UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA MECÁNICA

"Simulació de mecanitzat en 6 eixos amb un robot paral·lel"

TRABALL FI DE GRAU

Juny - 2025

AUTOR: Ferran Almiñana Marchirant

DIRECTOR: Adrián Peidro Vidal

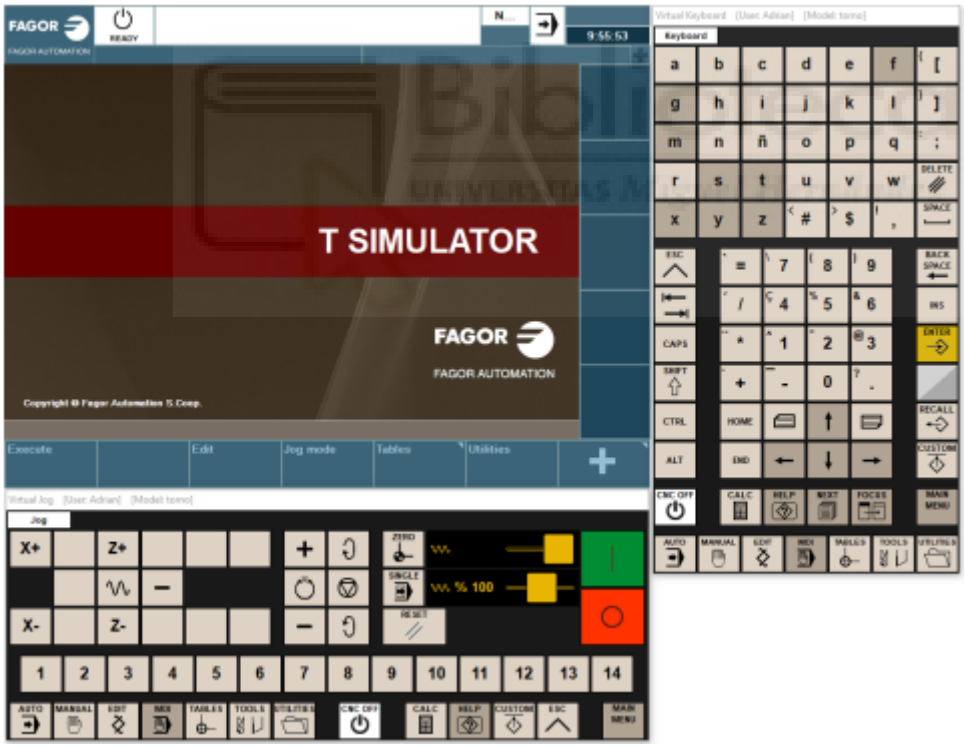
Índex

1. INTRODUCCIÓ

1.1.ANTECENDENTS

La simulació del mecanitzat realitzat per una màquina no es ninguna novetat. Ja fa temps que les empreses disposen de simuladors que permeten comprovar que els moviments que realitzarà la ferramenta i el resultat que s'obtindrà son els correctes, el que suposa un estalvi de diners per a les empreses, que ja no es gastaran els diners en bruts per comprovar si s'obté el resultat desitjat de la seua programació.

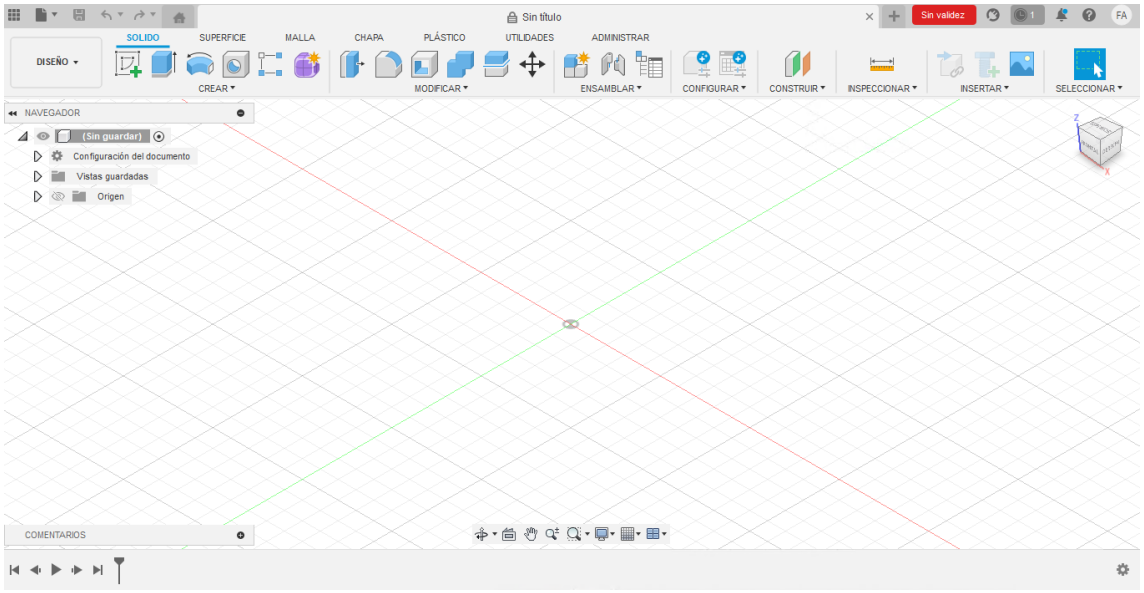
El torn es un exemple de màquina sostractiva amb simulador. L'empresa Fagor te un simulador CNC gratuït (Fagor Automation, 2016) amb el qual es pot vore com el torn realitza el moviment programat i el resultat que s'obté.



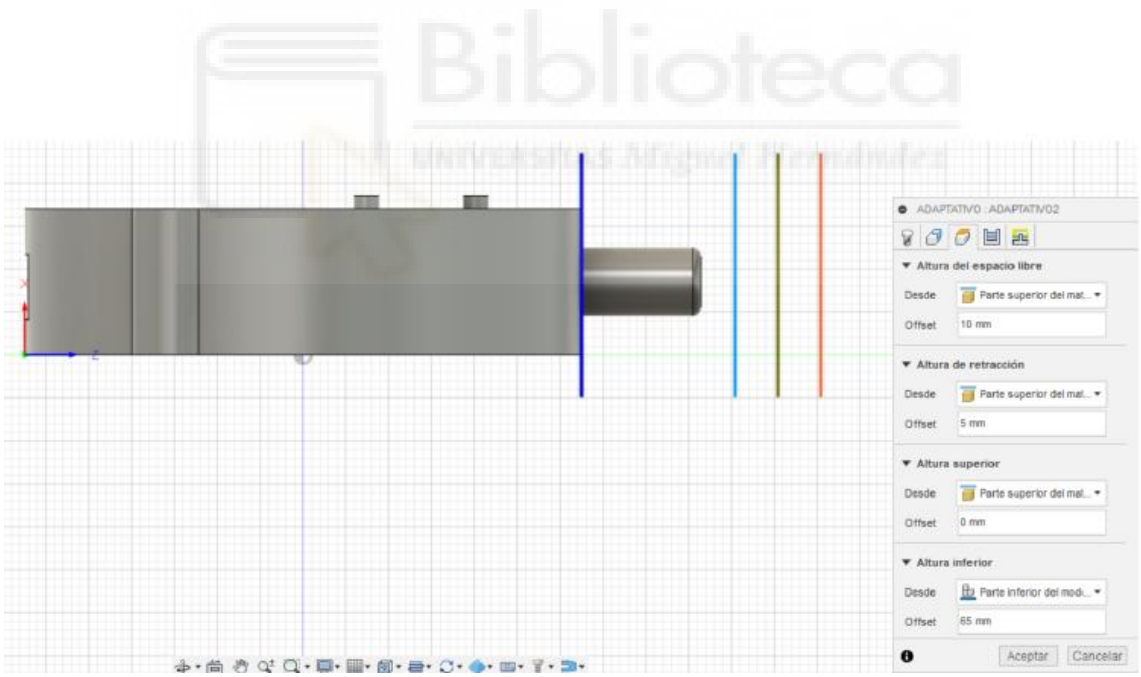
Il·lustració 1: Simulador CNC de FAGOR.

Per a màquines de 3 graus de llibertat (el torn sols en té 2), una de les màquines mes utilitzades son les fresadores. El propi simulador de Fagor ja mencionat també permet la simulació d'una fresadora. En aquest cas, l'empresa Autodesk presenta un simulador mes

complet i mes fàcil d'utilitzar, el Fusion 360, si be aquest no es gratuït com el de Fagor (Autodesk Inc, 2025).



Il·lustració 2: Entorn del Fusion 360.

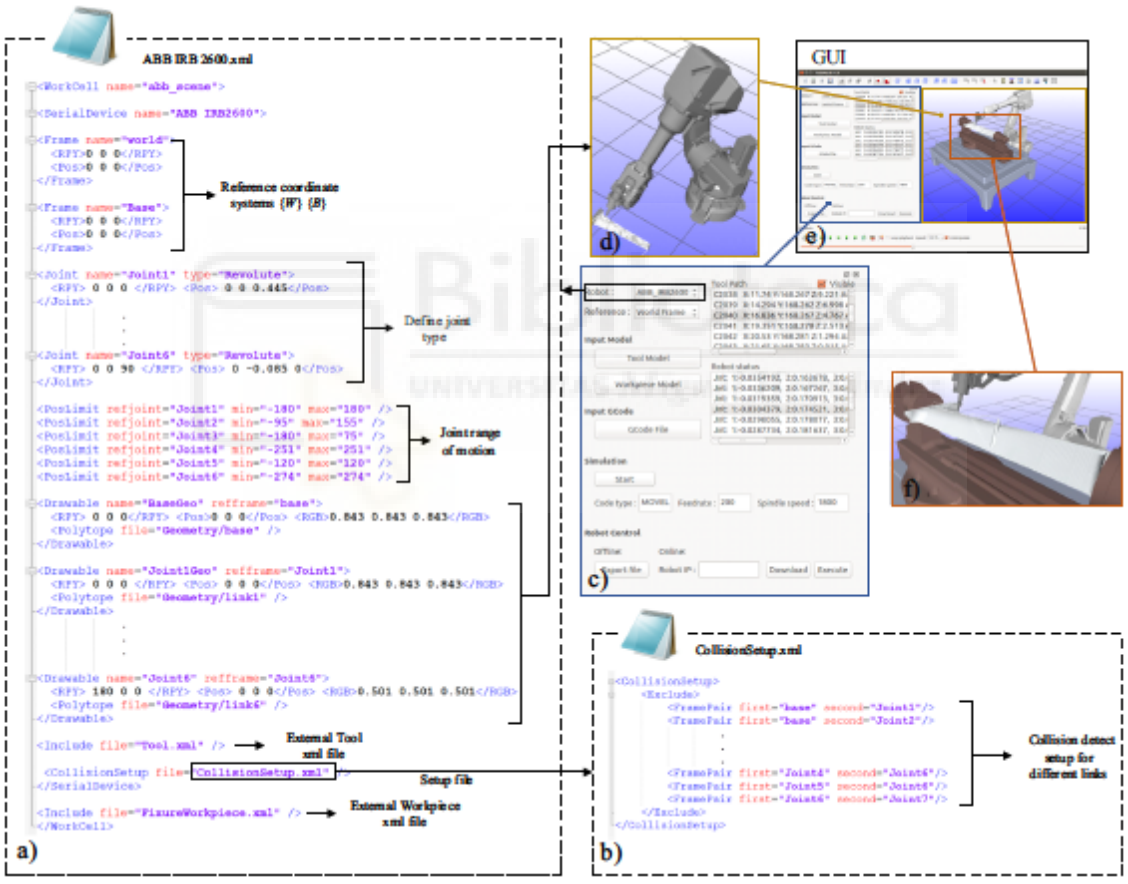


Il·lustració 3: Preparació del mecanitzat amb Fusion 360.

Fins ara, amb aquest simuladors les empreses podien comprovar la programació de les seues màquines abans de treballar amb el brut. El problema es que les empreses estan canviant aquestes màquines per robots. Aquests robots no estan limitats a 2 o 3 graus de llibertat, ja que poden tindre fins a 6 graus de llibertat. Per tant, es necessari disposar d'un simulador que siga capaç de representar el complex moviment que té un robot. I, com que

es tracta d'una tecnologia encara nova, es molt complicat trobar un simulador que complisca amb tots els requisits, ja que la majoria son exclusius per a les empreses per a les quals van ser desenrotllats o encara estan en fases molt primerenques.

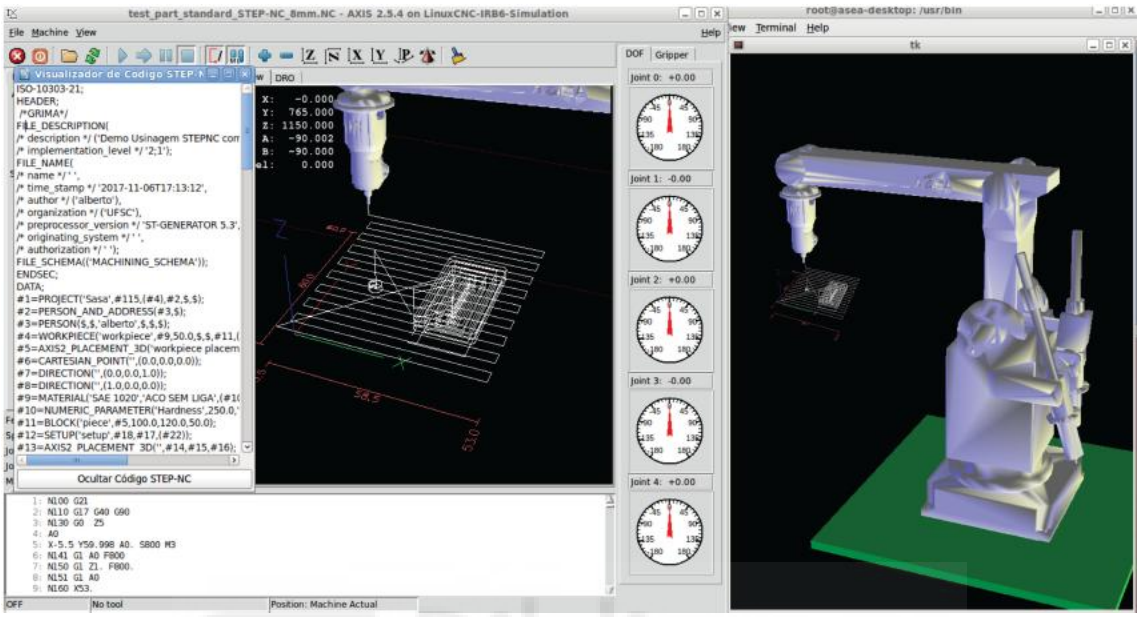
En el treball de Pan *et al.* (2021) es proposa una eina que permet traduir codi G a llenguatge RAPID mitjançant l'ús de la biblioteca RobotWork i la plataforma RobMach. A més de la traducció del codi, la ferramenta és capaç de simular les trajectòries generades pel robot. No obstant això, el sistema està pensat per a robots industrials de tipus sèrie, i en concret s'utilitza un ABB IRB 2600 com a model de prova. Per tant, no contempla el cas dels robots paral·lels ni les particularitats del seu comportament mecànic.



Il·lustració 4: Traductor de codi del article Pan et al. (2021)

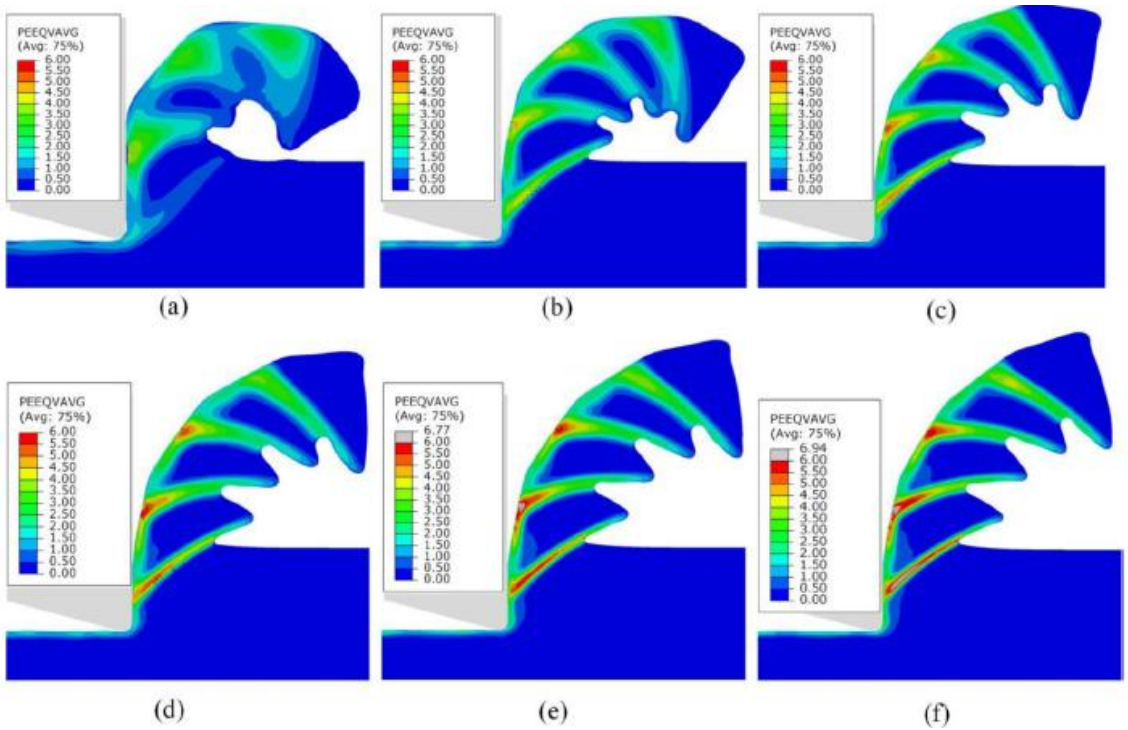
L'article d'Alvares *et al.* (2020) presenta una revisió diferents aplicacions orientades a la implementació del protocol STEP-NC com a alternativa moderna al codi G tradicional. Totes aquestes aplicacions permeten simular el comportament d'un mateix robot, l'ASEA IRB6, també de tipus sèrie. Tanmateix, totes elles comparteixen una limitació destacada: "la falta de controladors de robots industrials capaços d'interpretar directament un

programa de mecanitzat STEP-NC". Això posa de manifest la manca de compatibilitat nativa entre els llenguatges de programació avançats i les plataformes de control robòtic actuals.



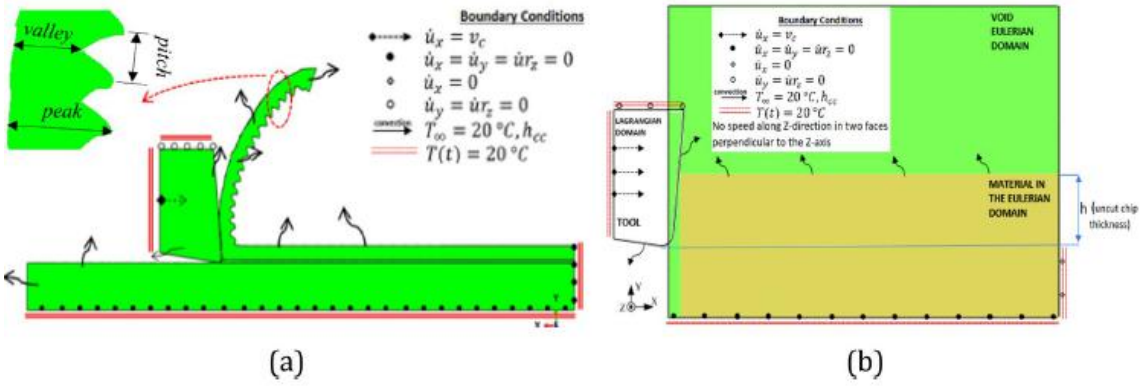
Il·lustració 5: Simulació d'Alvares et al.

