Desarrollo de robot paralelo tipo doble SCARA: diseño, fabricación, simulación, control y experimentación

Proyecto final de estudios
Ingeniería en Mecatrónica

Gino Avanzini, Gonzalo Fernández, Jeremías Pino Demichelis

Director: Ing. Eric Sanchez

Facultad de Ingeniería, Universidad Nacional de Cuyo
Mendoza, Argentina

2022

Agradecimientos

En primer lugar queremos agradecer a nuestras familias y amigos, por su absoluto apoyo durante este trayecto.

A todos los profesores que alimentaron la curiosidad en nosotros y que, con su incondicional dedicación, nos guiaron en este camino. En particular a Eric por su apoyo a lo largo del desarrollo de este proyecto.

A nuestros compañeros y colegas, con quienes transitamos estas aulas y pasillos.

A la Asociación de Mecatrónica, que nos dio un espacio para compartir ideas y aprender entre colegas.

A la educación pública por darnos esta posibilidad.

Resumen

En este documento se presenta el análisis, modelado, diseño, fabricación, puesta a punto y ensayos de un robot con estructura cinemática de tipo SCARA paralelo o mecanismo de cinco barras de tres grados de libertad, con fines educativos y didácticos, en el marco de la carrera de Ingeniería en Mecatrónica de la Facultad de Ingeniería de la Universidad Nacional de Cuyo. El desarrollo del modelo cinemático y dinámico del robot, basados en un fuerte desarrollo vectorial, nos permite generar tanto una simulación cinemática como una simulación dinámica, utilizadas para la evaluación del algoritmo de planificación de trayectoria. También se discute acerca del diseño mecánico y fabricación del robot, que tuvo como principal objetivo ser económicamente accesible. Se exponen los componentes de hardware, la arquitectura de control del robot y la implementación del firmware. Finalmente, se evalúa el desempeño del prototipo mediante la realización de trayectorias rectas y circulares.

Como resultado de este trabajo se obtiene una celda robótica de propósito general. Gracias a esto, los estudiantes disponen de una estructura donde aplicar conceptos de mecánica, control, visión artificial, electrónica de potencia, microcontroladores, robótica, etc. Además, cuentan con el estudio y el análisis matemático realizado sobre el robot planteado, que tiene como principal característica ser de configuración paralela, donde debieron utilizarse generalizaciones de conceptos aprendidos en la teoría de robots serie.

Palabras claves: *Robótica, sistemas embebidos, diseño mecánico, impresión 3D, robot paralelo, CAN, control.*

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

La robótica es una rama interdisciplinaria de la Ingeniería, en ella se integran saberes de la ingeniería mecánica, informática y electrónica, entre otros. Esto la hace un perfecto caso para la aplicación de la mecatrónica. En este trabajo se indagó en el diseño y la construcción de los robots paralelos, aplicando los conocimientos adquiridos durante el transcurso de la carrera.

Los robots de tipo paralelo han sido foco de estudio y se han desarrollado ampliamente en los últimos años, desde un punto de vista tanto teórico como de aplicaciones prácticas. Estos son mecanismos de cinemática cerrada que presentan muy buenas prestaciones en términos de precisión y velocidad comparados con un robot serie del mismo tamaño y peso. La estructura paralela permite la creación de robots más rígidos y ofrece un desempeño dinámico significativamente mejor. Esto es debido a que la carga a manipular es compartida por varias ramas del sistema, y por lo tanto, cada cadena cinemática acarrea solo con una fracción de la carga total. Estas arquitecturas hacen posible reducir la masa de los eslabones y, como resultado, permite el uso de actuadores de menor capacidad. Además de la excelente relación carga/peso, los manipuladores paralelos también presentan características interesantes, por ejemplo, la posición del efector final es más tolerante a los errores, a diferencia de los robots del tipo serie. Algunos de los argumentos en contra son los siguientes: el control es más complejo en comparación al de robots serie debido a la cinemática cerrada de la estructura; en general, su espacio de trabajo es más pequeño y mucho más complejo ya que estos robots incluyen no solo singularidades de tipo 1, sino también singularidades de tipo 2. Las singularidades son aquellas configuraciones que limitan el movimiento del robot manipulador, un análisis más detallado de esto podrá verse en la sección 2.3.3. Las características antes mencionadas permiten la creación de estructuras con alta capacidad de carga, alta capacidad dinámica y alta precisión, a un costo mucho menor que su equivalente serie.

