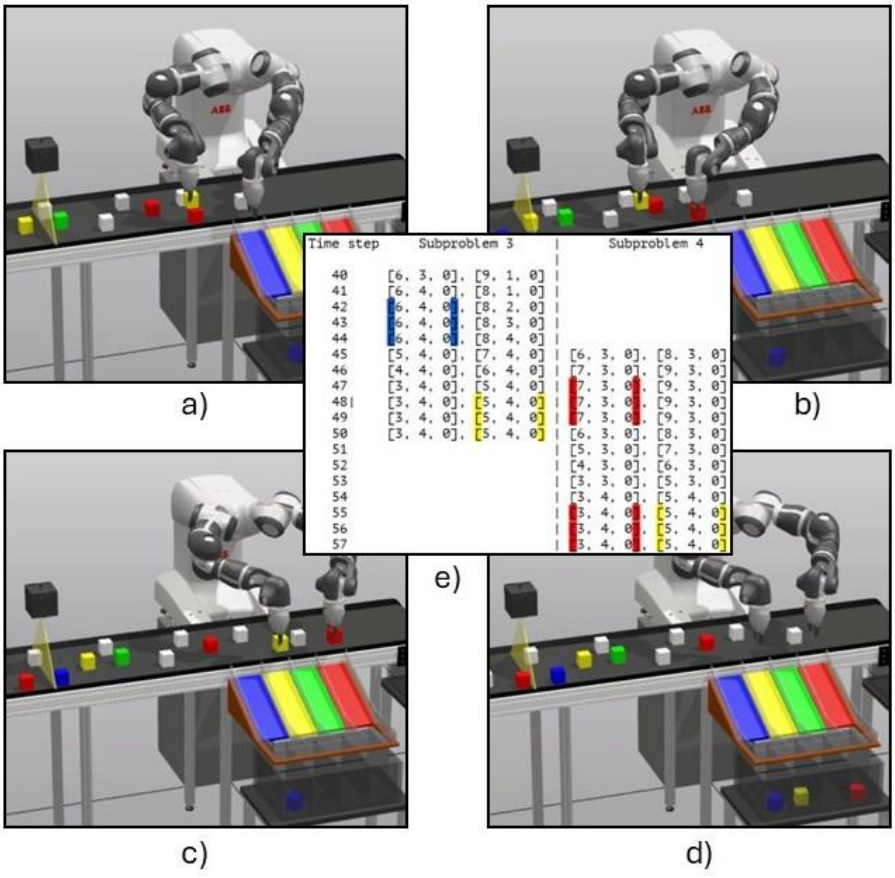
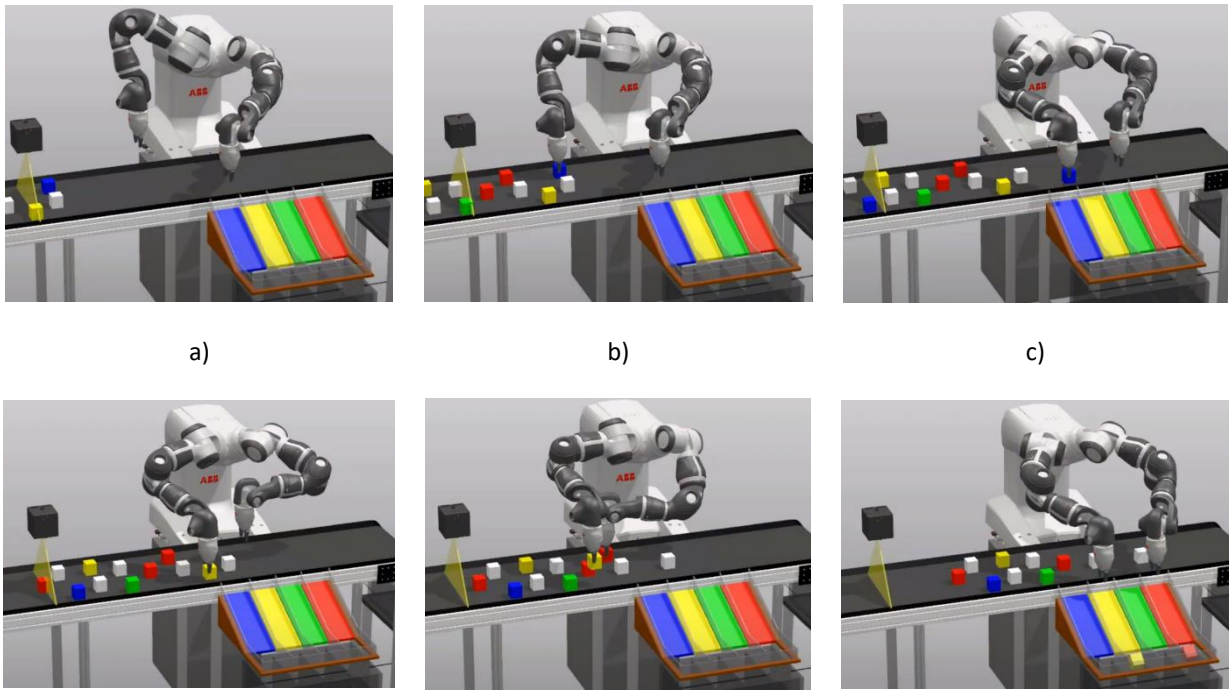


Figura 5.8 Representación de solapamiento entre subproblemas

Dado que existe un hueco para una nueva pieza en el subproblema y existen más piezas detectadas esperando ser procesadas, el subproblema actual se descarta y se crea uno nuevo que incluye la siguiente pieza (roja). De acuerdo con el plan que se acaba de descartar, referenciado como subproblema 3 en la figura 5.8e, la siguiente operación de pick-and-place consistiría en depositar la pieza amarilla por parte del brazo izquierdo. No obstante, el plan generado para el nuevo subproblema (mostrado como subproblema 4 en la figura) indica que la siguiente operación de pick-and-place es recoger la pieza roja. Esta operación se puede observar en la figura 5.8b. La razón detrás de este cambio de planes reside en que la solución óptima para este subproblema implica que el brazo derecho debe apartarse y esperar a que el brazo izquierdo recoja la pieza amarilla. Unos instantes de tiempo después (t=55), ambos brazos depositan simultáneamente sus respectivas piezas (figura 5.8c). Una vez depositadas, de nuevo aparecen nuevos huecos para introducir nuevas piezas en el subproblema. En estos momentos (figura 5.8d), se crea un nuevo subproblema que incluye las siguientes piezas en la cinta transportadora (roja y verde).



a) b) c)

d) e) f)

Figura 5.9 Solución óptima en cada subproblema

Por otro lado, la figura 5.9 representa una secuencia parcial de uno de los experimentos, donde se puede observar como la visibilidad de cada subproblema es limitada a N piezas (N=2 en el caso de este estudio), y que, aun generando una solución óptima en cada uno de estos subproblemas, no implica que la solución global sea necesariamente óptima. La primera escena (a) representa el momento en que la pieza amarilla es detectada. En este momento se crea un subproblema que incluye las piezas azul y amarilla. Observando la ubicación de los depósitos, intuitivamente, el brazo izquierdo debería recoger la pieza amarilla y el derecho la azul. No obstante, el plan generado para este subproblema indica que la situación óptima es que el brazo derecho procese ambas piezas, una después de la otra. La razón reside que en el depósito azul está muy próximo a la zona de entrada de la pieza azul, que es la pieza más avanzada en la cinta, y el tiempo invertido en depositar dicha pieza y volver a por la pieza amarilla es menor que esperar hasta que la pieza amarilla avance lo suficiente para que el brazo izquierdo pueda recogerla sin colisionar con el brazo derecho. La escena (c) representa el momento donde la pieza azul es depositada, y un nuevo subproblema es creado incluyendo la pieza amarilla pendiente de recoger y una nueva pieza (roja). De nuevo, intuitivamente, el brazo izquierdo debería recoger la pieza roja y el derecho la amarilla, y en esta ocasión, el plan coincide con esta decisión. La posición relativa de las piezas es tal que permite al brazo izquierdo recoger la pieza roja situándose por detrás del brazo derecho sin colisionar, como se observa en la escena (e). Finalmente, la escena (f) representa el momento donde ambas piezas son depositadas, en este caso, simultáneamente.

5.4.2 *Efectividad del controlador de velocidad*

El plan calculado para cada subproblema asume una cierta velocidad conocida de la cinta transportadora y del robot. En concreto, asume que el robot se desplaza cinco veces más rápido. Es decir, cada vez que el robot realiza un movimiento, cuya distancia es de 100mm, la cinta avanza 20mm. En un proceso real, estas velocidades pueden sufrir variaciones, desvirtuando la expectativa de avance de 20mm cada paso de tiempo. La simulación en RobotStudio® incluye un cierto ruido tanto en la velocidad de la cinta como la del robot. La figura 5.10 representa la situación donde el controlador de velocidad no está integrado en el sistema. La gráfica (a) representa la velocidad de la cinta transportadora que fluctúa ligeramente entorno a 40mm/s. La gráfica (b) representa el valor de paso de tiempo que

además de presentar fluctuaciones exhibe una desviación constante apreciable de una décima de segundo. Es decir, al robot se le indica que el movimiento debe realizarlo en 0.5s y el tiempo invertido acaba siendo 0.6s. En este caso, el error crítico es la desviación media de la velocidad del robot, ya que cada paso de tiempo dura un 20% más y por lo tanto la pieza avanza proporcionalmente un 20% más (24mm). El efecto de esto error se aprecia claramente en la gráfica (c), donde se puede observar como el error de posición de las piezas aumenta constantemente. Específicamente, las piezas cada vez están más adelantadas. Dado que las dimensiones de la pieza son 40x40x40mm y ésta es agarrada por el centro, el error de posición máximo permitido es 20mm. Si el error es superior a ese valor, significa que la pinza no agarrará la pieza cuando se cierre.

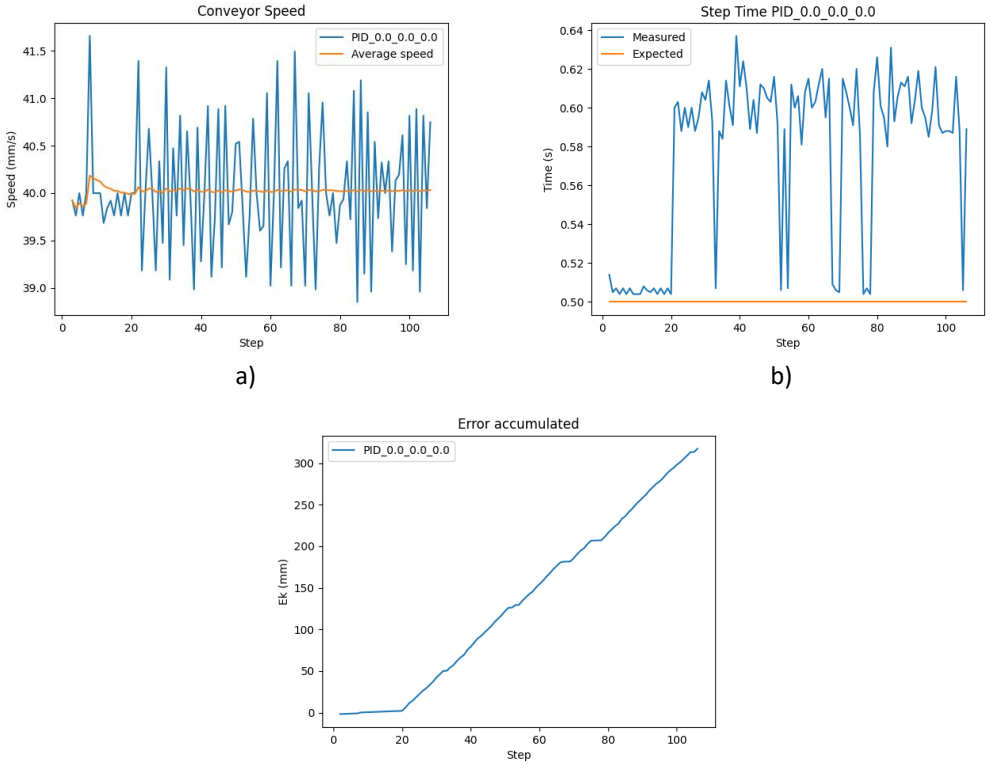
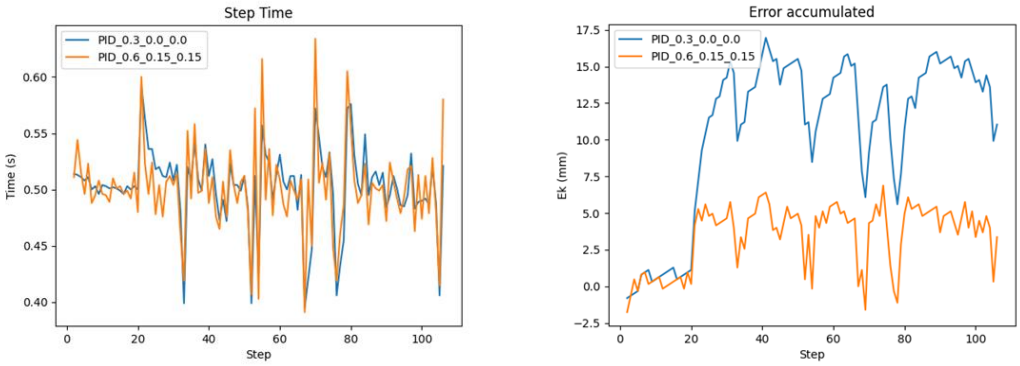


Figura 5.10 Solución sin controlador de velocidad: a) velocidad de la cinta, b) valor del paso de tiempo, c) error de posición de la pieza

En cambio, la figura 5.11 muestra el resultado de introducir el controlador de velocidad en el sistema. La velocidad de la cinta transportadora es independiente del controlador. No se muestra en esta figura ya que es exactamente la misma que la mostrada en la figura 5.10. La gráfica (a) muestra el valor calculado para el paso de tiempo con diferentes versiones del controlador PID. El control del valor del paso de tiempo tiene como propósito permitir avanzar más o menos a la cinta transportadora en cada paso de tiempo, compensando el error de avance de la pieza, para mantenerlo lo más próximo posible al valor de 20mm por paso de tiempo. La gráfica (b) muestra el error de posición de la pieza en función del tiempo (número de pasos de tiempo). Claramente el controlador representado por la línea naranja consigue unos mejores resultados. En concreto, es capaz de contener el error de posición en un rango aproximadamente de 7mm. Dado que hay un margen significativo hasta el error máximo permitido (20mm), esto significa que la operación de pick funcionará a lo largo de todo el proceso.



a) b)

Figura 5.11 Solución con controlador de velocidad: a) valor del paso de tiempo, b) error de posición de la pieza

5.4.3 *Coste computacional*

Dado que la operación de pick-and-place se divide en un conjunto de problemas consecutivos y solapados, el coste computacional hace referencia a cada uno de estos subproblemas. La tabla 5.5 muestra los valores medios obtenidos durante todos los experimentos.

Los experimentos se han realizado en un portátil 5560 Lattitude que cuenta con un i7-12800H y 16GB de RAM. La generación del MDP es relativamente rápida, a diferencia del estudio anterior, dado que se ha limitado cada subproblema a un máximo de dos piezas, y se soporta exclusivamente el modo de navegación 1 definido en el anterior estudio (movimientos ortogonales en el plano). Este proceso se realiza una única vez, y el MDP generado es válido para todos los experimentos y todos los subproblemas dentro de cada experimento. En cada experimento en particular, el primer paso es actualizar el MDP una vez se conoce la posición inicial de las piezas. Esta operación se repite para cada subproblema dentro de cada experimento y su coste es cercano a un minuto. Este valor es relativamente alto (~50s) dado que la implementación de esta parte está realizada en Python. Una implementación en C reduciría notablemente este valor. El segundo paso es generar una solución. En este estudio se ha optado por usar el algoritmo de Value Iteration. El tiempo requerido por este algoritmo es relativamente alto (4s) para una operación en tiempo real. No obstante, basado en los resultados del anterior estudio, si optáramos por otro método, tal como BFS, este tiempo sería inferior a 0.2s. Finalmente, la tabla muestra los recursos de memoria RAM y de disco necesarios, que están bastante contenidos para un ordenador actual.

Tabla *5.5* Información del coste computacional

Description	Value
Offline time	27m 12s
Online time (MDP update)	53s
Online time (MDP solve)	4.6s
Number of MDP states	~1.3M
Number of brute states	~10M
RAM required	2GB
DISK required	394MB

5.5 Discusión

La arquitectura de control definida en este estudio en una evolución de la descrita en el estudio anterior. El objetivo es afrontar dos limitaciones de ésta que están presentes en la tarea analizada en este estudio, la escalabilidad con el número de piezas y las piezas en

movimiento. La tarea analizada en este estudio es un proceso de pick-and-place que integra una cinta transportadora sobre la cual reposan piezas que deben ser clasificadas y depositadas en su ubicación correspondiente. El flujo de piezas se considera continuo, lo cual implica una acumulación considerable de piezas en la zona de trabajo. Igualmente, dado que las piezas están en movimiento, determinar su posición de recogida es un desafío añadido.

El problema de escalabilidad se ha afrontado desde la perspectiva de divide y vencerás. El problema general se ha divido en subproblemas menores que se resuelven separadamente. La solución obtenida para cada uno de estos problemas es óptima en referencia al número de pasos necesarios para completarla. No obstante, la solución global resultante no es necesariamente óptima ya que los subproblemas no son realmente independientes. Considerar los problemas como independientes resultaría en una solución ineficiente, ya que en el momento en el que restan menos piezas que robots, alguno o algunos de los robots deben esperar a que el resto procesen su pieza. En el caso de este estudio, el número de robots es dos (específicamente un robot con dos brazos) y el número de piezas soportado por cada subproblema es dos también. Si un brazo deposita su pieza antes que el otro, lo más eficiente es descartar el resto del subproblema y crear uno nuevo que incluya la pieza pendiente y una nueva. De este modo, el brazo que ha quedado liberado tiene la oportunidad de procesar una nueva pieza. En resumen, el problema se divide en un conjunto de subproblemas solapados.

La otra limitación de la arquitectura del estudio anterior es la ausencia de soporte para piezas en movimiento. En condiciones estáticas, la posición donde se realiza la operación de pick (posición inicial de la pieza) está definida al comienzo de cada operación. En el caso de piezas en movimiento, el sistema tiene que determinar el momento óptimo en el que esta operación ha de realizarse para cada pieza, y eso determina la posición en la que el robot y la pieza deben encontrarse en dicho momento. La desventaja, por naturaleza, es que el tamaño del problema (espacio de estados) aumenta con esta nueva característica, al necesitar tener presente la posición de cada pieza en cada momento. Para compensar este aumento de la complejidad del problema se han limitado algunas características variables en el anterior estudio, tales como el número de piezas por subproblema (2), y los modos de navegación soportados (solo movimientos ortogonales en el plano).

Las propiedades del problema son las mismas que en el estudio anterior: determinista, discreto y finito, por lo tanto, se podría aplicar cualquiera de los tres métodos explorados en el estudio anterior. En este caso, se ha empleado solo el método basado en MDP, ya que no es

el objeto de este estudio comparar de nuevo los diferentes tipos de métodos. Los experimentos indican que el coste computacional de la operación realizada durante el proceso es entorno a los cinco segundos. Este tiempo no es exactamente tiempo real en un proceso con un flujo continuo de piezas. No obstante, si el método actual es reemplazado por el método basado en grafos descrito en el anterior estudio, el tiempo invertido en esta operación sería inferior a 0.2s, lo cual permite una aplicación en tiempo real del sistema.

Al igual que en el estudio anterior, la arquitectura está basada en dos capas. La capa superior se encarga de resolver las tareas de alto nivel, como determinar el momento adecuado para recoger cada pieza, o las trayectorias de los robots. La capa inferior, de la que es responsable el controlador interno del robot, se encarga de realizar las transiciones físicas del robot entre dos puntos. Esto permite generar una solución precisa, y robusta. No obstante, a diferencia del estudio anterior, las piezas están en constante movimiento, por lo cual, errores en la estimación de la posición de las piezas o del robot pueden ser fatales para el éxito de la operación de pick. Es común que la velocidad de la cinta transportadora, o la de los robots no sea exactamente la velocidad configurada, con lo cual esta diferencia de velocidad sobre la esperada puede provocar situaciones de desincronización entre piezas y robot, llevando en el límite a un escenario en el que la operación de pick se realiza cuando la pieza no está en la posición adecuada. Para resolver estos problemas de desincronización se ha introducido un controlador de velocidad en el sistema. Se ha optado por utilizar un único controlador que influye en la velocidad del robot en lugar de introducir un controlador independiente para la cinta y otro para el robot. La razón estriba en que ajustar la velocidad del robot y de la cinta a sus respectivas velocidades configuradas no es el objetivo principal de la tarea, sino simplemente asegurar que piezas y robot permanecen sincronizados en cada momento. Si la cinta se mueve un poco más lento, pero el robot hace lo propio, ambas partes seguirán sincronizadas. Lo mismo ocurre si ambos se mueven más rápido.

La simulación en RobotStudio incluye ruido tanto en la velocidad del robot como la de la cinta. En los experimentos se puede observar como el controlador de velocidad del robot ayuda a contener el error de posición de las piezas en un rango (0-7mm) inferior al margen de error permitido en el sistema (20mm). También se observa como un controlador PID tiene un factor de corrección claramente mejor que un controlador P en este proceso, ya que el efecto de la acción derivativa permite tener en cuenta la tendencia o velocidad de cambio del error, con ello evitando aplicar acciones correctivas excesivas que intentan corregir el problema instantáneamente, provocando situaciones de oscilación (donde se aplica una acción excesiva, y a continuación una insuficiente para compensar el exceso, y así continuamente).

5.6 Conclusiones

En este estudio se ha extendido la arquitectura de control definida en el capítulo anterior para soportar operaciones de pick-and-place donde las piezas se encuentran en movimiento. La solución generada es capaz de optimizar la trayectoria de los robots, el orden de manipulación de las piezas, la asignación entre piezas y robots y el lugar de recogida de las piezas, teniendo en cuenta que éstas se desplazan solidarias a la cinta transportadora. La optimización en la solución hace referencia a minimizar el número de pasos en la que la operación de pick-and-place es completada, lo que es equivalente al tiempo invertido en ésta.

Gracias a la arquitectura basada en dos capas, el sistema es portable a otros modelos de robots, del mismo o diferente fabricante. No obstante, un cambio de modelo de robot requiere generar de nuevo la base de datos que contiene información específica del robot dentro del entorno del problema (qué posiciones es capaz de alcanzar cada brazo, donde se producen situaciones de colisión, etc ..). La base de datos en este estudio ha sido generada con ayuda de RobotStudio, que es un software proporcionado ABB para sus modelos de robot. Si el robot perteneciera a otro fabricante, sería necesario consultar las capacidades del simulador que éste ofrezca.

Por otro lado, la solución generada proporciona un plan completo para cada uno de los subproblemas en los que se descompone el problema principal. Estos planes asumen una cierta velocidad de la cinta y robot. Si estas velocidades sufren variaciones respecto a los valores esperados por el algoritmo, la operación de recogida de la pieza fallaría al no encontrarse la pieza en el lugar apropiado a la hora de cerrar la pinza. En los experimentos realizados se demuestra la efectividad del controlador PID integrado en el sistema para regular la velocidad de los brazos del robot,

Finalmente, a diferencia del estudio anterior, en éste se ha optado por emplear únicamente el método basado en MDP. El coste computacional está contenido (~5s), aunque no cumple las expectativas de un procesado en tiempo real. No obstante, si este método se reemplaza por el método basado en grafos, analizado en el anterior estudio, el tiempo invertido en cada subproblema resulta inferior a 0.2, lo cual cumple a la perfección con los requisitos de un proceso en tiempo real.

6

CONCLUSIONES

6.1 Conclusión

Esta Tesis presenta un sistema capaz de automatizar el proceso de control de una tarea de pick-and-place en la cual operan múltiples robots en un espacio de trabajo compartido. El objetivo de este sistema es optimizar el tiempo necesario para completar la tarea de pick-and-place, con ello aumentando la productividad del proceso. Este objetivo requiere resolver otras tareas en segundo plano, como la asignación de piezas a robots, el orden de procesamiento apropiado de las piezas, y las trayectorias de los robots para aproximar y transportar las piezas evitando situaciones de colisión entre ellos. Todas estas tareas se resuelven en tiempo real, dado que su solución óptima depende de las conficiones iniciales de la operación de pick-and-place en cada iteración (posición inicial de las piezas y de los brazos).

Los experimentos realizados integran un robot bimanual de 14 grados de libertad, conocido como YuMi, Es un robot colaborativo comercial del fabricante ABB. No obstante, el sistema es portable a otros modelos de robots, del mismo o diferente fabricante. El sistema cuenta con una base de datos que contiene información relevante del robot en el sistema, la cual ha de generarse una única vez al inicio del proyecto. Esta separación de la parte específica del robot permite portar la solución a otros modelos de una manera relativamente sencilla.