En el treball de Xu *et al.* (2021), es presenta una eina de simulació avançada per a processos de mecanitzat d'alta velocitat, amb una capacitat de detall molt elevada. El model de tall que proposen permet simular amb precisió la formació i geometria de l'encenall, aportant una gran fidelitat al procés de mecanització. Tot i això, l'eina se centra en la simulació del material i no aborda la cinemàtica o l'estructura del robot que sosté la ferramenta, i menys encara si es tracta d'un robot paral·lel.



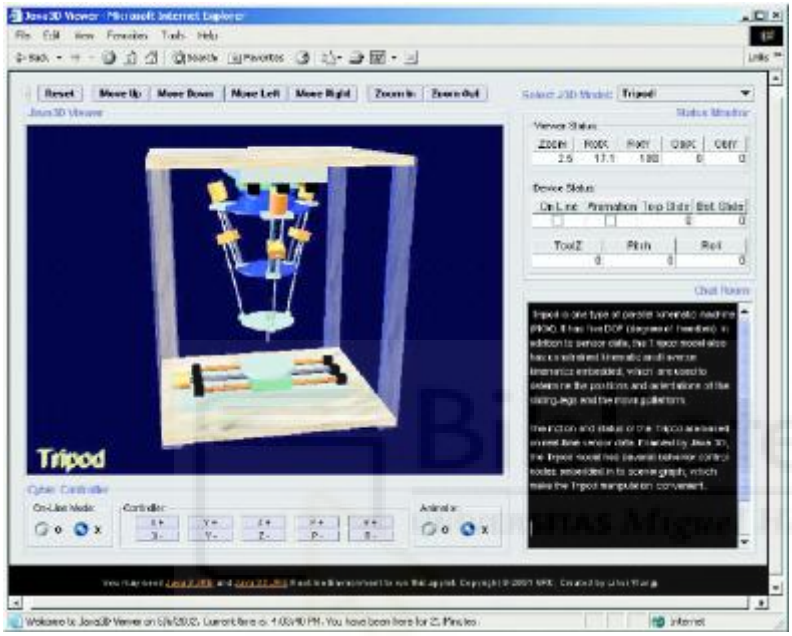
Il·lustració 6: Simulació de la morfologia de l'encenall.

En la seua revisió, Korkmaz *et al.* (2022) analitzen els mètodes de modelatge i simulació aplicats al mecanitzat, des dels enfocaments empírics fins a les tècniques basades en models físics i numèrics. L'article posa de manifest la complexitat creixent dels processos i la necessitat d'integrar informació cinemàtica, dinàmica i del comportament del material. Tot i que s'hi fa una revisió extensa de la literatura i dels reptes futurs en simulació, el treball no presenta cap eina que permeta simular la interacció entre un robot paral·lel i el material, centrant-se més en l'aspecte del mecanitzat que en el sistema mecànic que el du a terme.



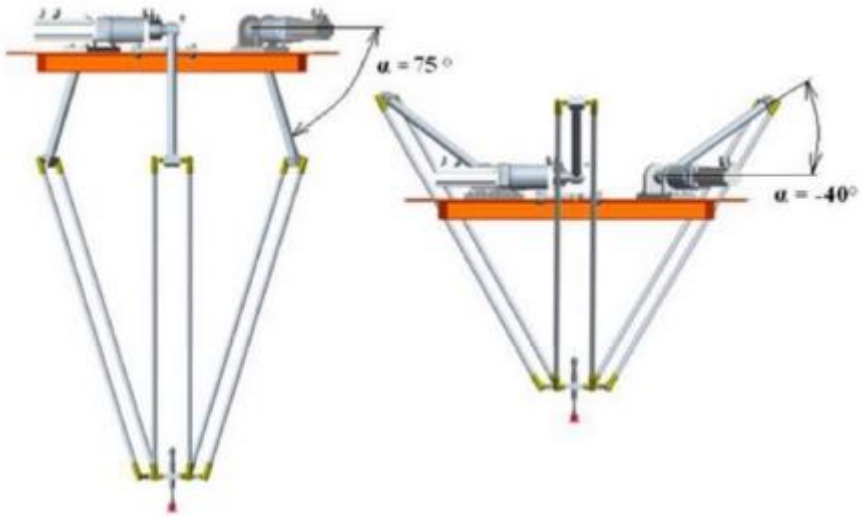
Il·lustració 7: Simulació de mecanitzat de Korkmaz et al.

D’altra banda, Zhang *et al.* (2005) proposen un entorn virtual integrat per al disseny, anàlisi i simulació de màquines amb cinemàtica paral·lela. Aquest entorn permet estudiar diversos aspectes relacionats amb el disseny mecànic i la planificació de trajectòries. No obstant això, l’aplicació desenvolupada no està orientada al procés de mecanitzat ni inclou la simulació de l’eliminació de material, per la qual cosa no cobreix les necessitats d’un sistema que represente tant el robot com el mecanitzat en temps real.



Il·lustració 8: Entorn virtual de Zhang et al.

En el treball de Scerba *et al.* (2025), es desenvolupa un model de simulació orientat a la modelització cinemàtica de mecanismes paral·lels, mitjançant una plataforma virtual interactiva basada en CAD. El projecte permet visualitzar com canvia la configuració d’un robot paral·lel en funció de les variables d’entrada, però no inclou cap interacció amb un entorn de treball ni amb un material a mecanitzar, i tampoc no permet interpretar codi de control com el G-code.



Il·lustració 9: Robot simulat per Scerba et al.

Finalment, *Russo et al.* (2024) realitzen una revisió actualitzada sobre el disseny, el modelatge i les aplicacions de les màquines eina amb cinemàtica paral·lela. L'article destaca els avantatges que ofereixen aquestes arquitectures, especialment en termes de rigidesa i precisió, i exposa diversos camps d'aplicació. Tot i això, entre les limitacions actuals s'esmenta la manca de sistemes de simulació que integren el comportament mecànic i el procés de mecanitzat, especialment en entorns accessibles i didàctics.

Amb aquesta anàlisi, es constata que, tot i l'existència de nombrosos estudis sobre robots paral·lels i sobre simulació de mecanitzat, cap eina actual agrupa en un sol entorn la representació cinemàtica del robot, la interpretació de codi G i la visualització del procés de mecanitzat sobre un brut. Aquesta absència justifica la realització d'aquest Treball de Fi de Grau, que té com a objectiu desenvolupar un simulador senzill però funcional, capaç de cobrir aquest buit.

1.2.OBJECTIUS I CONTRIBUCIÓ

Com s'ha explicat en l'apartat anterior (1.1 Antecedents), és necessari disposar d'una eina que permeta simular de manera precisa els moviments d'un robot paral·lel utilitzat per mecanitzar, i la seua interacció amb els bruts definits per l'usuari, tot seguint la programació establida. Per aquest motiu, l'objectiu principal d'aquest Treball de Fi de

Grau és desenvolupar una simulació que represente correctament tant la cinemàtica com la interacció entre un robot paral·lel i el material a mecanitzar.

Atés que aquest treball s'emmarca dins del grau d'Enginyeria Mecànica, l'enfocament no se centrarà en el desenvolupament avançat del programari del simulador, sinó en el càlcul, representació i comprovació rigorosa de la cinemàtica del robot i de la seua capacitat d'interactuar amb el brut.

Amb l'assoliment d'aquest objectiu, el treball oferirà una eina útil tant per a l'àmbit professional com per al formatiu. D'una banda, es podrà emprar en entorns industrials per validar trajectòries i assegurar que la ferramenta execute els moviments de manera correcta abans de la mecanització real. De l'altra, el simulador podrà utilitzar-se en entorns universitaris com a recurs didàctic per a explicar el funcionament dels robots paral·lels i per a facilitar la pràctica en la programació d'aquestes màquines per part de l'alumnat.

1.3.ESTRUCTURA DE LA MEMÒRIA

Aquesta memòria s'estructura en set capítols principals, cadascun dedicat a un aspecte fonamental del treball realitzat:

- Capítol 1 – Introducció: Es contextualitza el treball mitjançant una revisió dels antecedents, es defineixen els objectius i la contribució del projecte, i s'explica breument l'estructura global de la memòria.

- Capítol 2 – Cinemàtica: Es presenta el model cinemàtic del robot paral·lel utilitzat. Inclou la descripció del mecanisme, el desenvolupament de les equacions de la cinemàtica directa i inversa, l'anàlisi de velocitats i la identificació de configuracions singulars.

- Capítol 3 – Descripció de la programació del simulador: Es detalla el procés de desenvolupament de la simulació, incloent la selecció de la plataforma EJS, la implementació gràfica i funcional del robot, la generació i manipulació del brut,

el tractament de les trajectòries, la interpretació del codi G i funcions avançades com el canvi de ferramenta, el control de la velocitat i la detecció de singularitats.

- Capítol 4 – Exemples: Es mostren diferents trajectòries de mecanitzat executades amb el simulador, amb l'objectiu d'il·lustrar el funcionament del sistema, tant en termes de moviment com d'orientació de la ferramenta.

- Capítol 5 – Conclusions i futurs treballs: Es recullen les conclusions derivades del projecte i es proposen possibles línies de treball futures per ampliar les funcionalitats del simulador.

- Capítol 6 – Bibliografia: S'hi recullen totes les referències consultades, incloent articles científics, llibres i recursos tècnics utilitzats durant el desenvolupament del treball.

- Capítol 7 – Apèndix del codi: Es documenta el codi font del simulador, agrupat per finestres de programació i amb explicacions per a cada bloc funcional, a fi de facilitar la seva comprensió i reutilització.

2. CINEMÀTICA

El robot utilitzat en aquest projecte és un mecanisme paral·lel amb sis graus de llibertat (GDL), dissenyat per realitzar operacions de mecanitzat amb una elevada precisió i rigidesa. Els sis GDL del sistema permeten controlar tant la posició com l'orientació de la ferramenta en l'espai tridimensional.

En aquesta secció es descriu detalladament la seua arquitectura, el funcionament cinemàtic, i les equacions que permeten determinar la posició i orientació de la plataforma mòbil en funció dels paràmetres d'entrada. Així mateix, s'analitzen tant la cinemàtica directa com la inversa, així com les condicions i restriccions pròpies del sistema.

2.1. DESCRIPCIÓ DEL ROBOT

El robot simulat en aquest treball és un mecanisme paral·lel de sis graus de llibertat. Està format per dues bases: una base inferior fixa i una base superior mòbil, sobre la qual s'instal·la la ferramenta. El moviment del robot s'obté mitjançant sis actuadors lineals de tipus UPS, que comprèn una articulació universal o cardànica (U) connectada a la base fixa, una articulació prismàtica (P) que permet modificar la seua longitud de manera independent a diferents velocitats, i una articulació esfèrica (S) connectada a la base superior mòbil.

Les barres connecten la base fixa amb la base mòbil mitjançant articulacions universals, tant en els extrems inferiors com superiors, fet que permet la llibertat de moviment necessària per orientar i posicionar la plataforma mòbil. Es podrien utilitzar en ambdós extrems articulacions esfèriques i el moviment seria el mateix, però apareixeria una rotació parasítica lliure, que s'ha eliminat canviant l'articulació inferior per una universal.

La posició de la base mòbil queda definida per les coordenades espacials x, y i z, mentre que la seua orientació es modelitza mitjançant els angles d'Euler α (alpha), β (beta) i γ (gamma), seguint una seqüència XYZ. Aquests sis paràmetres constitueixen els paràmetres generals del sistema.

2.2. CINEMÀTICA DIRECTA

Tal com s'ha indicat anteriorment, la posició i l'orientació del robot estan determinades per la longitud de les barres que el conformen. Per tant, en el model de cinemàtica directa, es parteix de la longitud de cada barra per tal d'obtenir la posició i orientació de la base mòbil.

Per formalitzar aquest càlcul, es defineixen les següents variables:

- Lj: escalar que conté la longitud de cada barra (j = 1,...,6);
- aj: matriu que conté les coordenades fixes (en el sistema global) de les articulacions ancorades a la base inferior;
- Bj: matriu que conté les coordenades de les articulacions corresponents a la base mòbil (també expressades en coordenades globals), i que depenen dels valors x, y, z, α, β, γ.

L'equació que relaciona aquestes variables és la següent:

$$L_j = \|a_j - B_j\|$$

És a dir, la longitud de cada barra és la distància euclidiana entre el punt fix aj i el punt mòbil Bj.

Tot i això, aquest problema és computacionalment complex: hi ha 6 equacions, com a resultat d'aplicar l'equació anterior, per a cada una de les barres j=1...6. En eixes equacions, són conegudes les longituds Lj, mentre que les incògnites són x y z alpha beta gamma, que hi intervenen a través de Bj. Es tracta d'un sistema de sis equacions amb sis incògnites que no és gens fàcil de resoldre, ja que es requereixen complicades manipulacions algebraiques per tal d'eliminar incògnites en cascada, arribant finalment a una única equació polinòmica amb un grau molt elevat. Segons l'article de Raghavan, M.

(1993), el problema pot arribar a presentar fins a 40 solucions reals per a una mateixa configuració de barres.

Tot i l'interés teòric de la cinemàtica directa, en el context d'aquest treball no resulta útil aplicar-la. El motiu és que en la simulació desenvolupada, és l'usuari qui defineix en tot moment la posició i orientació que ha de tenir la ferramenta. Per tant, la matriu Bj queda determinada directament pels valors que introdueix l'usuari, mentre que la matriu aj és constant.

Per aquest motiu, el càlcul rellevant per al projecte és el de la cinemàtica inversa, que consisteix a determinar, a partir d'una posició i orientació desitjades, la longitud que ha de tenir cadascuna de les sis barres perquè el robot assolisca l'estat desitjat.

2.3. CINEMÀTICA INVERSA

Per tal d'obtenir les longituds de les barres que permeten al robot assolir una posició i orientació determinades, és necessari primer calcular la matriu Bj, que conté les coordenades de les articulacions mòbils expressades en el sistema de coordenades globals. Aquesta matriu s'obté a partir de bj, que és la matriu de coordenades locals de les articulacions sobre la base mòbil.

Per transformar les coordenades locals a globals, s'utilitza la matriu de transformació T, formada per:

- El vector de posició P, que representa la ubicació del grup mòbil en l'espai tridimensional;

$$P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- I la matriu de rotació R, construïda a partir dels angles d'Euler XYZ (α, β, γ).

$$R = \begin{bmatrix} \cos(\beta)\cdot\cos(\gamma) & -\cos(\beta)\cdot\sin(\gamma) & \sin(\beta) \\ \cos(\alpha)\cdot\sin(\gamma)+\sin(\alpha)\cdot\sin(\beta)\cdot\cos(\gamma) & \cos(\alpha)\cdot\cos(\gamma)-\sin(\alpha)\cdot\sin(\beta)\cdot\sin(\gamma) & -\sin(\alpha)\cdot\cos(\beta) \\ \sin(\alpha)\cdot\sin(\gamma)-\cos(\alpha)\cdot\sin(\beta)\cdot\cos(\gamma) & \cos(\alpha)\cdot\sin(\beta)\cdot\sin(\gamma)+\sin(\alpha)\cdot\cos(\beta) & \cos(\alpha)\cdot\cos(\beta) \end{bmatrix}$$

Aquesta transformació es pot expressar com:

$$T = \begin{bmatrix} R & P \\ 0_{1x3} & 1 \end{bmatrix}$$

Aplicant aquesta transformació a la matriu bj, s'obté Bj:

$$B_j = T \cdot b_j$$

Un cop obtinguda Bj, es pot calcular la longitud de cada barra mitjançant l'equació següent:

$$L_j = \|a_j - B_j\|$$

Aquest càlcul permet obtenir les sis longituds necessàries per posicionar i orientar la plataforma mòbil segons els valors definits per l'usuari en x, y, z, α, β, γ.

2.4. ANÀLISI DE VELOCITATS

L'equació anterior que relaciona cada longitud amb la posició i orientació del robot es pot reescriure per a cada barra com:

$$L_1 = \|a_1 - B_1(x\ y\ z\ \alpha\ \beta\ \gamma)\|$$
$$L_2 = \|a_2 - B_2(x\ y\ z\ \alpha\ \beta\ \gamma)\|$$
$$L_3 = \|a_3 - B_3(x\ y\ z\ \alpha\ \beta\ \gamma)\|$$
$$L_4 = \|a_4 - B_4(x\ y\ z\ \alpha\ \beta\ \gamma)\|$$
$$L_5 = \|a_5 - B_5(x\ y\ z\ \alpha\ \beta\ \gamma)\|$$
$$L_6 = \|a_6 - B_6(x\ y\ z\ \alpha\ \beta\ \gamma)\|$$

És a dir, que cada longitud Lj es pot considerar com una funció fj(x, y, z, α, β, γ).

Per analitzar la velocitat de variació de cada barra, cal derivar aquestes funcions respecte del temps. Aplicant la regla de la cadena, s'obtenen les derivades de les longituds:

$$\dot{L_1} = \dot{f_1} = \frac{\delta f_1}{\delta x}\dot{x} + \frac{\delta f_1}{\delta y}\dot{y} + \frac{\delta f_1}{\delta z}\dot{z} + \frac{\delta f_1}{\delta \alpha}\dot{\alpha} + \frac{\delta f_1}{\delta \beta}\dot{\beta} + \frac{\delta f_1}{\delta \gamma}\dot{\gamma}$$

$$\dot{L_2} = \dot{f_2} = \frac{\delta f_2}{\delta x}\dot{x} + \frac{\delta f_2}{\delta y}\dot{y} + \frac{\delta f_2}{\delta z}\dot{z} + \frac{\delta f_2}{\delta \alpha}\dot{\alpha} + \frac{\delta f_2}{\delta \beta}\dot{\beta} + \frac{\delta f_2}{\delta \gamma}\dot{\gamma}$$

$$\dot{L_3} = \dot{f_3} = \frac{\delta f_3}{\delta x}\dot{x} + \frac{\delta f_3}{\delta y}\dot{y} + \frac{\delta f_3}{\delta z}\dot{z} + \frac{\delta f_3}{\delta \alpha}\dot{\alpha} + \frac{\delta f_3}{\delta \beta}\dot{\beta} + \frac{\delta f_3}{\delta \gamma}\dot{\gamma}$$

$$\dot{L_4} = \dot{f_4} = \frac{\delta f_4}{\delta x}\dot{x} + \frac{\delta f_4}{\delta y}\dot{y} + \frac{\delta f_4}{\delta z}\dot{z} + \frac{\delta f_4}{\delta \alpha}\dot{\alpha} + \frac{\delta f_4}{\delta \beta}\dot{\beta} + \frac{\delta f_4}{\delta \gamma}\dot{\gamma}$$

$$\dot{L_5} = \dot{f_5} = \frac{\delta f_5}{\delta x}\dot{x} + \frac{\delta f_5}{\delta y}\dot{y} + \frac{\delta f_5}{\delta z}\dot{z} + \frac{\delta f_5}{\delta \alpha}\dot{\alpha} + \frac{\delta f_5}{\delta \beta}\dot{\beta} + \frac{\delta f_5}{\delta \gamma}\dot{\gamma}$$

$$\dot{L_6} = \dot{f_6} = \frac{\delta f_6}{\delta x}\dot{x} + \frac{\delta f_6}{\delta y}\dot{y} + \frac{\delta f_6}{\delta z}\dot{z} + \frac{\delta f_6}{\delta \alpha}\dot{\alpha} + \frac{\delta f_6}{\delta \beta}\dot{\beta} + \frac{\delta f_6}{\delta \gamma}\dot{\gamma}$$

Per a les sis barres, això es pot expressar en forma matricial com:

$$\begin{bmatrix} \dot{L_1} \\ \dot{L_2} \\ \dot{L_3} \\ \dot{L_4} \\ \dot{L_5} \\ \dot{L_6} \end{bmatrix} = \begin{pmatrix} \frac{\delta f_1}{\delta x} & \frac{\delta f_1}{\delta y} & \frac{\delta f_1}{\delta z} & \frac{\delta f_1}{\delta \alpha} & \frac{\delta f_1}{\delta \beta} & \frac{\delta f_1}{\delta \gamma} \\ \frac{\delta f_2}{\delta x} & \frac{\delta f_2}{\delta y} & \frac{\delta f_2}{\delta z} & \frac{\delta f_2}{\delta \alpha} & \frac{\delta f_2}{\delta \beta} & \frac{\delta f_2}{\delta \gamma} \\ \frac{\delta f_3}{\delta x} & \frac{\delta f_3}{\delta y} & \frac{\delta f_3}{\delta z} & \frac{\delta f_3}{\delta \alpha} & \frac{\delta f_3}{\delta \beta} & \frac{\delta f_3}{\delta \gamma} \\ \frac{\delta f_4}{\delta x} & \frac{\delta f_4}{\delta y} & \frac{\delta f_4}{\delta z} & \frac{\delta f_4}{\delta \alpha} & \frac{\delta f_4}{\delta \beta} & \frac{\delta f_4}{\delta \gamma} \\ \frac{\delta f_5}{\delta x} & \frac{\delta f_5}{\delta y} & \frac{\delta f_5}{\delta z} & \frac{\delta f_5}{\delta \alpha} & \frac{\delta f_5}{\delta \beta} & \frac{\delta f_5}{\delta \gamma} \\ \frac{\delta f_6}{\delta x} & \frac{\delta f_6}{\delta y} & \frac{\delta f_6}{\delta z} & \frac{\delta f_6}{\delta \alpha} & \frac{\delta f_6}{\delta \beta} & \frac{\delta f_6}{\delta \gamma} \end{pmatrix} \cdot \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix}$$

Per simplificar:

$$\dot{L_J} = J_{inv} \cdot v$$

On la matriu Jinv (Jacobiana inversa) conté totes les derivades parcials $\frac{\delta f_j}{\delta}$ (variable), i representa com es propaguen les velocitats de les coordenades generals a les velocitats dels actuadors lineals (les barres).