Se decidió diseñar y construir un robot planar de 2 grados de libertad y adicionarle un tercer grado de libertad con la intención de poder utilizarlo en aplicaciones de *pick and place* o corte plasma. El robot consta de dos cadenas cinemáticas, que lo dotan de la posibilidad de moverse en el plano xy, una compuesta por 3 articulaciones rotacionales ($\bar{R}RR$)[1] y la otra por 2 ($\bar{R}R$), una base fija y una plataforma móvil con una articulación prismática para posicionar el efector en el eje z. En este trabajo se intenta construir una maquina de precisión y alto rendimiento dinámico, buscando ser una alternativa a los robot SCARA y DELTA ampliamente utilizados para tareas de *pick and place*. En la Fig. 1.1 se puede observar el robot construido.

Este capítulo se organizó en diferentes secciones. En la sección 1.1 se describen los modelos de robot existentes, más adelante, en las secciones 1.2 y 1.3, se detalla el problema a resolver y los objetivos del trabajo. Luego, en la sección 1.4, se describen las herramientas utilizadas. En la sección 1.5, se introducen los conceptos teóricos utilizados para el desarrollo del trabajo y finalmente en la sección 1.6, se detalla la organización del informe.

[1] $\bar{X}$: articulación activa; P: articulación prismática; R: articulación rotacional

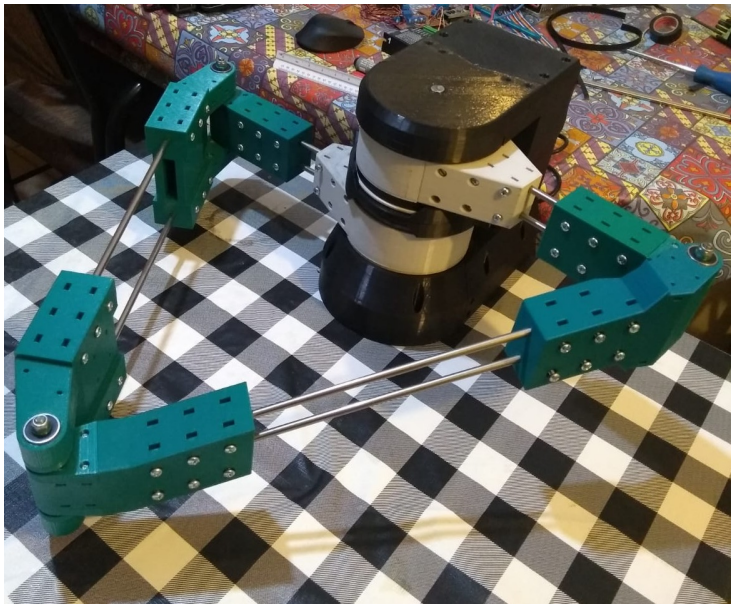


Figura 1.1: Doble *SCARA* paralelo o mecanismo de cinco barras construido.

1.1. Estado del arte

Investigando sobre manipuladores robóticos paralelos basados en mecanismos de cinco barras, se encontraron dos desarrollos previos de interés. Por un lado, el robot *Dextar* ($\bar{R}RRR\bar{R}$), presentado en Bourbonnais, Bigras y Bonev [1], donde proponen un diseño en el cual todas las articulaciones son rotacionales y todos los eslabones de la cadena cinemática tienen la misma longitud, como se observa en la Fig. 1.2, y lo comparan con el robot "*double-SCARA*" ofrecido por *Mitsubishi Electric*. En dicho trabajo también proponen una estrategia para evitar singularidades, cambiando de modo de trabajo[2]. Como resultado de su desarrollo lograron incrementar significativamente el espacio de trabajo del robot. Por otro lado, se encontró el robot *ParaPlacer* ($\bar{P}RRR\bar{P}$), que consiste en un diseño donde las articulaciones actuadas son prismáticas, y optimizan el espacio de trabajo realizando movimientos continuos entre diferentes modos de ensamble[3]. Además, presentan un prototipo que puede observarse en la Fig. 1.2.