Todos los experimentos han sido realizados en un simulador. ABB ofrece un software llamado RobotStudio, basado en el motor de físicas AGX de Algoryx, que permite realizar simulaciones fidedignas. En concreto, el propio fabricante asegura que lo que se observa en el simulador se corresponde con lo que ocurre en el proceso real (incluyendo robot y otros elementos incluidos en el proyecto). La calidad de este simulador confiere un alto grado de confianza en los resultados obtenidos

El problema bajo estudio presenta diversos desafíos. El objetivo principal es minimizar el tiempo invertido en la tarea de pick-and-place, en la cual operan múltiples robots en el mismo espacio de trabajo. Esto requiere de un trabajo de carácter colaborativo entre los robots, tanto para optimizar las trayectorias como para evitar colisiones entre ellos. Por otro lado, las

condiciones iniciales en cada iteración del proceso varían, lo cual no permite prefijar una serie de trayectorias y acciones que se repiten en cada ciclo. Además, otro de los propósitos del sistema es que sea aplicable en un robot comercial, en un proceso real y funcionando en tiempo real. Esta característica tiene un impacto directo tanto en los requisitos de coste computacional del sistema, como en la fiabilidad, y precisión de la solución generada. Finalmente, el sistema aplicado en esta Tesis para el robot YuMi debería ser portable a robots de otros modelos. Para afrontar todos estos retos, se ha definido una nueva arquitectura de control, que se ha construido incrementalmente a lo largo de los tres estudios presentados en esta Tesis.

El primero de los estudios sienta las bases de la arquitectura de control definida en esta Tesis. A diferencia de otros estudios enfocados en el control directo de las articulaciones del robot, la arquitectura de control planteada está basada en las coordenadas cartesianas del efector final del robot. Este enfoque permite reducir la complejidad del problema, lo cual es esencial para su potencial aplicabilidad en un proceso real. Además, la arquitectura define una división del controlador en dos capas. La capa superior, agnóstica respecto al robot integrado, aborda los retos de alto nivel, como la generación de las trayectorias del robot, o el orden de procesamiento de las piezas. La capa inferior, representada por el propio robot, permite delegar el movimiento de los brazos en el controlador interno que éste integra. Esta división permite conseguir soluciones robustas y precisas, aprovechando las garantías de control que ofrece el fabricante.

El segundo estudio reemplaza el robot simple de dos brazos, con dos grados de libertad cada uno, por un robot colaborativo comercial, el IRB14000 (YuMi). En este estudio se introducen el resto de características ignoradas inicialmente en el primer estudio, como la existencia del efector final (pinza), la posibilidad de colisión entre brazos, la diversidad de posibles configuraciones articulares para una mismo posición final, etc.. Este estudio demuestra que la arquitectura diseñada sigue siendo válida al introducir un robot comercial en el sistema.

Finalmente, el tercer estudio extiende la arquitectura de control para abarcar tareas de pick-and-place donde las piezas están en movimiento. Esta característica presenta un nuevo desafío ya que la posición de las piezas varía durante la operación y el sistema debe encontrar la posición ideal donde cada pieza debe ser recogida para optimizar el proceso.

Dentro de esta arquitectura de control, en este Tesis se comparan tres métodos diferentes para generar una solución. El método basado en MDP permitiría en un futuro afrontar variantes del problema donde se introdujera incertidumbre en el sistema. El método

basado en grafos es el que, con diferencia, obtiene una solución en un menor tiempo, lo cual es un factor clave para su aplicabilidad en tiempo real. Por último, el método basado en PDDL es el que mejor escala con el número de piezas y el único capaz de resolver el problema en los experimentos que incluyen más piezas.

Una de las debilidades de este sistema es la pobre escalabilidad con respecto al número de robots y de piezas. Para afrontar esta situación se propone la opción de reducir la cantidad de movimientos disponibles del robot. En particular se definen cuatro modos de navegación. El modo de navegación más sencillo solo permite movimientos en el plano, y esto contribuye a reducir el coste computacional del problema. El modo de navegación más completo dota de mayor flexibilidad en el tipo de movimientos (movimientos ortogonales y diagonales en las tres dimensiones) a costa de aumentar considerablemente el coste computacional. En los experimentos realizados se ha observado que el segundo modo de navegación (movimientos ortogonales y diagonales en el plano) resulta ser el más eficiente.

6.2 Trabajo futuro

Durante la elaboración de esta tesis doctoral han quedado algunas líneas de trabajo abiertas. Estas líneas se pueden resumir en los siguientes puntos:

- Explorar enfoques multi-agente, MDP jerárquico u otros que permitir reducir el coste computacional del sistema
- Experimentar implementación de algoritmo Value Iteration (MDP) en HW especializado (GPU) dado que está basado principalmente en operación con matrices.
- Implementar en un lenguaje compilado (c) el proceso de generación offline del MDP, que es la parte más costosa en tiempo del sistema, aunque solo ha de ejecutarse una única vez por proyecto.
- Modelar enlaces del robot mediante formas geométricas (ej. elipsoides) para eliminar la dependencia de simuladores externos (ej. RobotStudio) para generar la información relativa a colisiones entre brazos.

BIBLIOGRAFÍA

[1] J. Hermann, A. David, A. Wagner, and M. Ruskowski. “Considering interdependencies for a dynamic generation of process chains for production as a service.” Procedia Manufacturing, vol. 51, pp. 1454–1461, 2020.

[2] M. Ruskowski, A. Herget, J. Hermann, W. Motsch, P. Pahlevannejad, A. Sidorenko, S. Bergweiler, A. David, C. Plociennik, J. Popper, K. Sivalingam, and A. Wagner, “Production Bots fur Production Level Skill-basierte Systeme fur die Produktion der Zukunft,” atp magazin, vol. 62, pp. 62–71, 2020.

[3] S. Wrede, O. Beyer, C. Dreyer, M. Wojtynek, and J. Steil, “Vertical integration and service orchestration for modular production systems using business process models,” Procedia Technology, vol. 26, pp. 259–266, 2016.

[4] C.J. Sellers and S. Y. Nof. “Performance analysis of robotic kitting systems.” Robotics and Computer-Integrated Manufacturing. Volume 6, Issue 1, pp. 15-24, 1989. https://doi.org/10.1016/0736-5845(89)90081-1.

[5] N. Kimura et al., “Mobile dual-arm robot for automated order picking system in warehouse containing various kinds of products,” in Proc. IEEE/SICE Int. Symp. Syst. Integr. (SII), Dec. 2016, pp. 332–338.

[6] T. Tamei, T. Matsubara, A. Rai, and T. Shibata, “Reinforcement learning of clothing assistance with a dual-arm robot,” in Proc. 11th IEEE-RAS Int. Conf. Humanoid Robots, Oct. 2011, pp. 733–738.

[7] Y. Li, C. Yang, W. Yan, R. Cui, and A. Annamalai, “Admittance- based adaptive cooperative control for multiple manipulators with output constraints,” IEEE Trans. Neural Netw. Learn. Syst., vol. 30, no. 12, pp. 3621–3632, Dec. 2019.

[8] Z. Li and C.-Y. Su, “Neural-adaptive control of single-master–multiple- slaves teleoperation for coordinated multiple mobile manipulators with time-varying communication delays and input uncertainties,” IEEE Trans. Neural Netw. Learn. Syst., vol. 24, no. 9, pp. 1400–1413, Sep. 2013.

[9] J. Borrell Méndez, C. Perez-Vidal, J. V. Segura Heras and J. J. Pérez-Hernández, "Robotic Pick-and-Place Time Optimization: Application to Footwear Production," in IEEE Access, vol. 8, pp. 209428-209440, 2020, doi: 10.1109/ACCESS.2020.3037145.

[10] Borrell, J., Perez-Vidal, C. & Segura, J.V. Optimization of the pick-and-place sequence of a bimanual collaborative robot in an industrial production line. Int J Adv Manuf Technol 130, 4221–4234 (2024). https://doi.org/10.1007/s00170-023-12922-9

[11] T. He, H. Wang, and S. Won Yoon. “Comparison of Four Population-Based Meta-Heuristic Algorithms on Pick-and-Place Optimization”. Procedia Manufacturing, vol. 17, pp. 944-951, 2018.

[12] X. Ye, Y. Zhang, C. Ru, J. Luo, S. Xie and Y. Sun, "Automated Pick-Place of Silicon Nanowires," in IEEE Transactions on Automation Science and Engineering, vol. 10, no. 3, pp. 554-561, July 2013, doi: 10.1109/TASE.2013.2244082.

[13] J. Gao, X. Zhu, A. Liu, Q. Meng, and R. Zhang, "An Iterated Hybrid Local Search Algorithm for Pick-and-Place Sequence Optimization," Symmetry, vol. 10, no. 11, p. 633, Nov. 2018, doi: 10.3390/sym10110633.

[14] J. Kennedy and R. Eberhart, "Particle swarm optimization," Proceedings of ICNN'95 - International Conference on Neural Networks, Perth, WA, Australia, 1995, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.

[15] M. Kumar, M. Husian, N. Upreti, and D. Gupta. "Genetic Algorithm: Review and Application." International Journal of Information Technology and Knowledge Management, vol. 2, no. 2, pp. 451-454, July-December 2010.

[16] M. Dorigo, M. Birattari and T. Stutzle, "Ant colony optimization," in IEEE Computational Intelligence Magazine, vol. 1, no. 4, pp. 28-39, Nov. 2006, doi: 10.1109/MCI.2006.329691.

[17] A. R. Alazzam, "Using BUA algorithm to solve a sequential pick and replace problem," 2018 International Conference on Information and Computer Technologies (ICICT), DeKalb, IL, USA, 2018, pp. 144-149, doi: 10.1109/INFOCT.2018.8356858.

[18] S. Daoud, H. Chehade, F. Yalaoui, and L. Amodeo. "Efficient metaheuristics for pick and replace robotic systems optimization." Journal of Intelligent Manufacturing, vol. 25, pp. 27–41 (2014). https://doi.org/10.1007/s10845-012-0668-z

[19] Gafur, N.; Kanagalingam, G.; Wagner, A.; Ruskowski, M. Dynamic Collision and Deadlock Avoidance for Multiple Robotic Manipulators. IEEE Access 2022, 10, 55766–55781.

[20] Y. Zhang and J. Wang, "Obstacle avoidance for kinematically redundant manipulators using a dual neural network," IEEE Trans. Syst. Man, Cybern. B, Cybern., vol. 34, no. 1, pp. 752–759, Feb. 2004.

[21] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," Auton. Robot., vol. 13, no. 3, pp. 207–222, 2002.

[22] G. Wen, S. S. Ge, F. Tu, and Y. S. Choo, "Artificial potential-based adaptive H∞ synchronized tracking control for accommodation vessel," IEEE Trans. Ind. Electron., vol. 64, no. 7, pp. 5640–5647, Mar. 2017.

[23] L. Palacios, M. Ceriotti, and G. Radice, "Close proximity formation flying via linear quadratic tracking controller and artificial potential function," Adv. Space Res., vol. 56, no. 10, pp. 2167–2176, Nov. 2015.

[24] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in Autonomous Robot Vehicles. Newbury Park, CA, USA: Sage, 1986.

[25] R. Volpe and P. Khosla, "Manipulator control with superquadric artificial potential functions: Theory and experiments," IEEE Trans. Syst., Man, Cybern., vol. 20, no. 6, pp. 1423–1436, 1990.

[26] D. Guo and Y. Zhang, “A new inequality-based obstacle-avoidance MVN scheme and its application to redundant robot manipulators,” IEEE Trans. Syst., Man, Cybern. C, Appl. Rev., vol. 42, no. 6, pp. 1326–1340, Nov. 2012.

[27] W. Sun, L. G. Torres, J. V. D. Berg, and R. Alterovitz, “Safe motion planning for imprecise robotic manipulators by minimizing probability of collision,” in Robotics Research. Cham, Switzerland: Springer, 2016.

[28] J. Oh, H. Bae, and J.-H. Oh, “Analytic inverse kinematics considering the joint constraints and self-collision for redundant 7DOF manipula- tor,” in Proc. 1st IEEE Int. Conf. Robot. Comput. (IRC), Apr. 2017, pp. 123–128.

[29] T. Kivelä, J. Mattila, J. Puura, and S. Launis, “Redundant robotic manipulator path planning for real-time obstacle and self-collision avoidance,” in Proc. Int. Conf. Robot. Alpe-Adria Danube Region, 2017, pp. 208–216.

[30] D. Nicolis, M. Palumbo, A. M. Zanchettin, and P. Rocco, “Occlusion- free visual servoing for the shared autonomy teleoperation of dual-arm robots,” IEEE Robot. Autom. Lett., vol. 3, no. 2, pp. 796–803, Apr. 2018.

[31] Gracia Calandin, LI.; Sala Piqueras, A.; Garelli, F. (2014). Robot coordination using task-priority and sliding-mode techniques. Robotics and Computer-Integrated Manufacturing. 30(1):74-89. doi:10.1016/j.rcim.2013.08.003.

[32] Mateu-Gomez, D.; Martínez-Peral, F.J.; Perez-Vidal, C. Multi-Arm Trajectory Planning for Optimal Collision-Free Pick-and-Place Operations. Technologies 2024, 12, 12. https://doi.org/10.3390/technologies12010012

[33] Gafur, N.; Kanagalingam, G.; Wagner, A.; Ruskowski, M. Dynamic Collision and Deadlock Avoidance for Multiple Robotic Manipulators. IEEE Access 2022, 10, 55766–55781.

[34] Y. Yang, J. Yang, H. Zeng and J. Li, "A 3D Vision-Based Conveyor Tracking System for Pick-and-Sort Robotic Applications," 2021 6th International Conference on Robotics and Automation Engineering (ICRAE), Guangzhou, China, 2021, pp. 231-236, doi: 10.1109/ICRAE53653.2021.9657787.

[35] Anton, F., Borangiu, T., Anton, S., Răileanu, S., Lişiţă, A. (2022). An Evaluation of Pick on the Fly Methods for High-Speed Part Processing in Low Cost Digital Manufacturing. In: Borangiu, T., Trentesaux, D., Leitão, P., Cardin, O., Joblot, L. (eds) Service Oriented, Holonic and Multi-agent Manufacturing Systems for Industry of the Future. SOHOMA 2021. Studies in Computational Intelligence, vol 1034. Springer, Cham. https://doi.org/10.1007/978-3-030-99108-1_32

[36] Ku, YD., Yang, JH., Fang, HY. et al. Optimization of Grasping Efficiency of a Robot Used for Sorting Construction and Demolition Waste. Int. J. Autom. Comput. 17, 691–700 (2020). https://doi.org/10.1007/s11633-020-1237-0

A

CONTRIBUCIONES PRINCIPALES

La presente Tesis Doctoral está sustentada por un compendio de trabajos previamente publicados o en actual revisión en revistas de impacto, indexadas según JCR Science Edition. El cuerpo de dicha tesis queda constituido por los siguientes artículos, cuyas referencias bibliográficas completas se indican a continuación:

Mateu-Gomez, D.; Martínez-Peral, F.J.; Perez-Vidal, C. <<Multi-Arm Trajectory Planning for Optimal Collision-Free Pick-and-Place Operations>>. Technologies 2024, 12, 12. https://doi.org/10.3390/technologies12010012

- Título de la revista: Technologies
- Factor de impacto (JCR 2022): 3.6
- Categoría: ENGINEERING, MULTIDISCIPLINARY. Cuartil Q1 (41/178)

Mateu-Gomez, D.; Martínez-Peral, F.J.; Perez-Vidal, C. <<Dual-Arm Coordination for Mutual-Collision-Avoidance to Implement a Pick-on-the-Fly Application>>

- Título de la revista: Robotics and Computer-Integrated Manufacturing
- Factor de impacto: (JCR 2022): 10.4
- Categoría: COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS. Cuartil Q1, Decil D1 (7/110)
- Estado: en revisión

A continuación, se anexan los documentos correspondientes a dichas publicaciones.

technologies

MDPI

Article

Multi-Arm Trajectory Planning for Optimal Collision-Free Pick-and-Place Operations

Daniel Mateu-Gomez [1], Francisco José Martínez-Peral [2,*] and Carlos Perez-Vidal [2]

[1] Analog Devices Spain, Parc Cientific Universitat de Valencia, C/Catedratico Agustin Escardino 9, 46980 Paterna, Spain; daniel.mateu@analog.com
[2] Instituto de Investigación en Ingeniería I3E, Miguel Hernández University, Av. de la Universidad s/n, 03202 Elche, Spain; carlos.perez@umh.es
* Correspondence: francisco.martinezp@umh.es

Abstract: This article addresses the problem of automating a multi-arm pick-and-place robotic system. The objective is to optimize the execution time of a task simultaneously performed by multiple robots, sharing the same workspace, and determining the order of operations to be performed. Due to its ability to address decision-making problems of all kinds, the system is modeled under the mathematical framework of the Markov Decision Process (MDP). In this particular work, the model is adjusted to a deterministic, single-agent, and fully observable system, which allows for its comparison with other resolution methods such as graph search algorithms and Planning Domain Definition Language (PDDL). The proposed approach provides three advantages: it plans the trajectory to perform the task in minimum time; it considers how to avoid collisions between robots; and it automatically generates the robot code for any robot manufacturer and any initial objects' positions in the workspace. The result meets the objectives and is a fast and robust system that can be safely employed in a production line.

Keywords: pick-and-place operations; robotic sequence order optimization; dual-arm collision avoidance; Markov Decision Process; PDDL in robotic manipulation

Citation: Mateu-Gomez, D.; Martínez-Peral, F.J.; Perez-Vidal, C. Multi-Arm Trajectory Planning for Optimal Collision-Free Pick-and-Place Operations. *Technologies* **2024**, *12*, 12. https://doi.org/10.3390/technologies12010012

Academic Editor: Ahmad Lotfi

Received: 9 November 2023
Revised: 22 December 2023
Accepted: 16 January 2024
Published: 22 January 2024

1. Introduction

Production lines increasingly require short and customized manufacturing batches tailored to customer demand, which affects the configuration of the factories and their level of automation [1]. Factory systems must become more flexible and adaptable [2,3]. On the other hand, robots are transitioning from being rigid, pre-programmed, and almost "blind" to being collaborative, easy to reprogram and equipped with a large number of sensors. Consequently, this new generation of robots is increasingly being used in assembly, disassembly, or packaging lines of small factories or reduced batches. By operating in a shared workspace with human operators, production systems can further increase efficiency and minimize the workspace area. However, this creates a problem in terms of collision control. Figure 1 shows a group of robots sharing the same workspace and performing a common pick-and-place task. In this case, the group of robots needs to place pieces of the same color in their corresponding trays without colliding with each other and completing the task as efficiently as possible.

Typically, robot trajectories are programmed to ensure there is no collision between the robot and a static environment. Since robots are often used for repetitive tasks, it is sufficient to plan collision-free trajectories only once. If certain parts of the production process are modified, it becomes necessary to re-plan collision-free trajectories and reprogram all manipulators. In cases where the environment is unpredictable, such as when multiple robots are present, operators modify the workspace, or there are changes in the production process conditions (e.g., the repositioning of production elements), modular approaches are

required where each robot can react to changes in real time. The ability to optimally adapt to flexible manufacturing needs, would be an advantage in new production processes.

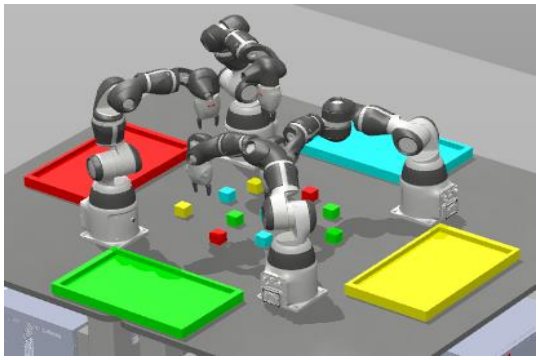
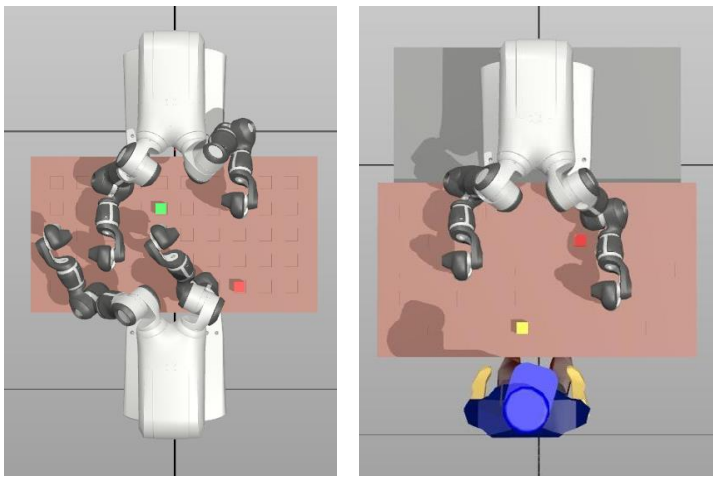


Figure 1. Pick-and-place scenario with four collaborative ABB IRB 14,050 robots.

Object manipulation involves not only performing tasks while avoiding collisions between robots and the environment but also doing so in the shortest possible time. Figure 2 depicts a representation of a production process where a series of pick-and-place operations need to be performed in a non-predefined order. This means that a certain number of pieces (e.g., kitting operation [4]) must be manipulated in any order, while considering the establishment of a pick-and-place operation sequence that minimizes the operation time. In this sense, a set of robotic arms, as shown in Figure 2a, or a dual-arm robot and a human operator, as shown in Figure 2b, could be considered. This last case is particularly relevant because the system needs to be adapted to the "disappearance" of pieces grabbed by the operator, which entails an instantaneous change from one state to another, requiring the recalculation of the optimal solution.



(**a**) (**b**)

Figure 2. Pick-and-place scenario with: (**a**) two collaborative ABB YuMi robots (IRB 14,000); (**b**) one collaborative robot and a human operator sharing the workspace.

A particular case, as shown in Figure 3, is when a specific robot (e.g., left robot) requires the collaboration of another (e.g., right robot) to reach a piece and place it in the desired location. This type of task involves more than just trajectory planning; it requires high-level task planning, which becomes even more complex when a task must be completed in the shortest possible time.

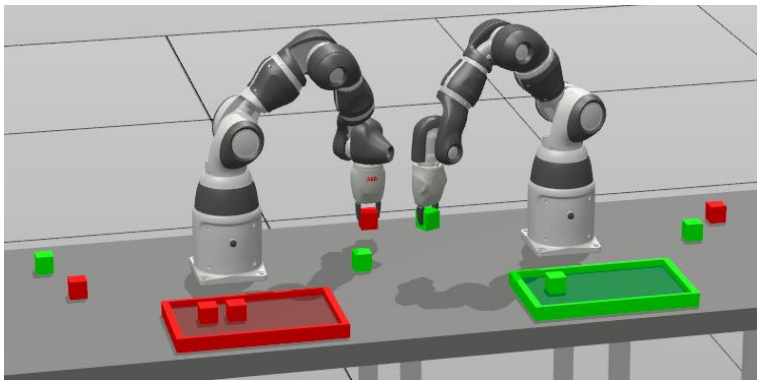


Figure 3. Collaboration required between both arms to complete the task.

This research introduces a new approach for effectively and efficiently solving pick-and-place tasks by using machine learning algorithms and smart architectural design. It becomes more significant in collaborative scenarios with multiple pieces and multiple robots sharing a common workspace. It plays an automation role, eliminating the requirement for a human to manually create the trajectories for the robots, and placing emphasis on task time minimization. This study shows this approach applied to a scenario comprising multiple pieces, ranging from 2 to 10, and a pair of robots, specifically a collaborative robot with two arms.

This article is organized as follows. Section 2 provides a brief analysis of the current state-of-the-art in pick-and-place algorithms. Section 3 describes the proposed approach, formulated as a discretization of the robot's workspace. The actions that the robot can perform to solve the problem and collision avoidance system are presented. Section 4 focuses on solving the problem, addressed as a Markov Decision Process (MDP) and as a planning problem using Planning Domain Definition Language (PDDL). The study's results are presented in Section 5, where, in addition to demonstrating the algorithm's performance, the results obtained in a visual environment are shown. Simulations provide an intuitive evaluation of the outcome. Finally, Section 6 presents the conclusions and proposes future work in this field.

2. Related Work

In recent years, there has been an increasing emphasis on developing algorithms to optimize the pick-and-place task, with a particular focus on minimizing operation time. This challenge has been addressed in different applications, such as shoe production [5], PCB assembly [6], and the pick-and-place of individual nanowires in Scanning Electron Microscopy (SEM) [7]. Previous research has addressed different subproblems related to pick-and-place optimization, including placement sequencing and feeder allocation [8], among others. Minimizing operation time has been extensively explored in the field using metaheuristic algorithms. Given the complexities associated with these problems, heuristic and metaheuristic algorithms are widely employed to solve them. Examples of such algorithms include Particle Swarm Optimization (PSO) [9], the Genetic Algorithm (GA) [10], and Ant Colony Optimization (ACO) [11]. These methods employ stochastic search techniques that emulate biological or natural evolution and natural selection. One advantage of these methods is that they often provide solutions that are close to optimal in most cases [12].

Several studies have employed mathematical algorithms in industrial pick-and-place processes. In [13], an ACO algorithm was proposed, which outperformed the Exhaustive Enumeration Method (EEM), although the study did not specifically address multi-objective pick-and-place processes. GA and ACO algorithms presented better solutions in terms of execution time in pick-and-place problem-solving for PCB assembly, compared to PSO and SFLA algorithms due to their optimized structures [8]. The Hybrid Iterated

Local Search (IHLS) algorithm combines local search and integer programming to drastically reduce computing time for large-scale processes compared to other heuristics. In [12], a novel metaheuristic algorithm called Best Uniformity Algorithm (BUA) was developed. It generates random solutions within a search space to find the optimal pick-and-place sequence. The BUA algorithm achieved remarkable results in less time when compared to the GA algorithm, although for smaller problem sizes. Its implementation focused on a sequential pick-and-place process involving a single head moving one object at a time. A similar approach was presented in [5], where a Decision Tree Algorithm optimized the pick-and-place sequence using a two-armed robot to place shoe components into a mold. The Decision Tree Algorithm was employed to recognize patterns and calculate the best real-time optimized sequence. The results indicated that implementing the algorithm improves performance compared to random planning.