Aquesta matriu és clau per a l'anàlisi del comportament dinàmic del robot, per a la generació de trajectòries i per a detectar possibles singularitats del sistema, ja que en aquests punts la matriu Jacobiana pot esdevenir no invertible o indeterminada.

2.5. SINGULARITATS

Com s'ha explicat anteriorment, la derivada de la funció que determina la longitud de cadascuna de les barres (Lj') permet calcular la velocitat amb què aquestes s'han d'estendre o contrareure per tal d'assolir una velocitat desitjada de moviment (v) en la plataforma mòbil del robot. No obstant això, en un sistema físic real, la relació que es produeix és la contrària: són els actuadors els que s'estenen o es contrauen a determinades velocitats, i això dona lloc a una translació i/o rotació de la ferramenta.

Per tant, en la pràctica, l'equació que descriu el comportament del sistema és la següent:

$$v = J \cdot \dot{L_J}$$

On *J* és la matriu Jacobiana directa, és a dir, la inversa de la Jacobiana inversa (Jinv).

Tanmateix, aquest model presenta una limitació crítica en les anomenades singularitats. Una singularitat es produeix quan el determinant de la matriu Jacobiana inversa (Jinv) és zero, la qual cosa implica que la matriu Jacobiana directa (J) no existeix. En aquest cas, no es pot establir una relació única entre les velocitats dels actuadors i la velocitat de la plataforma. Això comporta una pèrdua de control del sistema, ja que per a una determinada velocitat de Lj, en primer lloc no està garantit que aquesta velocitat es puga produir, i en segon lloc, en cas que es puga produir, el moviment de translació i rotació

de la plataforma mòbil no és únic, és a dir, pot moure's de infinites formes diferents sense poder predir quin serà el moviment del sistema real.

Aquest fenomen pot provocar moviments no desitjats, oscil·lacions, o directament la inoperabilitat del robot en certes configuracions. Per aquest motiu, la detecció i evitació de singularitats és un aspecte fonamental en el disseny i el control de mecanismes paral·lels, especialment en aplicacions de precisió com el mecanitzat.

3. DESCRIPCIÓ DE LA PROGRAMACIÓ DEL SIMULADOR

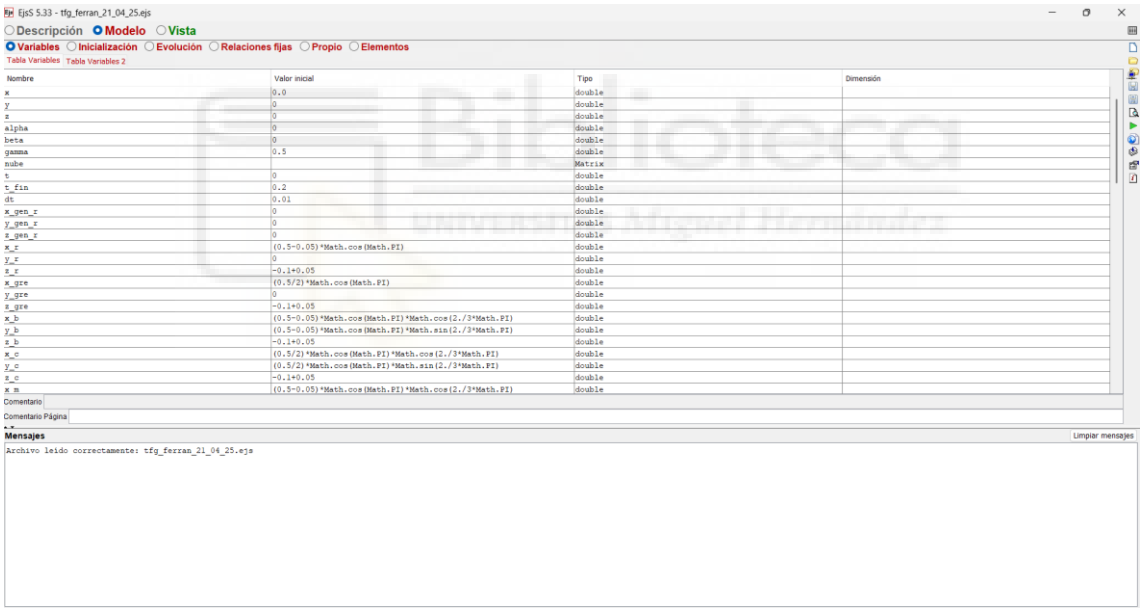
En aquesta secció es descriu el desenvolupament del projecte, explicant pas a pas com s'ha construït la simulació del robot de sis graus de llibertat, així com les decisions tècniques preses en cada etapa.

3.1.BREU DESCRIPCIÓ DE EJS

Per a començar la programació, calia decidir el llenguatge a utilitzar. Python i Javascript eren les dues grans opcions. Python és un llenguatge més fàcil d'utilitzar, però finalment es va optar per Javascript perquè permetia l'ús de diferents dispositius (telèfon mòbil, ordinadors...) d'una manera més senzilla.

Per poder utilitzar la simulació en diversos dispositius, era necessària la programació HTML. I com que el present Treball de Fi de Grau pertany al grau d'Enginyeria Mecànica i, per tant, l'objectiu no era que el focus del treball fos la programació detallada de tota la interfície de simulació, sinó centrar-nos en la simulació de les trajectòries i el mecanitzat, vam decidir utilitzar una eina de desenvolupament que facilitara tota la creació de la interfície de control amb la mínima programació. És per aquest motiu que es va decidir utilitzar el programa EJS. El programa EJS (Easy JavaScript Simulations) és una ferramenta de autoria gratuïta escrita en Java que ajuda als qui no son programadors a crear simulacions interactives en Java o Javascript, principalment amb objectius didàctics. EJS va ser creat per Francisco Esquembre (Universidad de Murcia) i forma part del projecte de Física de Codi Obert (Esquembre, 2015).

Il·lustració 10: Vista general de la ferramenta EJS.



Il·lustració 11: Vista de les diferents finestres per a programar.

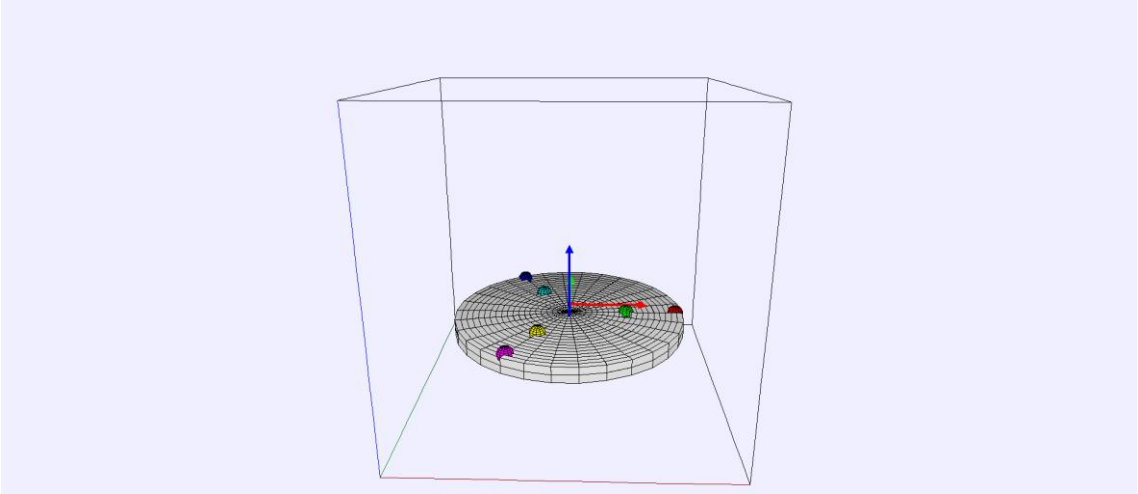
EJS permet generar codi HTML automàticament a partir del codi Javascript programat en l'aplicació (que s'integra perfectament amb el codi HTML generat automàticament); i a partir de la interfície construïda mitjançant la paleta de la pestanya "View" o "Vista", on se seleccionen els elements d'una paleta i es construeix un arbre jeràrquic d'elements gràfics.

El problema va aparèixer quan es volia representar el brut a mecanitzar. El brut consisteix en un conjunt de punts que formen un cub. L'aplicació amb Java té preestablida una figura adequada per al brut: el conjunt de punts. La versió en Javascript no disposava d'aquesta opció, i el més semblant era el conjunt d'esferes. Però calcular cada vegada el radi, la posició i altres paràmetres de cadascuna de les esferes que conformaven el brut alentia excessivament la simulació, per molt menudes que foren aquestes. Per aquest motiu, finalment la simulació s'ha creat utilitzant el llenguatge Java, també acceptat per l'aplicació. (Per comprendre com es representa el brut, mirar Il·lustració 18: Vista del brut a mecanitzar.)

Amb la versió Java, l'aplicació no genera codi HTML, sinó que genera automàticament un codi Java que pot executar-se des de l'escriptori com un fitxer JAR (Java ARchive).

3.2.IMPLEMENTACIÓ DE L'ANIMACIÓ

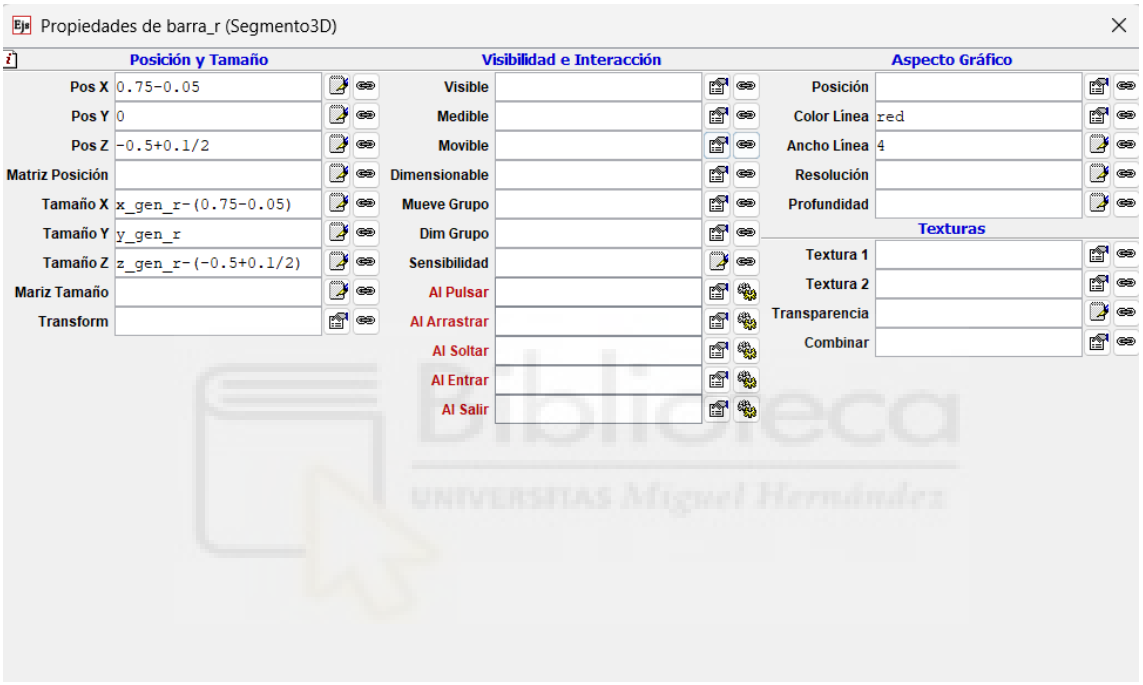
Per poder realitzar la simulació, el primer pas és representar el robot. La base del robot és un cilindre de color gris amb la posició, el radi i l'altura ja determinats. A la base es poden veure unes esferes, les quals tenen també la posició definida i representen els punts d'ancoratge de les barres que s'encarregaran de moure el robot. Els seus colors segueixen el patró RGBCMY (Roig, Verd, Blau, Cian, Magenta i Groc, per les sigles en anglès Red, Green, Blue, Cyan, Magenta, Yellow).



Il·lustració 12: Representació de la base del robot amb els punts d'ancoratge.

Les barres, cadascuna amb el color del punt d'ancoratge al qual està unida, tenen la posició definida (la mateixa que el punt d'ancoratge), però el seu tamany està definit per la diferència entre la posició del punt d'ancoratge mòbil i el fix. La barra roja, per exemple, té definit el seu tamany en l'eix X segons l'equació:

$$tamañoX = x\_gen\_r - (0.75 - 0.05)$$



Il·lustració 13: Definició de les propietats de la barra roja.

on x_gen_r és el nom de la variable que guarda la posició del punt d'ancoratge respecte a l'eix X del sistema de coordenades general.

Les variables que defineixen la posició dels punts d'ancoratge mòbils respecte a l'eix de coordenades general (les utilitzades per calcular el tamany de les barres) es nomenen com "eix_gen_color". Són un total de 18 variables (3 per a cadascun dels 6 colors) i es calculen amb la funció actualizarT(), que està detallada a l'apèndix 7.4.1

Per realitzar aquest càlcul, la funció construeix les matrius P, R i b. La matriu o vector P representa la posició del grup mòbil respecte de l'eix de coordenades general:

$$P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

La matriu R és una matriu de rotació definida a partir dels angles d'Euler XYZ (alpha, beta, gamma):

$$R = \begin{bmatrix} \cos(\beta) \cdot \cos(\gamma) & -\cos(\beta) \cdot \sin(\gamma) & \sin(\beta) \\ \cos(\alpha) \cdot \sin(\gamma) + \sin(\alpha) \cdot \sin(\beta) \cdot \cos(\gamma) & \cos(\alpha) \cdot \cos(\gamma) - \sin(\alpha) \cdot \sin(\beta) \cdot \sin(\gamma) & -\sin(\alpha) \cdot \cos(\beta) \\ \sin(\alpha) \cdot \sin(\gamma) - \cos(\alpha) \cdot \sin(\beta) \cdot \cos(\gamma) & \cos(\alpha) \cdot \sin(\beta) \cdot \sin(\gamma) + \sin(\alpha) \cdot \cos(\beta) & \cos(\alpha) \cdot \cos(\beta) \end{bmatrix}$$

La matriu b representa les coordenades locals dels punts d'ancoratge:

$$b = \begin{bmatrix} x_r & x_{gre} & x_b & x_c & x_m & x_y \\ y_r & y_{gre} & y_b & y_c & y_m & y_y \\ z_r & z_{gre} & z_b & z_c & z_m & z_y \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

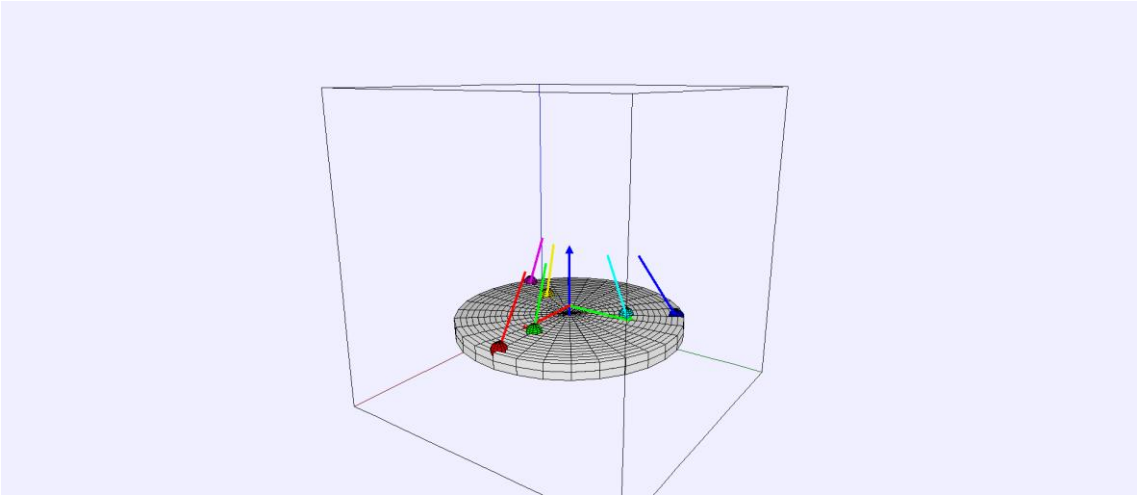
Amb les matrius P i R es construeix la matriu de transformació T:

$$T = \begin{bmatrix} R & P \\ 0_{1x3} & 1 \end{bmatrix}$$

Finalment, la matriu B, que conté les coordenades generals dels punts d'ancoratge, s'obté mitjançant:

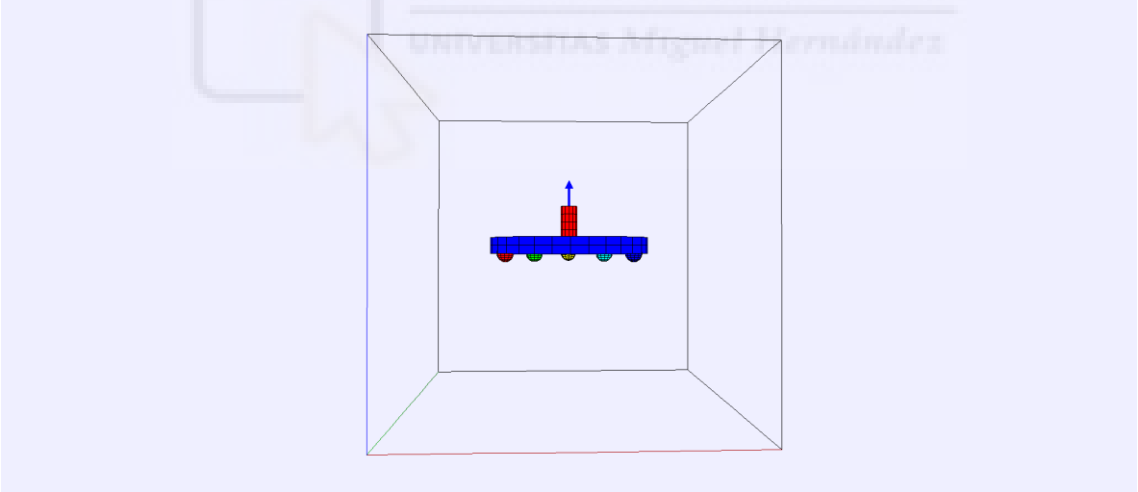
$$B = T \cdot b$$

Tant la base, com els punts d'ancoratge fixos i les barres formen part del conjunt "grup fix", un conjunt que no es mourà en tota la simulació.