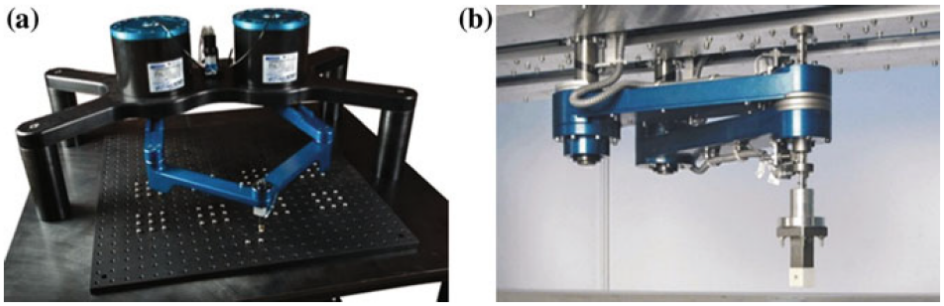


Figura 1.2: a) Robot *Dextar*. b) Robot *ParaPlacer*. Imagen extraída de Khalil y Briot [2].

De los estudios previamente presentados se obtuvo un esquema inicial para el robot a desarrollar y una primera idea de qué actuadores utilizar para las articulaciones activas.

Además de las publicaciones mencionadas, existe trabajo desarrollado previamente sobre esta clase de robot por parte de los autores, comenzado con un Proyecto Tipo C impulsado por la Universidad

[2]Modos de trabajo es un concepto que se discute más adelante, pero básicamente consiste en diferentes configuraciones de la cadena cinemática para una misma posición del efector final.

[3]El modo de ensamble es un concepto que se discute más adelante, pero básicamente consiste en diferentes configuraciones de la cadena cinemática para una misma posición de las articulaciones activas.

Nacional de Cuyo Avanzini, Fernández y Pino [3], en cuyo desarrollo se iniciaron los estudios cinemáticos y dinámicos, el diseño mecánico de un primer prototipo y su sistema de control.

1.2. Formulación y fundamentación del problema

Como estudiantes de la carrera de Ingeniería en Mecatrónica de la Facultad de Ingeniería de la Universidad Nacional de Cuyo, hemos observado a través del intercambio con colegas graduados, profesores, y profesionales de carreras afines, una creciente aplicación de celdas de automatización con robots paralelos en la industria. Esta clase de robots, a pesar de sus dificultades en su análisis matemático y físico, y su control, han demostrado tener grandes cualidades en cuanto a velocidad y precisión.

En la carrera de Ingeniería en Mecatrónica se estudia en detalle la robótica serie, que se basa en cadenas cinemáticas abiertas, mientras que los robots paralelos solo se caracterizan, estudiando conceptos generales y reflexionando sobre sus ventajas y desventajas.

Por lo tanto, esta experiencia con robótica paralela puede brindarle el día de mañana a nuestros pares una herramienta distintiva en el mercado laboral. Se propuso la creación de una celda robótica prototipo genérica con una estructura cinemática paralela simple, como lo es un mecanismo de 5 barras. Esta celda permite a aquellos estudiantes interesados en incorporar esta rama de la robótica trabajar en el modelado del mecanismo, ver los resultados en un prototipo físico, y así hallar dificultades presentes en la práctica que en un análisis teórico se pasan por alto.

Además, la disponibilidad de un prototipo físico genérico, con toda la documentación adjunta que incluye modelos matemáticos y físicos, software de planificación de trayectorias, y software para control de actuadores, permite a los estudiantes la realización de prácticas y experiencias que no necesariamente estén vinculadas a la robótica paralela, sino también a sistemas de control, inteligencia artificial, sistemas embebidos, autómatas y otras ramas de la robótica; experiencias que se complementan con laboratorios ya existentes en la carrera, o que podrían implementarse en algunas cátedras.

1.3. Objetivos

El objetivo principal del trabajo es diseñar, fabricar e implementar una celda robótica prototipo. Este robot tiene como principal propósito servir de material didáctico para los estudiantes de la carrera de Ingeniería en Mecatrónica. Además, como aplicación industrial de referencia, se decide imitar tareas que tienen preponderancia sobre un plano horizontal de trabajo, como pueden ser el corte láser, el corte plasma o tareas de *pick and place*.