While [14] shares a similar objective, it diverges in terms of approach. This work focuses on collision-free path planning using priority arbitration based on the distance to the goal. However, unlike the referenced work, it does not incorporate prior planning of the operation order to minimize the task execution time. This aspect has been successfully addressed in the algorithm proposed in this article.

3. Approach to This Research

3.1. Materials

The setup used in this research comprises multiple pieces, ranging from 2 to 10, and it is operated by one of the most widespread collaborative bimanual robots, the ABB YuMi IRB 14000. The YuMi robot is designed for small-piece manipulation tasks and can cooperate with humans in the same environment without the need for a protection cell, as shown in Figure 2b. It is a redundant robot, featuring 7 degrees of freedom (DOF) per arm and it is equipped with a gripper, used to manipulate the pieces.

The overall goal of the task is to carry a certain number of pieces to their designated location. Specifically, for that study, the destination of the pieces has been predetermined and the source location is provided in real time when the task is about to start. That information might be provided by a device such as the Intel D415 Stereo Depth Camera, but for this study that is emulated by arbitrarily generating those positions on each run. All the experiments in this study have been done using RobotStudio®.

3.2. Design of the System Architecture

The task under study requires considering several factors: which piece to assign to each robot, what order should the pieces be taken in, how to avoid collision between robots, which trajectory should follow a robot, and, in the end, how to minimize the overall task time. This study approaches this project from the perspective of a decision-making problem and the purpose is to devise a solution that can be implemented in real-world processes in the industry. When adopting a decision-making approach, it is essential to define the set of actions that can be taken within the system and understand the concept of the current state of the system. YuMi is a high-precision bi-manual robot featuring 14-DOF. A strategy based on direct control of the arm's joint configuration would be impractical from the perspective of a solution designed for real-world processes, due to the explosion on the combinatory. An over-discretized, low-resolution joints scenario would lack enough precision to undertake the task and would still be impractical; a hypothetical scenario with low 10 degrees resolution, would lead to an average joint bin size of 20 values and a combinatory exceeding a quadrillion combinations.

The approach presented in this study is based on the Cartesian coordinates of the robot. The robots' workspace, defined by their area of influence, has been discretized as a 3D constellation of points, henceforth grid, used as waypoints to guide the trajectory of the robots. The resolution of this grid is configurable, although, for this work, a grid of ten columns, five rows, and three heights has been defined, resulting in a total of one hundred

and fifty possible waypoints, each separated by 100 mm. Following the strategy of Cartesian coordinates and based on the real-world applicability requirement, the proposed architecture consists of a dual-layer structure. At a high level, the decision-making controller manages all the high-level concepts previously mentioned, such as trajectory generation, pieces assignment, etc. At a low level, the manufacturer's software and hardware layers are responsible for the robots' transitions, ensuring precision and robustness. This layering division enables the solution to be scaled to other robot models. Another key aspect of the design concerns the robot's movements. The robot is allowed to move towards any of its neighboring waypoints in the 3D grid. This decision facilitates the robot arms' synchronization and reduces the complexity of the problem. This synchronization happens at each time-step, where all robots point to their next waypoint target. Regarding the complexity, the number of waypoints to choose from is reduced to 27, which includes the option of staying at the current location if that is the best option. This is a significant improvement over the possibility of moving to any of the 150-grid waypoints. Based on that, robot trajectories have been represented as an ordered set of waypoints in the grid. At a low level, the manufacturer layer will execute a continuous and smooth trajectory guided by these waypoints (see Figure 4 for more information).

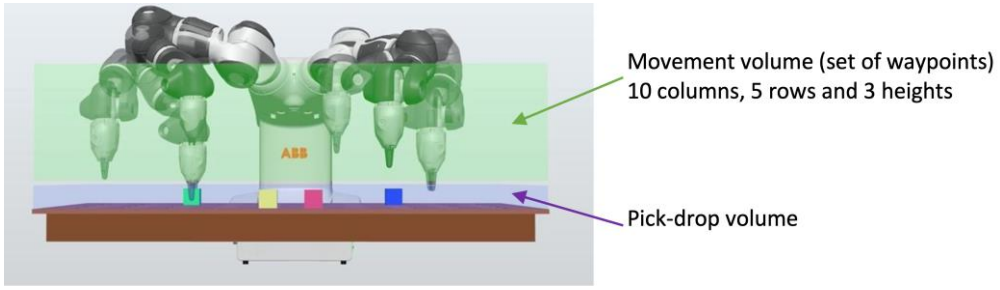


Figure 4. Visual description of volumes defined in the system architecture.

This architecture models the system as a set of robots and pieces, all defined as 3D points within the workspace. This, by itself, is not enough to account for some factors, such as the fact that robot bodies are not single points in the workspace, causing the problem of collisions, robots have a limited area of influence, gripper grasp operation details, and robot joints have angular range limitations, causing discontinuities in the movement if a joint limit is violated. To address these gaps, some additional design measures have been adopted. A prior analysis was carried out to collect information about the area of influence of each robot within the grid, the potential collision between robots for all grid location combinations, and the preferred robot joint configuration (see Section 3.3.5) for each arm and every grid location. All this information is stored in a database that the controller has access to. So, in essence, this architecture represents the robots as 3D points in the workspace, but accounting for their body presence and range of actuation by means of the database information. The last key factor of the design is the strategy followed to execute any robot trajectory. This architecture is, by nature, Cartesian-coordinates-based (transition from point A to point B), dealing with gripper 3D locations, but the actions taken by the controller are joint-based (transition from joint configuration Q1 to joint configuration Q2). The purpose is to gain direct control over the robots' joint configuration, instead of leaving that decision to the robot criteria (there are multiple valid joint configurations that satisfy the criteria for being in point B). This helps to ensure a natural posture avoiding discontinuities or undesired movements during transitions between neighboring waypoints.

3.3. Design of the System Model

The system's model represents the real-world problem by ignoring the irrelevant details and only keeping the essential ones for the solution. Based on the system's

architecture, which has established the fundamentals of how the system works, it is essential to keep track of the location of robots, the assignment and status of the pieces, and the source and destination location of the pieces. Other relevant inputs, such as collision information or robot location viability, are derived from the fundamental features or internal variables that determine the system model. Specifically, this information is stored in a database that the model has access to.

3.3.1. State-Space

The state of the model is the representation of the current situation. For this study, it has been designed to be fully observable, therefore it captures all the relevant information that the controller needs in order to make a decision, as shown next:

$state = f(robotPos_i, robotStatus_i, piecesStatus_j, initPiecesPos_j, endPiecesPos_j)$

$robotPos_i$ = *3D location of the* $robot_i$ *gripper,* $\forall i = 0...N$

$robotStatus_i$ = *Indication of the piece, if any, carried out by the* $robot_i$, $\forall i = 0...N$

$piecesStatus_j$ = *Indication whether the* $piece_j$ *has been processed,* $\forall j = 0...K$

$initPiecesPos_j$ = *Source 3D location of the* $piece_j$, $\forall j = 0...K$

$endPiecesPos_j$ = *Target 3D location of the* $piece_j$, $\forall j = 0...K$

N = *Number of robots*

K = *Number of pieces*

Similarly, it has been designed to be discrete and single agent. The state is defined at the system level, not at the individual robot level, and captures, at the same time, the information of all the elements in the system. Following the defined dual-layer system architecture, the model sits on the high-level layer, where the robots transition from the current waypoint to the next one. Even though the final solution achieves smooth and continuous movements, thanks to the dual-layer design, the model is made up of a set of discrete variables.

The state-space is a concept that captures all the feasible configurations of the system. In this case, the state-space is finite, since all internal model variables are discrete, and its size evolves according to the equation shown in Equation (1).

$$Num\ states = (G + K \cdot P)^N \cdot (K + 1)^N \cdot N^K \quad (1)$$

where G is the 3D grid size, K is the number of pieces, N is the number of robots, and P is the number of steps for pick or drop.

For this work, the number of robots is 2, the number of pieces ranges from 2 to 10, and the grid size is 150 (10 × 5 × 3). The $K{\cdot}P$ term comes as a result of applying a special treatment technique for the pick-and-place operations. The technique decouples the area of movement of the robots, used to reach and transport the pieces, from the pick-and-place operations themselves. The source and destination locations of the pieces are on an inferior plane of the volume filled by the grid. This technique prevents the need to enlarge the grid to accommodate the pieces on it. P refers to the number of time-steps required for the vertical moves during pick and place operations.

3.3.2. Action-Space

The system architecture defines the robot navigation as a sequence of transitions between neighboring waypoints in the grid. The number of neighboring waypoints is always 26 in a 3D grid, except for the boundary, as shown in Figure 5. Additionally, in a multi-robot setup, there is the option to stay in the same position. This might be the most effective way to evade a collision, or only because there is nothing else to process for that robot. Pick-up and drop-down operations are handled in different ways. Since the source and destination location of the pieces sit on a working table, on an inferior plane to the grid, this action cannot be executed by the regular intra-grid navigation moves. By design, those operations must be carried out by the robot in the lower plane of the grid and in the same XY coordinates as the source or destination location of the piece. These operations involve

pure vertical moves and gripper open or close actions. In order to ensure a successful operation, the configuration of the robot places the gripper pointing towards the piece and the pieces have been defined as cubic blocks, just as it was done in [15].

The total number of available actions per robot is:

$$Num\ robot\ actions\ = \left(N_{navigate} + N_{stay} + N_{pick} + N_{drop}\right) = 26 + 1 + 1 + 1 = 29$$

At a system level, an action is defined as the aggregate of all actions performed by the robots. Since the action taken by a robot is independent from the action taken by the others, the total number of available system actions exponentially increases with the number of robots. In this case, for two robots it is:

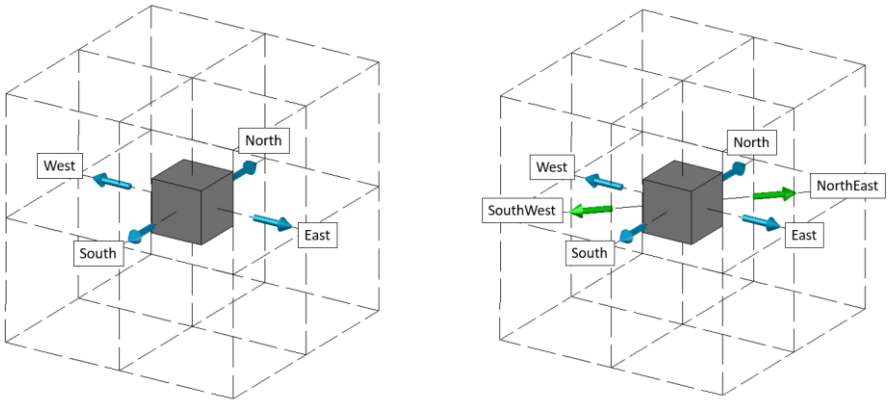
$$Num\ system\ actions = Num\ robot\ actions^{N} = 29^{2} = 841$$

Navigation Nodes:

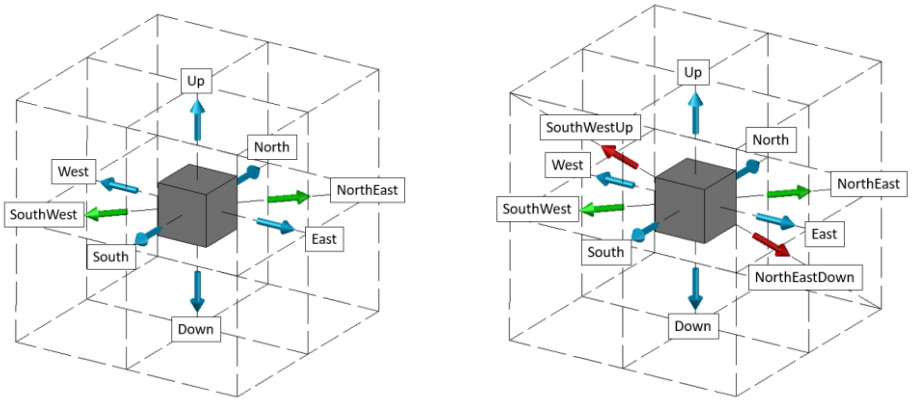
The computational complexity of the problem is related to the size of both state and action spaces. For the state-space, experiments with a different number of pieces were conducted. For the action-space, experiments with different sets of supported system actions have been conducted as well. To capture this idea, three additional incremental action modes have been specified, each containing a subset of the possible navigation actions:

1. Mode 1: Only orthogonal actions are allowed (North, South, East, West), as shown in Figure 5a, on the inferior plane of the grid.
2. Mode 2: This Mode adds diagonal actions (e.g., SouthWest, NorthEast, etc.), as depicted in Figure 5b, on the inferior plane of the grid.
3. Mode 3: This Mode adds Up and Down actions, as shown in Figure 5c. They enable robots to navigate the whole 3D grid.
4. Mode 4: This Mode considers all neighboring waypoint directions. It is the most flexible one, considering combinations of three orthogonal movements, as shown in Figure 5d (e.g., SouthWestUp, NorthEastDown, etc.).

As the robot increases its mobility with more degrees of freedom, combining movements along different axes, the number of time-steps required to complete the task might be reduced, at the expense of an increase in computation. That trade-off is evaluated in performed tests.



(**a**) (**b**)



(**c**) (**d**)

Figure 5. Action-space Modes defined in the project: (**a**) Orthogonal movements (Mode 1); (**b**) Orthogonal and diagonal movements (Mode 2); (**c**) Addition Up/Down movement (Mode 3); (**d**) Movement to all neighboring waypoints (Mode 4).

3.3.3. Reward-Space

The cost is a feedback indication, received by the controller when executing an action, and it is exclusively dependent on the current state of the system and the action taken on it. The controller makes use of it to plan a solution in the long term. This term is also known as reward in the Reinforcement Learning domain. By design, there are only two kinds of feedback, a general one to penalize any action and another one to indicate the success of the task. Specific values depend on the type of algorithm applied to solve the problem.

3.3.4. Integration with the Controller

Conceptually, the task involves multiple robots collaborating to transport a certain number of pieces in the least amount of time. For each piece, a robot must navigate to its location, pick it up, transport it to the designated location and drop it down. Robot navigation or piece transportation entails the gripper following a trajectory from point A to point B, which, according to defined system architecture, is formulated as a sequence of transitions between neighboring waypoints. The proposed system model operates at this abstraction level and takes advantage of robot capabilities to execute these transitions (from point A to point B).

The task starts at t_0 with the system model at state s_{t0}. The source and destination location of the pieces is not shown in the state representation because their values stay invariant during the whole task:

$$s_{t0} = (robotPos_0 \dots robotPos_{N-1}, robotStatus_0 \dots robotStatus_{N-1}, pieceStatus_0 \dots pieceStatus_{N-1})$$

$robotPos_i$ = *arbitrary position*, $\forall i = 0 \dots N$

$robotStatus_i$ = *CARRYING NO PIECE*, $\forall i = 0 \dots N$

$piecesStatus_j$ = *PIECE NOT PROCESSED*, $\forall j = 0 \dots K$

An action a_{t0} is taken and the system state evolves to s_{t1} according to the system model behaviour. The pseudo-code simulating model behavior is shown in Algorithm 1. That process is repeated until the state s_{tM} becomes a goal state. There might be multiple goal states in the state-space. In this case, a goal state is defined by:

$$pieceStatus_i = PIECE\ PROCESSED, \forall i = 0 \dots K$$

Algorithm 1: Pseudo-code to emulate system model behavior.

```
// State is a tuple containing all internal Model variables
// Action is a tuple containing the actions taken by each robot
ModelBehaviour(state, action)
    Check valid action in this state
    if action is not valid then
        return state, HIGH_PENALTY
    end if
   Compute expected next_state based on current state and action
   Check grid boundaries for each robot
   Check collision between robots
   for I from 1 to N do
      Check accessibility of robot i to its new location
      if any check fails then
         # state is unchanged
         reward = HIGH_PENALTY
      else
         state = next_state
         reward = SMALL_PENALTY
       end if
    end for
return state, reward
```

Regarding the collision check on the pseudo-code, it is worth noting that it is carried out both before and after taking the action. Since the grid is small enough, it is assumed there is no collision during this transition if there is not either before or after, except for the case when robots exchange location. That assumption was tested in RobotStudio® as well.

3.3.5. Database

The database contains key information used to obtain the robot joint configuration, detect collision between robots, and locations not reachable by any robot. All this information is not encoded in the state-space because it is dependent on other internal variables of the state-space, specifically the robot location. The collision detection and accessibility of the robots to a specific location are critical inputs for the controller in order to make a good decision.

A preliminary study of collisions and accessibility was performed using RobotStudio®. Collision validation is based on the combination of all robots' locations in the grid. Its size exponentially grows with the number of robots, $(X \cdot Y \cdot Z)^N$, where X, Y, Z corresponds to the size of all three axes of the grid, and N to the number of robots. The output is a Boolean indication of collision for each test. For this study, $(X \cdot Y \cdot Z)^N = (10 \cdot 5 \cdot 3)^2 = 22{,}500$ tests. Accessibility validation is individually carried out per robot, and it is based on the robot location and its area of influence. Its size is equal to the size of the grid, so $(X \cdot Y \cdot Z) = 150$ tests. The output is a Boolean indication of a feasible location. All this information is stored in a database (DB) that the controller has access to.

Figure 6 graphically depicts some examples of collision indication for some arbitrary configurations stored in the DB.

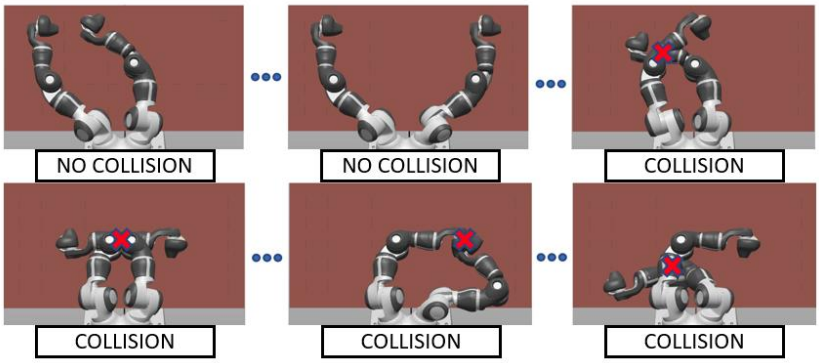


Figure 6. Visual representation of situations of collision detection.

The pseudo-code shown in Algorithm 2 has been used to collect the DB information, including the collision indication minimum distance between robots, and the accessibility and joint configuration ($\vec{q}$) per robot. This distance can be optionally used as a safety measure, forcing collision indication when it is lower than an established threshold. Its calculation involves obtaining the minimum distance between all links of one arm (α) and all links of the other arm (β), and keeping within the minimum among them, as indicated in Equations (2) and (3).

$$\delta_{ij} = \left\| \lambda i_{\alpha} - \lambda j_{\beta} \right\| \quad (2)$$

$$\delta_{min} = \left(\delta_{ij} \right) \quad (3)$$

where δ_{ij} is the distance between links i and j, i_{α} is the i link of the α arm, i_{β} is the i link of the β arm, and δ_{min} is the minimum value of all δ_{ij} calculated for a specific configuration.

Algorithm 2: Pseudo-code for data base generation (including collision detection, reachability, and joint configuration).

```
// For each arm of the robot get robot configuration
Move robot to initial position
for i from 1 to (X × Y × Z) do
    Read coordinates from file and RobTarget calculation
    if RobTarget = reachable then
        JointTarget calculation from RobTarget
        if JointPosError = not reachable or OutReach then
         Pos not reachable and RobTarget values set to 0
        end if
    Write Coordinates, Reachability and JointTarget data in file
    end if
end for
// For each height of the grid get collision data
Move α & β to initial position
for i from 1 to (X × Y × Z) do
    for j from 1 to (X × Y × Z) do
        Read α and β joints values from file
        Move α and β
        Check positions reachability
```

```
20:        if α & β = reachable then
21:            Measure distances between links
22:            Distance comparison and get δmin
23:            Write Coordinates & δmin info in file
24:        else
25:            Write Coordinates & δmin = 0 info in file
26:        end if
27:      end for
28:    end for
```

Another key piece of information stored in the database is the joint configuration to be used by any robot based on the grid location. This association between grid location and joint configuration is unique. The selection of appropriate joint configurations is based on certain criteria. All the configurations meeting these criteria are called feasible configurations and all are equally eligible. Finally, one configuration among all feasible ones is stored in the database. The criteria used in this study consider the following factors: the robot gripper is always pointing towards the worktable, the arm's elbow is correctly positioned to avoid collisions with the objects in the working area, singularities (reaching an angular limit) during operation must not occur, ensuring a smooth transition between adjacent cells, and in turn, through the whole trajectory. At a high level, the database solves the kinematics of the robots.

The database is accessed as a LookUp Table (LUT). The LUT is indexed by the robot's location to obtain either the robot accessibility information or the associated joint configuration, and it is also indexed by the location of all the robots to obtain collision information.

4. Solution of the Problem

The problem has been addressed as a decision-making challenge. Reinforcement Learning (RL), a growing field in Artificial Intelligence, is dedicated to this domain, encompassing a wide range of situations, including uncertainty, vast or continuous state and action spaces, the absence of a model, multi-agent environments, and so on. RL necessitates the representation of the model as a Markov Decision Process (MDP).

In this study, the architecture of the system and the model has been defined by design, hence the model is perfectly known, the state and action spaces are discrete, finite, deterministic, and relatively huge, but do not exceed the limits, requiring some kind of function approximator. The architecture is designed such that the controller acts as a single brain for all robots, making it a single agent. Given these properties, methods from other areas, such as graph search or planning languages, may be applied as well.

Figure 7 represents the whole experiment process for all three approaches (MDP, Graph search, and PPDL). The diagram illustrates a sequence of tasks that each approach needs to complete in a specific order. The flow is divided into three stages: Project Level, Task Level, and Execution Level.

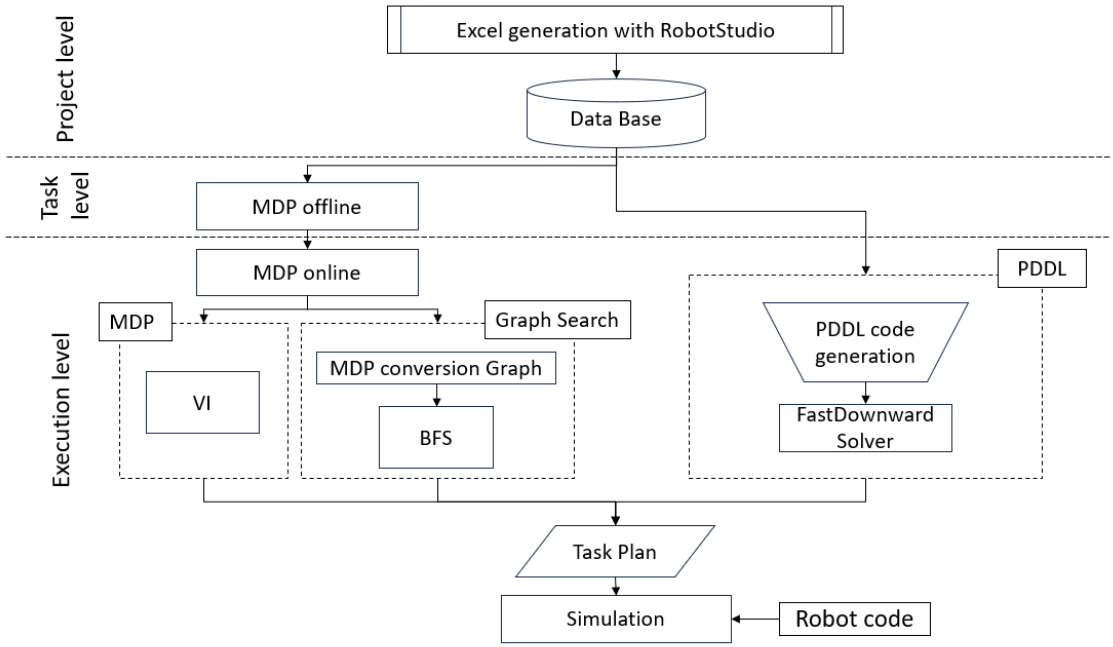


Figure 7. Process description for all three approaches (MDP, Graph search, and PDDL).

The Project Level stage is common for all approaches and involves any task depending on predefined and invariable design decisions for the project, such as the robot selection (YuMi) or the grid resolution (10 × 5 × 3). This stage needs to be performed only once, unless any of those design decisions change, such as using a new robot. This stage is responsible for generating the system database (described in Section 3.3.5), which in turn, is an input for the Task Level stage. The database includes information related to robot location feasibility, the robots' collision, and the robots' joint configuration.

The Task Level stage is related to any task that involves parameter changes in successive experiments. This stage only needs to be re-executed if any of those parameters change. These parameters are the number of available actions, the grid size, and the number of pieces. The number of available actions and the grid size depend on the mode of action (described in Section 3.3.2) selected for the experiment. While still keeping the same grid resolution (100 mm between cells), some experiments restrict the robot movement to a plane (reducing the grid to 10 × 5 × 1). The number of pieces ranges from 2 to 10. This stage only applies to approaches based on the MDP (MDP and Graph search) and it is responsible for building the MDP model and saving it to disk. This stage is performed offline, and the resulting MDP model is valid for any parameter changes in the next stage.