Il·lustració 14: Representació del grup fix del robot.

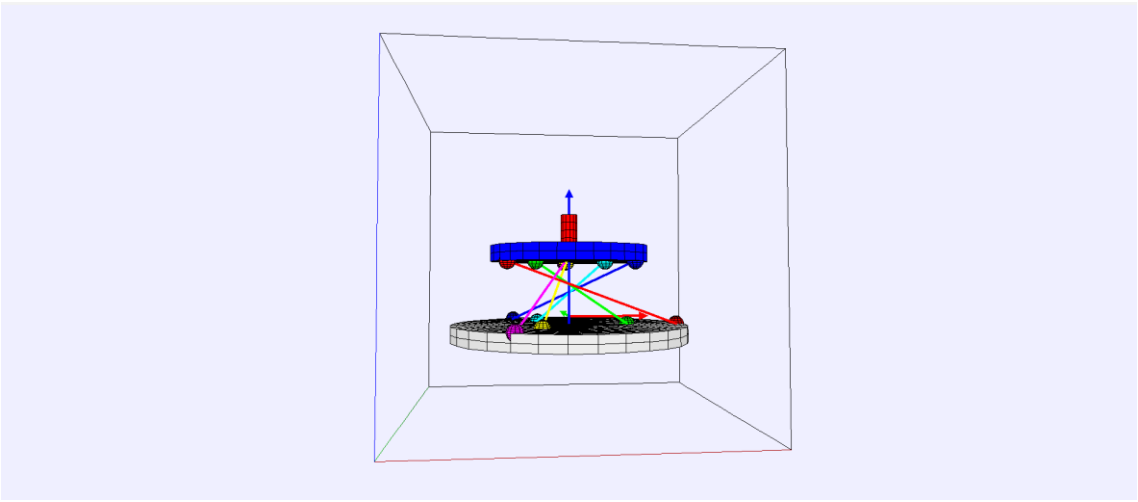
En canvi, el "grup mòbil" està conformat per la base mòbil (un cilindre blau), els punts d'ancoratge mòbils (representats per esferes) i la ferramenta (un cilindre roig). Cadascuna d'aquestes figures té el seu tamany definit respecte a les coordenades locals del grup mòbil. Així, quan el grup mòbil es desplaça, la base roman sempre situada al seu centre.



Il·lustració 15: Representació del grup mòbil del robot.

Una vegada representat tot el robot, cal definir-ne el moviment. Per a poder realitzar moviments lineals bàsics, la posició del grup mòbil està definida per les variables x, y, z. Respecte a l'orientació, aquesta es modela mitjançant tres angles d'Euler (alpha, beta, gamma), corresponents als eixos X, Y i Z respectivament. Amb tot açò, el robot queda

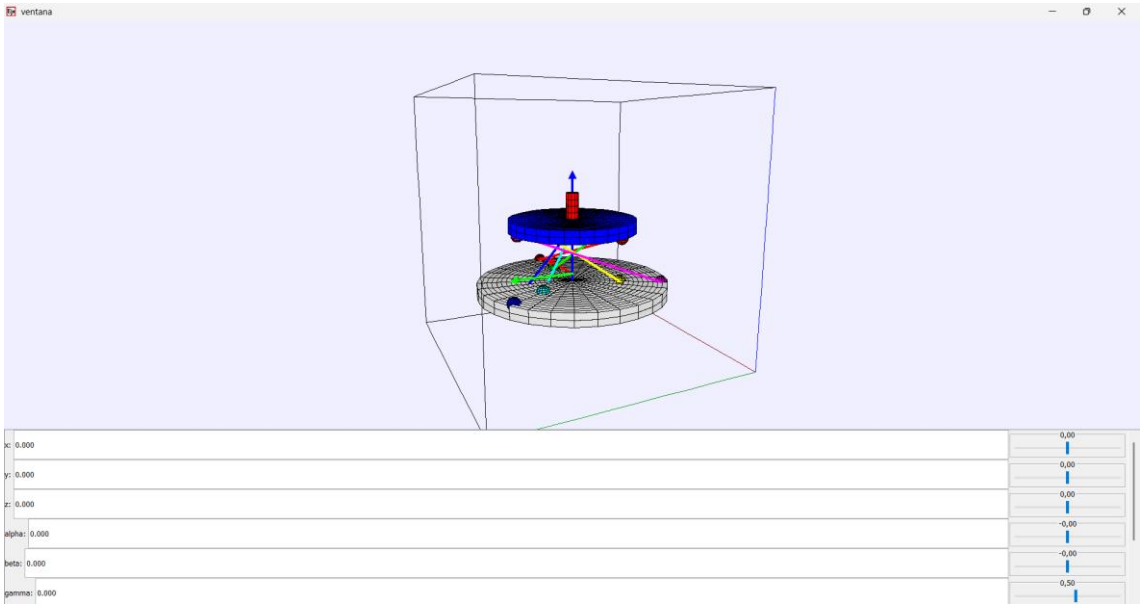
representat i el seu moviment es pot apreciar quan es modifiquen els valors de les variables esmentades.



Il·lustració 16: Representació completa del robot.

3.3.PROGRAMACIÓ DE LA INTERFICIE

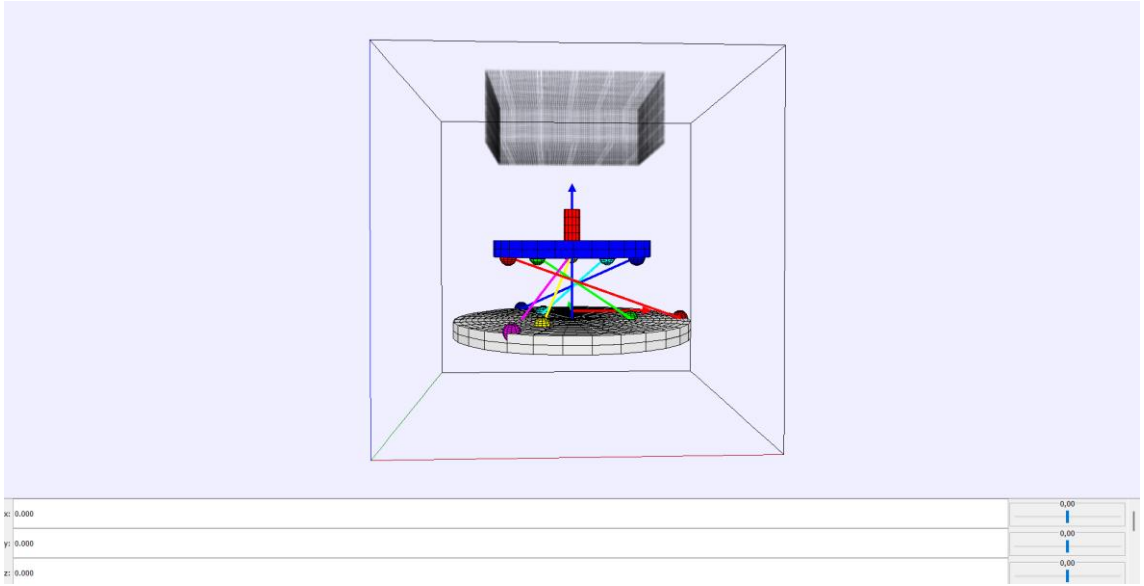
Amb tot el que s'ha presentat fins ara, per veure el moviment del robot, es necessari canviar els valors de les variables x, y, z, alpha, beta i gamma des de la pròpia programació. Per a major comoditat, es programa una interfície. Aquesta consta d'un camp numèric i d'una lliscadora per a cadascuna de les variables, ambdós elements ja preestablits en EJS. Amb el camp numèric es pot modificar directament el valor de les variables, mentre que amb la lliscadora es pot veure el moviment del robot amb més fluïdesa.



Il·lustració 17: Vista de la interfície programada.

3.4.SIMULACIÓ DEL BRUT

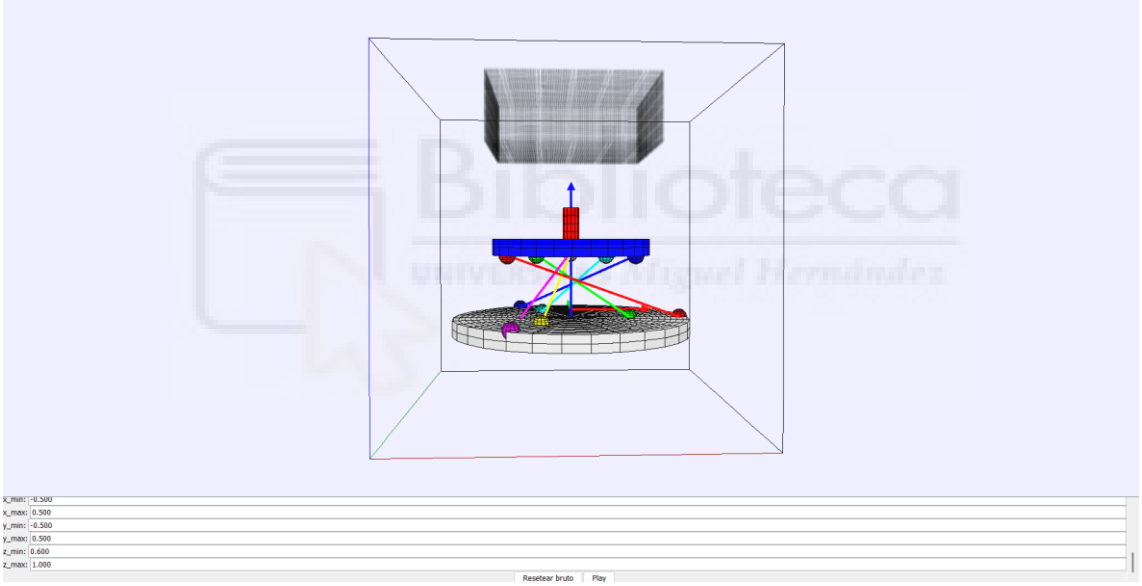
Amb el robot i el seu moviment ja representats, el següent pas és representar el brut i com interactua amb la ferramenta del robot. El brut està format per un conjunt de punts, un element ja definit en EJS al qual només cal passar-li la matriu que conté les coordenades de cada punt. Aquesta matriu s'anomena "nube_mecanizada".



Il·lustració 18: Vista del brut a mecanitzar.

La matriu es construeix amb la funció crearBruto() (explicada a l'apèndix 7.4.4), que depèn únicament dels límits establits per l'usuari. L'usuari definirà la mida del brut i, en executar la simulació, el programa cridarà a la funció crearBruto(), que calcularà quants punts caben en l'espai definit i construirà una matriu de tres columnes i tantes files com punts hi haja.

Per a major comoditat de l'usuari, a la interfície s'ha afegit un camp numèric que modifica el valor de cadascuna de les variables que defineixen la mida del brut. A més, també s'ha afegit un botó que reconstruirà el brut (en cas que l'usuari haja eliminat una part incorrectament) i un altre botó per iniciar o parar la simulació.



Il·lustració 19: Vista dels botons de la interfície.

Una vegada representat el brut, la interacció amb la ferramenta es calcula comprovant si algun dels punts del brut toca la ferramenta. Si és així, el punt s'elimina de la matriu i la representació s'actualitza.

En aquest treball, la interacció que s'ha programat és purament geomètrica, és a dir, una comprovació d'interferència mecànica entre el brut i la ferramenta, i no s'han modelat forces d'interacció, perquè aquesta part es deixa per a una possible continuació futura d'aquest Treball de Fi de Grau.

Tornant a la interacció geomètrica programada en aquest treball, aquesta es realitza a la pestanya “Relaciones fijas” (aquesta està explicada amb mes detall a l’apèndix 7.3). El codi d’aquesta pestanya s’executa cada vegada que s’actualitza la simulació. Així, quan es realitza un moviment, aquest codi s’executa per a cada increment de les variables.

El primer pas del codi de “Relaciones fijas” és seleccionar un punt del brut (el programa obté la seua posició en X, Y i Z). A continuació, el programa transforma les coordenades del punt des del sistema de referència global (P_{nube_global}) al sistema de referència mòbil (P_{local}) mitjançant les equacions següents:

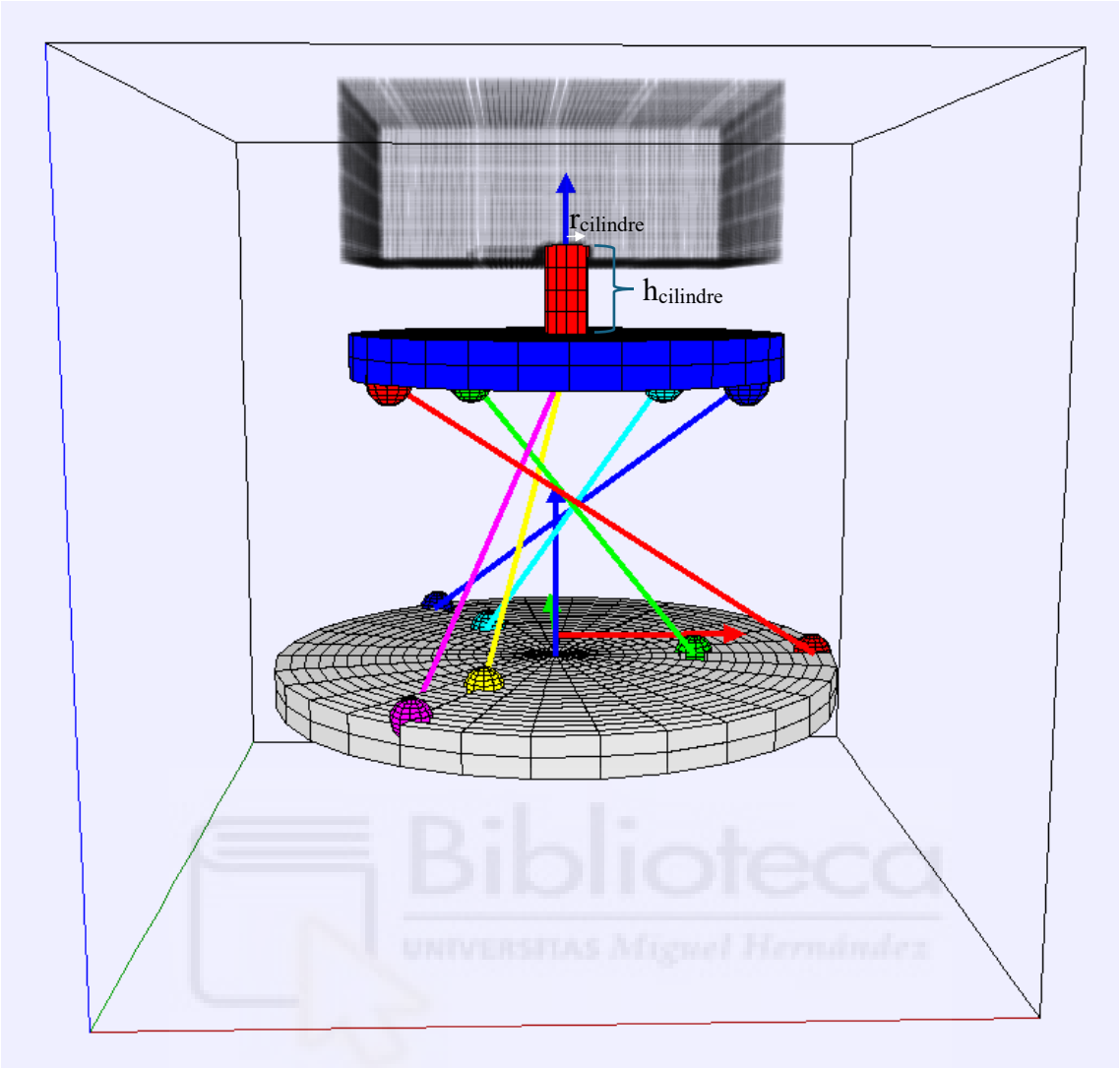
$$P_{nube\_global} = T \cdot P_{local}$$
$$P_{local} = invT \cdot P_{nube\_global}$$

on invT és la matriu inversa de T.

Amb les coordenades locals del punt, el programa ja pot comprovar si el punt toca la ferramenta. La comprovació es fa mitjançant les dues condicions següents:

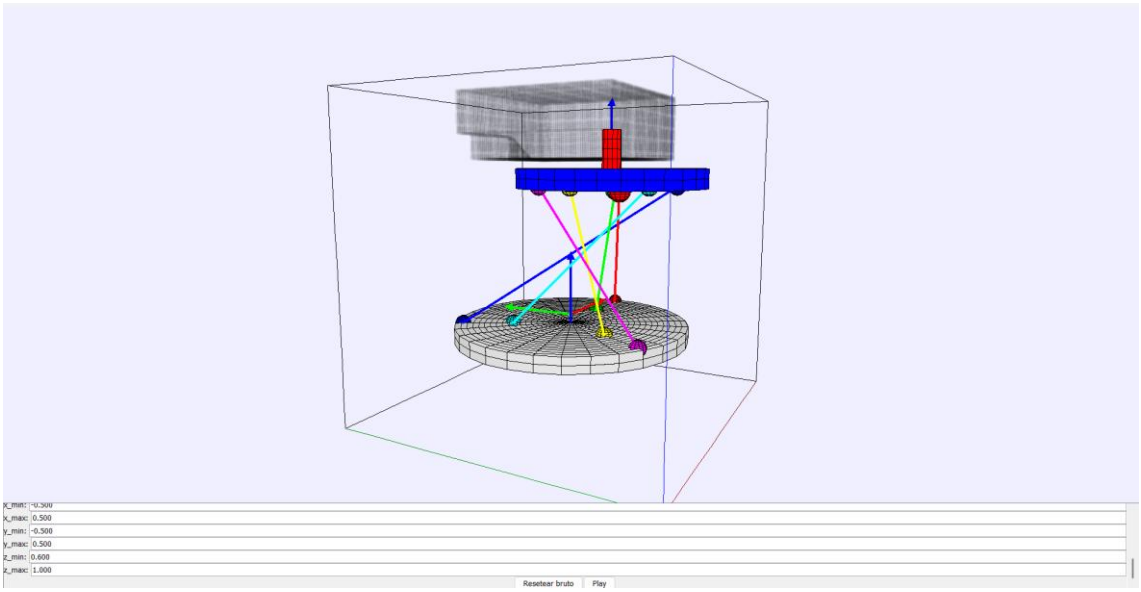
$$x^2 + y^2 \leq {r_{cilindre}}^2$$
$$0 \leq z \leq h_{cilindre}$$

On x, y, z són les coordenades del punt del brut considerat, expressat en coordenades locals de la ferramenta.