Para cumplir con este objetivo se trabajó sobre los siguientes puntos:

- Diseño cinemático y dinámico del robot.
- Simulación.
- Diseño mecánico.
- Fabricación del mismo.
- Diseño e implementación del control de los actuadores.
- Utilización de técnicas de comunicación industriales robustas, como protocolo CAN y CANopen.
- Planificación de trayectorias y control del robot.

1.4. Herramientas

Para el desarrollo de este proyecto se utilizaron las siguientes herramientas:

- Software: Para la programación de los algoritmos de control se utilizaron dos lenguajes diferentes: para el control a bajo nivel se utilizó *C*, y para el planificador de trayectoria se utilizó *Python*. Para la simulación del robot se decidió utilizar el simulador *CoppeliaSim*, que se programó en *Python* y una pequeña parte en *Lua*. Para el análisis cinemático y dinámico se utilizó *Matlab* y *Simscape*. Además, se utilizó la herramienta de control de versiones *Git*.

- Hardware: Como placas del control de motores y nodo primario CAN se utilizaron 3 microcontroladores *Stm32f103*, alias *Blue Pill*.
- Mecánica: Para el diseño mecánico se utilizaron dos software, *SolidEdge* en el diseño del primer prototipo y luego *SolidWorks* para el modelado del robot final. Para la construcción de los prototipos mediante fabricación aditiva se utilizo una impresora 3D Artillery Genius con el slicer *Cura*.

1.5. Marco teórico

A continuación se detallan los conceptos teóricos utilizados para describir geométricamente al robot y para la generación de trayectorias. Para la descripción geométrica del robot se utilizan los parámetros modificados de Denavit-Hartenber, se expone un resumen del capítulo 4, sección 4.2.1, del libro *Dynamics of Parallel Robots - Khalil y Briot [2]*. En dicho libro se puede encontrar ejemplos de aplicación de los parámetros y una explicación más detallada del tema. Además se presenta una introducción al bus CAN, partiendo de sus características eléctricas y de capa de enlace y desarrollando brevemente el protocolo de aplicación CANopen, muy utilizado en aplicaciones de control de movimiento. Como se explicará posteriormente, estos protocolos serán utilizados en el subsistema electrónico y de control.

1.5.1. Parámetros modificados Denavit-Hartenberg

Consideraremos una cadena cinemática con estructura de árbol compuesta por $n+1$ cuerpos (físicos o virtuales), donde el cuerpo i se denota como B_i. Esta cadena cinemática está compuesta por n articulaciones asociadas a los cuerpos B_i $(i = 1, ..., n)$ como se muestra en la Fig. 1.3. Numeramos cada cuerpo i de forma creciente desde la base denotada como cuerpo B_0. Para un robot industrial fijado en el suelo, el cuerpo B_0 sería el cuerpo fijado en el suelo.

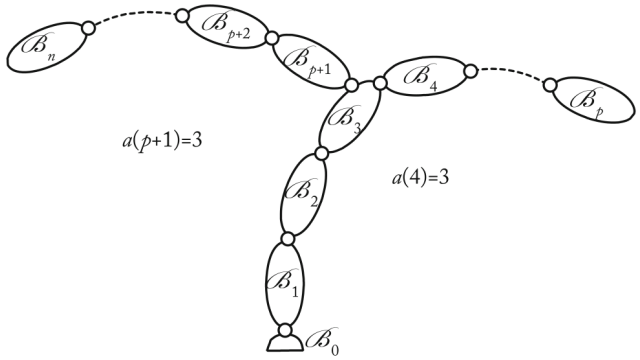


Figura 1.3: Estructura de árbol extraída de Khalil y Briot [2].