The Execution Level is also known as the online stage. At this stage, the process receives information about the location of the pieces and robots. The entity providing the location of the pieces is out of the scope of this study. At this stage, the flow for each approach diverges.

The first step in the MDP approach is to update the MDP model. Even though the MDP model is built in the previous stage, the location of the pieces is unknown at that point, hence the state-action pairs related to pick or drop operations need to be updated in this stage once this information is available. MDP generation is split to reduce the online execution time, as most of the computation is performed offline. Once the MDP is built, an MDP solver can be applied to generate a solution. Value Iteration was selected (described in Section 4.1) for this study.

The first step in the Graph search flow is common with the MDP flow, the MDP model update. Once the MDP is fully defined, there is a conversion task devoted to

transforming the MDP model into a graph (described in Section 4.2). This graph could be solved by any search algorithm. BFS was selected for this study (described in Section 4.2).

The last flow, PDDL, involves two tasks, the generation of the PDDL source code and generating a solution for the PDDL description. The first task is automated. This automation takes some inputs: the number of pieces, their locations, the robot locations, and the target, and generates the PDDL description for this particular problem. The solution generation is carried out by a PDDL planner solver. FastDownward was selected for this study.

All three approaches converge again in the Task Plan, where a common file format solution is generated. Specifically, for this study, this file contains the sequence of joint configurations and gripper actions required for each robot to complete the overall task.

The final step of the process involves implementing the solution to either a real or simulated problem. In this study, RobotStudio® was used to simulate the process. RobotStudio® requires the solution file, the process project (including the robot, the pieces, and the worktable), and the robot source code as inputs. The robot source code is written in RAPID, which is a programming language used for ABB robots. The robot program was used in all experiments to parse the solution file and execute the actions on the robots (see simulation video links in Section 5.2).

4.1. MDP Model

A Markov Decision Process is a mathematical framework for modeling decision-making problems under uncertainty. The four core components of an MDP model are: a set of states S, a set of actions A, the state transition function $P(s'|s,a)$, and the reward function $R(s,a,s')$. The transition function indicates the probability of reaching a state s' if an action a is taken in state s. The reward function indicates the reward obtained for that same transition. The key property of an MDP is the Markov property, which states that the effects of an action taken in a state only depend on that state and not on the prior history. Figure 8 represents this concept, where the next state s_{t+1} and reward r_{t+1} only depend on the current state s_t and the action taken in it.

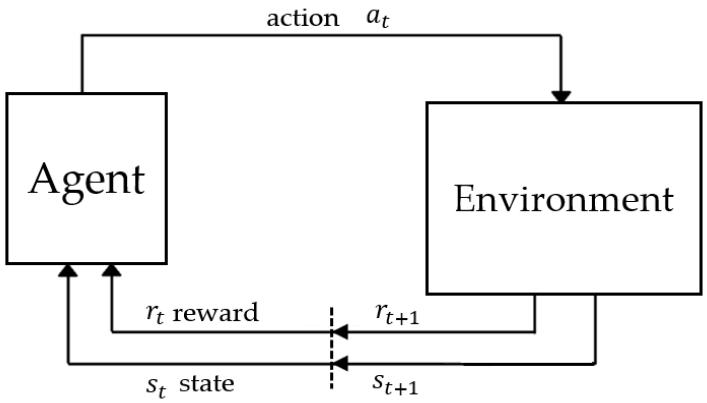


Figure 8. MDP model and building elements (for deterministic scenario).

In the context of this study, the agent is the brain making decisions, and the environment is the system under control, including the pieces and robots. Depending on the nature of the problem, the MDP may be fully known, partially known (i.e., the reward function is unknown), or totally unknown. In this work, the MDP is totally known (all four core components are defined by design). The state-space and action-space are described in detail in Sections 3.3.2 and 3.3.3, respectively.

The system is deterministic, so the state transition function in this study is not expressed in terms of probabilities $P(s,a,s')$, but as a function $T(s,a)$, which specifies the next state of the system given its current state and the action taken. This component has

been modelled in the software implementation as a LookUp Table (LUT), which is indexed by the current state and action and returns the next state. The logic used to populate the table has been implemented in Python and takes all the information into consideration (i.e., the grid boundary exceeded by a robot, collision between robots, location not reachable, invalid action on the current state, etc.).

For the same reason (deterministic system), the reward function is expressed in this study as $R(s,a)$ instead of $R(s,a,s')$. This component is also modelled as a LUT, and is indexed the same way as the transition function, but it returns the reward value. The logic to populate this LUT is implemented in Python as well. By design, there are three kinds of defined rewards:

$$R(s,a) = \begin{cases} 100 & if\ s' = s_{goal} \\ -1 & else\ if\ (s,a)\ is\ valid \\ -20 & else \end{cases} \quad (4)$$

A big reward (+100) indicates that the goal is met (all pieces have been transported). A big penalty (−20) indicates that the action taken on the state s is not valid (i.e., exceeding grid boundaries, collision between robots, etc.). Finally, a small penalty is applied in all other cases as a strategy for RL algorithms to find the optimal solution (smallest number of steps), since their purpose is to maximize the long-term reward.

The core idea behind this framework is simple. At any point in time, the agent takes an action based on the current system state. As a result, the system state changes, according to the Transition function, and the agent receives a feedback or reward, according to the Reward function.

The purpose of any RL algorithm over an MDP model is to find the optimal action to take on any state. This concept is called Policy, $\pi(s,a)$, and based on that policy the agent can make a sequence of optimal decisions from the initial state up to the goal state. Typically, RL algorithms are classified into two groups, model-based and model-free. Unlike model-free algorithms, model-based ones explicitly learn the model (reward and transition functions) of the system, and then apply planning over it. In this work, the model is known, and consequently, only the planning stage is required.

MDP Solver

There are many reinforcement learning algorithms that aim to learn the MDP model either implicitly or explicitly with the goal of finding a good policy [16]. In this study, the MDP is defined by design, hence there is no need to learn it. In this scenario, finding a good policy is a matter of planning. There are several dynamic programming algorithms that are well suited to this task, such as Value Iteration and Policy Iteration. Value Iteration (VI) was selected for this study.

The Value Iteration core component is called the Value function (V) and represents, for any given state s, the expected cumulative reward from that state s to the goal state, following a certain policy π. Initially, the policy $\pi(s)$ is usually arbitrary, so the Value function $V^{\pi}(s)$ presented in Equation (5) is not a good estimation of the cumulative reward. The aim of Value Iteration is to find a policy that maximizes V, which is known as V^{*} presented in Equation (6).

$$V^{\pi}(s) = E_{\pi}\left[\sum_{k=0}^{H} \gamma^{\,k} R_{k+1} \middle| s_{t=s}\right] \quad (5)$$

$$V^{*}(s) = \max_{\pi}\left[\sum_{k=0}^{H} \gamma^{\,k} R_{k+1} \middle| \pi, s_{t=s}\right] \quad (6)$$

Indeed, it is an iterative algorithm, where $V^{\pi}(s)$ is gradually improved on each iteration by applying the Bellman equation presented as (7).

$$\mathrm{V}(s) = \max_{a} \sum_{s'} P_a(s'|s)[R(s,a,s') + \gamma \times V(s')] = \max_{a}[R(s,a) + V(s')] \quad (7)$$

The General Bellman equation may be simplified for deterministic environments when using additive discounting (a small negative reward for moving to non-terminal states). Under this scenario, the multiplicative discount (discount factor) may be set to 1, and the expression is reduced to its simplest form, as seen in (7).

The algorithm stops when V^{π} converges to V^{*}, since the Value function is not going to improve anymore, that is, the V *matrix* will remain the same on additional iterations. In practice, the algorithm usually stops when the variation in V is less than a certain predefined threshold. At this point, the optimal policy π^{*} can be directly derived from V^{*} and it is presented in Equation (8).

$$\pi^{*}(s) = \underset{a}{argmax}\left[R(s,a) + \gamma \times \sum_{s'} T(s,a,s') \times V^{*}(s')\right] = \underset{a}{argmax}[R(s,a) + V^{*}(s')] \quad (8)$$

Applying the same reasoning as in (7), the discount factor is set to 1 in (8). T is used to model the uncertainty in the environment. The system in this study is deterministic, hence there is only one state s' out of the whole state-space with a probability bigger than 0 (actually, probability 1). Under this scenario, this term can be removed, and the expression is reduced to its simplest form, as seen in (8).

It is worth noting the optimal V^{*} is unique, but there might be multiple π^{*} leading to this optimal Value function. This means that different sequences of actions might lead to the same result. This has a direct impact on this problem, specifically in the situation where a robot has nothing else to do (i.e., there are two robots and only one piece left).

The implementation of this algorithm was written in Python. The simplified Bellman Equation (7) was expressed in matrix form, and the matrix computations were performed using a numerical library called NumPy. Once the Value function has converged, the optimal policy is obtained using the simplified policy Equation (8).

From the point of view of the algorithm, it is irrelevant if one of the robots is idle or randomly moving. In practice, it is usually preferred that this robot keeps idle. To tackle this point, a smart ordering of the system actions was performed (system actions that affect only one robot's movement have priority over those in which both robots move), so in case of multiple actions leading to the same result, the preferred one will be returned by the argmax operation in Equation (8).

4.2. Graph Model

In this approach, the system model was represented as a graph. Specifically, the graph was directly derived from the matrix version of the MDP model (transition and reward functions) described in Section 4.1. A graph basically consists of three types of elements: nodes, transitions, and costs. There is a direct mapping between MDP states and graph nodes, and between MDP state transitions and graph node transitions. Graph costs are derived from the MDP rewards, although the mapping is not direct and specific transformations have been performed. The resulting graph is a reduced version of the MDP model that incorporates the following reward-cost transformations, as shown in Table 1.

Table 1. MDP reward conversion to Graph cost.

MDP	Graph	Meaning
−20	-	Invalid MDP state-action transition
+100	+1	Direct-to-goal transition (very last transition of the path)
−1	+1	Any other transition

All invalid state-action pairs (i.e., an action in a specific state results in exceeding the grid boundaries) have not been included in the graph. The small penalty (−1) reward in the MDP is transformed as a small positive reward (+1), since the target of a graph search algorithm is to minimize the path cost, unlike MDP-based algorithms where their target is to maximize the long-term reward.

Finally, all direct-to-goal transitions in MDP (transitions from state s to s' where s' is a goal state) have the same reward value (+100). The Graph search algorithm's target is to minimize the path cost, and the key point is that every solution (optimal or not) will include one and only one of those direct-to-goal transitions (the very last transition in the path). Based on that property, any arbitrary cost value would be valid for those transitions as long as all of them keep the same value. The cost value chosen for those transitions is +1, resulting in a graph where all costs are the same value (unweighted graph).

Graph Search

There is a wide range of graph search algorithms. Commonly, they are classified as informed search algorithms (i.e., Dijkstra [17] or A* [18]) or uninformed search algorithms (i.e., BFS [19] or DFS [20]). Uninformed search algorithms explore the state space in a blind manner, while informed search algorithms take advantage of additional information (such as the transition cost values) to guide and reduce the state-space exploration.

Given the characteristics of the current graph model, where all the costs have the same value, which is equivalent to say that there is no information at all to guide the search, an informed search algorithm does not provide any advantage over an uninformed search one. Moreover, under this specific scenario, an uninformed search implementation is easier and lighter, resulting in slightly better execution times. The BFS (uninformed search) algorithm was selected for this approach.

BFS is an algorithm used to explore a graph or a tree-like data structure. BFS guarantees to find an optimal solution if the graph is finite and deterministic [21]. The graph representing the system in this work meets those properties. Additionally, it is an unweighted graph, hence the computational cost might be further reduced since the algorithm can be stopped at the depth level where the first solution is found. Any solution found on a deeper depth level would imply a path with more actions involved.

It is worth noting that the common mental representation of a graph solution is the shortest path between two points, seen as a physical distance magnitude. Actually, the path between two nodes may involve multiple concepts other than physical distance. In this study, the shortest path between the node representing the initial state of the system and the node representing the goal state includes all kinds of trajectories carried out by every robot, the optimal assignment of the pieces, the order in which they are processed, etc.

The main language used for the implementation of this study was Python. Generally, a Python application is not as efficient as one written in a compiled language, such as C. On the other hand, Python supports using modules written in other languages, such as NumPy [22], a very optimized numerical library written in C. The Value Iteration algorithm described in Section 4.1 was implemented in Matrix form, taking advantage of the NumPy matrix operations. In order to enable a fair comparison between algorithms, BFS was implemented in C as well, and imported in Python as a module. Its pseudo-code can be seen in Algorithm 3.

Algorithm 3: Pseudo-code to implement Breadth First Search algorithm.

```
1:  BFS (G, s)      // Where G is the graph, and s is the source node
2:      let Q be the FIFO queue.
3:      // Inserting s in queue
4:      Q.enqueue(s)
5:      mark s as visited
6:      // Loop until the queue is empty
7:      while (Q is not empty)
8:          // Remove the first node from the queue
9:          v = Q.dequeue
```

```
10:       // Process all its neighbors
11:       for all neighbors w of v in Graph G
12:          if w is not visited
13:           // Insert w in Q
14:              Q.enqueue(w)
15:        end if
16:       end for
17:    end while
```

4.3. Planning Domain Definition Language

Planning Domain Definition Language (PDDL) is an attempt to standardize Artificial Intelligence (AI) planning languages. PDDL has a variety of versions, such as Probabilistic PDDL (PPDDL) [23] to account for uncertainty, Multi-Agent PDDL (MA-PDDL) to allow planning for multiple agents, etc. In this study, the problem was designed as single-agent and deterministic, so the official PDDL language was used. PDDL models the problem in two separate files: the Domain file and the Problem file. The Domain file contains the predicates and actions; predicates are facts of interest, such as "is *i* robot in XYZ location?", "is there collision in the current state?", and "is *i* robot carrying the piece *j*?", and those predicates are evaluated as true or false; actions are the counterpart of the transition function in the MDP model; that is, the way the system state changes. The Problem file contains the initial state, the goal specification, and the objects; the initial state and goal specification are straightforwardly associated with the initial and goal state in the MDP model; the objects are the counterpart of the internal variables of the MDP model (robot location, robot status, and piece status).

Unlike MDP or graph search methods, this approach ends with the model problem definition. Then a well-proved software planner is used to obtain a solution. Some of the most used planners are FastDownward [24], LPG [25], and PDDL.jl [26]. In this project, FastDownward was used. FastDownward is a planner that relies on a heuristic search to address several planning problems, including PDDL. It is recognized as one of the most efficient planners currently available. Unlike other planners, FastDownward employs an alternative translation known as "multivalued planning tasks" instead of using the propositional representation of PDDL. This approach significantly enhances search efficiency.

5. Results

5.1. Validation by Comparison

RobotStudio® was used to validate the results obtained in this work. This simulator is based on Algoryx [27] and provides high precision in both kinematic and dynamic simulations. In this way, it is possible to simulate a robotic station with a high level of accuracy regarding movements and/or collisions between robots. Therefore, the conclusions reached using this software are considered to faithfully reflect the behavior of the real system. In this project, values and data generated by the simulator were accepted as valid.

5.2. Comparison of Results

To comparatively assess the results obtained from the proposed algorithms, the following data have been considered: the number of steps performed by the solution provided by each algorithm required off-line resources (RAM and disk storage) and on-line computational cost (associated with scalability and implementation feasibility in a production line).

The initial comparison among algorithms focuses on the number of steps or movements that algorithms need to perform to solve the task. This is the factor to be minimized as it determines how fast the task is performed. The number of pieces used in each

industrial operation can significantly vary. Some assemblies only require a few pieces to obtain the intermediate or final product, while in other cases, hundreds of pieces are needed. This work is focused on SimplicityWorks Europe manufacturing process [5], where the company carries out pick-and-place operations without any specific order constraints. The goal is to minimize the task time by selecting the best sequence for placing between 4 and 9 pieces. Table 2 shows a comparison between solutions provided by several algorithms: Value Iteration (VI), Breadth First Search (BFS) and Planning Domain Definition Language (PDDL). Multiple tests for different scenarios (number of pieces and action mode) have been conducted. Initial conditions for each test are the same for every algorithm, including the source and destination location of the pieces and the initial location of the robots. The primary observation indicates that every algorithm reaches a solution with the same number of steps. This makes sense as all three algorithms are able to find an optimal solution to the problem. Those solutions may be different but with the same level of quality. The second point to note is that the PDDL algorithm scales better than the others, being the only one able to solve all tests. Finally, both VI and BFS solve the same test cases. This is because both of them rely on the MDP model, which does not scale well when the number of pieces and actions increases.

Table 2. Number of steps required to complete the pick-and-place task using two robotic arms.

NUMBER OF STEPS				
NAVIGATION MODE	**# OF PIECES**	**VI**	**BFS**	**PDDL**
1	2	28	28	28
	4	46	46	46
	6	55	55	55
	8	68	68	68
	9	77	77	77
	10	-	-	88
2	2	25	25	25
	4	37	37	37
	6	46	46	46
	8	56	56	56
	9	-	-	67
	10	-	-	75
3	2	25	25	25
	4	37	37	37
	6	46	46	46
	8	-	-	56
	9	-	-	67
	10	-	-	75
4	2	24	24	24
	4	35	35	35
	6	-	-	44
	8	-	-	56
	9	-	-	66
	10	-	-	73

The number of steps required for a set of robots to manipulate N pieces depends on their initial and final locations. This information is accessible just before the task starts. At that moment, the MDP must be updated (for all state-action pairs where the action is pick or drop), and after that, the solution for this particular MDP must be found. This whole process is carried out during the process, hence it is called "online computation" in this work. There is only a few seconds or minutes available in the best case scenario. Table 3

shows the time required for each algorithm to find a solution. The BFS algorithm is clearly the fastest whenever it can provide a solution. The comparison between BFS and PDDL is more intriguing. There is not a clear winner in all test scenarios. As a reference, VI is slower than PDDL in the test case "Mode 1 and 9 pieces" (5 min 57.5 s vs. 1 min 15.2 s), but it is faster in the test case "Mode 4 and 4 pieces" (6 min 27.4 s vs. 15 min 37.3 s).

Table 3. Online computation time of each algorithm for a different number of pieces and different Navigation Modes.

ONLINE TIME				
NAVIGATION MODE	**# OF PIECES**	**VI**	**BFS**	**PDDL**
1	2	0.4 s	0.001 s	15 s
	4	4.6 s	0.016 s	16 s
	6	39 s	0.125 s	19 s
	8	4 min 37 s	0.973 s	36 s
	9	5 min 57 s	2.918 s	1 min 15 s
	10	-	-	2 min 56 s
2	2	0.8 s	0.016 s	39 s
	4	10 s	0.047 s	41 s
	6	1 min 26 s	0.267 s	47 s
	8	9 min 45 s	1.851 s	1 min 20 s
	9	-	-	2 min 38 s
	10	-	-	5 min 49 s
3	2	8.2 s	0.032 s	3 min 39 s
	4	1 min 26 s	0.376 s	4 min 02 s
	6	11 min 18 s	2.907 s	5 min 53 s
	8	-	-	16 min 08 s
	9	-	-	36 min 54 s
	10	-	-	1 h 29 min 24 s
4	2	37 s	0.110 s	14 min 30 s
	4	6 min 27 s	1.146 s	15 min 37 s
	6	-	-	22 min 19 s
	8	-	-	1 h 27 min 18 s
	9	-	-	2 h 14 min 46 s
	10	-	-	5 h 24 min 10 s

Another important issue is the offline resources required to calculate the VI and BFS algorithms described in Tables 2 and 3. Both algorithms use the MDP architecture described in Section 4.1, which describes the number of states and their behavior according to the actions taken. The number of states of the MDP can be calculated according to Equation (9), where ξ_σ is the number of states and N is the number of pieces.

$$\xi_\sigma = ((M \cdot N \cdot Z) + (K \cdot P))^2 \cdot (2 + T)^K \cdot (K + 1)^2 \quad (9)$$

where M, N, and Z are the number of rows, columns, and heights of the grid, K is the number of pieces to be picked, T is the number of possible intermediate positions, and P is the number of states required for the pick and drop operations. Intermediate positions (T) are useful in case a piece cannot be transported by any robot, due to accessibility problems, either on the source or destination location of the piece. In this case, the piece can be partially transported by a robot up to a certain intermediate position, and then another robot can complete the transportation. These kinds of problems are represented in Figure 3.

Table 4 shows the computational cost (time) and disk resources and memory required to build the MDP. This computation is performed only once for a specific problem topology (e.g., a certain grid size and number of pieces). This is particularly relevant because

the MDP may take several hours to be calculated on a medium-power computer such as the one used in this project (e.g., 4 h 39 min and 42 s to calculate an MDP in Navigation Mode 4 with four pieces to be manipulated). In addition, the MDP size on disk exceeded 5GB in some test cases. Actually, this size grew as the numbers of pieces increased.

Table 4. Offline computational cost (time), DISK memory used to store the state-transition table (GB), RAM memory necessary (MB), number of states of the MDP, and number of brute states of the MDP.

STATE-TRANSITION TABLE (MDP)						
NAVIGATION MODE (# ACTIONS)	**# PIECES**	**TIME**	**DISK (GB)**	**RAM (MB)**	**# MDP STATES**	**# BRUTE STATES**
1 (49)	2	12 s	0.004	8	13,568	104,976
	4	2 min 06 s	0.043	81	138,240	1,345,600
	6	15 min 57 s	0.328	619	1,052,672	12,054,784
	8	1 h 43 min 58 s	2.142	4.037	6,864,896	90,326,016
	9	4 h 28 min 04 s	5.265	9.923	16,875,520	236,748,800
	10	-	-	-	-	607,129,600
2 (121)	2	36 s	0.010	20	13,568	104,976
	4	6 min 04 s	0.103	201	138,240	1,345,600
	6	45 min 46 s	0.783	1.523	1,052,672	12,054,784
	8	4 h 49 min 30 s	5.107	9.968	6,864,896	90,326,016
	9	-	-	-	-	236,748,800
	10	-	-	-	-	607,129,600
3 (169)	2	05 min 42 s	0.090	28	13,568	853,776
	4	54 min 58 s	0.863	280	138,240	9,985,600
	6	6 h 47 min 04 s	6.383	2.135	1,052,672	82,301,184
	8	-	-	-	-	571,401,216
	9	-	-	-	-	1,445,068,800
	10	-	-	-	-	3,580,825,600
4 (841)	2	28 min 40 s	0.444	137	13,568	853,776
	4	4 h 39 min 42 s	4.235	1.395	138,240	9,985,600
	6	-	-	-	-	82,301,184
	8	-	-	-	-	571,401,216
	9	-	-	-	-	1,445,068,800
	10	-	-	-	-	3,580,825,600

There are empty cells on some test cases for all the columns except for the BRUTE STATES because either the computer ran out of memory when solving the MDP or it could not be done in a reasonable amount of time. BRUTE STATES refer to the number of states in the state-space according to Equation (9). However, there is a considerable subset of those states that are not valid data (i.e., states where the robots collide, a robot is not able to reach a specific grid location). The MDP states are the number of actual states used in the MDP after filtering invalid states in the original BRUTE state-space. This information is provided by the software when building the MDP. The number of MDP states exponentially increases with the number of pieces, and the other variables (disk, RAM, and time) depends on this size, hence they exponentially increase as well.

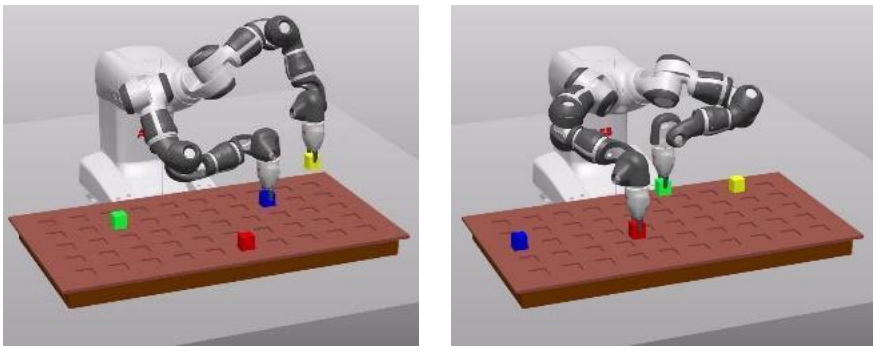
Disk resource information required to store the MDP is directly obtained from the filesystem. The time and MDP states columns are provided by the software when building the MDP. The RAM required is computed based on the number of states, the number of actions, and the number of bytes needed to encode the MDP information. In this implementation, the state value is represented by a 32-bit value and the reward by a 16-bit value, making a total of 6 bytes per state-action pair. Additionally, during the Value Iteration algorithm execution, the implementation keeps a copy of the Value function in the previous time-step in order to be able to check if it converges.

The PC used in this research is an Intel Core i7-12700KF 12th Gen 3.60Ghz, with 64GB of RAM, 1TB SSD, a Nvidia Quadro P100 graphics card, and a Windows 11 Pro 22H2 operating system.

5.3. Data Validation

The results presented in this section show the behavior of the algorithms being compared (VI, BFS, and PDDL). For this purpose, RobotStudio® software was used. It provides a graphical representation of the setup, including a dual arm collaborative robot ABB IRB 14,000 (also called YuMi), multiple pieces, and a worktable. It also monitors the whole simulation, detecting any fault, such as collisions.