Il·lustració 20: Vista de les variables que intervenen en la comprovació de la interacció ferramenta-brut.

Si el punt compleix ambdues condicions, es considera en contacte amb la ferramenta i, per tant, s'elimina. Comprovat el punt, el programa seleccionara el següent punt. El procés es repetirà per a tots els punts que conformen el brut.



Il·lustració 21: Vista de la interacció ferramenta-brut.

3.5.TRAJECTÒRIA LINEAL I CIRCULAR

Ara que el robot es pot moure i interactua amb el brut, cal programar les trajectòries lineals i circulars que descriuen el moviment. En una primera versió, el moviment del robot era instantani, és a dir, quan s'indicava una nova posició en les caixes numèriques de la interfície, el robot es movia instantàniament a la posició indicada, en lloc de mostrar una trajectòria fent el moviment vers a la nova posició (excepte si s'utilitzen les lliscadores, que proporcionen un moviment més progressiu).

El tipus de trajectòria dependrà del valor de la variable "tray_lineal": si és "true", el moviment serà lineal; si és "false", serà circular. El codi que programa les trajectòries es troba a la finestra "Evolución" (aquesta finestra s'explica amb mes detall a l'apèndix 7.2). En ambdós casos, cal definir les variables targetX, targetY, targetZ, targetA, targetB i targetG, que representen el valor final de cada coordenada.

TRAJECTÒRIA LINEAL

Per a trajectòries lineals, el nou valor de cada variable s'obté mitjançant interpolació lineal. Per exemple, a l'eix X:

$$x = x_0 + (targetX - x_0) \cdot \frac{t}{t_{fin}}$$

on x_0 és el valor inicial de "x", t representa el temps i t_{fin} és el temps total (de moment amb un valor arbitrari).

TRAJECTÒRIA CIRCULAR

Per a trajectòries circulars en el pla XY, els valors de "x" i "y" es calculen amb:

$$x = a + r \cdot \cos(\theta)$$
$$y = b + r \cdot \sin(\theta)$$

on (a, b) son les coordenades del centre del cercle, r és el seu radi i θ és l'angle. El radi i el centre es poden determinar mitjançant l'equació d'una circumferència:

$$(x - a)^2 + (y - b)^2 = r^2$$

A partir d'aquest plantejament es resol un sistema d'equacions amb tres punts coneguts: el punt inicial "x", el punt final "targetX" i un punt intermedi "xm" (i el seus respectius "y"):

$$(x - a)^2 + (y - b)^2 = r^2$$
$$(targetX - a)^2 + (targetY - b)^2 = r^2$$
$$(xm - a)^2 + (ym - b)^2 = r^2$$

Els angles inicial i final es calculen així:

$$\theta_{inicio} = \tan^{-1}\left(\frac{y - b}{x - a}\right)$$

$$\theta_{final} = \tan^{-1}\left(\frac{targetY - b}{targetX - a}\right)$$

I el valor instantani de θ s'obté mitjançant interpolació:

$$\theta = \theta_{inicio} + \left(\frac{t}{t_{fin}}\right) \cdot \left(\theta_{final} - \theta_{inicio}\right)$$

El càlcul de la trajectòria continua mentre t siga menor que t_{fin}.

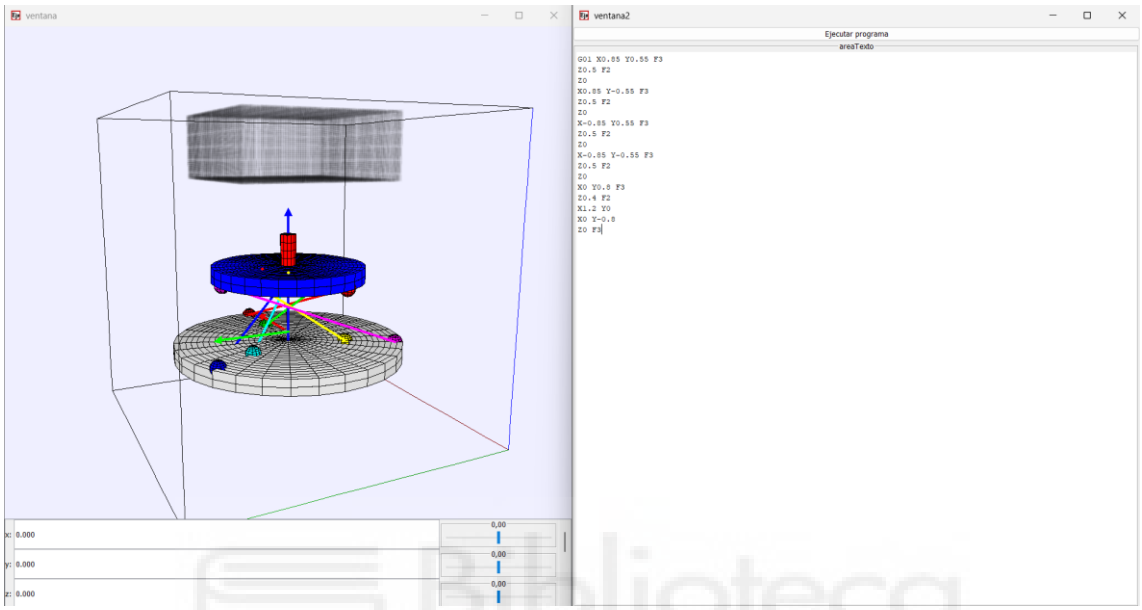
3.6.INTÈRPRET DE CODI G

S'ha creat un espai perquè l'usuari puga programar el moviment. Quan vol provar el codi, prem el botó d'inici i el programa llig el contingut escrit amb la funció leerPrograma(), que està explicada a l'apèndix 7.4.2. Aquesta funció processa el codi línia a línia i l'executa abans de passar a la següent.

El primer que comprova és si la línia conté la lletra "G", que indica si es tracta d'un moviment lineal o circular. Si es detecta "G01", es dona a la variable "tray_lineal" el valor "true" , i els següents moviments seran lineals fins que s'indique el contrari. Per als moviments circulars s'utilitza "G09". Aquestes ordres són modals, és a dir, es mantenen actives fins que es reemplacen.

Hi ha altres maneres de programar trajectòries circulars amb codi G, com les funcions G2 i G3 (que requereixen indicar el punt de destí i el radi o les coordenades del centre del cercle), o G8 (que fa una trajectòria circular tangent a la trajectòria prèvia), però en aquest treball s'ha optat per la programació de cercles amb G09 perquè aquesta funció ofereix una programació més semblant a la que altres llenguatges de programació de robots ofereixen, com el llenguatge RAPID de ABB, que requereix indicar un punt intermedi de la circumferència.

La resta del codi assigna valors a les variables indicades. Per exemple, si s'escriu "X 0.5", la funció assignarà "0.5" a la variable "x". Com que el moviment no s'executa fins que es processa tota la línia, es poden modificar diverses variables en una sola instrucció, cosa que permet moviments diagonals.



Il·lustració 22: Vista de l'espai destinat a la programació de l'usuari.

3.7.FXXX, CANVI DE FERRAMENTA I SINGULARITATS

La simulació ja funciona correctament amb les funcionalitats descrites en els apartats anteriors. Per tal d'optimitzar-ne el rendiment, s'ha decidit no representar tot el volum del brut, sinó únicament els punts que conformen la seua superfície, gràcies a la funció extraer_frontera(), descrita a l'apèndix 7.4.5. Aquesta estratègia evita carregar els punts interiors, cosa que accelera significativament l'execució de la simulació. A més, quan la ferramenta elimina un punt, els punts interns que queden exposats es tornen visibles, mantenint així la coherència visual del procés de mecanitzat.

També s'ha implementat la possibilitat de modificar la velocitat de la ferramenta mitjançant la comanda F (per exemple, F3). El codi calcula la distància que la ferramenta ha de recórrer i la divideix per la velocitat indicada per determinar el valor de tfin, que és el temps total que ha de durar l'animació.

CÀLCUL DE LA DISTÀNCIA

Per a trajectòries circulars, la distància és calcula com la longitud d'un arc de circumferència:

$$dist\grave{a}ncia = (\theta_{fin} - \theta_{ini}) \cdot r$$

Per a trajectòries lineals, es considera la diferència entre les coordenades inicials i finals, incloent tant posició com orientació. La fórmula emprada és:

$$distancia = \sqrt{\begin{array}{r}(x - targetX)^2 + (y - targetY)^2 + (z - targetZ)^2 + \\ (\alpha - targetA)^2 + (\beta - targetB)^2 + (\gamma - targetG)^2\end{array}}$$

A més, s'ha afegit la funcionalitat de canvi de ferramenta. Quan l'usuari escriu la lletra T, el programa interpreta que es vol realitzar un canvi de ferramenta. El valor numèric que acompanya a la lletra T es la posició de la nova ferrament dins de la llista de ferramentes disponibles.

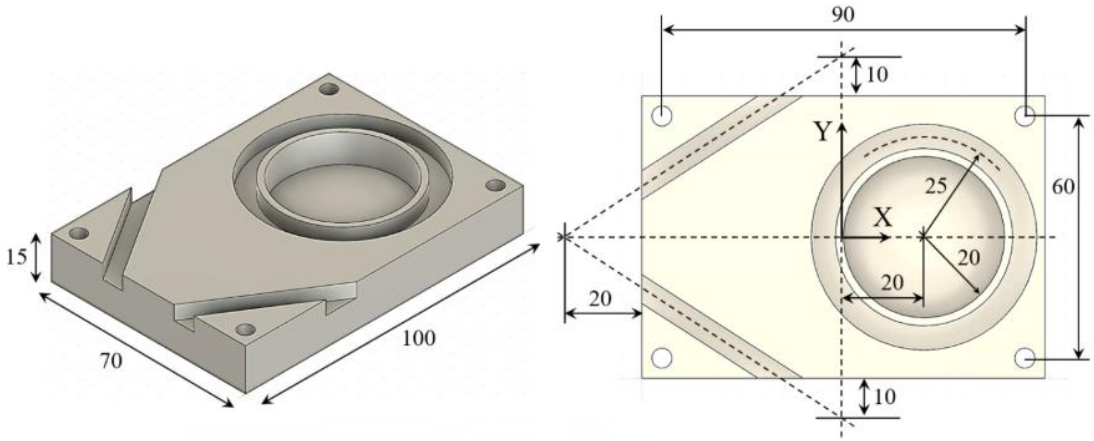
D'altra banda, s'ha implementat la comprovació del determinant de la matriu Jacobiana, amb l'objectiu de detectar punts singulars durant la trajectòria. Aquesta comprovació s'ha dut a terme amb la funció deterJac(), descrita a l'apèndix 7.4.7. El determinant es mostra tant en un quadre de text, que mostra en cada instant el se valor, com un gràfic per visualitzar la seua evolució durant la simulació.

Finalment, s'ha modificat la representació del brut quan la simulació està aturada per fer-la més visual i clara per a l'usuari. Quan el robot està en repòs, el núvol de punts que representa la frontera del brut es mostra amb punts més grans i amb un cert nivell de transparència, fet que permet visualitzar l'interior del volum mecanitzat. Quan s'executa una trajectòria de mecanitzat, els punts es redueixen i es desactiva la transparència, ja que aquest efecte penalitza el rendiment gràfic a causa dels càlculs addicionals que ha de fer el motor per gestionar les oclusions entre punts.

4. EXEMPLES

4.1. PRIMER EXEMPLE

En aquest primer exemple es simula una peça senzilla que mostra les trajectòries rectes i circulars que es capaç de fer el robot. La peça desitjada es la següent:



Il·lustració 23: Imatge de la peça desitjada, enunciada a les pràctiques de FAO.

En aquest exemple s'ometrà la caixera circular, ja que aquest TFG no ha contemplat la programació de caixeres.

DIMENSIONS DEL BRUT

Les coordenades que delimiten el brut inicial són les següents:

X_min: -1

X_max: 1

Y_min: -0.7

Y_max: 0.7

Z_min: 0.6

Z_max: 0.75

CODI DE LA TRAJECTÒRIA PROGRAMADA

A continuació es mostra el codi introduït per a executar la simulació.

C70

```
G01 X0.85 Y0.55 F3
Z0.5 F2
Z0
X0.85 Y-0.55 F3
Z0.5 F2
Z0
X-0.85 Y0.55 F3
Z0.5 F2
Z0
X-0.85 Y-0.55 F3
Z0.5 F2
Z0
X0 Y0.8 F3
Z0.4 F2
X1.2 Y0
X0 Y-0.8
Z0
G01 X-0.8 Y0
Z0.4
G09 X0.05 Y0 I-0.4 J0.4
G09 X-0.8 Y0 I-0.4 J-0.4
G01 Z0
```

Aquest conjunt d'instruccions simula el moviment de la ferramenta en diferents punts del bloc. La trajectòria descriu una sèrie de forats o perforacions en diferents punts, seguida d'una trajectòria més llarga cap a la part central i inferior del bloc.

RESULTAT DE LA SIMULACIÓ

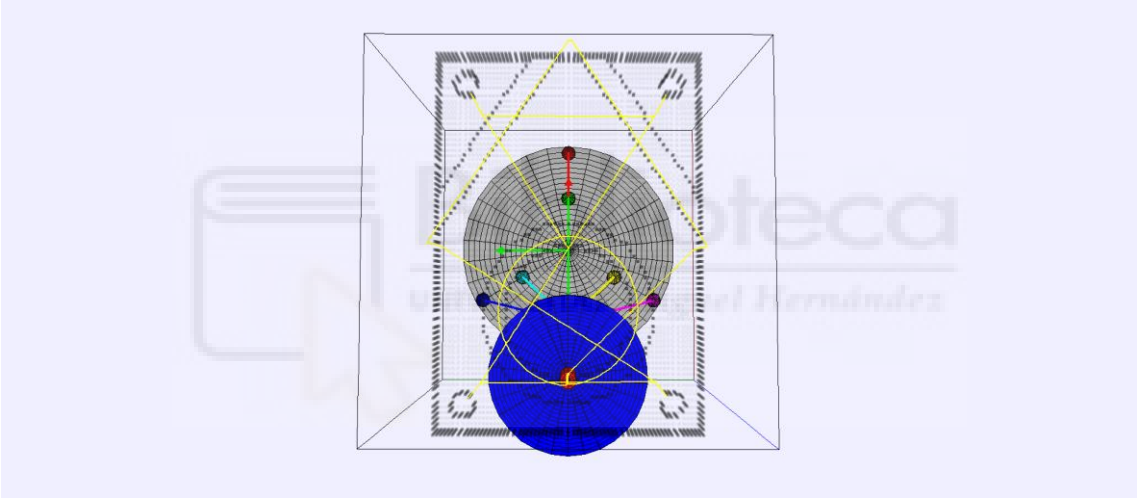
El resultat visual d'aquest codi mostra un conjunt de perforacions rectangulars als quatre extrems del bloc, juntament amb una trajectòria central que travessa el material en direcció longitudinal. Es pot observar com la ferramenta:

- Baixa repetidament per realitzar mecanitzats verticals,

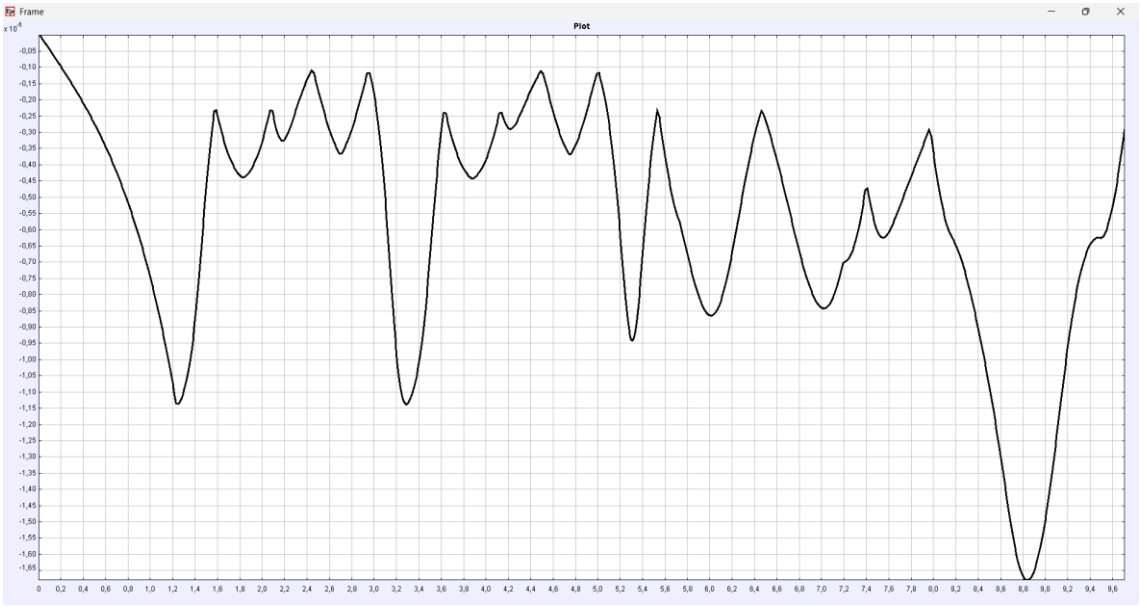
- Es desplaça de manera fluïda entre punts,

- I respecta els valors de velocitat especificats en cada segment.

Aquest primer exemple demostra l'eficàcia del simulador a l'hora de reproduir trajectòries definides per l'usuari i visualitzar el procés de mecanitzat amb alta fidelitat i resposta immediata a les instruccions del codi G.



Il·lustració 24: Resultat del primer exemple.



Il·lustració 25: Gràfica del determinant de la Jacobiana inversa del primer exemple.

4.2. SEGON EXAMPLE

Aquest segon exemple té com a objectiu mostrar la capacitat del robot per adoptar múltiples orientacions durant el moviment de la ferramenta, mantenint el control sobre els seus sis graus de llibertat. A diferència del primer exemple, en aquest cas la trajectòria programada inclou variacions significatives en els angles d'Euler, cosa que permet observar com el sistema respon davant canvis d'orientació en temps real.

DIMENSIONS DEL BRUT

Les dimensions que té el brut en aquest segon exemple són:

X_min: -0.5

X_max: 0.5

Y_min: -0.5

Y_max: 0.5

Z_min: 0.6

Z_max: 1

CODI DE LA TRAJECTÒRIA PROGRAMADA

A continuació es mostra el codi introduït per a executar la simulació.