Definimos una relación de antecedencia a entre los cuerpos. Si el cuerpo B_i es antecedente del cuerpo B_j, entonces

$$i = a(j)$$

Las reglas de numeración garantizan entonces que $i < j$. Por definición, en una estructura cinemática de árbol, un cuerpo puede tener como máximo un cuerpo antecedente (ninguno en el caso de la base) pero varios cuerpos sucesores. Si un cuerpo no tiene ningún cuerpo sucesor, éste es uno de los cuerpos terminales de la cadena. Un sistema en el que todos los cuerpos tienen un solo cuerpo sucesor se denomina cadena abierta simple, o cadena tipo serie. Un sistema con por lo menos un cuerpo con más de un sucesor y con cadenas cerradas (un cuerpo terminal y uno o más bucles cerrados) se denomina cadena tipo paralela. Los valores de $a(j)$ definen completamente la topología de la estructura, que puede analizarse como un grafo (véase la Fig. 1.3).

A cada cuerpo B_j se le fija un marco de referencia local F_j con origen O_j. La parametrización modificada de Denavit-Hartenberg (MDH) permite construir cada marco F_j de forma sencilla aplicando las siguientes reglas para $i = 1, ..., n$:

- El marco $F_i = (O_i, x_i, y_i, z_i)$ se fija con respecto al cuerpo B_i.
- El eje z_i está a lo largo del eje de la articulación i.
- El eje x_i se toma a lo largo de la normal común entre z_i y uno de los ejes de la articulación sucesiva, que se fija en el eslabón B_i. Entonces los siguientes casos son posibles:
  - El cuerpo B_i es un cuerpo terminal y no tiene cuerpo siguiente; x_i puede fijarse arbitrariamente (siempre que sea ortogonal a z_i).
  - El cuerpo B_i, $i = a(j)$, sólo tiene un cuerpo sucesor; x_i debe estar entonces a lo largo de la perpendicular común a z_i y z_j.
  - El cuerpo B_i tiene varios cuerpos sucesivos; uno de los cuerpos sucesivos debe ser elegido para construir el eje x_i. En la práctica, el cuerpo siguiente en el que se define x_i puede ser elegido como el que se encuentra en la trayectoria que lleva al eslabón terminal, pero esto no es una obligación.
- El eje y_i se toma por la regla de la mano derecha tal que x_i , y_i , z_i sean una base ortonormal.

En el caso de que, para el cuerpo B_j que sucede al cuerpo B_i ($i = a(j)$), x_i no sea ortogonal a z_j , construimos un vector adicional u_j a lo largo de la ortogonal común a z_i y z_j. Nótese que u_j está fijado en el cuerpo B_i.

Primero se definen los ejes z_j de todos los cuerpos. Luego se fija la dirección de los ejes x_j (y u_k si es necesario). Con la definición sistemática de cada marco de cuerpo, es posible definir un conjunto de 6 parámetros para cada marco F_j, que se denotan como los parámetros MDH, que son para $i = a(j)$:

- γ_j : ángulo entre x_i y u_j alrededor de z_i.
- b_j : distancia entre x_i y u_j a lo largo de z_i.
- α_j : ángulo entre z_i y z_j alrededor de u_j.
- d_j : distancia entre z_i y z_j a lo largo de u_j.
- θ_j : ángulo entre u_j y x_j alrededor de z_j.
- r_j : distancia entre u_j y x_j a lo largo de z_j.

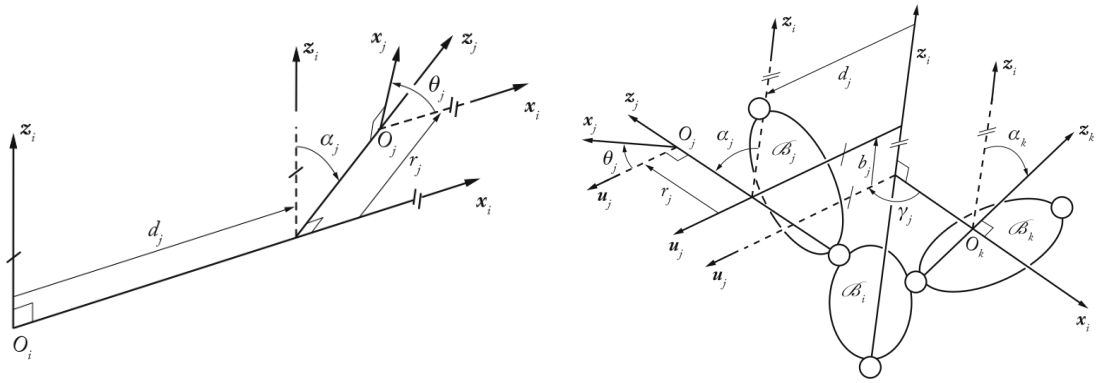
Estos seis parámetros son necesarios cuando el eje x_i del cuerpo antecedente B_i no es perpendicular a z_j y cuando se ha construido un vector adicional u_j. En los demás casos, sólo son necesarios cuatro parámetros: α_j , d_j , θ_j y r_j. De hecho, en el último caso, el vector u_j no tiene ningún papel ya que está alineado con x_i y los parámetros MDH pasan a ser igual es a los DH con $\gamma_j = 0$ y $b_j = 0$.