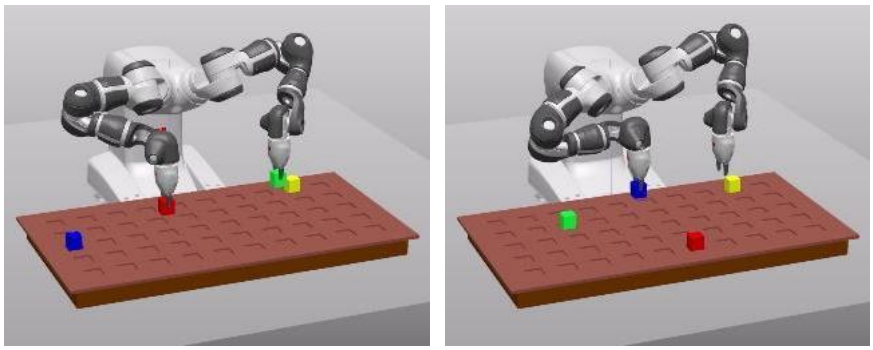
The first simulation involves a YuMi robot only supporting orthogonal movements in a horizontal plane (except for the pick and place operations). This corresponds to Navigation Mode 1 (see Section 3.3.2). Figure 9 shows two frames of the simulation, and a demonstrative video is available on https://www.youtube.com/watch?v=3ysBDbBr5mQ (accessed on 6 June 2023). This approach offers a collision-free optimized solution to the manipulation problem, as the robot is capable of completing the task in 46 steps. It is worth noting that the robot's speed for each movement is configured to take approximately 0.5 s per transition (time to move from one state to the next one). This means that the entire task is completed in 23 s.



(**a**) (**b**)

Figure 9. Simulation of a pick-and-place operation with Mode 1 movements. (**a**) t = 9 s; (**b**) t = 22 s.

Figure 10 shows another simulation of the same pick-and-place process. In this case, the robot supports both orthogonal and diagonal movements in the horizontal plane (Navigation Mode 2 described in Section 3.3.2). By giving the system the ability to diagonally move, the robot can combine two movements into one and reduce the number of steps. The simulation can be viewed on https://www.youtube.com/watch?v=3ysBDbBr5mQ (accessed on 6 June 2023). The number of steps is lower than in the previous case, even though the initial and final positions of pieces are the same. In this case, the task is completed in 37 steps, taking 18.5 s. It confirms that with a more flexible combination of movements, the algorithm finds a better solution to complete the task.

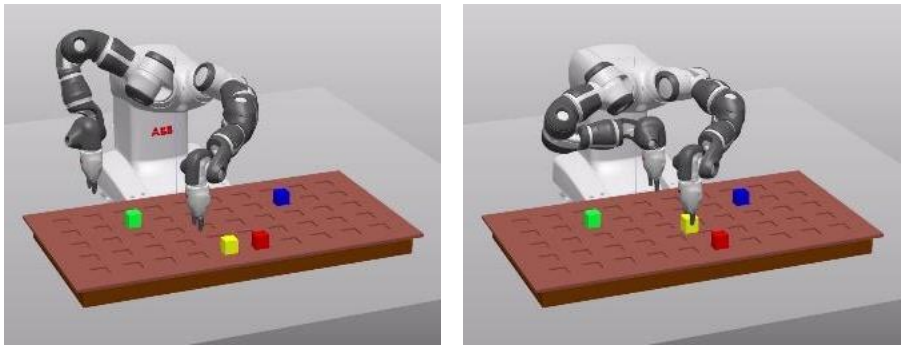


(**a**) (**b**)

Figure 10. Simulation of a pick-and-place operation with Navigation Mode 2 movement. (**a**) t = 10 s; (**b**) t = 19 s.

A third simulation is presented, where the robot supports moving through a 3D virtual grid, enabling the arm's intersection without collision by operating on different planes. This configuration corresponds to Navigation Mode 4 described in Section 3.3.2. The objective of this simulation is to check if the algorithm finds a solution with a lower number of steps. The full video is available at https://www.youtube.com/watch?v=kqKprKYazKk (accessed on 6 June 2023). In this case, the task is completed in 35 steps, taking 17.5 s for the robot to place all pieces in their final positions. Once again, with a larger set of movements, the algorithm finds a better solution for the task than in previous cases. On this occasion, arms have the ability to cross each other to minimize the operation time, avoiding collisions.

Figure 11 shows a set of frames extracted from this simulation, representing a couple of situations where the algorithm takes advantage of both arms moving on different planes. The first situation, shown in frames (a), (b) and (c) of Figure 11 involves the left arm picking up and placing the first piece (yellow), and the right arm picking up the second piece (blue). In this sequence, the left arm transitions to a higher plane, allowing the right arm to approximate the blue piece without interrupting the left arm trajectory. In the second situation, shown in frames (c), (d) and (e) of Figure 11, the algorithm finds optimal to move the right arm to a higher plane in order to maintain its course without collision. At t = 14 s. (Figure 11d), the right arm is about to pick up the blue piece and the left arm is approximating the green piece. After that, the right arm starts moving towards the red piece. At t = 16 s. (Figure 11e), the right arm is moving at a higher plane than the left arm to avoid a collision. After that, both arms progress in parallel in the same direction. At t=20sec. (Figure 11f), the left arm is still transporting the green piece while the right arm has already picked up the red piece and is moving toward its destination.



(**a**) (**b**)

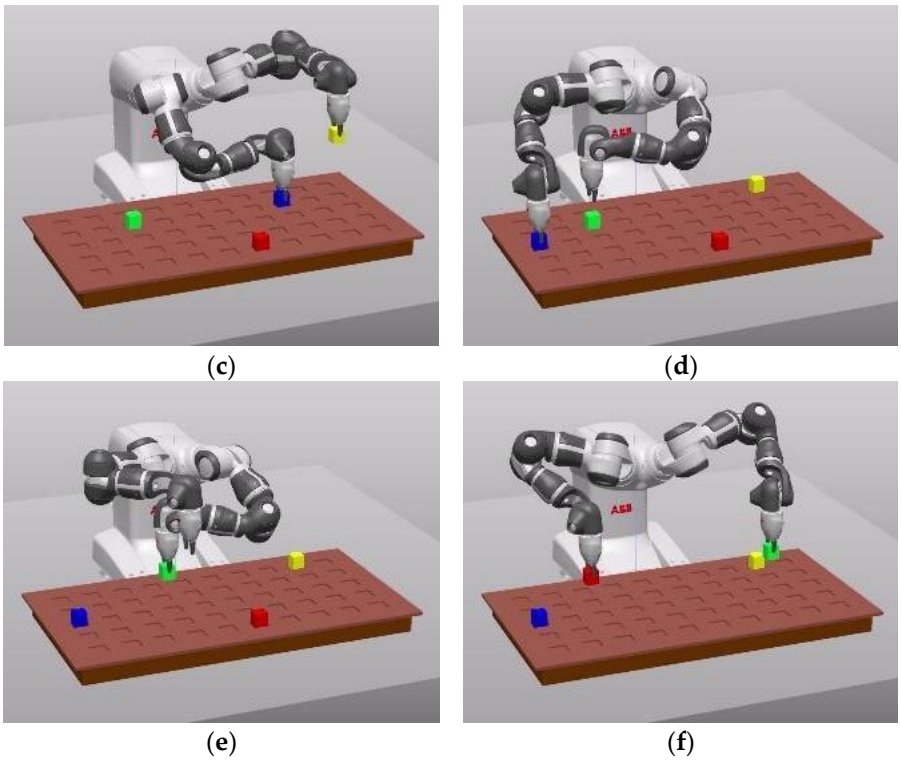
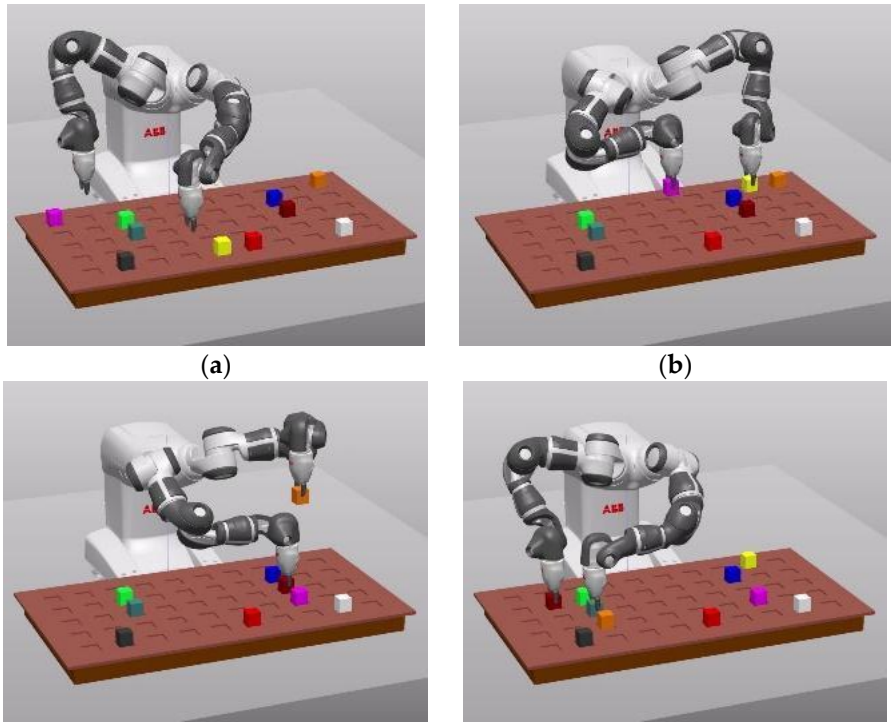


Figure 11. Simulation of a pick-and-place operation with Navigation Mode 4 movements (movements in 3 different heights). (**a**) t = 3 s; (**b**) t = 6 s; (**c**) t = 8 s; (**d**) t = 14 s; (**e**) t = 16 s; (**f**) t = 20 s.

Finally, a new simulation is presented. The robot still supports moving through the virtual 3D grid, but the number of pieces is increased to 10. In this case, the task is successfully performed in 73 steps and a total time of 36.5 s. Note that in frames (c), (f), and (h) of Figure 12, both arms are crossing, which is possible because the algorithm places them at different heights. As seen in Table 2, only the PDDL algorithm is capable of finding a solution for such a high number of pieces, scaling the problem much better than the VI and BFS options to solve the proposed MDP.



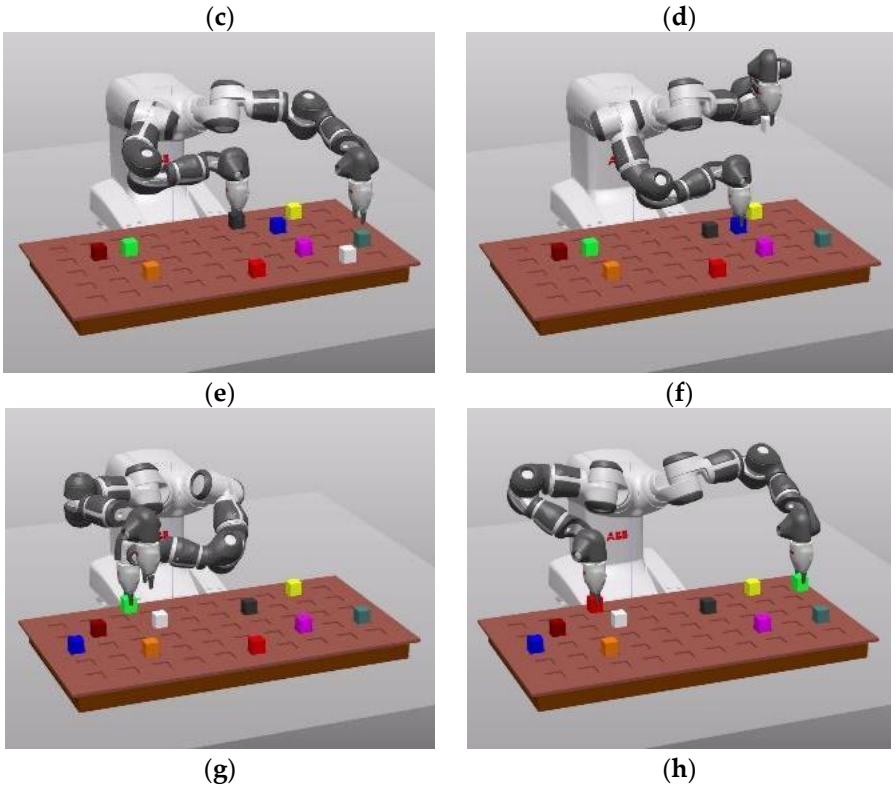


Figure 12. Simulation of a pick-and-place operation solved with the PDDL algorithm for 10 pieces. (**a**) t = 3 s; (**b**) t = 8 s; (**c**) t = 12 s; (**d**) t = 18 s; (**e**) t = 24 s; (**f**) t = 28 s; (**g**) t = 35 s; (**h**) t = 40 s.

A comprehensive collection of simulations, classified per algorithm, can be found at the following links: Value Iteration (https://www.youtube.com/watch?v=cXEfuw8WPqA, accessed on 8 November 2023), BFS (https://www.youtube.com/watch?v=2T0z_3_9az8, accessed on 8 November 2023), and PDDL (https://www.youtube.com/watch?v=vV5W80SKIOo, accessed on 8 November 2023). Additionally, at https://www.youtube.com/watch?v=jl47uA9LN_w (accessed on 8 November 2023) is presented a test case where both arms must cooperate to transport one of the pieces. In this case, one of the arms can only access the initial location of the piece, and the other one can only access its destination location. The first arm leaves the piece in an intermediate position so that the second one can finish the subtask. In this case, the approach proposed can solve the problem presented in Figure 3, where two or more robots should coordinately collaborate between them and in the minimum time.

6. Conclusions and Future Work

This article presents a method to coordinate robotic manipulators, avoiding collisions between arms during pick-and-place operations, and minimizing the task time by selecting the best manipulation sequence and the best trajectory planning. Due to the assumed simplifications in the system (deterministic, single-agent, and fully observable), the Markov Decision Process (MDP) proposed can be transformed into a graph that can be solved using a search algorithm, such as Breadth First Search (BFS). Additionally, the Planning Domain Definition Language (PDDL) has been used to generate an alternative solution that can be compared with VI and BFS.

For every test case where a solution can be obtained, all three algorithms consistently produce an optimal solution, which is not unique and may differ between all algorithms. In this context, optimal refers to the least number of steps required to complete the task. Regarding the computational cost to find a solution, the graph search method (BFS) turns out to be the fastest one, although limited in terms of scalability. In fact, the generated

graph is derived from the MDP model, so actually both MDP and BFS approaches suffer this same scalability issue, directly related to the number of pieces and robots. On the other hand, the PDDL scales better and is able to solve all tested cases. The MDP approach is neither the fastest nor the least eager on resources, but provides a general framework capable of scaling to more complex problems, including uncertainty, the continuous state and action spaces, the partially observable state, etc. The last factor, irrespective of the algorithm used, is the flexibility of the robot's navigation. A significant enhancement of the solution is observed when diagonal movements are incorporated into the 2D plane (as outlined in Mode 2 in Section 3.3.2). While the inclusion of extra movements for 3D space navigation results in a marginal improvement in the solution, it comes at a substantial expense due to the computational cost being exponential to the number of robots (in this case, quadratic, as there are two robots).

However, the proposed method has several disadvantages. Firstly, the system does not scale well. The provided solution is feasible for certain production processes where the number of pieces to manipulate is small (less than 10 elements). Secondly, the movements that the robots must perform are constrained within a grid, which is less natural compared to other solutions such as [15], although it provides a robust and faster industrial solution to the problem. Lastly, a preliminary study using RobotStudio® is required to create a database, including information about collision situations between arms.

In future work, the intention is to explore new techniques, such as hierarchical reinforcement learning or multi-agent learning, that can overcome the scalability limitation of the current single-agent approach. The goal is to introduce a larger number of pieces and robots that could operate within the same workspace. Another objective is to investigate alternative implementation strategies, such as using specialized HW (GPU), since the algorithm (VI) core is heavily based on matrix operations. Additionally, the preliminary study of robot collisions demands considerable time and complexity. Therefore, an alternative method is being considered. A possible solution is to frame the robot links within a set of ellipsoids and check for collisions between these ellipsoids in space. This approach would eliminate the need to rely on a graphical environment such as RobotStudio®, reducing offline computational costs but potentially increasing online computational requirements. Hence, an analysis of the performance of this new approach would be required.

Author Contributions: Conceptualization, D.M.-G., F.J.M.-P. and C.P.-V.; methodology, D.M.-G. and C.P.-V.; software, D.M.-G. and F.J.M.-P.; validation, D.M.-G. and F.J.M.-P.; formal analysis, C.P.-V.; investigation, D.M.-G. and F.J.M.-P.; resources, D.M.-G. and F.J.M.-P.; writing—original draft preparation, D.M.-G., F.J.M.-P. and C.P.-V.; writing—review and editing, D.M.-G., F.J.M.-P. and C.P.-V.; visualization, F.J.M.-P.; supervision, C.P.-V.; project administration, C.P.-V.; funding acquisition, C.P.-V.; All authors have read and agreed to the published version of the manuscript.

Funding: This research has been partly funded by project CPP2021-008593, grant MCIN/AEI/10.13039/501100011033 and by the European Union-NextGenerationEU/PRTR.

Data Availability Statement: All files created can be found in the following Git repository: https://github.com/da-mago/Two-arms-pick-place-planning (accessed on 8 November 2023).

Acknowledgments: This project would not have been possible without the help of ABB Spain, which has provided software licenses and hardware.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hermann, J.; David, A.; Wagner, A.; Ruskowski, M. Considering interdependencies for a dynamic generation of process chains for production as a service. *Procedia Manuf.* **2020**, *51*, 1454–1461. https://doi.org/10.1016/j.promfg.2020.10.202.
2. Ruskowski, M.; Herget, A.; Hermann, J.; Motsch, W.; Pahlevannejad, P.; Sidorenko, A.; Bergweiler, S.; David, A.; Plociennik, C.; Popper, J.; et al. Production Bots fur Production Level Skill-basierte Systeme fur die Produktion der Zukunft. *Atp Mag.* **2020**, *62*, 62–71. https://doi.org/10.17560/atp.v62i9.2505.
3. Wrede, S.; Beyer, O.; Dreyer, C.; Wojtynek, M.; Steil, J. Vertical integration and service orchestration for modular production systems using business process models. *Procedia Technol.* **2016**, *26*, 259–266. https://doi.org/10.1016/j.protcy.2016.08.035.
4. Sellers, C.J.; Nof, S.Y. Performance analysis of robotic kitting systems. *Robot. Comput.-Integr. Manuf.* **1989**, *6*, 15–24. https://doi.org/10.1016/0736-5845(89)90081-1.
5. Méndez, J.B.; Perez-Vidal, C.; Heras, J.V.S.; Pérez-Hernández, J.J. Robotic Pick-and-Place Time Optimization: Application to Footwear Production. *IEEE Access* **2020**, *8*, 209428–209440, https://doi.org/10.1109/ACCESS.2020.3037145.
6. He, T.; Wang, H.; Yoon, S.W. Comparison of Four Population-Based Meta-Heuristic Algorithms on Pick-and-Place Optimization. *Procedia Manuf.* **2018**, *17*, 944–951. https://doi.org/10.1016/j.promfg.2018.10.112.
7. Ye, X.; Zhang, Y.; Ru, C.; Luo, J.; Xie, S.; Sun, Y. Automated Pick-Place of Silicon Nanowires. *IEEE Trans. Autom. Sci. Eng.* **2013**, *10*, 554–561. https://doi.org/10.1109/TASE.2013.2244082.
8. Gao, J.; Zhu, X.; Liu, A.; Meng, Q.; Zhang, R. An Iterated Hybrid Local Search Algorithm for Pick-and-Place Sequence Optimization. *Symmetry* **2018**, *10*, 633. https://doi.org/10.3390/sym10110633.
9. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95—International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948. https://doi.org/10.1109/ICNN.1995.488968.
10. Kumar, M.; Husian, M.; Upreti, N.; Gupta, D. Genetic Algorithm: Review and Application. *Int. J. Inf. Technol. Knowl. Manag.* **2010**, *2*, 451–454. http://dx.doi.org/10.2139/ssrn.3529843.
11. Dorigo, M.; Birattari, M.; Stutzle, T. Ant colony optimization. *IEEE Comput. Intell. Mag.* **2006**, *1*, 28–39. https://doi.org/10.1109/MCI.2006.329691.
12. Alazzam, A.R. Using BUA algorithm to solve a sequential pick and place problem. In Proceedings of the 2018 International Conference on Information and Computer Technologies (ICICT), DeKalb, IL, USA, 23–25 March 2018; pp. 144–149. https://doi.org/10.1109/INFOCT.2018.8356858.
13. Daoud, S.; Chehade, H.; Yalaoui, F.; Amodeo, L. Efficient metaheuristics for pick and place robotic systems optimization. *J. Intell. Manuf.* **2014**, *25*, 27–41. https://doi.org/10.1007/s10845-012-0668-z.
14. Gafur, N.; Kanagalingam, G.; Ruskowski, M. Dynamic collision avoidance for multiple robotic manipulators based on a non-cooperative multi-agent game. *arXiv* **2021**. https://doi.org/10.48550/arXiv.2103.00583.
15. Gafur, N.; Kanagalingam, G.; Wagner, A.; Ruskowski, M. Dynamic Collision and Deadlock Avoidance for Multiple Robotic Manipulators. *IEEE Access* **2022**, *10*, 55766–55781, https://doi.org/10.1109/ACCESS.2022.3176626.
16. AlMahamid, F.; Grolinger, K. Reinforcement Learning Algorithms: An Overview and Classification. In Proceedings of the 2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), Virtually, 12–17 September 2021; pp. 1–7. Available online: https://doi.org/10.48550/arXiv.2209.14940.
17. Javaid, A. Understanding Dijkstra's algorithm. *SSRN Electron. J.* **2013,** *10*, 1-27. https://dx.doi.org/10.2139/ssrn.2340905.
18. Foead, D.; Ghifari, A.; Kusuma, M.B.; Fiah, N.H.; Gunuwan, E. A systematic literature Review of A* PathFinding. In Proceedings of the 5th International Conference on Computer Science and Computational Intelligence 2020, Online, 19–20 November 2020. https://doi.org/10.1016/j.procs.2021.01.034.
19. Gao, P.; Liu, Z.; Wu, Z.; Wang, D. A Global Path Planning Algorithm for Robots Using Reinforcement Learning. In Proceedings of the 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO), Dali, China, 6–8 December 2019. https://doi.org/10.1109/ROBIO49542.2019.8961753.
20. Paulino, L.; Hannum, C.; Varde, A.S. Search Methods in Motion Planning for Mobile Robots. In Proceedings of the Intelligent Systems and Applications: Proceedings of the 2021 Intelligent Systems Conference (IntelliSys), Amsterdam, The Netherlands, 2–3 September 2021. https://doi.org/10.1007/978-3-030-82199-9_54.
21. Sadik, A.M.J.; Dhali, M.A.; Farid, H.M.A.B.; Rashid, T.U.; Syeed, A. A Comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory. In Proceedings of the 2010 International Conference on Artificial Intelligence and Computational Intelligence, Sanya, China, 23–24 October 2010; pp. 52–56, https://doi.org/10.1109/AICI.2010.18.
22. Shen, G.; Liu, Q. *Performance Analysis of Linear Regression Based on Python*; En Cognitive Cities, IC3 2019, CCIS 1227; Springer Nature: Singapore, 2020; pp. 695–702. https://doi.org/10.1007/978-981-15-6113-9_80.
23. Younes, H.L.S.; Littman, M.L. PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162* **2004**, *2*, 99.
24. Helmert, M. The fast downward planning system. *J. Artif. Intell. Res.* **2006**, *26*, 191–246. https://doi.org/10.1613/jair.1705.

25. Gerevini, A.; Saetti, A.; Serin, I. Planning through stochastic local search and temporal action graphs in LPG. *J. Artif. Intell. Res.* **2003**, *20*, 239–290. https://doi.org/10.1613/jair.1183.
26. Tan, Z.-X. PDDL.jl: An Extensible Interpreter and Compiler Interface for Fast and Flexible AI Planning. Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 2022.
27. Backman, A. Algoryx—interactive physics. In Proceedings of the SIGRAD 2008, the Annual SIGRAD Conference Special Theme: Interaction, Stockholm, Sweden, 27–28 November 2008; Linköping University Electronic Press: Linköping, Sweden, 2008; pp. 87–87.

Dual-Arm Coordination for Mutual-Collision-Avoidance to Implement a Pick-on-the-Fly Application

Abstract:

This article tackles the challenge of automating the pick-and-place operation of multiple moving objects by a dual-arm robot avoiding collisions between manipulators minimising the execution time. The system is able to determine the sequence of actions to be undertaken. Leveraging its versatility in handling decision-making scenarios, the system is framed within the mathematical framework of Markov Decision Process (MDP). To do so, the workspace has been discretized to get a finite set of states connected by a set of actions. In this study, the model is tailored to suit a deterministic setup, where a single agent oversees the entire operation and enjoys full visibility into the system. The proposed methodology offers several advantages: it devises optimal trajectories for completing tasks satisfactorily, accounts for collision avoidance between manipulators and automatically generates robot code compatible with various manufacturers. The outcome fulfils the outlined objectives, yielding a rapid and robust solution.