C70 T4

G01 X0.6 F3

A30

Y0.100

Z0.29

X-0.6

A-30

Y-0.100

X0.6

A0.0

Y0.6

X0

B30

X-0.100

Y0.15

Y0.6

B-30

X0.100

Y0.15

Y0.6

B0

X0.6

Y-0.6

B-30

X0.100

Y-0.15

Y-0.6

B30

X-0.100

Y-0.15

Y-0.6

B0.0

Z0.25

X0.12

Y-0.15

X0.22

Y-0.5

X0.32

Y-0.15

X0.42

Y-0.5

X0.52

Y-0.15

X0.6

Y0.6

X0.12

Y0.15

X0.22

Y0.5

X0.32

Y0.15

X0.42

Y0.5

X0.52

Y0.15

X0.6

Y0.6

X-0.12

Y0.15

X-0.22

Y0.5

X-0.32

Y0.15

X-0.42

Y0.5

X-0.52

Y0.15

X-0.6

Y0.6

Y-0.6

X-0.22

Y-0.15

X-0.32

Y-0.5

X-0.42

Y-0.15

X-0.52

Y-0.5

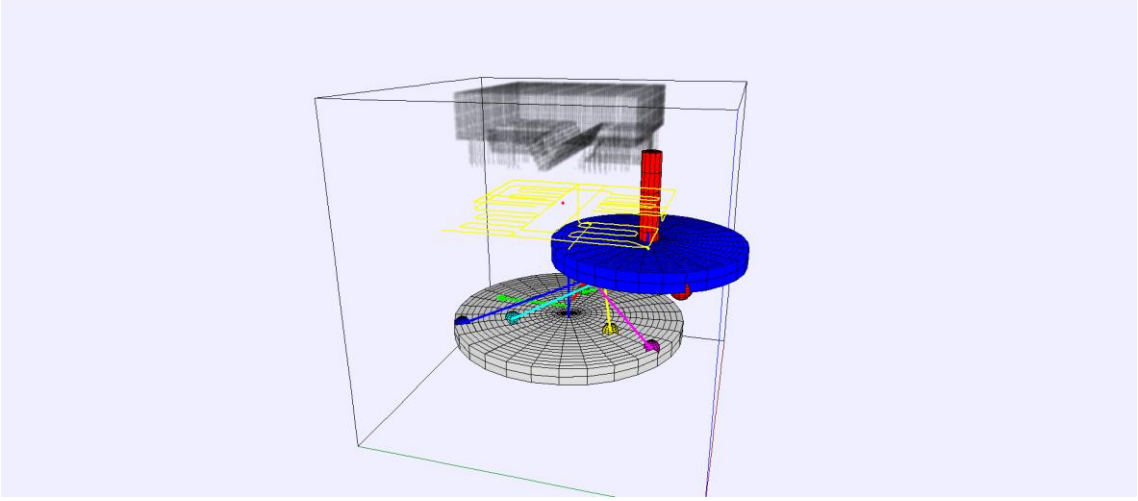
X-0.6 Y-0.6

Aquest codi mostra un conjunt de trajectòries planes que travessen el brut formant dues piràmides invertides perpendiculars entre si, com si es tractara d'un relleu creuat a través del material.

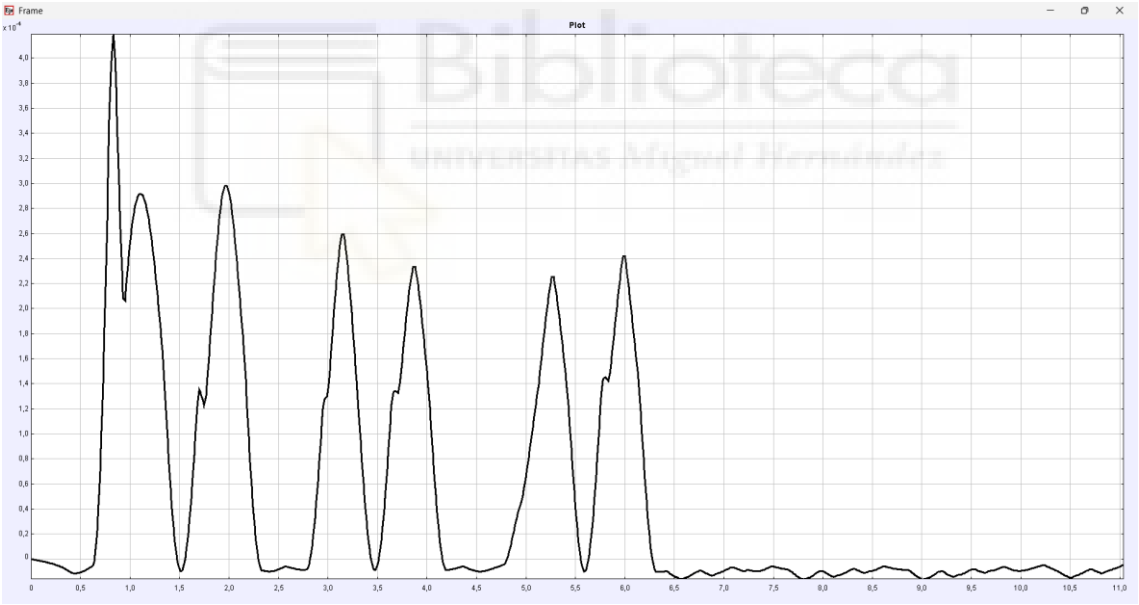
RESULTAT DE LA SIMULACIÓ

La simulació mostra clarament com el robot és capaç de mantenir la precisió dels moviments fins i tot quan varia l'orientació, i com s'adapta per executar les trajectòries sense problemes de control o col·lisions. El model gràfic reflecteix fidelment tant el moviment de translació com els canvis d'orientació de la ferramenta, permetent a l'usuari observar l'efecte combinat sobre el brut.

Aquesta prova demostra la versatilitat del simulador a l'hora de representar moviments amb graus de llibertat complets i reforça la seua utilitat com a eina educativa i de validació prèvia en entorns de mecanitzat complex.



Il·lustració 26: Resultat del segon exemple.



Il·lustració 27: Gràfica del determinant de la Jacobiana inversa del segon exemple.

5. CONCLUSIONS I FUTURS TREBALLS

La simulació desenvolupada acompleix satisfactòriament l'objectiu principal establit a l'apartat 1.2 Objectius i contribució. El sistema implementat permet representar de manera precisa el funcionament d'un robot paral·lel de sis graus de llibertat, mostrant tots els elements estructurals fonamentals del mecanisme.

Concretament, la simulació visualitza correctament les dues bases del robot (fixa i mòbil), els punts d'ancoratge i les sis barres que controlen la posició i orientació de la plataforma mòbil. Posteriorment, el programa genera el brut de mecanitzat a partir de les dimensions definides per l'usuari, i l'integra visualment dins de l'espai de treball.

A més, el sistema és capaç d'interpretar instruccions en llenguatge G simplificat proporcionades per l'usuari, executant els moviments corresponents de la ferramenta i mostrant en temps real la seua interacció amb el brut. Aquest comportament comprén l'entrada i eixida de la ferramenta dins del material, així com l'eliminació progressiva dels punts afectats pel mecanitzat.

Malgrat les fites aconseguides, el simulador actual es centra exclusivament en l'àmbit de la cinemàtica del robot. Per tant, una línia de desenvolupament especialment rellevant per a futurs treballs seria la incorporació de la dinàmica del sistema. Per exemple, es podria modelitzar la resistència que ofereix el material del brut en el moment de contacte amb la ferramenta, afegint així una resposta física més realista. Igualment, es podria estudiar el control del robot tenint en compte la interacció dinàmica entre la ferramenta i el material mecanitzat, així com la simulació de l'escalfament de la ferramenta com a conseqüència del treball realitzat.

Altres línies de millora possibles serien:

- Implementar una ferramenta amb punta esfèrica (en lloc de plana com en la versió actual), per tal de simular nous escenaris de mecanització.
- Introduir el càlcul i compensació del radi de la ferramenta, amb l'objectiu de millorar la precisió del mecanitzat simulat i acostar-se a un entorn industrial real.

- Afegir la detecció automàtica de col·lisions entre les sis barres que mouen el robot, per evitar configuracions físicament invàlides.
- Desenvolupar una planificació optimitzada de trajectòries, que permeta evitar col·lisions i singularitats, millorant així l'estabilitat i eficiència del procés simulat.

Aquestes ampliacions obririen la porta a una simulació molt més completa, capaç d'integrar no només els aspectes cinemàtics, sinó també les condicions físiques i operatives reals del mecanitzat. Això convertiria el simulador en una eina encara més potent i versàtil, tant per a l'àmbit acadèmic com per a aplicacions professionals en el camp de la robòtica aplicada al mecanitzat.

6. BIBLIOGRAFIA

Alvares, A. J., Rodriguez, E., Jaimes, C. I. R., Toquica, J. S., & Ferreira, J. C. (2020). STEP-NC architectures for Industrial Robotic Machining: Review, implementation and validation. IEEE Access, 8, 152592-152610. (https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9170533)

Autodesk Inc. (2025), "Autodesk Products Fusion 360 Page", URL: https://www.autodesk.com/products/fusion-360/overview?us_oa=dotcom-us&us_si=6c8ff03f-3e6b-4fa5-a379-2851a04cddb3&us_st=fusion%20360&us_pt=NINVFUS, [Data de consulta: 26 de maig de 2025]

Esquembre, F. (2015). Facilitating the creation of virtual and remote laboratories for science and engineering education. Ifac-Papersonline, 48(29), 49-58. (https://www.sciencedirect.com/science/article/pii/S2405896315024702)

Fagor Automation (2016), "Fagor Automation Home Page", URL: https://www.fagorautomation.com/es/disponible-la-nueva-version-del-simulador-cnc-gratuito, [Data de consulta: 26 de maig de 2025]

Korkmaz, M. E., & Gupta, M. K. (2023). A state of the art on simulation and modelling methods in machining: future prospects and challenges. Archives of Computational Methods in Engineering, 30(1), 161-189. (https://link.springer.com/content/pdf/10.1007/s11831-022-09794-9.pdf)

Pan, J., Fu, Z., Xiong, J., Lei, X., Zhang, K., & Chen, X. (2022). RobMach: G-Code-based off-line programming for robotic machining trajectory generation. The International Journal of Advanced Manufacturing Technology, 1-15. (https://link.springer.com/article/10.1007/s00170-021-08082-3)

Peidro, A., Reinoso, O., Gil, A., Marín, J. M., & Paya, L. (2018). A method based on the vanishing of self-motion manifolds to determine the collision-free workspace of redundant robots. *Mechanism and Machine Theory*, *128*, 84-109.

Raghavan, M. (1993). The Stewart platform of general geometry has 40 configurations. (https://asmedigitalcollection.asme.org/mechanicaldesign/article-abstract/115/2/277/417494/The-Stewart-Platform-of-General-Geometry-Has-40)

Russo, M., Zhang, D., Liu, X. J., & Xie, Z. (2024). A review of parallel kinematic machine tools: Design, modeling, and applications. International Journal of Machine Tools and Manufacture, 196, 104118. (https://www.sciencedirect.com/science/article/pii/S089069552400004X)

Scerba, M., Sevcikova, R., Al-Rabeei, S., Mir, S., & Daneshjo, N. (2025). Simulation Modeling of Kinematic Structures of Parallel Mechanisms. Engineering, Technology & Applied Science Research, 15(2), 20714-20721. (https://mail.etasr.com/index.php/ETASR/article/view/9898)

Xu, X., Outeiro, J., Zhang, J., Xu, B., Zhao, W., & Astakhov, V. (2021). Machining simulation of Ti6Al4V using coupled Eulerian-Lagrangian approach and a constitutive model considering the state of stress. Simulation Modelling Practice and Theory, 110, 102312. (https://www.sciencedirect.com/science/article/abs/pii/S1569190X21000344?cas

a_token=vtDenb_a5KMAAAAA:fA-B9wswIRhCnuHPeWwCZLMB8E6MRVIjswjvZ_RwqRdVsi261_oLGb5FQZ0MM0VZznIsVUYe5m8)

Zhang, D., Wang, L., & Lang, S. Y. (2005). Parallel kinematic machines: design, analysis and simulation in an integrated virtual environment. (https://www.diva-portal.org/smash/get/diva2%3A1090203/FULLTEXT01.pdf)

7. APÈNDIX DEL CODI

En aquesta secció s'inclou el codi Java utilitzat per programar la simulació, juntament amb una breu explicació de cadascuna de les parts. El codi està separat per les diferents finestres en què ha estat programat dins de l'entorn EJS.

7.1.CODI DE LA FINESTRA "INICIALIZACIÓN"

```
1  actualizarT();
2  crearBruto;
3  calcularL(x, y, z, alpha, beta, gamma);
4  deterJac();
5  color_nube = new Color(0,0,0,10);
6  Tools = new double[][] {
7  {0.1,0.2}, // radio 0.1, altura 0.2 (T0)
8  {0.05,0.4}, // radio 0.05, altura 0.4 (T1)
9  {0.2,0.1}, // radio 0.1, altura 0.1 (T2)
10 {0.05, 0.25}, //radio 0.05, altura 0.25 (la original)(T3)
11 {0.05, 0.5}, //radio 0.05, altura 0.5 (T4)
12 };
```

Aquest codi s'executa una única vegada, just a l'inici de la simulació. El seu objectiu és preparar l'entorn i generar les estructures necessàries per al correcte funcionament del sistema.

La línia 1 crida la funció actualizarT(), que calcula la matriu de transformació T i la matriu B amb les coordenades globals dels punts d'ancoratge mòbils. La línia 2 crida la funció crearBruto(), que construeix la matriu de punts del brut a mecanitzar. De manera similar, les línies 3 i 4 s'encarreguen de calcular el vector de longituds de les barres Lj i el determinant de la Jacobiana, respectivament, amb l'objectiu de detectar possibles singularitats durant la simulació.

A continuació, es defineix el color inicial del brut (color_nube), amb un nivell baix d'opacitat per facilitar la visualització inicial del seu interior.

Finalment, s'inicialitza la variable Tools, que conté les diferents ferramentes disponibles per a la simulació.

Aquest conjunt d'instruccions garanteix que, des del principi, tant el robot com el brut estiguen correctament definits i representats a l'entorn gràfic

7.2.CODI DE LA FINESTRA "EVOLUCIÓN"

```
1  if (tray_lineal) {
2     double[] inicio = {
3        x0, y0, z0, alpha0, beta0, gamma0
4     };
5     double[] destino = {
6        targetX, targetY, targetZ, targetA, targetB, targetG
7     };
8
9    double distancia = 0;
10    for (int i = 0; i < 6; i++) {
11    double diferencia = inicio[i] - destino[i];
12       distancia += diferencia * diferencia;
13    }
14    distancia = Math.sqrt(distancia);
15
16    t_fin = distancia / velocidadH;
17
18    if (t_fin == 0) {
19       iterador_programa++;
20       _pause();
21       x0 = x;
22       y0 = y;
23       z0 = z;
24       alpha0 = alpha;
25       beta0 = beta;
26       gamma0 = gamma;
27       leerPrograma();
28    }
29    else {
30       x = x0 + (targetX - x0) * t / t_fin;
31       y = y0 + (targetY - y0) * t / t_fin;
32       z = z0 + (targetZ - z0) * t / t_fin;
33       alpha = alpha0 + (targetA - alpha0) * t / t_fin;
34       beta = beta0 + (targetB - beta0) * t / t_fin;
35       gamma = gamma0 + (targetG - gamma0) * t / t_fin;
36
```

```
37          if (t >= t_fin) {
38              t = t_fin;
39              x = x0 + (targetX - x0) * t / t_fin;
40              y = y0 + (targetY - y0) * t / t_fin;
41              z = z0 + (targetZ - z0) * t / t_fin;
42              alpha = alpha0 + (targetA - alpha0) * t / t_fin;
43              beta = beta0 + (targetB - beta0) * t / t_fin;
44              gamma = gamma0 + (targetG - gamma0) * t / t_fin;
45
46              iterador_programa++;
47              _pause();
48              x0 = x;
49              y0 = y;
50              z0 = z;
51              alpha0 = alpha;
52              beta0 = beta;
53              gamma0 = gamma;
54              leerPrograma();
55          }
56      }
57
58      t = t + dt;
59  } else {
      // La ecuación de la circunferencia es(x0-a)^2 + (y0-b)^2 = r^2, donde (a, b)
60  son las coordenadas del centro de la circunferencia
61    // Esta ecuacion se cumple para todos los valores de x e y.
      // Si establecemos un sistema de ecuaciones con la ecuación para los 3 valores
62  de x e y (x0, xm y targetX), podemos calcular
63    // el centro de la circunferencia y su radio.
64
65    double A1 = 2 * (x0 - xm);
66    double B1 = 2 * (y0 - ym);
67    double C1 = x0 * x0 + y0 * y0 - xm * xm - ym * ym;
68
69    double A2 = 2 * (targetX - xm);
70    double B2 = 2 * (targetY - ym);
71    double C2 = targetX * targetX + targetY * targetY - xm * xm - ym * ym;
72
73    // Resolver el sistema de ecuaciones para (a, b)
74    double det = A1 * B2 - A2 * B1;
75      a = (C1 * B2 - C2 * B1) / det;
76      b = (A1 * C2 - A2 * C1) / det;
77
78    // Calcular radio: (x0-a)^2 + (y0-b)^2 = r^2
      double r = Math.sqrt((targetX - a) * (targetX - a) + (targetY - b) * (targetY -
79  b));
80
81      distanciaR = Math.PI * r;
82      t_fin = distanciaR / velocidadH;
83
```

```
84      if (t_fin == 0) {
85          iterador_programa++;
86          _pause();
87          x0 = x;
88          y0 = y;
89          z0 = z;
90          alpha0 = alpha;
91          beta0 = beta;
92          gamma0 = gamma;
93          leerPrograma();
94      }
95
        // Con el centro y el radio, podemos usar las ecuaciones polares para obtener
96  los valores de X e Y.
97      // Falta obtener el valor del ángulo theta, que variará en función del tiempo.
98
99      // Calcular ángulos inicial y final
100     theta0 = Math.atan2(y0 - b, x0 - a);
101     thetaM = Math.atan2(ym - b, xm - a);
102     thetaF = Math.atan2(targetY - b, targetX - a);
103
104     // Ajustar la dirección del giro
        if ((thetaM > theta0 && thetaM < thetaF) || (thetaM < theta0 && thetaM >
105 thetaF)) {
106         // Si thetaM está entre theta0 y thetaF, la dirección es correcta
107     } else {
108         // Si no, invertir el sentido sumando 2π
109         if (thetaF < theta0) {
110             thetaF += 2 * Math.PI;
111         } else {
112             theta0 += 2 * Math.PI;
113         }
114     }
115
116     // Calcular el ángulo theta
117     theta = theta0 + (thetaF - theta0) * (t / t_fin); // Interpolación angular
118
119     // Ecuaciones polares
120     x = a + r * Math.cos(theta);
121     y = b + r * Math.sin(theta);
122
123     if (t >= t_fin) {
124         iterador_programa++;
125         _pause();
126         x0 = x;
127         y0 = y;
128         z0 = z;
129         alpha0 = alpha;
130         beta0 = beta;
131         gamma0 = gamma;
```

```
132         leerPrograma();
133     }
134
135     if (Math.abs(t - t_fin) <= 0.0001) {
136         t = t_fin;
137         x = x0 + (targetX - x0) * t / t_fin;
138         y = y0 + (targetY - y0) * t / t_fin;
139         z = z0 + (targetZ - z0) * t / t_fin;
140         alpha = alpha0 + (targetA - alpha0) * t / t_fin;
141         beta = beta0 + (targetB - beta0) * t / t_fin;
142         gamma = gamma0 + (targetG - gamma0) * t / t_fin;
143
144         iterador_programa++;
145         _pause();
146         x0 = x;
147         y0 = y;
148         z0 = z;
149         alpha0 = alpha;
150         beta0 = beta;
151         gamma0 = gamma;
152         leerPrograma();
153     }
154
155     t = t + dt;
156 }
```

Aquest codi s'executa contínuament mentre la simulació està activa. Controla els moviments del robot, tant lineals com circulars.

Primerament, es comprova si la trajectòria ha de ser lineal (tray_lineal == true) o circular. Segons aquest valor, s'executa el bloc de codi corresponent.

TRAJECTÒRIES LINEALS

Es defineixen dos vectors: "inicio" i "destino", que contenen les coordenades inicials i finals del moviment. A continuació, es calcula la distància euclidiana entre aquestes dues posicions (tenint en compte també l'orientació).