La Fig. 1.4 ilustra esta forma de parametrizar.

Utilizando esos parámetros, la matriz de transformación homogénea que nos permite transformar el marco F_i fijado al cuerpo B_i en el marco F_j fijado al cuerpo B_j puede escribirse como:

$$^iT_j = Rot(z,\gamma_j)Trans(z,b_j)Rot(x,\alpha_j)Trans(x,d_j)Rot(z,\theta_j)Trans(z,r_j)$$

Para los robots paralelos se abren virtualmente las cadenas cinemáticas cerradas y se procede de la misma forma que con estructuras en forma de árbol.



(a) Estructura cinemática serie. (b) Estructura cinemática de árbol.

Figura 1.4: Parámetros MDH. Figura extraída de Khalil y Briot [2].

1.5.2. Planificación y generación de trayectorias.

En robótica, la generación de trayectorias involucra el cálculo de posiciones, velocidades, aceleraciones y jerks de cada una de las articulaciones que componen el robot a lo largo de un camino geométrico de referencia, [4]. Para la planificación de una trayectoria determinada, es necesario especificar un camino geométrico, función vectorial paramétrica $p(s)$, que describe el movimiento en el espacio mediante la función o ley de tiempo $s(t)$, que caracteriza la evolución en el tiempo a lo largo de dicho camino geométrico. Por lo tanto, la trayectoria se describe como $p(t) = p(s(t))$, y es el movimiento en el tiempo a lo largo del camino geométrico. Este tipo de planteo resulta en una trayectoria coordinada o isocrona.

En la práctica, el camino geométrico se obtiene de una secuencia de puntos en el espacio cartesiano o espacio de tarea. Se verán tres tipos de movimientos entre dichos puntos en el espacio:

- Movimiento PTP: Del inglés *Point To Point*, donde cada articulación evoluciona desde su posición inicial a la final sin realizar consideración alguna sobre el estado o evolución de las demás articulaciones, Barrientos y col. [5], esto conduce a trayectorias no intuitivas en el espacio cartesiano.
- Movimiento LIN: Donde el camino geométrico entre un punto y otro es una línea recta, y se obtiene mediante interpolación en el espacio cartesiano.
- Movimiento CIRC: Donde el camino geométrico entre un punto y otro es un arco de circunferencia, y se obtiene mediante interpolación en el espacio cartesiano.

Es importante diferenciar entre **planificación de tarea** y **planificación en espacio articular**. La primera, es la determinación de la secuencia de puntos o nodos en el espacio de tarea, mientras que la segunda es la secuencia de nodos en el espacio articular. Según sea la complejidad del trayecto a realizar, será necesario primero realizar una planificación de tarea para luego pasar a la planificación en espacio articular o simplemente trabajar desde el inicio en espacio articular.

La planificación en espacio de tarea tiene ventajas como la fácil visualización de la trayectoria generada y simplifica la evasión de obstáculos. En cambio, la planificación en espacio articular no necesita cálculos de cinemática inversa y gracias a esto es posible sortear las singularidades que pudieren estar presentes en nuestro espacio de trabajo.

Como bibliografía para ampliar los conocimientos sobre la generación de trayectorias se recomiendan los siguientes 3 libros: Siciliano y col. [6], Corke [7] y Biagiotti y Melchiorri [4].