Authors:

Daniel Mateu-Gomez[a] ; Francisco José Martínez-Peral*[b] ; and Carlos Perez-Vidal[b]

[a] Analog Devices Spain, Parc Cientific Universitat de Valencia, C/ Catedratico Agustin Escardino, 9, Paterna, 46980, Valencia, Spain;

[b] Instituto de Investigación en Ingeniería I3E, Miguel Hernández University, Av. de la Universidad s/n, Elche, 03202, Alicante, Spain;

* Corresponding author: francisco.martinez74@goumh.umh.es

Keywords:

Pick-and-place operations; Robotic sequence order Optimization; Dual-arm collision avoidance; Markov Decision Process; PID Controller; Pick-on-the-fly pieces;

1. Introduction

Robotic manipulation processes are evolving to accommodate shorter and more personalised production runs in the manufacturing industry [1], manipulation of elements in the pharmaceutical industry [2] or the automation of laboratories for biological samples analysis [3][4], amongst other applications. These new applications impact on how automation and robotics can be integrated in more complex environments. Moreover, robots are evolving from inflexible, pre-programmed machines with limited sensory capabilities to machines with flexible programs and equipped with advanced sensors. This transformation enables them to effectively participate in tasks such as assembly, disassembly, manipulation of hazard or fragile products, amongst others [5], partly thanks to collaborative robots [6] and redundant robots, with more than 6 Degrees of Freedom (DOF) [7]. In addition, cooperation between different robots further enhances productivity while minimising spatial requirements. However, this cooperation introduces challenges in collision avoidance [8][9], added to redundancy resolution. Figure 1 illustrates a scenario where two collaborative and redundant robots (or a two arms robot of this kind) work in a shared workspace to complete a pick-and-place task efficiently, ensuring that each element is placed in his designated position as quickly as possible.

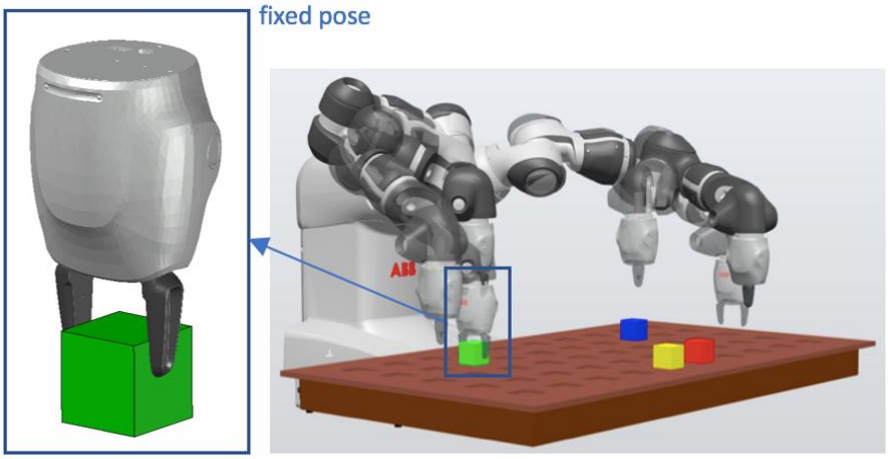


Figure 1. Pick-and-place scenario with a two arms ABB IRB 14000 robot

Dual robotic manipulators are becoming increasingly popular as they can perform cooperative tasks [10][11], such as working in an automated order picking system [12] and assisting elderly and disabled people with dressing [13]. In this context, there are some challenges in the fields of dual redundant robotic manipulator operation: 1) Redundancy Resolution Problem: It is difficult to generate the joint trajectories to the desired path in real time and there exist innumerable solutions to joint trajectories and; 2) Real-Time Mutual-Collision-Avoidance Strategy: When the dual robot manipulators are executing an end-effector movement, a problem that cannot be negligible is the mutual-collision.

Usually, robots' trajectories are programmed to ensure they do not collide with a static environment. Since robots often handle repetitive tasks, planning collision-free trajectories once is usually enough. However, if the positions of production elements change, re-planning collision-free trajectories is required for all manipulators. Another challenge arises when trying to minimise operation time or enhance process efficiency [14][15]. Moreover, the movement of objects, common in industries using conveyor belts, introduces a special challenge into the system. The environment includes moving objects, and each robot should pick one of them at a time, avoiding collisions with other robots that will be trying to pick a different moving object in the same workspace.

To solve these problems, an innovative approach is necessary. By retrieving the literature, there are no references considering mutual-collision-avoidance for manipulating moving objects, commonly known as, on-the-fly manipulation. Due to the effectiveness of the proposed approach based on the Markov Decision Problem (MDP) model, the dual arm robot can efficiently solve the manipulation task. The main contributions of this article are described as follows:

1) The system can prevent the two robots from colliding in a shared workspace by pre-examining the robot configurations in a discretized representation of this workspace;
2) The proposed approach considers the movement of objects to be manipulated. No references considering manipulation of moving objects by dual-arm robots have been found in bibliography and this is the main novelty of this research;
3) By representing the system as a Markov Decision Problem (MDP), it's possible to find an optimal solution that completes task with the least number of actions;
4) The redundancy issue arising from the 7 DOFs in each YuMi arm, is effectively addressed by utilising a Look-Up Table (LUT), that captures distinct joint positions across the entire discretized workspace; and
5) The suggested approach has the potential to be applied to other robot models because its architecture builds upon the fundamental capabilities shared by all robots.

This article is structured as follows. In Section 2, a brief analysis of the current state of manipulating moving objects is provided. Following that, Section 3 presents the architecture developed in this research and explains how this architecture is incorporated within the MDP framework. Section 0 is devoted to tackle the problem framed as an MDP model. The findings of the research are outlined in Section 5, highlighting the algorithm's efficacy through data and visual demonstrations. Finally, Section 6 offers conclusions and outlines prospective endeavours in this subject.

2. Related work

To address the problem of potential collisions between the arms of a bimanual robot, it is necessary to consider a specific approach. In recent years, many studies have been conducted to prevent collisions between robotic manipulators, all of which require measuring the minimum distance between manipulators. The calculated minimum distance is used at each sample period to achieve real-time control and avoid collisions, requiring fast distance calculation. Some approaches compute the distance using the relationship between two straight lines in different planes [16], but this method entails a high computational cost. Other works propose a simplification of the earlier case to measure the minimum distance between two-line segments representing robot links. This method can provide the distance quickly, but considering robots as a set of lines generates an imprecise value of distance, dismissing the actual shape of robot links. On the other hand, links can be considered as ellipsoids, as done in [9], reaching a compromise between precision and computational cost, but the issues of operation time optimization and redundancy resolution remain.

The use of Artificial Potential Fields (APF) has been established as an effective technique for preventing collisions between robots [17]-[19]. Khatib et al. [20] developed a real-time obstacle avoidance method based on creating an APF, applicable to both manipulators and mobile robots, thus enabling operations in complex environments. On the other hand, Volpe et al. [21] introduced a superquadratic potential function capable of generating repulsive forces to avoid obstacles. Despite their advantages, these APF methods face limitations in handling several forms of obstacles and require intensive computations, especially in scenarios with three-dimensional obstacles. Therefore, it is concluded that these approaches are more suitable for mobile robots and non-redundant manipulators than for redundant manipulators. Zhang et al. [16] proposed a strategy based on a quadratic programming (QP) problem that integrates the minimum velocity norm, desired trajectory tracking, and obstacle avoidance, then solve the QP problem using a recurrent neural network (RNN). This approach helps to reduce the computational cost associated with calculating the pseudoinverse. On the other hand, Guo et al. [22] made improvements to Zhang et al. scheme by adjusting certain parameters in the obstacle avoidance constraint to reduce discontinuous velocity changes. Other researchers [23]-[25] proposed schemes to prevent collisions. Sun et al. [23] introduced a method to calculate the collision probability of a manipulator's motion planner, considering Gaussian motion assumptions and detection with uncertainty.

Oh et al. [24] proposed an analytical inverse kinematics solution considering joint limits and self-collision avoidance for a 7-DOF manipulator with spherical shoulder and wrist joints. However, this work only focuses on a single redundant manipulator. This approach could be extended to two arms of a bimanual manipulator, but its computational cost would be extremely high. Nicolis et al. [26], among other works, proposed an RNN to achieve synchronous motion of a multi-arm robot for cooperative manipulation. In this case, all manipulators grasp the same object, so that the movement of each one is constrained by the others and vice versa. This problem is more common in the state of the art, and several references can be found in this sense [27][28]. However, in this article, a cooperative problem is presented where the end effectors are not connected by the piece to be manipulated, but each one of them manipulates a piece, similarly to [8] and [9]. In these cases, pieces are static, and the main difference between them is that [9] provides an adequate (continuous)

solution, and [8] provides an optimal (discretized) solution in terms of its operation time. Regarding moving pieces, no references have been found in the literature that address this issue beyond the known conveyor tracking systems for handling pieces on transport belts used in the industry.

In this regard, Yang et al. [29] developed a system to perform pick-and-place on an airport conveyor belt to classify luggage according to its destination. In [30], Anton et al. presented a comparative analysis of the performance and cost of three different methods for executing pick-on-the-fly applications: Using only a camera; Using a camera and an encoder and; Using a camera and a laser-based sensor. The experiments showed that the use of encoders offers superior performance at a low cost. On the other hand, in [31], Dong Ku et al. develop a system to achieve greater efficiency in the classification of Construction and Demolition Waste (CDW), achieving high productivity but where the workspaces of the robots are not overlapped.

3. Approach to This Research

3.1. Materials

The setup used in this project comprises a 2x7 DOFs collaborative bimanual robot, the ABB YuMi IRB 14000, an industrial conveyor belt, a group of 5 different types of pieces to be classified and a Line Scan Camera (LSC) [32], the JAI Sweep+ Series LT-200-CL. This approach is particularly interesting because a hyperspectral camera can be added to the system to recognize objects based on their material, not their colour. This opens up a new range of applications in the sorting of materials for recycling or simply for the recovery of objects.

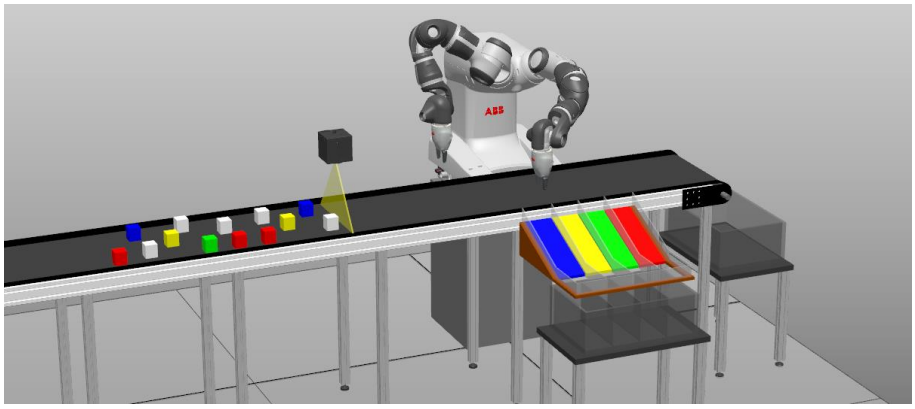


Figure 2. Project set-up to perform Dual-Arm Coordination for Mutual-Collision-Avoidance Pick-on-the-Fly Objects

A LSC in an industrial conveyor belt captures images by scanning objects line by line as they move past the camera. This method allows for continuous imaging of objects, ensuring high-speed image acquisition. The captured image data is then processed in real-time to identify the object as shown in Figure 2. Trigger signal of the LSC needs to be synchronised with the movement of the conveyor belt to avoid image deformation. This synchronisation ensures that the camera captures each line of pixels precisely as objects move past it. To do so, an encoder is required to detect the movement of the conveyor belt and send a signal to the camera to trigger the image capture at the right moment. Both the camera and the encoder processing are out of scope in this study, but the system operates behind the same principles. The system is notified about a new piece when this piece reaches the camera X axis location.

3.2. Design of the System Architecture

The task proposed in this project involves classifying a continuous flow of pieces on a conveyor belt. The robot (dual-arm robot) must pick-on-the-fly those pieces and drop them in their corresponding deposit. The architecture described in this article builds upon and expands the one previously outlined in [8], addressing its primary shortcomings. This previous study is focused on the pick and place operation for a reduced number of static pieces.

Figure 3 shows the block diagram of the process. The conveyor belt carries on a set of parts at a certain speed. There exist five types of parts and each one needs to be sorted out into a different deposit. Four of them must be picked up by the robots and dropped off in their corresponding deposit. The fifth type should be disregarded, and it will automatically fall into the collection area at the far end of the conveyor belt. The dual-arm robot is able to handle up to two pieces simultaneously. The assignment between the robots and the pieces, and the trajectories performed by each robot to approach and transport the pieces is determined by the main controller. This controller gathers information about the XY position of the parts through the LSC and the conveyor belt encoder. The LSC supplies image data to a buffer that accumulates the information until a dynamically sized image is obtained. Once the system identifies an object within the image, the main controller receives a notification and detailed information regarding the type of the piece and its XY coordinates. Subsequent to that moment, the location of the pieces is deduced by utilising an encoder within the conveyor belt. The speed control block is responsible for synchronising the progress of the robots and the pieces to guarantee that their positions align precisely during the pick-up operation. The main controller, described in the Section 0, is based on the MDP framework, and takes advantage of a predefined Database that includes specific and pertinent information about the YuMi robot within this setup, such as collision detection between arms, arm location feasibility and robots' joint configuration based on the robots' XY location.

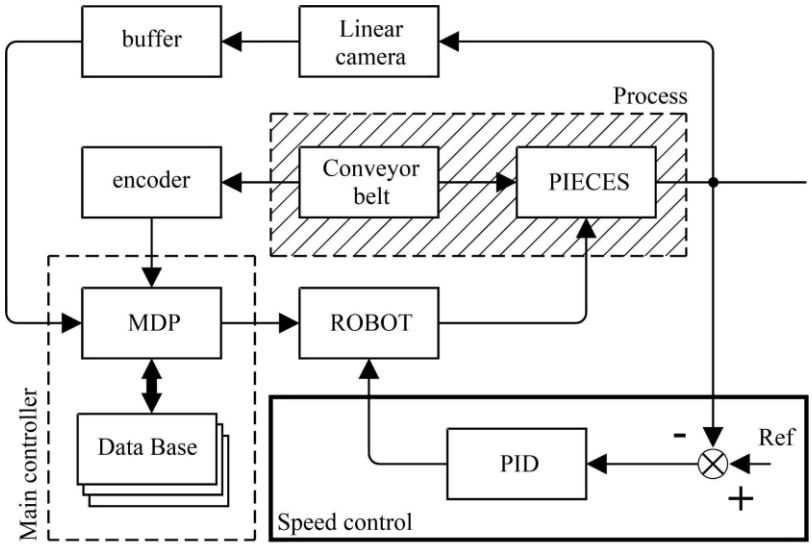


Figure 3. Software and hardware elements of the configuration setup

Maintaining the same core strategy of the previous study, the robot workspace area is discretized into a 2D constellation of points (2D grid). These points then serve as waypoints to guide the robots' trajectories. All robots and pieces are represented as single points in the space, and the robots' trajectories are encoded as an ordered sequence of grid points. All robots move synchronously at a predefined rate, executing one transition per time step. The trajectories are limited to transitions between orthogonal adjacent grid points in order for the robots to run the same distance on every time step. This strategy ensures that both robots move from one location to another almost simultaneously and there is no need to introduce unnecessary waiting periods for any specific robot.

The computed plan applied by the main controller contains all the trajectories and actions (pick/drop) that the robots need to perform to complete the task. This approach significantly streamlines the problem by limiting the potential locations of the points representing the robots, but ignores some key factors, such as the robot body, which occupies a specific region within its workspace, or the fact that several locations within the grid are inaccessible to the robot. That information is introduced separately by building a database that includes the positions outside the robots' effective range and the locations where the robots collide. Another crucial aspect of this architecture is that it relies on the robot's capabilities, taking advantage of the robustness and precision of the movements performed by the robot itself. At each time step, the robot is requested to move to a new configuration by executing standard robot instructions, such as MOVL for the ABB robots. The transition to the new configuration is managed internally by the robot. The last key point of this approach is that the trajectories are described as a sequence of joint configurations. Each of those joint configurations corresponds to a specific XYZ position of the robot within the grid. Understanding the precise joint configurations of each robot at specific grid locations enables efficient collision detection.

A notable constraint in the previous study lies in its scalability concerning the quantity of pieces involved, and the problem explored in this study centres on a continuous and unlimited flow of pieces. The fundamental concept behind the new architecture involves breaking down the problem into a series of subproblems. These subproblems can then be addressed using an architecture similar to the one defined in the previous study. Each of these smaller problems imposes a strict limit on the number of pieces ensuring they remain small and predetermined. On the other hand, addressing the challenge posed by the moving pieces necessitates an extension of the previous architecture. For effective operation, the system must track the precise location of each piece at all times. Since all the pieces move along the conveyor belt at the same speed, their positions at any given time step can be calculated based on their initial location, the number of time steps, and the conveyor belt's speed.

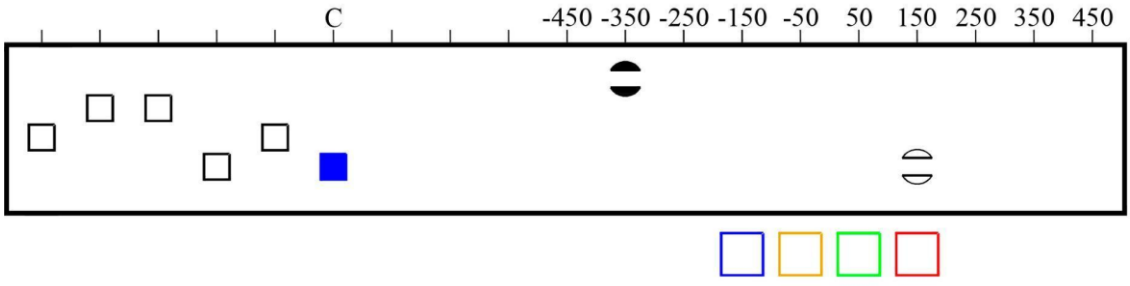
For ease of the pick-up procedure, the operation is restricted to XY coordinates that align with the grid locations within the robot workspace areas. An additional limitation is that the conveyor belt speed must be a fraction of the robot's speed (as indicated in Equation (1)) with the speed ratio (R) required to be an integer value.

$$\varsigma_{\rho} = \varsigma_{\pi} \cdot \mathbf{R} \tag{1}$$

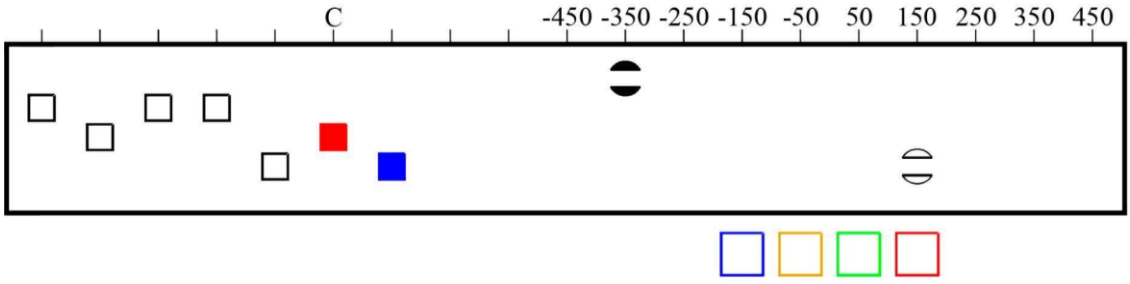
where ς_{π} represents the speed of pieces, ς_{ρ} represents the speed of the robot and $\mathbf{R}$ is a configurable parameter in the system, although the experiments performed in this study uses $\mathbf{R}$=5, hence the pieces are moving five times slower than the robot. Specifically, the robots' speed is set to 200mm/s and the conveyor belt speed is set to 40mm/s. In practical scenarios, it is unlikely that both the conveyor belt and the robots will precisely match their exact expected speed. The new architecture can account for deviations or small variations in the conveyor belt and robot speeds, provided that these speed errors are quantifiable, as discussed in Section 3.4.

Each subproblem operates on a small subset of pieces. The pieces selected for each subproblem are the ones farthest to the right on the conveyor belt. This selection is due to the left-to-right movement of the pieces. Once a piece extends beyond the robot's working area, it becomes irretrievable. Dividing the problem into non-overlapping subproblems (i.e.: the first subproblem including the pieces 1 and 2, the second subproblem including the pieces 3 and 4, and so on …) would lead to an inefficient global solution, because if one robot deposits a piece before the other, it would be idle, wasting valuable time until the remaining piece is processed. Given the continuous flow of pieces into the workspace area, once a piece is completed, it is likely that a new piece will be ready to be inserted into the subproblem. Figure 4 is a representation of the subproblem division. The ruler on the top of the picture indicates the position in millimetres for the robot workspace area. The letter 'C'

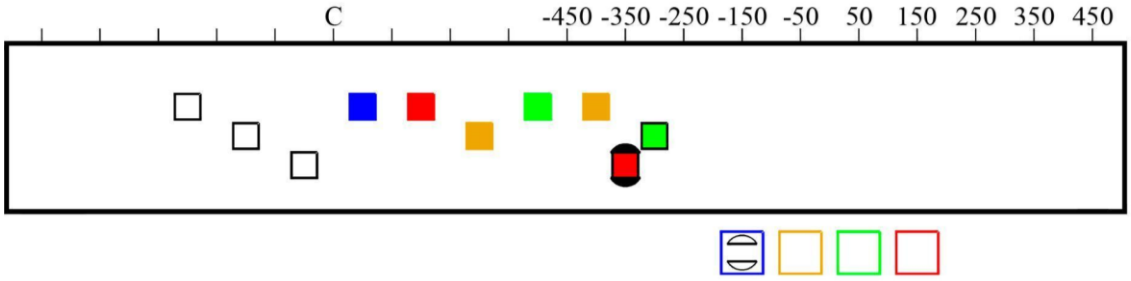
represents the camera detection position (in this case at -850mm with respect to the 0mm in the workspace area). The pieces are represented by a square box. As soon as a piece is detected by the camera, the corresponding box is filled with colour. The colour identifies the type of piece or the deposit where it should be dropped. The pieces involved in the current subproblem are outlined with dark colour. The deposits are represented with larger non-filled square boxes following the same colour convention as the pieces. Finally, the robots' grippers are represented by a split circle. The right arm one is filled with black colour, and the left one with white colour. When the gripper carries on a piece, the gripper is depicted as firmly holding the piece.



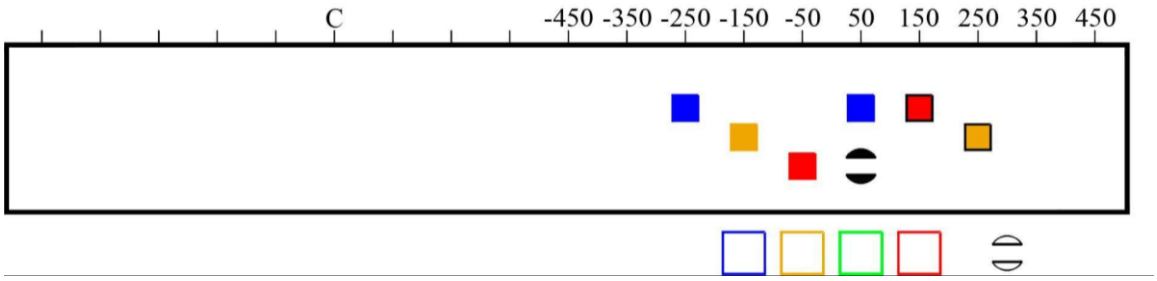
a)



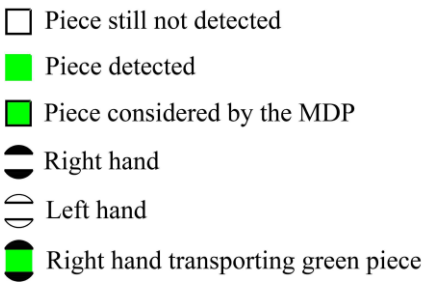
b)



c)



d)



e)

Figure 4. Representation of the problem: a) First subproblem: one piece detected; b) Second subproblem: two pieces detected; c) Third subproblem: the first piece (blue) has been processed; d) N^{th} subproblem; e) Description of visual elements

Figure 4 provides a glimpse of several visual snapshots representing the algorithm's internal information management. Initially, the robot workspace area is empty, and the pieces are moving (from left to right) but the system is unaware of them. At this point, it is not necessary to formulate any plan for the robots, and they keep in idle state. After some time, the first piece (blue in this particular case) is detected by the camera and notified to the system. The system generates and activates a plan for the robots to account for that piece (Figure 4a). A few time steps later, the robots still keep in idle state (Figure 4b) because the pieces are still far away from the workspace area, and the algorithm determines that there is no need for the robots to move yet. At this same moment (Figure 4b), the second piece is detected by the camera and notified to the system. The system generates a new plan for the two first pieces, deactivates the current plan and activates the new plan that accounts for both pieces from that time step on. After some time, the robots pick up the pieces and carry them on to their respective deposits. Figure 4c represents the moment where the blue piece is dropped off. Then the corresponding robot is released and ready to process a new piece (the green piece). At this point, a new plan is generated for both the red and the green pieces and the current one is replaced by this new plan. This procedure is repeated once and again after any piece has been processed, as long as there are new pieces detected by the camera pending to be processed. Otherwise, there is no need to build a new plan, and the current one can be used until the subproblem is completed. Figure 4a, Figure 4b and Figure 4c represent the first three overlapping subproblems in this example.

The first plan contains the whole sequence of actions required by the robots to pick, transport, and drop the first piece in its corresponding deposit. Upon detection of the second piece, the initial plan is discarded, despite the fact that the first piece hasn't been pick up yet, and the subsequent plan is implemented, the new plan includes the whole sequence of actions required by the robots to pick, transport, and drop the first two pieces, taking into account the current location of the pieces and the robots. Similarly, when the first piece is deposited, the second plan is immediately abandoned and replaced by the third one, even though the red piece is still in-progress.