A partir de la distància i de la velocitat (velocidadH), es determina el temps total t_fin del moviment.

Finalment, es realitza la interpolació per a cada variable (posició i angles) en funció del temps t.

TRAJECTÒRIES CIRCULARS

Es resol un sistema d'equacions per obtenir les coordenades del centre del cercle (a, b) i el radi r. Seguidament, es calcula el recorregut total i, com en el cas anterior, el temps t_fin a partir de la velocitat.

Es determinen els angles inicial, intermedi i final del moviment mitjançant la funció atan2.

Finalment, es calcula l'angle actual θ i es representen les coordenades x i y mitjançant funcions polars.

En ambdós casos, quan t ≥ t_fin, es pausa la simulació, s'actualitzen les variables i es passa a la següent línia del programa G amb leerPrograma().

7.3.CODI DE LA FINESTRA "RELACIONES FIJAS"

```
1  actualizarT();
2  calcularL(x, y, z, alpha, beta, gamma);
3  deterJac();
4
5  Matrix P_local = new Matrix(4, 1);
6  Matrix invT = T.inverse();
7  nube_mecanizada = new Matrix(tam, 3);
8
9  double x_punto, y_punto, z_punto;
10 double radio = 0.5;
11 int contador2 = 0;
12
13 Matrix P_nube_global = new Matrix(4, 1);
14 P_nube_global.set(3, 0, 1);
15
16 double x_punto_local, y_punto_local, z_punto_local;
```

```
17
18  fila = tam;
19
20  for (int contador1 = 0; contador1 <= fila - 1; contador1++) {
21      x_punto = nube.get(contador1, 0);
22      y_punto = nube.get(contador1, 1);
23      z_punto = nube.get(contador1, 2);
24
25      P_nube_global.set(0, 0, x_punto);
26      P_nube_global.set(1, 0, y_punto);
27      P_nube_global.set(2, 0, z_punto);
28
29      P_local = invT.times(P_nube_global);
30      x_punto_local = P_local.get(0, 0);
31      y_punto_local = P_local.get(1, 0);
32      z_punto_local = P_local.get(2, 0);
33
        if (x_punto_local * x_punto_local + y_punto_local * y_punto_local <=
    radio_hta * radio_hta && z_punto_local >= 0 && z_punto_local <= altura_hta)
34  {
35      }
36      else {
37          nube_mecanizada.set(contador2, 0, x_punto);
38          nube_mecanizada.set(contador2, 1, y_punto);
39          nube_mecanizada.set(contador2, 2, z_punto);
40          contador2++;
41      }
42  }
43
44  nube_mecanizada = nube_mecanizada.getMatrix(0, contador2 - 1, 0, 2);
45  nube = nube_mecanizada;
46  nube_mecanizada_Array = nube_mecanizada.getArray();
47
48  int n_fr = 65;
    borde_nube_mecanizada                                                    =
    extraer_frontera_pointCloud(nube_mecanizada_Array, n_fr, n_fr, n_fr, new
49  boolean[] {
50      true
51  });
52  tam = nube_mecanizada_Array.length;
53
54  if (_isPlaying()) {
55      color_nube = new Color(0, 0, 0, 255);
56      grosor_nube = 1;
57  }
58  else {
59      color_nube = new Color(0, 0, 0, 10);
60      grosor_nube = 5;
61  }
```

Aquest codi s'executa cada vegada que es modifica la simulació (per exemple, quan es mou la ferramenta). El seu objectiu és gestionar la interacció entre la ferramenta i el brut.

Primer es calculen les coordenades locals de cada punt del brut respecte al sistema mòbil mitjançant la matriu inversa invT. A continuació, per cada punt, es comprova si aquest es troba dins del cilindre que representa la ferramenta: si està dins, s'elimina; si no, es manté i s'afegeix a la matriu nube_mecanizada. Només es representen els punts situats a la superfície externa del volum del brut, és a dir, aquells punts que formen la frontera bidimensional que delimita el sòlid mecanitzable. Aquesta selecció s'aconsegueix mitjançant la funció extraer_frontera_pointCloud(), la qual redueix la càrrega gràfica i millora el rendiment. Finalment, s'assigna color i gruix diferents al brut segons si la simulació està activa o no.

7.4.CODI DE LA FINESTRA "PROPIO"

L'última finestra amb codi s'anomena "Propio". Aquesta finestra conté totes les funcions personalitzades creades per al correcte funcionament de la simulació. Hi ha un total de 7 funcions: actualizarT(), leerPrograma(), volverOrigen(), crearBruto(), extraer_frontera(), calcularL() i deterJac(). A continuació es descriu cadascuna:

7.4.1. FUNCIÓ ACTUALIZART()

```
1  public void actualizarT () {
2    //Definir variables
3
4    //Construir matriz P
5    p = new Matrix(4,1);
6    p.set(0, 0, x);
7    p.set(1, 0, y);
8    p.set(2, 0, z);
9    p.set(3, 0, 1);
10
11
12   //Construir matriz A
13   Matrix A = new Matrix(3,3);
14   A.set(0, 0, 1);
15   A.set(0, 1, 0);
16   A.set(0, 2, 0);
```

```
17
18  A.set(1, 0, 0);
19  A.set(1, 1, Math.cos(alpha));
20  A.set(1, 2, -Math.sin(alpha));
21
22  A.set(2, 0, 0);
23  A.set(2, 1, Math.sin(alpha));
24  A.set(2, 2, Math.cos(alpha));
25
26
27  //Construir matriz B
28  Matrix B = new Matrix(3,3);
29  B.set(0, 0, Math.cos(beta));
30  B.set(0, 1, 0);
31  B.set(0, 2, Math.sin(beta));
32
33  B.set(1, 0, 0);
34  B.set(1, 1, 1);
35  B.set(1, 2, 0);
36
37  B.set(2, 0, -Math.sin(beta));
38  B.set(2, 1, 0);
39  B.set(2, 2, Math.cos(beta));
40
41
42  //Construir matriz C
43  Matrix C = new Matrix(3,3);
44  C.set(0, 0, Math.cos(gamma));
45  C.set(0, 1, -Math.sin(gamma));
46  C.set(0, 2, 0);
47
48  C.set(1, 0, Math.sin(gamma));
49  C.set(1, 1, Math.cos(gamma));
50  C.set(1, 2, 0);
51
52  C.set(2, 0, 0);
53  C.set(2, 1, 0);
54  C.set(2, 2, 1);
55
56  //Matriz con las coordenadas de todos los anclajes
57  Matrix b = new Matrix(4, 6);
58  b.set(0, 0, x_r);
59  b.set(0, 1, x_gre);
60  b.set(0, 2, x_b);
61  b.set(0, 3, x_c);
62  b.set(0, 4, x_m);
63  b.set(0, 5, x_y);
64
65  b.set(1, 0, y_r);
66  b.set(1, 1, y_gre);
```

```
67   b.set(1, 2, y_b);
68   b.set(1, 3, y_c);
69   b.set(1, 4, y_m);
70   b.set(1, 5, y_y);
71
72   b.set(2, 0, z_r);
73   b.set(2, 1, z_gre);
74   b.set(2, 2, z_b);
75   b.set(2, 3, z_c);
76   b.set(2, 4, z_m);
77   b.set(2, 5, z_y);
78
79   b.set(3, 0, 1);
80   b.set(3, 1, 1);
81   b.set(3, 2, 1);
82   b.set(3, 3, 1);
83   b.set(3, 4, 1);
84   b.set(3, 5, 1);
85
86   // Multiplicar A y B
87   Matrix AB = new Matrix(3,3);
88   AB = A.times(B);
89
90   // Multiplicar el resultado por C
91   Matrix R = new Matrix(3,3);
92   R = AB.times(C);
93
94   //Construir matriz T
95   T = new Matrix(4, 4);
96   T.set(3, 3, 1);
97   double valorR, valorP = 0;
98
99   for (int i = 0; i < 3; i++) {
100    for (int j = 0; j < 3; j++) {
101      valorR = R.get(i, j);
102      T.set(i, j, valorR);
103    }
104  }
105
106  for (int k = 0; k < 3;k++) {
107    valorP = p.get(k, 0);
108    T.set(k, 3, valorP);
109  }
110
111
112  //ConstBjruir B
113  Bj = new Matrix(4,6);
114  Bj = T.times(b);
115
116  //Coordenadas anclajes móviles en global
```

```
117    x_gen_r = Bj.get(0, 0);
118    y_gen_r = Bj.get(1, 0);
119    z_gen_r = Bj.get(2, 0);
120    x_gen_g = Bj.get(0, 1);
121    y_gen_g = Bj.get(1, 1);
122    z_gen_g = Bj.get(2, 1);
123    x_gen_b = Bj.get(0, 2);
124    y_gen_b = Bj.get(1, 2);
125    z_gen_b = Bj.get(2, 2);
126    x_gen_c = Bj.get(0, 3);
127    y_gen_c = Bj.get(1, 3);
128    z_gen_c = Bj.get(2, 3);
129    x_gen_m = Bj.get(0, 4);
130    y_gen_m = Bj.get(1, 4);
131    z_gen_m = Bj.get(2, 4);
132    x_gen_y = Bj.get(0, 5);
133    y_gen_y = Bj.get(1, 5);
134    z_gen_y = Bj.get(2, 5);
135
136  }
```

Aquesta funció calcula totes les matrius necessàries per transformar coordenades locals en globals: el vector de posició P, la matriu de rotació R, la matriu de coordenades locals dels punts d'ancoratge b, la matriu de transformació T i la matriu de coordenades gerenals dels punts d'ancoratge B.

El resultat final s'utilitza per representar correctament les barres en la posició adequada.

7.4.2. FUNCIÓ LEERPROGRAMA()

```
1   public void leerPrograma() {
2      targetX = x;
3      targetY = y;
4      targetZ = z;
5      targetA = alpha;
6      targetB = beta;
7      targetG = gamma;
8    String programa = _view.areaTexto.getText();
9      String[] lines = programa.split("\\r?\\n");
10     if (iterador_programa < lines.length) {
11        String[] splited = lines[iterador_programa].split("\\s+");
12        for (int j = 0; j < splited.length; j++) {
```

```
13          if (splited[j].length() > 0) {
14
15      char comando = splited[j].charAt(0);
16
17      String numero = "";
18          for (int k = 1; k < splited[j].length(); k++) {
19       char character = splited[j].charAt(k);
20              numero = numero + character;
21          }
22
23      double valor_numerico = Double.parseDouble(numero);
24
25          if (splited[j].charAt(0) == 'G' || splited[j].charAt(0) == 'g') {
26              if (valor_numerico == 1 || valor_numerico == 01) {
27                  tray_lineal = true;
28              }
29              else {
30                  tray_lineal = false;
31              }
32          }
33
34          if (splited[j].charAt(0) == 'X' || splited[j].charAt(0) == 'x') {
35              targetX = valor_numerico;
36          }
37
38          if (splited[j].charAt(0) == 'Y' || splited[j].charAt(0) == 'y') {
39              targetY = valor_numerico;
40          }
41
42          if (splited[j].charAt(0) == 'Z' || splited[j].charAt(0) == 'z') {
43              targetZ = valor_numerico;
44          }
45
46          if (splited[j].charAt(0) == 'I' || splited[j].charAt(0) == 'i') {
47              xm = valor_numerico;
48          }
49          if (splited[j].charAt(0) == 'J' || splited[j].charAt(0) == 'j') {
50              ym = valor_numerico;
51          }
52          if (splited[j].charAt(0) == 'F' || splited[j].charAt(0) == 'f') {
53              velocidadH = valor_numerico;
54          }
55          if (splited[j].charAt(0) == 'A' || splited[j].charAt(0) == 'a') {
56              targetA = valor_numerico * Math.PI / 180;
57          }
58          if (splited[j].charAt(0) == 'B' || splited[j].charAt(0) == 'b') {
59              targetB = valor_numerico * Math.PI / 180;
60          }
61          if (splited[j].charAt(0) == 'C' || splited[j].charAt(0) == 'c') {
62              targetG = valor_numerico * Math.PI / 180;
```

```
63                }
64                if (splited[j].charAt(0) == 'T' || splited[j].charAt(0) == 't') {
65            int valor_entero = (int) valor_numerico;
66                    radio_hta = Tools[valor_entero][0];
67                    altura_hta = Tools[valor_entero][1];
68
69                }
70              }
71            }
              System.out.println("Ejecutando linea: " +
72  lines[iterador_programa]);
73            t = 0;
74            _play();
75        }
76  }
```

La funció leerPrograma() Interpreta el codi escrit per l'usuari en format G-code simplificat:

Primerament, inicialitza les variables targetX, targetY, etc., amb la posició actual. A continuació, llig una línia del codi i la divideix en tokens. Amb els tokens, comprova el tipus d'instrucció (G, X, Y, F, etc.) i assigna els valors numèrics a les variables corresponents. Finalment, reprodueix la línia llegida i activa la simulació amb _play().

La funció permet modificar múltiples variables en una sola línia i admet ordres modals (com G01, F, etc.), es a dir, que l'última comanda de moviment utilitzada es queda memoritzada i no cal estar repetint-la en cada nou moviment que es vulga fer.

7.4.3. VOLVERORIGEN()

```
1   public void volverOrigen() {
2     x = 0;
3     y = 0;
4     z = 0;
5     alpha = 0;
6     beta = 0;
7     gamma = 0;
8     xm = 0.2;
9     ym = 0.1;
10    t = 0;
11  }
```

Mou la ferramenta al seu punt d'origen (x=y=z=0, angles a 0) i reinicia el temps a 0. És útil quan l'usuari vol reconstruir el brut i evitar que la ferramenta interfereixi amb ell.

7.4.4. CREARBRUTO()

```
1  public void crearBruto () {
2    //Creación del bruto
3    double x = x_min;
4    double y = y_min;
5    double z = z_min;
6
7    int nxpieza = 70;
8    int nypieza = 70;
9    int nzpieza = 70;
10
11   dxpieza = (x_max-x_min)/(nxpieza-1);
12   dypieza = (y_max-y_min)/(nypieza-1);
13   dzpieza = (z_max-z_min)/(nzpieza-1);
14
15   tam = nxpieza * nypieza * nzpieza;
16   nube = new Matrix(tam, 3);
17   int columna;
18   fila = 0;
19   x = x_min;
20   for(int i=0;i<nxpieza;i++) {
21     y = y_min;
22     for(int j=0;j<nypieza;j++) {
23       z = z_min;
24       for(int k=0;k<nzpieza;k++) {
25         columna = 0;
26         nube.set(fila, columna, x);
27         columna = columna + 1;
28         nube.set(fila, columna, y);
29         columna = columna + 1;
30         nube.set(fila, columna, z);
31         columna = columna + 1;
32         fila = fila + 1;
33         z = z + dzpieza;
34       }
35       y = y + dypieza;
36     }
37     x = x + dxpieza;
38   }
39 }
```

Genera el brut a mecanitzar. Divideix l'espai delimitat pels límits mínim i màxim de cada eix (x_min, x_max, etc.) en una malla de 70 punts per eix. Calcula les distàncies entre punts i ompli una matriu amb totes les coordenades 3D.

7.4.5. EXTRAER_FRONTERA()

```
      double[][] extraer_frontera_pointCloud(double[][] WS, int nx, int ny, int nz,
1     boolean[] exists) {
2       double[][] envelope = get_point_cloud_envelope(WS);
3
4       double x_min = envelope[0][0];
5       double x_max = envelope[1][0];
6
7       double y_min = envelope[0][1];
8       double y_max = envelope[1][1];
9
10      double z_min = envelope[0][2];
11      double z_max = envelope[1][2];
12
13      double dx, dy, dz;
14      dx = (x_max - x_min)/(nx-1);
15      dy = (y_max - y_min)/(ny-1);
16      dz = (z_max - z_min)/(nz-1);
17
18      double[][][][] malla = new double[nx][ny][nz][7];
19
20      double x, y, z;
21      x = x_min;
22      for(int i=0;i<nx;i++) {
23        y = y_min;
24        for(int j=0;j<ny;j++) {
25          z = z_min;
26          for(int k=0;k<nz;k++) {
27            malla[i][j][k][0] = x;
28            malla[i][j][k][1] = y;
29            malla[i][j][k][2] = z;
30            z += dz;
31          }
32          y += dy;
33        }
34        x += dx;
35      }
36
```

```
37    double x_r, y_r, z_r;
38    double x_core, y_core, z_core;
39    int i, j, k;
40    for(int u=0;u<WS.length;u++) {
41     if(true) {
42       x_r = WS[u][0];
43       y_r = WS[u][1];
44       z_r = WS[u][2];
45       x_core = x_min + dx*Math.round((x_r-x_min)/dx);
46       y_core = y_min + dy*Math.round((y_r-y_min)/dy);
47       z_core = z_min + dz*Math.round((z_r-z_min)/dz);
48       i = (int) Math.round((x_core-x_min)/dx);
49       j = (int) Math.round((y_core-y_min)/dy);
50       k = (int) Math.round((z_core-z_min)/dz);
51       malla[i][j][k][3] = malla[i][j][k][3] + 1;
52       malla[i][j][k][4] = malla[i][j][k][4] + x_r;
53       malla[i][j][k][5] = malla[i][j][k][5] + y_r;
54       malla[i][j][k][6] = malla[i][j][k][6] + z_r;
55     }
56    }
57
58    for(i=0;i<nx;i++) {
59     for(j=0;j<ny;j++) {
60       for(k=0;k<nz;k++) {
61         malla[i][j][k][4] = malla[i][j][k][4]/malla[i][j][k][3];
62         malla[i][j][k][5] = malla[i][j][k][5]/malla[i][j][k][3];
63         malla[i][j][k][6] = malla[i][j][k][6]/malla[i][j][k][3];
64       }
65     }
66    }
67
68    double[][] fr_ws = new double[nx*ny*nz][3];
69    int l=0;
70    for(i=0;i<nx;i++) {
71     for(j=0;j<ny;j++) {
72       for(k=0;k<nz;k++) {
73         if(malla[i][j][k][3]>0) {
74           int vecinos=0;
75           if( (i>0 && i<nx-1) && (j>0 && j<ny-1) && (k>0 && k<nz-1) ) {
76             vecinos = vecinos + (malla[i+1][j][k][3]>0? 1 : 0);
77             vecinos = vecinos + (malla[i-1][j][k][3]>0? 1 : 0);
78             vecinos = vecinos + (malla[i][j+1][k][3]>0? 1 : 0);
79             vecinos = vecinos + (malla[i][j-1][k][3]>0? 1 : 0);
80             vecinos = vecinos + (malla[i][j][k+1][3]>0? 1 : 0);
81             vecinos = vecinos + (malla[i][j][k-1][3]>0? 1 : 0);
82           }
83           if(vecinos<6) {
84             fr_ws[l][0] = malla[i][j][k][4];
85             fr_ws[l][1] = malla[i][j][k][5];
86             fr_ws[l][2] = malla[i][j][k][6];
```

```
87              l++;
88            }
89          }
90        }
91      }
92    }
93
94    if(l>0) {
95      fr_ws = Arrays.copyOfRange(fr_ws,0,l);
96    }
97    else {
98      fr_ws = new double[][] {
99        {
100         -999,-999,-999
101       }
102     } ;
103   }
104
105   return fr_ws;
106 }
107
108 double[][] get_point_cloud_envelope(double[][] cloud) {
      // Dada una nube de N puntos con dimension D, almacenados en la matriz
109 'cloud' de la siguiente forma:
110   //
111   // cloud:
112   //
113   // [ x_11 x_12 x_13 ... x_1D ]
114   // [ x_21 x_22 x_23 ... x_2D ]
115   //   .
116   //   .
117   //   .
118   // [ x_N1 x_N2 x_N3 ... x_ND ]
119   //
      // Esta funcion obtiene la minima caja D-dimensional que encierra esa nube de
120 puntos, en el siguiente formato:
121   //
122   // [ min_1 min_2 min_3 ... min_D ]
123   // [ max_1 max_2 max_3 ... max_D ]
124   //
      // Es decir, retorna una matriz 2xD, con el minimo y el maximo en cada una
125 de las D dimensiones.
126
127   int N = cloud.length;
128   int D = cloud[0].length;
129
130   double[] minimos, maximos;
131   minimos = Arrays.copyOf(cloud[0],D);
132   maximos = Arrays.copyOf(cloud[0],D);
133
```

```
134   for(int i=1;i<N;i++) {
135     for(int j=0;j<D;j++) {
136       minimos[j] = Math.min(minimos[j],cloud[i][j]);
137       maximos[j] = Math.max(maximos[j],cloud[i][j]);
138     }
139   }
140
141   double[][] retorno = new double[2][D];
142   retorno[0] = Arrays.copyOf(minimos,D);
143   retorno[1] = Arrays.copyOf(maximos,D);
144   return retorno;
```

Filtra i mostra només els punts exteriors del brut mitjançant l'anàlisi de veïnat. Això redueix la càrrega computacional i fa la simulació més fluïda. No és essencial per a entendre el funcionament del robot, però sí per millorar-ne el rendiment visual.