1.5.2.1. Ley de tiempo

Para obtener una determinada trayectoria, el camino es regido por una ley de tiempo. Esta depende de:

- Especificaciones de tarea: Características que tiene la tarea a realizar, como detenerse en algún punto del espacio, moverse a velocidad constante, etc.

- Criterio de optimización: Realizar los caminos en el menor tiempo posible, minimizar energía, camino con las configuraciones articulares que provoquen menor torque sobre la estructura del robot, etc.
- Restricciones diversas: Las impuestas por las capacidades de los actuadores o por la tarea en sí, máximo torque, máxima velocidad, máxima aceleración, etc.

Los requisitos en general para la generación de una ley de tiempo son que los perfiles de posición y velocidad (por lo menos) sean continuos en el tiempo; que los algoritmos para su generación sean relativamente eficientes; y evitar o minimizar efectos indeseados como curvaturas irregulares, etc.

Las principales técnicas para su generación son el uso de perfiles de velocidad trapezoidal o perfiles S, con velocidad, aceleración y *jerk*[4] acotado; o el uso de polinomios de tercer o quinto grado.

Para el análisis se optó por una ley de tiempo con perfil de velocidad trapezoidal (o también conocida en ingles como *linear segments with parabolic blends* LSPB) y un polinomio de grado quinto.

Para el perfil trapezoidal, se divide el movimiento en tres etapas:

$$\text{Aceleración} \rightarrow \text{Velocidad constante} \rightarrow \text{Desaceleración}$$

Las distintas ecuaciones utilizadas para generar nuestra curva de posición son:

- Etapa de aceleración: $s(t) = at^2/2,\ t \in [0, \tau]$
- Etapa de velocidad constante: $s(t) = s(\tau) + v(t - \tau),\ t \in [\tau, T]$
- Etapa de desaceleración: $s(t) = s(T) + v(t - T) - a(t - T)^2/2,\ t \in [T, T + \tau]$

Donde $v(t) = \frac{ds(t)}{dt}$ es la velocidad en función del tiempo, $a(t) = \frac{d^2s(t)}{dt^2}$ es la aceleración en función del tiempo, τ es el tiempo de aceleración y desaceleración y T es el tiempo en el cual se comienza a desacelerar. Con estas ecuaciones se puede obtener una ley de tiempo como la de la Fig. 1.5.

A las ecuaciones anteriores se imponen condiciones de borde como las que siguen:

$$\begin{cases} s(0) = 0 \\ s(T + \tau) = 1 \\ \dot{s}(0) = v_{inicial} \\ \dot{s}(T + \tau) = v_{final} \end{cases} \tag{1.1}$$

Siendo $v_{inicial}$ y v_{final}, la velocidad inicial y final, estas podrían variar si se quiere inicial o terminar con una velocidad deseada. Para nuestro caso se decide hacer una perfil de velocidad trapezoidal con velocidad inicial y final iguales a 0.

[4] Jerk: Derivada de la aceleración respecto al tiempo, o cuarta deriva de la posición respecto al tiempo.

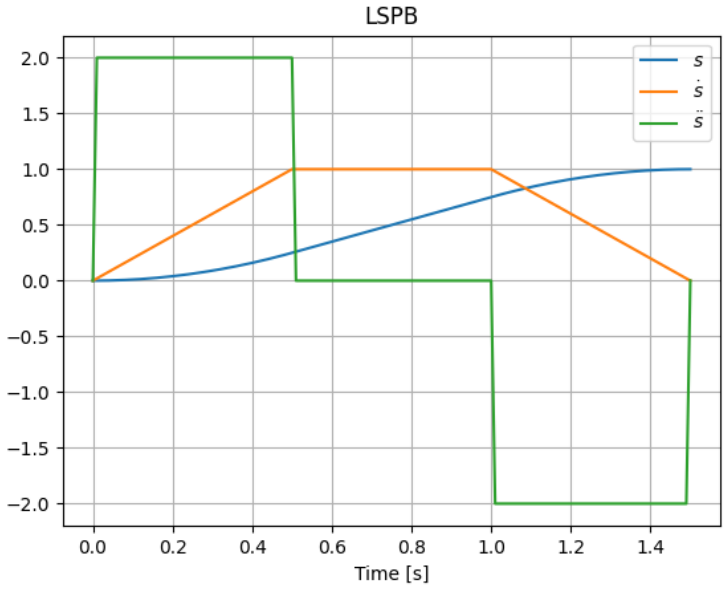


Figura 1.5: Ley de tiempo con perfil de velocidad trapezoidal.