Each subproblem supports up to two pieces. The Algorithm 1 shows the pseudocode that outlines the reasoning used to identify the necessity of a new subproblem and the implementation of the calculated plan to resolve this subproblem Initially, there are no pieces in the workspace area, hence there is no subproblem defined and the robots keep in idle state. The algorithm operates at the granularity of the time step. At each time step, the algorithm dispatches the subsequent action (line 24) for the robots from the existing plan, if available. If specific criteria are fulfilled, the whole plan is redeveloped (line 14), and the system progresses with the updated plan.

Algorithm 1. Pseudo-code to emulate system model behaviour.

```
1:  // Initially, there isn't any plan active (robots keep idle)
2:  time_step = 0
3:  plan = None
4:  plan_max_pieces = N          // N = 2 chosen for the experiments
5:  LOOP
6:    // Keep track of the pieces already detected and not yet assigned to any subproblem
7:    IF new piece/s detected THEN
8:        add new piece/s to pieces_detected list
9:    ENDIF
10:   // Determine if a new subproblem needs to be generated
11:   IF pieces_detected list is not empty THEN
12:       IF there is no active plan OR subproblem_num_pieces < plan_max_pieces THEN
13:         subproblem = createNewsubproblem()
14:         plan = generateNewPlan(subproblem)
15:         time_step = 0          // start the new plan
16:       ENDIF
17:   ENDIF
18:   // Deactivate the plan when is done
19:   IF plan is completed THEN
20:     plan = None
21:   ENDIF
22:   // Execute the actions for the corresponding time step in the plan
23:   IF plan != None THEN
24:     execute plan step
25:   ENDIF
26:   // Prepare for the next time step
27: ENDLOOP
```

3.3. Design of the System Model

The system model refers to the set of MDP models associated with each of the subproblems resulting from the decomposition of the original problem. Each of these models shares a common structure, but they differ according to initial conditions of the subproblem. The MDP model is specifically designed for a particular arrangement of the pieces, as accommodating any arbitrary location would substantially amplify the complexity of the problem. The pickup operation is only feasible at certain locations where the XY coordinates of the piece and the robot match, and these locations are time dependent. The actual number of pieces involved on each subproblem doesn't affect the MDP model structure, which always accounts for two pieces. If the subproblem involves only one piece (i.e. first subproblem), the second piece is set up as already processed and its location turns out to be insignificant.

By design, each of the subproblems involve two robots moving through a 2D grid to approach and transport up to two pieces. The pickup and drop off operations start at any arbitrary location of this 2D grid and consist of a predefined sequence of movements. The pieces are carried out by the conveyor belt, so their pickup location depends on the time step where this operation is performed, the initial location of the pieces and the conveyor belt speed. The pieces' assignment to the robots is part of the solution of the problem. All this information needs to be incorporated into the model of the system. A portion of this information remains constant:

Variable	Description
$pieceInitPos_j$	*Initial XY coordinates (20x5 2D grid) of the piece j, for j=1...K*
$pieceEndPos_j$	*End XY coordinates (20x5 2D grid) of the piece j, for j=1...K*
K	Number of pieces
N	Number of robots
G	Number of pick/drop states

Table 1. Invariant state information

Variable	Description
$robotPos_i$	*XY coordinates (10x5 2D grid) of the robot i gripper, for i=1...N*
$piecePos_j$	*XY coordinates (20x5 2D grid) of the piece j, for j=1...K*
$robotStatus_i$	*piece carried by the robot i (1...K) or none (0), for i=1...N*
$pieceStatus_j$	*piece j is in the conveyor belt (1) or not (0), for j=1...K*
$robotGripperStatus_i$	*robot gripper state (0...G-1) during the pick/drop operation, for i=1...N*

Table 2. Dynamic state information

PieceEndPos$_j$ matches the designated drop-off location (deposit) for this particular piece, and its value depends on the type of piece. *PieceInitPos*$_j$ variable indicates the initial location of the piece in the current subproblem. The present location of each piece on the conveyor belt at any particular time step can be deduced from its starting location in the subproblem and the count of time steps carried out within this subproblem plan (2). Since all the pieces move at the same speed on the conveyor belt, there is no need to track individually and explicitly the current location for every piece by adding the var *piecePos*$_j$ to the model. The model incorporates the *time_step* variable instead. The pieces' location will be inferred from it at each time step:

$$\mathrm{piecePos_j} = \mathrm{pieceInitPos_j} + \mathrm{conveyorBeltSpeed} \cdot time\_step \quad (2)$$

The computational cost is reduced by encoding a single and common *time_step* variable than *K* individual pieces locations. The range of the *time_step* variable is bounded by design. The pieces move forward at a regular speed on the conveyor belt, and their position needs to be monitored at each time step. The number of time_steps taken for any piece to progress from the camera detection point to the end of the robot workspace area is exactly the same number of various locations that the piece can take during this interval (one per time step). Hence, the number of all the possible combinations of all pieces' locations would increase exponentially with the number of pieces (time_step_range_max^K).

The time step range (0-65) required depends on the workspace area dimension, the camera location, and the conveyor belt speed. In this study, the workspace area extends in the X axis from -450mm to 450mm, the camera is located at -850mm with respect to the centre of the workspace area and the conveyor belt speed is set to ⅕ of the robot speed, that is, 20mm/s:

$$N_{ts}^{max} = (\ 450\mathrm{mm}-(-850\mathrm{mm})\ )/(20\mathrm{mm/s}) + 1 = 1300/20 + 1 = 66$$

Where N_{ts}^{max} is the maximum time_step. The time_step variable greatly increases (x66) the state space size of the model compared to the scenario where the pieces are static, hence their location at any time step is directly their initial location and there is no need to include a time_step variable in the system.

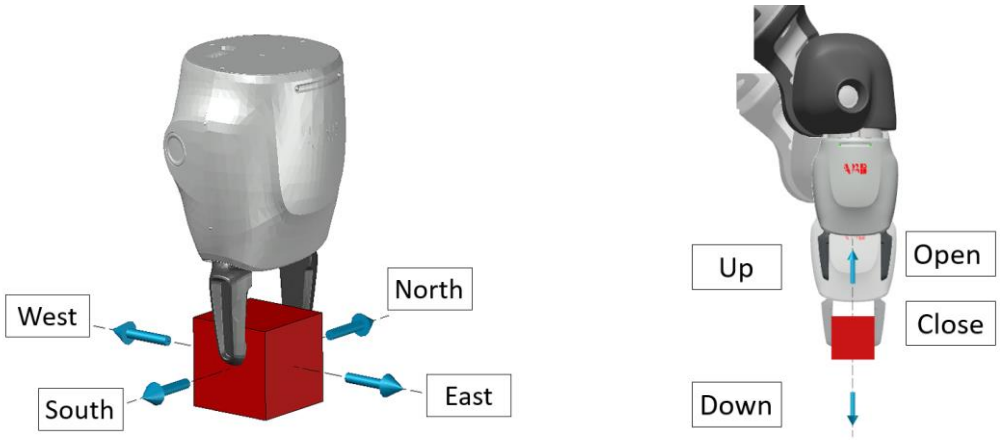
The number of pieces (K) on each subproblem is fixed to two, the number of robots for the entire problem is fixed to two (YuMi dual arm) as well, and the number of pick/drop states is three. The *robotPos*$_i$ variable depends on the actions determined by the current plan (move east, west, north, south, pick, drop or idle). This variable encodes the XY location of the robot gripper. The pick and drop operations involve a sequence of actions: move down vertically, close/open the gripper, and move up vertically. Those actions are executed one per time step (0...G-1). The 3D position of the robot during the pick and drop operations is not captured in the robotPos$_i$ variable. The robot only descends to a lower Z plane (workspace table) at certain grid points. It is more resource-efficient to store this information in a separate variable (*robotGripperPos*$_i$) than to expand the 2D workspace grid to a 3D workspace grid. During the standard robot movements across the 2D workspace grid, RobotGripperStatus$_i$ is assigned a value of 0. After the pick or drop operation is initiated, this variable is set to 1 to indicate that the robot has made a vertical downward movement and is located at a lower plane (workspace table), ready to pick or drop the piece. In the subsequent time step, this variable is assigned a value of 2 signifying that the gripper is closed (for pick) or opened (for drop). Finally, in the next time step, the variable is reset to 0 indicating that the robot has ascended vertically back to the 2D workspace grid.

The final two variables necessary for the model are *robotStatus*$_i$ and *pieceStatus*$_i$. RobotStatus$_i$ indicates whether the robot is holding any piece and identifies which piece it is. PieceStatus$_i$ denotes whether the piece is in transit on the conveyor belt, or if it has already been picked up by the robot. The *robotStatus*$_i$ and *pieceStatus*$_i$ variables are inherited between subproblems. When a new subproblem replaces the current one, its initial configuration is the configuration of the current subproblem at that point. Given these system variables, the size of the state space of the model is defined by (3).

$$\text{Num\_states} = (\text{G} + \text{K} \cdot \text{P})^{\text{N}} \cdot (\text{K} + 1)^{\text{N}} \cdot \text{N}^{\text{K}} \cdot N_{ts}^{max} \tag{3}$$

The movements supported for the robot in the 2D grid are the four orthogonal moves, as shown in Figure 5a. Considering that there are two robots, there might be a situation in the plan where it is most beneficial to have one robot remain idle, such as when there is only one piece left and two robots are available. Moreover, as the pieces progress along the conveyor belt and the pick-up operation is only possible at grid cell locations, there might be instances where one or even both robots stay idle during the execution of the plan to align the robot gripper's location with the piece. The last two types of actions are picking up and dropping off. Given these options, the size of the action space of the model is shown in (4).

$$\text{Num\_actions} = (\text{N} + \text{S} + \text{W} + \text{E} + \text{Idle} + \text{Pick} + \text{Drop})^{\text{N}} = 72 = 49 \tag{4}$$



a) b)

Figure 5. Supported robot movements: a) Movements along the grid, b) Movements to pick and drop pieces

3.4. Robot movements velocity adjustment

In a real-world process, the conveyor belt's actual speed often exhibits minor variations around the intended speed setting. Additionally, there might be a slight deviation (offset) from the configured speed due to external factors or mechanical tolerances. Essentially, the real-world speed may not align perfectly with the ideal or expected speed. The robots suffer an analogous situation. They move from one location to another according to a predefined plan. However, their actual speed during each time step might not precisely match the planned speed.

The simulations in RobotStudio also account for this noise in the conveyor belt and robot speed (as shown in Figure 10a and Figure 10b). In this case, the settings are 40mm/s and 200mm/s, respectively. The success of the pick-up operation hinges on precise positioning of both the robot and the piece when the gripper is closed. The planning is computed based on predefined speed settings for the conveyor belt and the robot. Deviation from these settings leads to desynchronization between the piece and the robot, and if the error is large enough to a failure in the pickup operation (the gripper is closed before or after the piece is in the pickup location). Figure 10c represents the actual positioning error of a virtual piece that is never picked up. All the pieces move at the same speed as this virtual piece (that is, the conveyor belt speed), so this figure is a good reference for any piece and to find out whether the pickup operation would succeed or fail on every time step. The size of the pieces in this study is 40x40x45 mm, and the pickup operation is planned to be done in the centre of the piece, which leaves a valid range of [-20 mm, +20 mm]. If the error of the piece is outside this range, the pickup operation will fail. Figure 10c illustrates the progressive accumulation of error in this experiment, leading to the failure of any pick operation post t=20. The error shown in this figure is actually the relative error between the piece and the robot. It considers both the errors in the conveyor belt speed and the robots' speed. If both errors happen in the same direction (the speed is higher or lower than the settings on both the piece and the robot), they could compensate each other in the ideal best case. If the errors happen in the opposite direction, the relative error between the piece and the robot would increase. In the end, even if actual speed of the conveyor speed matched precisely the settings, any error on the robot speed would affect the pickup operation, so this error is also modelled as the piece error (if the piece is 22mm apart from the gripper when the gripper is closed, it doesn't matter if the piece was slower than expected, the robot was faster than expected or any combination of the two).

As stated previously, the success of the operation exclusively depends on the synchronisation between the piece and the robot picking this piece up. Although matching the speed settings on both the conveyor belt and the robot would ensure this synchronisation, this is not the actual target. The approach in this study to align the positioning of the piece and the robot is to implement a single controller aimed at reducing the relative error between them, rather than introducing separate controllers for the conveyor belt and the robot, each dedicated to achieving its anticipated speed parameters. An additional benefit of this method is that the controller will solely influence the speed of the robot, eliminating the need for real-time regulation of the conveyor belt speed.

The controller integrated in the system is a PID controller (5). The error the controller tries to minimise is the difference between the expected position of the piece on each time step and its actual measured position. The plan is expecting the pieces to move forward 20mm on every time step. The controller (PID) acts directly over the time steps duration, by increasing or decreasing the speed of the robots, to compensate for the conveyor belt speed error. The objective is to keep the progress of the pieces as close as possible to the average ratio of 20mm/time_step.

$$a_k = \tau + \frac{1}{\psi} \cdot \left[K_p \cdot e_k + K_i \cdot \sum_{i=k-1}^{k-n} (e_i \cdot t_i) + K_d \cdot \frac{e_k - e_{k-m}}{\sum_{i=0}^{m} t_{k-i}} \right] \quad (5)$$

$$e_k = {\chi^{\varpi}}_k - {\chi^{\epsilon}}_k \quad (6)$$

Where ${\chi^{\varpi}}_k$ is the measured position of the virtual piece; ${\chi^{\epsilon}}_k$ is the expected position of the virtual piece; τ is the reference time (0.5 seconds); and ψ the expected conveyor velocity (40 mm/s).

4. Solution of the Problem

Figure 6 represents the high-level stages of the solution. The first two, project and task level, have a high computational cost, but they may be performed offline (before operating in the real process). The project level tasks are focused on generating a database containing sensitive information later required during the real time process, such as collision detection between robots, location reachability and joint configuration given the robots' location. The task level accounts for the most part of the MDP generation. These stages are one-time actions that apply throughout the entire project lifecycle unless there are changes in the problem conditions. The project level must be executed again if and only if another robot model replaces the robot. The task level would be executed again if the number of pieces involved on each subproblem, or the grid dimension or resolution changes.

The execution level corresponds to the real time process tasks. The first step is to complete the MDP generation. There exists a small subset of the MDP that strongly relies on the location of the pieces, and this information is not available during the earlier stages. As stated in Section 3.2, the problem is decomposed in a set of overlapping subproblems, each with a different configuration of the pieces (either the number of pieces or their initial location). The loop depicted at this level represents the need for creating a fresh MDP model that corresponds to the next subproblem. Once the MDP model is updated, the subproblem is solved by using the Value Iteration algorithm. The plan resulting from this solution replaces the current one in the plan task. The plan task operates at the level of individuals of time steps, where it carries out the actions associated with the current plan. Finally, the execution task, which is running inside the robot, is responsible for interpreting and executing those standard robot actions, such as 'close the gripper' or 'update the robot joint configuration'.

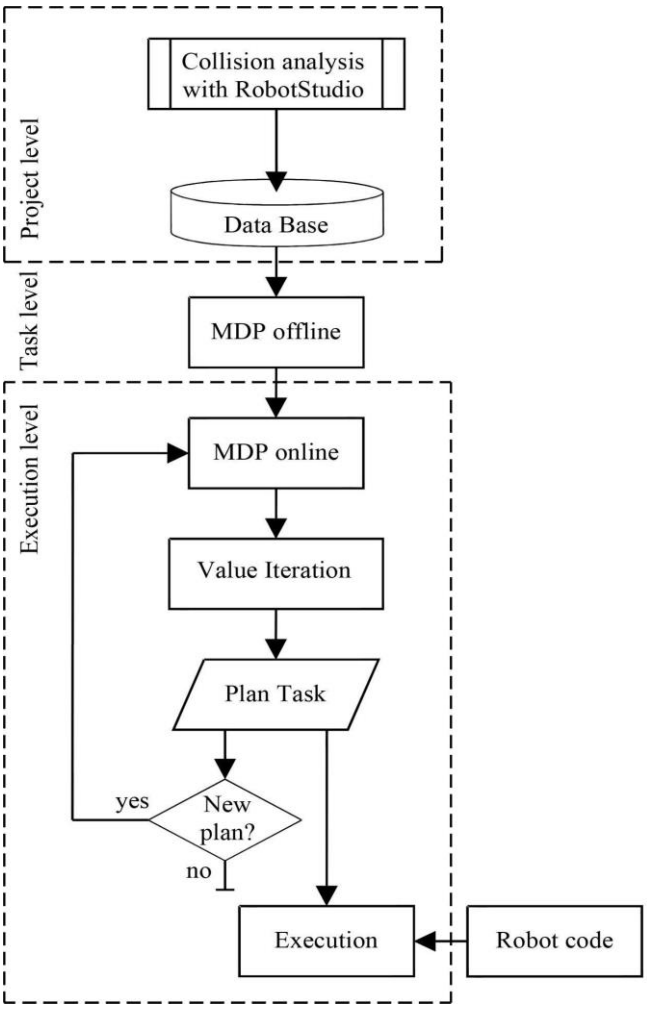


Figure 6. High level process description

4.1. MDP Model

A Markov Decision Process (MDP) is a mathematical framework used for decision-making in uncertain environments. It involves four main components: states, actions, a transition function, and a reward function. MDPs obey the Markov property, meaning future outcomes depend basically on the present state, taken action and uncertainty. In this project, the agent represents the decision-maker, and the environment includes the controlled system, as shown in Figure 7.

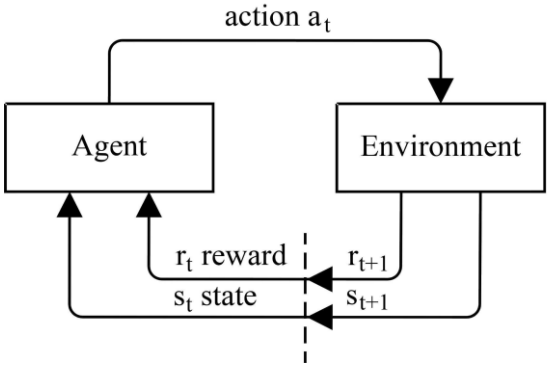


Figure 7. MDP model Agent-Environment

Depending on the problem, an MDP can be fully known, partially known, or even completely unknown. In this study, the MDP is fully known, with all components precisely defined. Given that the

system is discrete, finite and its size remains controlled, transition and reward functions are represented as LookUp Tables (LUTs). Furthermore, considering the system deterministic, these LUTs directly map the pair state (s_t)-action (a_t) to the corresponding next state (s_{t+1}) and reward (r_{t+1}). The MDP defined in this research models a pick-on-the-fly and place operation performed by an agent (brain of the robot) operating inside an environment (which includes the YuMi robot itself, the conveyor belt, and the pieces), as shown in Figure 7.

$$R(s,a) = \begin{cases} +K_\alpha & if\ s' = s_{goal} \\ -1 & else\ if\ (s,a)\ is\ valid \\ -K_\alpha & else \end{cases} \tag{7}$$

The agent operates by taking actions determined by the current state of the environment. In return, the environment modifies its present state according to the action executed by the agent and gives immediate feedback (reward) regarding that action. The reward function, as defined in the Equation (7), accounts for three diverse types of rewards. A big reward ($+K_\alpha$) to indicate that the task is complete, a big penalty ($-K_\alpha$) to signal invalid state-action pairs, such as an action leading to robots' collision, and a small reward (-1) for any other valid action. The goal of Reinforcement Learning algorithms is to maximize the total reward that the agent gathers over time. In this context, the small negative penalty (-1) incurred for each action compels the algorithm to reduce the number of actions needed to achieve the goal, thereby maximizing the overall reward. These algorithms can either learn the system's model before planning (model-based) or directly optimise actions without modelling (model-free) the system. In this case, since the system model is known, only planning is required.

The algorithm employed in this study to find an effective solution to the MDP is Value Iteration (VI). The Value Iteration core component is called Value function (V) and represents, for any given state s, the expected cumulative reward from that state s to the goal state, following a certain policy π. Initially, the policy $\pi(s)$ is usually arbitrary, so the Value function $V^\pi(s)$ presented in Equation (8) is not a good estimation of the cumulative reward. The aim of Value Iteration is to find a policy that maximizes V, which is known as V^* presented in Equation (9). Indeed, it is an iterative algorithm, where $V^\pi(s)$ is gradually improved on each iteration by applying the Bellman equation presented as (10). Given that the MDP is deterministic and is using additive discounting (small -1 penalty), the General Bellman equation can be simplified by eliminating the terms for probability and discount factor, as shown in Equation (10). Specifically, the discount factor (γ) is assigned a value of 1 and the probability term is also set to 1 for the sole action leading to s' (which is unique in a deterministic scenario), while it is set to 0 for all other supported actions. The algorithm stops when the Value function (V^π) converges to the optimal Value function (V^*), indicating that further iterations won't improve it. In practice, the algorithm is deemed to have converged when the change in V is less than a specified threshold. At this stage, the optimal policy (π^*) can be derived directly from V^* according to Equation (11).

$$V^{\pi}(s) = E_{\pi} \cdot \left[\sum_{k=0}^{H} \gamma \cdot R_{k+1} | s_{t=s} \right] \tag{8}$$

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \tag{9}$$

$$V_{k+1}(s) = \max_{a} \sum_{s'} P_a(s'|s) \cdot [R(s,a,s') + \gamma \cdot V_k(s')] = \max_{a} [R(s,a) + V_k(s')] \tag{10}$$

$$\pi^{*}(s) = \underset{a}{argmax} \left[R(s,a) + \gamma \cdot \sum_{S'} T(s,a,s') \cdot V^{*}(s') \right] = \underset{a}{argmax} [R(s,a) + V^{*}(s')] \tag{11}$$

The optimal policy Equation (11) adopts the same simplifications as Equation (10). The discount factor is assigned a value of 1 and the summatory (Σ) is removed because the MDP is deterministic, which implies that there is only one unique state s' that results from executing action a in state s. The transition function is set to 1 for this unique state s' and set to 0 for all the others. Although the optimal Value function (V^{*}) is unique, there may be multiple policies (π^{*}) that lead to the same optimal Value function, allowing for different action sequences to achieve the same outcome. This carries real-world consequences, especially in situations where a robot does not have an immediate assignment, and it is more desirable to keep the robot stationary rather than allowing it to move randomly. To address situations where multiple actions lead to the same outcome, a strategy is employed to prioritize certain actions over others, ensuring preferred actions are selected. The algorithm is implemented in Python, making use of NumPy for matrix calculations.

5. Results

5.1. Data Validation - RobotStudio Simulation

The experiments conducted in this study have been done in RobotStudio. RobotStudio is based on AGX Dynamics from Algoryx [33] to power simulations of multibody systems and hose dynamics. The setup integrates a YuMi Robot, a conveyor belt, multiple classification deposits and a continuous stream of pieces. The camera indicates the location where the pieces are detected. The conveyor belt is moving at a speed of 40 millimetres per second, while the robots are moving five times faster. The pieces and the storage deposits designed to hold those pieces share the same colour. The white colour pieces' deposit is located at the end of the conveyor belt.

The problem is decomposed in a set of overlapped smaller problems on the fly, and the overall solution includes the planning for each of those subproblems. Figure 8 depicts a partial sequence from one of the experiments, highlighting this particular behaviour. The sequence commences right after the blue piece has been placed. This happens at the time step number 40 (t = 40). At this point, the right arm is carrying on the yellow piece and the next planned operation is to drop it in its corresponding deposit. Given that only one piece (yellow piece) is involved in the current subproblem, and considering that there are additional detected pieces, the current subproblem is discarded and a new one (4th subproblem) that includes the yellow and red pieces is created. The plan computed for this new subproblem determines a different behaviour for the right arm. The optimal solution for this subproblem entails the left arm swiftly retrieving the red piece, while the right arm cooperates by ensuring sufficient space for the right arm to pick up the piece, as shown in Figure 8b. After the red piece is picked up, both robots are heading to deposit the pieces. In this particular case, both robots reach their corresponding deposit location at the same time and simultaneously perform the drop off operation, as shown in Figure 8c. This happens at t=58.

Since both pieces have been processed, the current subproblem is fully completed and given that there are additional detected pieces, a new subproblem (5th subproblem) is created. The 4th and 5h subproblems do not overlap in this case because all the pieces are dropped off at once.