7.4.6. CALCULARL()

```
      public void calcularL (double x, double y, double z, double alpha, double beta,
1     double gamma) {
2       //Construir matriz P2
3       Matrix p2 = new Matrix(4,1);
4       p2.set(0, 0, x);
5       p2.set(1, 0, y);
6       p2.set(2, 0, z);
7       p2.set(3, 0, 1);
8
9       //Matriz aj
10      Matrix a = new Matrix(4, 6);
11      a.set(0, 0, 0.75-0.05);
12      a.set(0, 1, 0.75/2);
13      a.set(0, 2, (0.75-0.05)*Math.cos(2./3*Math.PI));
14      a.set(0, 3, 0.75/2*Math.cos(2./3*Math.PI));
15      a.set(0, 4, (0.75-0.05)*Math.cos(2./3*Math.PI));
16      a.set(0, 5, 0.75/2*Math.cos(2./3*Math.PI));
17
18      a.set(1, 0, 0);
19      a.set(1, 1, 0);
20      a.set(1, 2, (0.75-0.05)*Math.sin(2./3*Math.PI));
21      a.set(1, 3, 0.75/2*Math.sin(2./3*Math.PI));
22      a.set(1, 4, -(0.75-0.05)*Math.sin(2./3*Math.PI));
23      a.set(1, 5, -0.75/2*Math.sin(2./3*Math.PI));
24
25      a.set(2, 0, -0.5+0.1/2);
```

```
26      a.set(2, 1, -0.5+0.1/2);
27      a.set(2, 2, -0.5+0.1/2);
28      a.set(2, 3, -0.5+0.1/2);
29      a.set(2, 4, -0.5+0.1/2);
30      a.set(2, 5, -0.5+0.1/2);
31
32      a.set(3, 0, 1);
33      a.set(3, 1, 1);
34      a.set(3, 2, 1);
35      a.set(3, 3, 1);
36      a.set(3, 4, 1);
37      a.set(3, 5, 1);
38
39      //Construir matriz A2
40      Matrix A2 = new Matrix(3,3);
41      A2.set(0, 0, 1);
42      A2.set(0, 1, 0);
43      A2.set(0, 2, 0);
44
45      A2.set(1, 0, 0);
46      A2.set(1, 1, Math.cos(alpha));
47      A2.set(1, 2, -Math.sin(alpha));
48
49      A2.set(2, 0, 0);
50      A2.set(2, 1, Math.sin(alpha));
51      A2.set(2, 2, Math.cos(alpha));
52
53
54      //Construir matriz B2
55      Matrix B2 = new Matrix(3,3);
56      B2.set(0, 0, Math.cos(beta));
57      B2.set(0, 1, 0);
58      B2.set(0, 2, Math.sin(beta));
59
60      B2.set(1, 0, 0);
61      B2.set(1, 1, 1);
62      B2.set(1, 2, 0);
63
64      B2.set(2, 0, -Math.sin(beta));
65      B2.set(2, 1, 0);
66      B2.set(2, 2, Math.cos(beta));
67
68
69      //Construir matriz C2
70      Matrix C2 = new Matrix(3,3);
71      C2.set(0, 0, Math.cos(gamma));
72      C2.set(0, 1, -Math.sin(gamma));
73      C2.set(0, 2, 0);
74
75      C2.set(1, 0, Math.sin(gamma));
```

```
76      C2.set(1, 1, Math.cos(gamma));
77      C2.set(1, 2, 0);
78
79      C2.set(2, 0, 0);
80      C2.set(2, 1, 0);
81      C2.set(2, 2, 1);
82
83      // Multiplicar A2 y B2
84      Matrix AB2 = new Matrix(3,3);
85      AB2 = A2.times(B2);
86
87      // Multiplicar el resultado por C2
88      Matrix R2 = new Matrix(3,3);
89      R2 = AB2.times(C2);
90
91      //Construir matriz T2
92      Matrix T2 = new Matrix(4, 4);
93      T2.set(3, 3, 1);
94      double valorR, valorP = 0;
95
96      for (int i = 0; i < 3; i++) {
97        for (int j = 0; j < 3; j++) {
98          valorR = R2.get(i, j);
99          T2.set(i, j, valorR);
100       }
101     }
102
103     for (int k = 0; k < 3;k++) {
104       valorP = p2.get(k, 0);
105       T2.set(k, 3, valorP);
106     }
107
108     //Matriz con las coordenadas de todos los anclajes
109     Matrix b = new Matrix(4, 6);
110     b.set(0, 0, x_r);
111     b.set(0, 1, x_gre);
112     b.set(0, 2, x_b);
113     b.set(0, 3, x_c);
114     b.set(0, 4, x_m);
115     b.set(0, 5, x_y);
116
117     b.set(1, 0, y_r);
118     b.set(1, 1, y_gre);
119     b.set(1, 2, y_b);
120     b.set(1, 3, y_c);
121     b.set(1, 4, y_m);
122     b.set(1, 5, y_y);
123
124     b.set(2, 0, z_r);
125     b.set(2, 1, z_gre);
```

```
126     b.set(2, 2, z_b);
127     b.set(2, 3, z_c);
128     b.set(2, 4, z_m);
129     b.set(2, 5, z_y);
130
131     b.set(3, 0, 1);
132     b.set(3, 1, 1);
133     b.set(3, 2, 1);
134     b.set(3, 3, 1);
135     b.set(3, 4, 1);
136     b.set(3, 5, 1);
137
138     //Construir Bj2
139     Matrix Bj2 = new Matrix(4,6);
140     Bj2= T2.times(b);
141
142
143     //Matriz Lj
144     Lj = new Matrix(6, 1);
        Lj.set(0, 0, Math.sqrt(Math.pow(a.get(0,0) - Bj2.get(0, 0), 2) +
        Math.pow(a.get(1,0) - Bj2.get(1, 0), 2) + Math.pow(a.get(2,0) - Bj2.get(2, 0),
145     2)));
        Lj.set(1, 0, Math.sqrt(Math.pow(a.get(0,1) - Bj2.get(0, 1), 2) +
        Math.pow(a.get(1,1) - Bj2.get(1, 1), 2) + Math.pow(a.get(2,1) - Bj2.get(2, 1),
146     2)));
        Lj.set(2, 0, Math.sqrt(Math.pow(a.get(0,2) - Bj2.get(0, 2), 2) +
        Math.pow(a.get(1,2) - Bj2.get(1, 2), 2) + Math.pow(a.get(2,2) - Bj2.get(2, 2),
147     2)));
        Lj.set(3, 0, Math.sqrt(Math.pow(a.get(0,3) - Bj2.get(0, 3), 2) +
        Math.pow(a.get(1,3) - Bj2.get(1, 3), 2) + Math.pow(a.get(2,3) - Bj2.get(2, 3),
148     2)));
        Lj.set(4, 0, Math.sqrt(Math.pow(a.get(0,4) - Bj2.get(0, 4), 2) +
        Math.pow(a.get(1,4) - Bj2.get(1, 4), 2) + Math.pow(a.get(2,4) - Bj2.get(2, 4),
149     2)));
        Lj.set(5, 0, Math.sqrt(Math.pow(a.get(0,5) - Bj2.get(0, 5), 2) +
        Math.pow(a.get(1,5) - Bj2.get(1, 5), 2) + Math.pow(a.get(2,5) - Bj2.get(2, 5),
150     2)));
151     }
```

Aquesta funció s'encarrega de calcular el vector de longituds de les sis barres que connecten la base fixa amb la plataforma mòbil del robot. Per fer-ho, primer construeix la matriu aj, que conté les coordenades fixes dels punts d'ancoratge sobre la base inferior.

A continuació, s'obté la matriu de rotació R i el vector de posició P, amb els quals es genera la matriu de transformació T. Aquesta matriu permet transformar les coordenades locals de les articulacions mòbils (bj) en coordenades globals (Bj).

Finalment, es calcula el vector Lj, que conté la longitud de cadascuna de les sis barres, mitjançant la distància euclidiana entre els punts corresponents de les matrius aj i Bj.

7.4.7. DETERJAC()

```
1   public void deterJac() {
2     double incremento = 0.00001;
3     double L1_0 = 0, L2_0 = 0, L3_0 = 0, L4_0 = 0, L5_0 = 0, L6_0 = 0;
      double L1_inc = 0, L2_inc = 0, L3_inc = 0, L4_inc = 0, L5_inc = 0, L6_inc =
4   0;
5     double df1x = 0, df2x = 0, df3x = 0, df4x = 0, df5x = 0, df6x = 0;
6     double df1y = 0, df2y = 0, df3y = 0, df4y = 0, df5y = 0, df6y = 0;
7     double df1z = 0, df2z = 0, df3z = 0, df4z = 0, df5z = 0, df6z = 0;
8     double df1a = 0, df2a = 0, df3a = 0, df4a = 0, df5a = 0, df6a = 0;
9     double df1b = 0, df2b = 0, df3b = 0, df4b = 0, df5b = 0, df6b = 0;
10    double df1g = 0, df2g = 0, df3g = 0, df4g = 0, df5g = 0, df6g = 0;
11
12    Matrix invJ = new Matrix(6, 6);
13
14      L1_0 = Lj.get(0, 0);
15      L2_0 = Lj.get(1, 0);
16      L3_0 = Lj.get(2, 0);
17      L4_0 = Lj.get(3, 0);
18      L5_0 = Lj.get(4, 0);
19      L6_0 = Lj.get(5, 0);
20
21
22      // Derivada de L1
23      calcularL(x + incremento, y, z, alpha, beta, gamma);
24      L1_inc = Lj.get(0, 0);
25      df1x = (L1_inc - L1_0) / incremento;
26      calcularL(x, y + incremento, z, alpha, beta, gamma);
27      L1_inc = Lj.get(0, 0);
28      df1y = (L1_inc - L1_0) / incremento;
29      calcularL(x, y, z + incremento, alpha, beta, gamma);
30      L1_inc = Lj.get(0, 0);
31      df1z = (L1_inc - L1_0) / incremento;
32      calcularL(x, y, z, alpha + incremento, beta, gamma);
33      L1_inc = Lj.get(0, 0);
34      df1a = (L1_inc - L1_0) / incremento;
35      calcularL(x, y, z, alpha, beta + incremento, gamma);
36      L1_inc = Lj.get(0, 0);
37      df1b = (L1_inc - L1_0) / incremento;
38      calcularL(x, y, z, alpha, beta, gamma + incremento);
```

```
39      L1_inc = Lj.get(0, 0);
40      df1g = (L1_inc - L1_0) / incremento;
41
42
43      // Derivada de L2
44      calcularL(x + incremento, y, z, alpha, beta, gamma);
45      L2_inc = Lj.get(1, 0);
46      df2x = (L2_inc - L2_0) / incremento;
47      calcularL(x, y + incremento, z, alpha, beta, gamma);
48      L2_inc = Lj.get(1, 0);
49      df2y = (L2_inc - L2_0) / incremento;
50      calcularL(x, y, z + incremento, alpha, beta, gamma);
51      L2_inc = Lj.get(1, 0);
52      df2z = (L2_inc - L2_0) / incremento;
53      calcularL(x, y, z, alpha + incremento, beta, gamma);
54      L2_inc = Lj.get(1, 0);
55      df2a = (L2_inc - L2_0) / incremento;
56      calcularL(x, y, z, alpha, beta + incremento, gamma);
57      L2_inc = Lj.get(1, 0);
58      df2b = (L2_inc - L2_0) / incremento;
59      calcularL(x, y, z, alpha, beta, gamma + incremento);
60      L2_inc = Lj.get(1, 0);
61      df2g = (L2_inc - L2_0) / incremento;
62
63
64      // Derivada de L3
65      calcularL(x + incremento, y, z, alpha, beta, gamma);
66      L3_inc = Lj.get(2, 0);
67      df3x = (L3_inc - L3_0) / incremento;
68      calcularL(x, y + incremento, z, alpha, beta, gamma);
69      L3_inc = Lj.get(2, 0);
70      df3y = (L3_inc - L3_0) / incremento;
71      calcularL(x, y, z + incremento, alpha, beta, gamma);
72      L3_inc = Lj.get(2, 0);
73      df3z = (L3_inc - L3_0) / incremento;
74      calcularL(x, y, z, alpha + incremento, beta, gamma);
75      L3_inc = Lj.get(2, 0);
76      df3a = (L3_inc - L3_0) / incremento;
77      calcularL(x, y, z, alpha, beta + incremento, gamma);
78      L3_inc = Lj.get(2, 0);
79      df3b = (L3_inc - L3_0) / incremento;
80      calcularL(x, y, z, alpha, beta, gamma + incremento);
81      L3_inc = Lj.get(2, 0);
82      df3g = (L3_inc - L3_0) / incremento;
83
84
85      // Derivada de L4
86      calcularL(x + incremento, y, z, alpha, beta, gamma);
87      L4_inc = Lj.get(3, 0);
88      df4x = (L4_inc - L4_0) / incremento;
```

```
89      calcularL(x, y + incremento, z, alpha, beta, gamma);
90      L4_inc = Lj.get(3, 0);
91      df4y = (L4_inc - L4_0) / incremento;
92      calcularL(x, y, z + incremento, alpha, beta, gamma);
93      L4_inc = Lj.get(3, 0);
94      df4z = (L4_inc - L4_0) / incremento;
95      calcularL(x, y, z, alpha + incremento, beta, gamma);
96      L4_inc = Lj.get(3, 0);
97      df4a = (L4_inc - L4_0) / incremento;
98      calcularL(x, y, z, alpha, beta + incremento, gamma);
99      L4_inc = Lj.get(3, 0);
100     df4b = (L4_inc - L4_0) / incremento;
101     calcularL(x, y, z, alpha, beta, gamma + incremento);
102     L4_inc = Lj.get(3, 0);
103     df4g = (L4_inc - L4_0) / incremento;
104
105
106     // Derivada de L5
107     calcularL(x + incremento, y, z, alpha, beta, gamma);
108     L5_inc = Lj.get(4, 0);
109     df5x = (L5_inc - L5_0) / incremento;
110     calcularL(x, y + incremento, z, alpha, beta, gamma);
111     L5_inc = Lj.get(4, 0);
112     df5y = (L5_inc - L5_0) / incremento;
113     calcularL(x, y, z + incremento, alpha, beta, gamma);
114     L5_inc = Lj.get(4, 0);
115     df5z = (L5_inc - L5_0) / incremento;
116     calcularL(x, y, z, alpha + incremento, beta, gamma);
117     L5_inc = Lj.get(4, 0);
118     df5a = (L5_inc - L5_0) / incremento;
119     calcularL(x, y, z, alpha, beta + incremento, gamma);
120     L5_inc = Lj.get(4, 0);
121     df5b = (L5_inc - L5_0) / incremento;
122     calcularL(x, y, z, alpha, beta, gamma + incremento);
123     L5_inc = Lj.get(4, 0);
124     df5g = (L5_inc - L5_0) / incremento;
125
126
127     // Derivada de L6
128     calcularL(x + incremento, y, z, alpha, beta, gamma);
129     L6_inc = Lj.get(5, 0);
130     df6x = (L6_inc - L6_0) / incremento;
131     calcularL(x, y + incremento, z, alpha, beta, gamma);
132     L6_inc = Lj.get(5, 0);
133     df6y = (L6_inc - L6_0) / incremento;
134     calcularL(x, y, z + incremento, alpha, beta, gamma);
135     L6_inc = Lj.get(5, 0);
136     df6z = (L6_inc - L6_0) / incremento;
137     calcularL(x, y, z, alpha + incremento, beta, gamma);
138     L6_inc = Lj.get(5, 0);
```

```
139     df6a = (L6_inc - L6_0) / incremento;
140     calcularL(x, y, z, alpha, beta + incremento, gamma);
141     L6_inc = Lj.get(5, 0);
142     df6b = (L6_inc - L6_0) / incremento;
143     calcularL(x, y, z, alpha, beta, gamma + incremento);
144     L6_inc = Lj.get(5, 0);
145     df6g = (L6_inc - L6_0) / incremento;
146
147
148     // Matriu Jinv
149     invJ.set(0, 0, df1x);
150     invJ.set(0, 1, df1y);
151     invJ.set(0, 2, df1z);
152     invJ.set(0, 3, df1a);
153     invJ.set(0, 4, df1b);
154     invJ.set(0, 5, df1g);
155
156     invJ.set(1, 0, df2x);
157     invJ.set(1, 1, df2y);
158     invJ.set(1, 2, df2z);
159     invJ.set(1, 3, df2a);
160     invJ.set(1, 4, df2b);
161     invJ.set(1, 5, df2g);
162
163     invJ.set(2, 0, df3x);
164     invJ.set(2, 1, df3y);
165     invJ.set(2, 2, df3z);
166     invJ.set(2, 3, df3a);
167     invJ.set(2, 4, df3b);
168     invJ.set(2, 5, df3g);
169
170     invJ.set(3, 0, df4x);
171     invJ.set(3, 1, df4y);
172     invJ.set(3, 2, df4z);
173     invJ.set(3, 3, df4a);
174     invJ.set(3, 4, df4b);
175     invJ.set(3, 5, df4g);
176
177     invJ.set(4, 0, df5x);
178     invJ.set(4, 1, df5y);
179     invJ.set(4, 2, df5z);
180     invJ.set(4, 3, df5a);
181     invJ.set(4, 4, df5b);
182     invJ.set(4, 5, df5g);
183
184     invJ.set(5, 0, df6x);
185     invJ.set(5, 1, df6y);
186     invJ.set(5, 2, df6z);
187     invJ.set(5, 3, df6a);
188     invJ.set(5, 4, df6b);
```

```
189     invJ.set(5, 5, df6g);
190
191     determinante = invJ.det();
192 }
```

Aquesta funció calcula la matriu Jacobiana inversa (Jinv) mitjançant aproximació numèrica, aplicant derivades parcials al vector de longituds Lj respecte de les sis variables generals de posició i orientació (x, y, z, α, β, γ).

Per obtenir cadascuna de les derivades, es realitzen crides successives a la funció calcularL() amb increments lleus en cada variable, i es calcula la diferència entre el valor inicial i l'incrementat. Un cop construïda la matriu Jinv, la funció en calcula el determinant, que permet detectar configuracions singulars del robot en què es perd el control de moviment de la plataforma.