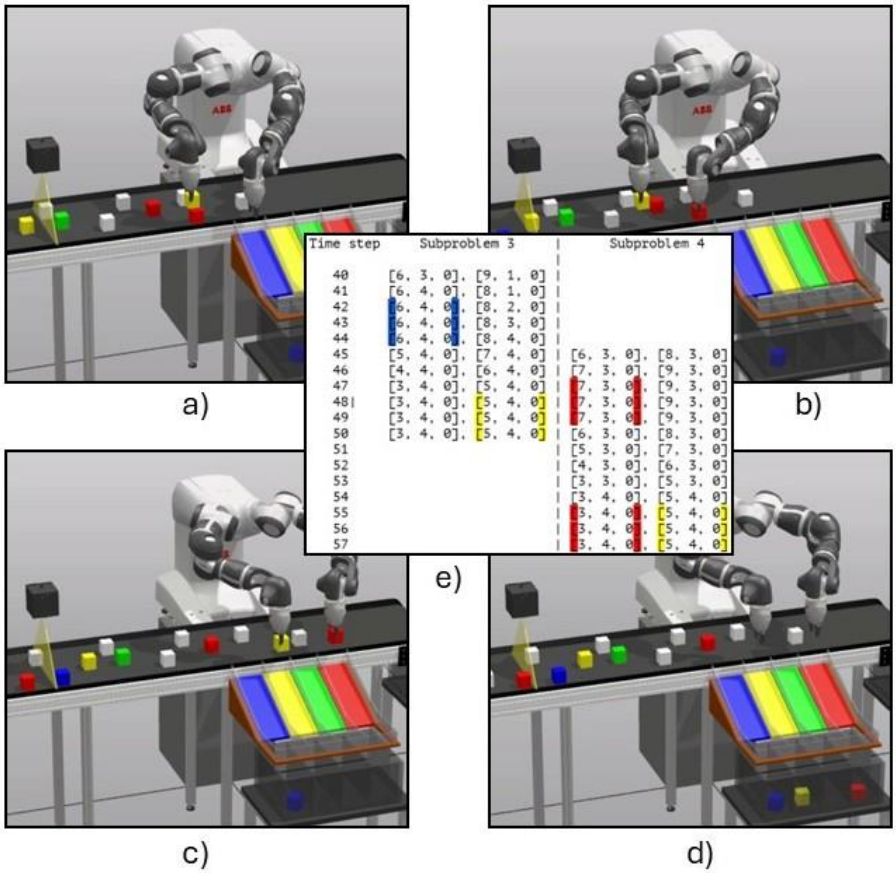
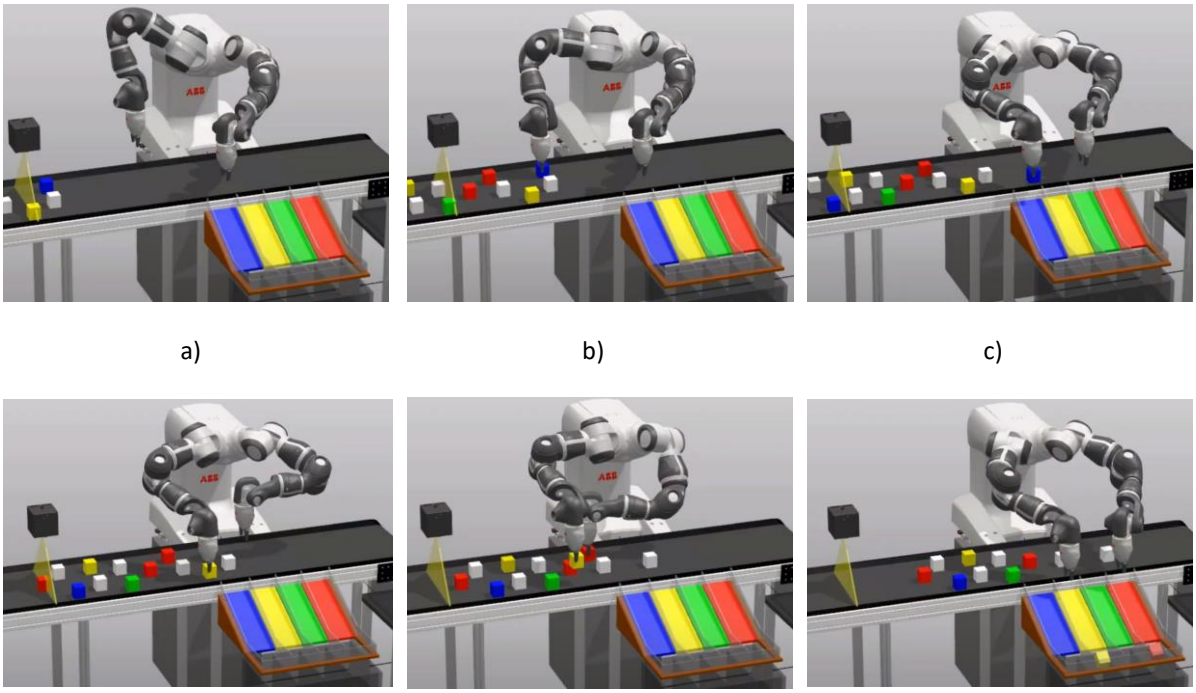


Figure 8. Overlapping between subproblems

The solution computed for every subproblem is optimal with respect to the time needed to complete those subtasks. The subproblems are updated on the fly to continuously account for the maximum number of pieces that the MDP model supports. It is more efficient than just splitting out the problem into fully separated subproblems, but the visibility is still limited to a small number of pieces, hence the global solution is likely not to be optimal. Figure 9 shows the initial sequence of another experiment. The first pieces involved in a subproblem are the blue and the yellow ones. Intuitively, the left arm should be assigned to the yellow piece and the right arm to the blue one because it would allow both robots to easily head to their respective deposits. However, the optimal plan computed for this subproblem indicates that both pieces must be processed by the right arm. The reason is subtle in this case. The yellow piece is still outside the robot workspace area (see Figure 9b) and the left arm would have to wait a few time_steps until the yellow piece is in the robot area. At the same time, the right arm wouldn't be able to immediately head to the blue deposit because it would need to cooperate with the left arm to pick up the yellow piece. After processing the blue piece (see Figure 9c), the subproblem is updated with a new piece (red piece), and a new plan is computed for it. Again, intuitively, the yellow piece should be processed by the right arm and the red piece with the left arm. In this case, the plan generated for this specific subproblem aligns with our intuitive understanding (see Figure 9d, Figure 9e and Figure 9f). The arrangement of the deposits, arms and pieces allows for the smooth retrieval of the red piece without blocking the movement of the yellow piece towards its designated deposit.



a) b) c)

d) e) f)

Figure 9. Optimal solution at subproblem level

5.2. Speed control based on PID

The plan computed for each subproblem assumes a predetermined speed for the robots and the pieces, hence the progress of the pieces on the conveyor belt should be 20mm per time step. This is crucial to ensure that the plan executes the pickup operation when both the robot and the piece are in the expected location. In a real process, none of those assumptions are likely to be true. The experiments conducted in RobotStudio include noise and offset with respect to the predefined values set to the speed of robots and the conveyor belt. Figure 10 illustrates the anticipated situation in an actual process assuming no errors were taken into account. Figure 10a shows the speed of the conveyor belt. The average speed aligns closely with the speed parameters, yet it experiences minor fluctuations over time. The time step within the system is characterised as the duration taken by the robots to finalise their transition. In this experiment, the time step is set to a fixed value of 0.5 seconds. This value is communicated to the robot, which subsequently modifies the robot's speed to accomplish the transition within that precise interval. The simulation incorporates both noise and offset errors in the robot's average speed, which are subsequently converted into the time step interval error. Figure 10b depicts the measured time step interval over time. The samples closer to 0.5s (up to t=20) correspond to the situation where both robots are in idle state, under which circumstances the time step settings are fulfilled. Conversely, when the robots are in motion, the noise and offset inaccuracies become more pronounced. The global error in the system depends on the error in both the robot's speed and the conveyor belt's speed. The width of the pieces is 40mm and given that the pickup operation is designed to occur at the piece's midpoint, the system can tolerate a maximum error of ±20mm. Specifically, the error must be confined within this specified range when the pickup operation is performed. Figure 10c shows the error in the system at every time step.

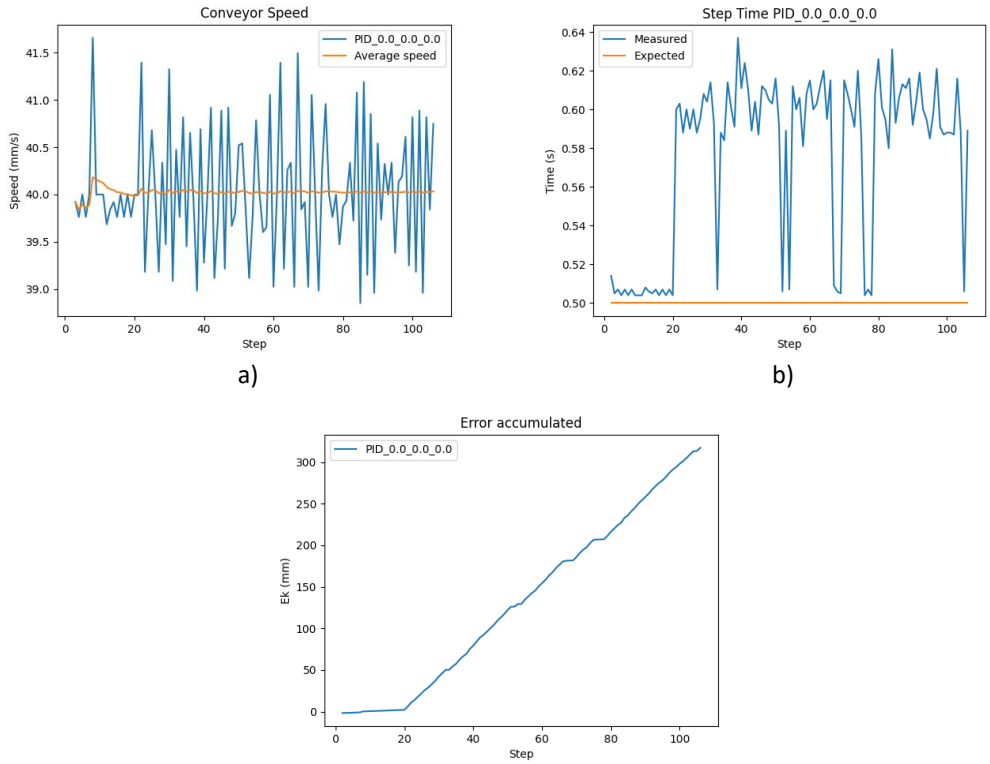


Figure 10. Open-Loop (no speed controller): a) conveyor belt speed, b) step time, c) piece error

To ensure that the system's error remains within an acceptable range, the robots and the pieces should closely adhere to the planned locations at each time step. The approach taken in this study involves integrating a close-loop control mechanism for managing the time steps intervals. The controller continuously updates the time step interval to compensate for the errors in the progress of the pieces that should be 20mm per time step. Figure 11 shows the speed of the robots and the step time value for two different experiments. The blue line corresponds to a proportional controller, and the orange line to a PID controller (which combines proportional, integral, and derivative control). The speed of the conveyor belt is unaffected by the presence of the controller.

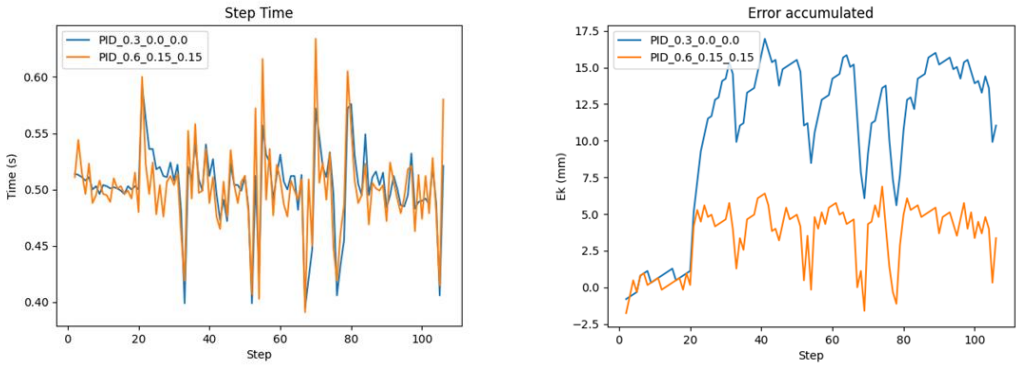


Figure 11. Close-Loop (PID controller): a) step time, b) piece error

The evolution of the step time value with the PID controller looks smoother than the one with the proportional controller. This is attributed to the derivative component by considering the rate of change of the error in the system, which helps to prevent the overshooting. However, the crucial

factor is the error made by the pieces over time. While both the proportional and PID controllers effectively manage the system error, the PID controller ensures that the error remains within a narrower range, as shown in Figure 11b.

5.3. Computational cost

The experiments have been performed in a 5570 Latittude laptop with i7-12800H processor and 16GB of RAM. Table 3 gathers information about the resources required to run the experiments. Those values represent the mean of all the experimental results obtained. The MDP model is created during the phase prior to the system being deployed. The cost in terms of time (offline time) is substantial. This procedure is currently implemented in pure python, and the execution time could be significantly decreased by transitioning the implementation to a compiled language. The number of states in the model as per the Equation (1) is identified as the brute states in the table. A clean-up operation is performed while building the MDP, which discards invalid states and reduces this figure to one-tenth. The MDP model is stored in disk and requires nearly 400MB. When the system is deployed, the MDP model is updated (MDP update) on the fly for each subproblem and a new plan is computed (MDP solve) for it. Both concepts are split in Table 3 considering that the MDP update is implemented in pure python and this timing could easily be decreased.

Description	Value
Offline time	27m 12s
Online time (MDP update)	0m 53s
Online time (MDP solve)	0m 0.01s
Number of MDP states	1302528
Number of brute states	10077696
RAM required	2 GB
DISK required	394 MB
PID values	Kp: 0.6 - Ki: 0.15 - Kd:0.15
Distance between pieces	100 mm

Table 3. Computational cost information

5.4. Experiments

Table 4 shows some experiments performed during this study:

Test Case	PID Controller Values			Success
	Kp	Ki	Kd	
1a[1]	0.0	0.0	0.0	❌
1b[2]	0.3	0.0	0.0	✅
1c[3]	0.6	0.15	0.15	✅
1d[4]	1.0	0.0	0.0	✅
2[5]	0.6	0.15	0.15	✅
3[6]	0.6	0.15	0.15	✅
4[7]	0.6	0.15	0.15	✅
5[8]	0.6	0.15	0.15	✅

Table 4. Results of experiments

[1] Link video 1: https://youtu.be/iUGcqjgjAXU
[2] Link video 2: https://youtu.be/UFQRLoTq6mI
[3] Link video 3: https://youtu.be/xKveW-h8XXs
[4] Link video 4: https://youtu.be/imff2zBpIh4
[5] Link video 5: https://youtu.be/Z_JyK3DiMVo
[6] Link video 6: https://youtu.be/RjpBKjtqta0
[7] Link video 7: https://youtu.be/ybqs59uY1sA
[8] Link video 8: https://youtu.be/AsrimBO3Epw

This table illustrates the results of experiments conducted across five distinct configurations. Each configuration varies from the others depending on the initial arrangement of the pieces. The optimal discovered parameters for the controller are Kp=0.6, Ki=0.15, and Kp-0.15. The first setup is repeated for different sets of controller parameters. Those experiments have been recorded and are accessible via the links provided above this paragraph.

6. Conclusions and Future Work

This paper presents a method for coordinating a dual-arm robotic manipulator, which automatically prevents collisions during pick-and-place operations of objects in motion. This operation is also known as pick-on-the-fly and place. Furthermore, the robot is required to reduce the time taken to complete the task by determining the optimal sequence of manipulations and the most efficient route planning. The task has been formulated as a decision-making problem within the context of the Markov Decision Process framework. Considering the system's conditions (deterministic, single-agent, and fully observable), the MDP model can be straightforwardly resolved using the Value Iteration algorithm. The proposed MDP is able to always find an optimal solution to the problem, set as a two-by-two-piece handling operation. Provided that the speed of the pieces in relation to the speed of the arms allows them to reach both pieces before they leave the workspace. Optimal in this context refers to the fewest number of steps required to complete the pick-on-the-fly and place task.

This approach of discretizing the workspace of a dual-arm robot, using a MDP and considering moving pieces is completely novel. It is the main contribution of this work, and no similar research can be found in the current state of the art.

However, the proposed method has several drawbacks. First, the system scales relatively poorly because the motion of the pieces exponentially increases the number of states. Second, movements to be performed by the robots are restricted within a grid, which is less natural compared to other solutions such as [9] (that only considers static pieces). However, the proposed approach provides a robust and faster industrial solution to the problem. Finally, a preliminary study is needed to create a database that includes information about collision situations between arms.

In future works, the aim is to explore alternative methodologies, such as hierarchical reinforcement learning or multi-agent learning, to improve the scalability drawback of the current single-agent approach. The objective is to enable a greater number of pieces and robots within the same operational area. Another goal is to evaluate alternative implementations, such as leveraging specialised hardware (multi-threaded based on OpenMP, GPU execution or FPGA implementation based on VHDL), given that the algorithm (Value Iteration) backbone heavily relies on matrix operations. Moreover, the initial research into robot collisions poses considerable time and complexity challenges. Hence, an alternative technique is under consideration. One potential resolution involves encapsulating the robot links within a set of ellipsoids and evaluating collisions among these ellipsoids in three-dimensional space. This method would eliminate the need of a graphical environment like RobotStudio®, probably reducing offline computational cost and eliminating dependency of proprietary software.

Funding

This research has been partly funded by project CPP2021-008593, grant MICIU/AEI/10.13039/501100011033 and by the European Union-NextGenerationEU/PRTR.

Acknowledgments

This project would not have been feasible without the assistance of ABB Spain, which generously provided both software licenses and hardware.

Data Availability Statement

The source code implemented for this study is accessible via GitHub: GitHub - da-mago/Two-arms-pick-place-planning at conveyor-belt

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work the author(s) used ChatGPT in order to improve language and readability. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

Bibliography:

1. Goel, R., & Gupta, P. (2020). Robotics and industry 4.0. A Roadmap to Industry 4.0: Smart Production, Sharp Business and Sustainable Development, 157-169.
2. Bhatnagar, N. (2020). Role of robotic process automation in pharmaceutical industries. In The International Conference on Advanced Machine Learning Technologies and Applications (AMLTA2019) 4 (pp. 497-504). Springer International Publishing.
3. Elpa, D. P., Prabhu, G. R. D., Wu, S. P., Tay, K. S., & Urban, P. L. (2020). Automation of mass spectrometric detection of analytes and related workflows: A review. Talanta, 208, 120304.
4. Alexovič, M., Urban, P. L., Tabani, H., & Sabo, J. (2020). Recent advances in robotic protein sample preparation for clinical analysis and other biomedical applications. Clinica Chimica Acta, 507, 104-116.
5. Borboni, A., Reddy, K. V. V., Elamvazuthi, I., AL-Quraishi, M. S., Natarajan, E., & Azhar Ali, S. S. (2023). The expanding role of artificial intelligence in collaborative robots for industrial applications: a systematic review of recent works. Machines, 11(1), 111.
6. Prabhakaran, S., Rishaa, P. S., Ashwath, R., & Suresh, M. (2024, March). A review on cobots: A man's helping hand. In AIP Conference Proceedings (Vol. 3035, No. 1). AIP Publishing.
7. Khaleel, H. Z., & Humaidi, A. J. (2024). Towards accuracy improvement in solution of inverse kinematic problem in redundant robot: A comparative analysis. International Review of Applied Sciences and Engineering.
8. Mateu-Gomez, D.; Martínez-Peral, F.J.; Perez-Vidal, C. Multi-Arm Trajectory Planning for Optimal Collision-Free Pick-and-Place Operations. Technologies 2024, 12, 12. https://doi.org/10.3390/technologies12010012
9. Gafur, N.; Kanagalingam, G.; Wagner, A.; Ruskowski, M. Dynamic Collision and Deadlock Avoidance for Multiple Robotic Manipulators. IEEE Access 2022, 10, 55766–55781.
10. Y. Li, C. Yang, W. Yan, R. Cui, and A. Annamalai, "Admittance- based adaptive cooperative control for multiple manipulators with output constraints," IEEE Trans. Neural Netw. Learn. Syst., vol. 30, no. 12, pp. 3621–3632, Dec. 2019.
11. Z. Li and C.-Y. Su, "Neural-adaptive control of single-master–multiple- slaves teleoperation for coordinated multiple mobile manipulators with time-varying communication delays and input uncertainties," IEEE Trans. Neural Netw. Learn. Syst., vol. 24, no. 9, pp. 1400–1413, Sep. 2013.
12. N. Kimura et al., "Mobile dual-arm robot for automated order picking system in warehouse containing various kinds of products," in Proc. IEEE/SICE Int. Symp. Syst. Integr. (SII), Dec. 2016, pp. 332–338.
13. T. Tamei, T. Matsubara, A. Rai, and T. Shibata, "Reinforcement learning of clothing assistance with a dual-arm robot," in Proc. 11th IEEE-RAS Int. Conf. Humanoid Robots, Oct. 2011, pp. 733–738.
14. J. Borrell Méndez, C. Perez-Vidal, J. V. Segura Heras, and J. J. Pérez-Hernández, "Robotic Pick-and-Place Time Optimization: Application to Footwear Production," in IEEE Access, vol. 8, pp. 209428-209440, 2020, doi: 10.1109/ACCESS.2020.3037145.
15. Borrell, J., Perez-Vidal, C. & Segura, J.V. Optimization of the pick-and-place sequence of a bimanual collaborative robot in an industrial production line. Int J Adv Manuf Technol 130, 4221–4234 (2024). https://doi.org/10.1007/s00170-023-12922-9
16. Y. Zhang and J. Wang, "Obstacle avoidance for kinematically redundant manipulators using a dual neural network," IEEE Trans. Syst. Man, Cybern. B, Cybern., vol. 34, no. 1, pp. 752–759, Feb. 2004.
17. S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," Auton. Robot., vol. 13, no. 3, pp. 207–222, 2002.
18. G. Wen, S. S. Ge, F. Tu, and Y. S. Choo, "Artificial potential-based adaptive H∞ synchronized tracking control for accommodation vessel," IEEE Trans. Ind. Electron., vol. 64, no. 7, pp. 5640–5647, Mar. 2017.
19. L. Palacios, M. Ceriotti, and G. Radice, "Close proximity formation flying via linear quadratic tracking controller and artificial potential function," Adv. Space Res., vol. 56, no. 10, pp. 2167–2176, Nov. 2015.
20. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in Autonomous Robot Vehicles. Newbury Park, CA, USA: Sage, 1986.
21. R. Volpe and P. Khosla, "Manipulator control with superquadric artificial potential functions: Theory and experiments," IEEE Trans. Syst., Man, Cybern., vol. 20, no. 6, pp. 1423–1436, 1990.
22. D. Guo and Y. Zhang, "A new inequality-based obstacle-avoidance MVN scheme and its application to redundant robot manipulators," IEEE Trans. Syst., Man, Cybern. C, Appl. Rev., vol. 42, no. 6, pp. 1326–1340, Nov. 2012.
23. W. Sun, L. G. Torres, J. V. D. Berg, and R. Alterovitz, "Safe motion planning for imprecise robotic manipulators by minimizing probability of collision," in Robotics Research. Cham, Switzerland: Springer, 2016.

24. J. Oh, H. Bae, and J.-H. Oh, “Analytic inverse kinematics considering the joint constraints and self-collision for redundant 7DOF manipula- tor,” in Proc. 1st IEEE Int. Conf. Robot. Comput. (IRC), Apr. 2017, pp. 123–128.
25. T. Kivelä, J. Mattila, J. Puura, and S. Launis, “Redundant robotic manipulator path planning for real-time obstacle and self-collision avoidance,” in Proc. Int. Conf. Robot. Alpe-Adria Danube Region, 2017, pp. 208–216.
26. D. Nicolis, M. Palumbo, A. M. Zanchettin, and P. Rocco, “Occlusion- free visual servoing for the shared autonomy teleoperation of dual-arm robots,” IEEE Robot. Autom. Lett., vol. 3, no. 2, pp. 796–803, Apr. 2018.
27. Gracia Calandin, LI.; Sala Piqueras, A.; Garelli, F. (2014). Robot coordination using task-priority and sliding-mode techniques. Robotics and Computer-Integrated Manufacturing. 30(1):74-89. doi:10.1016/j.rcim.2013.08.003.
28. Bai, Q., Li, P., Tian, W., Shen, J., Li, B., and Hu, J. (July 21, 2023). "Vision Guided Dynamic Synchronous Path Tracking Control of Dual Manipulator Cooperative System." ASME. J. Manuf. Sci. Eng. December 2023; 145(12): 121003. https://doi.org/10.1115/1.4062546
29. Y. Yang, J. Yang, H. Zeng, and J. Li, "A 3D Vision-Based Conveyor Tracking System for Pick-and-Sort Robotic Applications," 2021 6th International Conference on Robotics and Automation Engineering (ICRAE), Guangzhou, China, 2021, pp. 231-236, doi: 10.1109/ICRAE53653.2021.9657787.
30. Anton, F., Borangiu, T., Anton, S., Răileanu, S., Lişiţă, A. (2022). An Evaluation of Pick on the Fly Methods for High-Speed Part Processing in Low-Cost Digital Manufacturing. In: Borangiu, T., Trentesaux, D., Leitão, P., Cardin, O., Joblot, L. (eds) Service Oriented, Holonic and Multi-agent Manufacturing Systems for Industry of the Future. SOHOMA 2021. Studies in Computational Intelligence, vol 1034. Springer, Cham. https://doi.org/10.1007/978-3-030-99108-1_32
31. Ku, YD., Yang, JH., Fang, HY. et al. Optimization of Grasping Efficiency of a Robot Used for Sorting Construction and Demolition Waste. Int. J. Autom. Comput. 17, 691–700 (2020). https://doi.org/10.1007/s11633-020-1237-0
32. Maier, G., Pfaff, F., Pieper, C., Gruna, R., Noack, B., Kruggel-Emden, H., ... & Beyerer, J. (2020). Experimental evaluation of a novel sensor-based sorting approach featuring predictive real-time multiobject tracking. IEEE Transactions on Industrial Electronics, 68(2), 1548-1559
33. Backman, A. Algoryx—interactive physics. In Proceedings of the SIGRAD 2008, the Annual SIGRAD Conference Special Theme: Interaction, Stockholm, Sweden, 27–28 November 2008; Linköping University Electronic Press: Linköping, Sweden, 2008; p. 87.