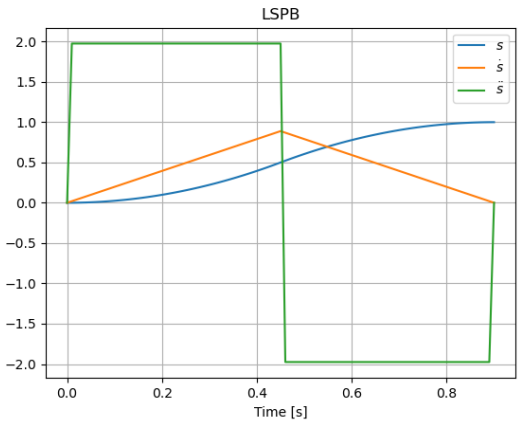
Partiendo de nuestro perfil de velocidad trapezoidal e integrándolo geométricamente para obtener la posición final se obtiene que $T = \frac{\Delta p}{V}$ siendo $\Delta p = p_f - p_i$ la diferencia entre la posición inicial y final, siendo esta igual a 1 debido a que es una curva paramétrica, se puede obtenerse la velocidad máxima escalada $V = 1/T$, luego por movimiento rectilíneo uniformemente variado se sabe que $a = \frac{V}{\tau}$, sustituyendo V se obtiene la aceleración máxima escalada $a = 1/(T\tau)$. Esto hace que nuestra ley de tiempo quede totalmente definida por los parámetros T y τ.

Las ecuaciones anteriores nos permiten expresar T y τ en función de la aceleración, velocidad máxima y el desplazamiento máximo cartesiano, y viceversa.

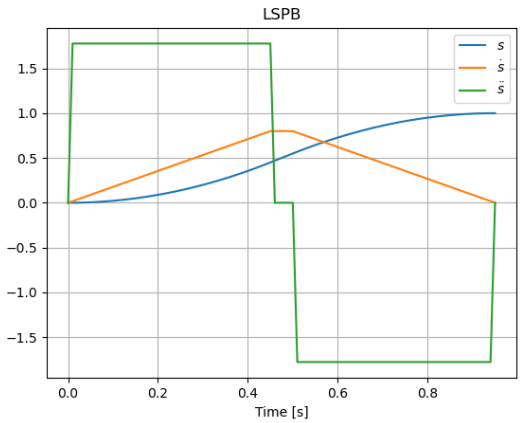
Teniendo en cuenta que el tiempo de la etapa de velocidad constante es $T - \tau$, esta puede ser 0 o negativa si $T \leq \tau$. Esto pasa cuando el desplazamiento, Δp, es relativamente pequeño. En este caso la máxima velocidad no es alcanzada si la máxima aceleración es impuesta, entonces, T y τ necesitan ser recalculados, Yoon y col. [8].

Para recalcular τ se usa la ecuación 1.2, ecuación utilizada para generar un perfil triangular. Ahora, para recalcular T, se tienen dos opciones: la primera es mantener el perfil triangular, este perfil no tendría una etapa de velocidad constante, y la segunda es mantener el perfil trapezoidal. Si se decide continuar con un perfil triangular se elige $T = \tau$, este perfil no posee una etapa de velocidad constante. Si se decide mantener el perfil trapezoidal se utiliza el valor de $T = \frac{v_{max}}{a_{max}}$ (valor previo calculado de τ que nos asegura que $T > \tau$). Esta ultima opción provoca que se alcance una velocidad menor a la máxima en esta trayectoria, pero reduce la exigencia de los motores al no tener que pasar de a_{max} a $-a_{max}$ en el mismo instante de tiempo como en el perfil triangular (ver Fig. 1.6).

$$\Delta p = v_{max}\tau = a_{max}\tau^2 \quad \Rightarrow \quad \tau = \sqrt{\frac{\Delta p}{a_{max}}} \tag{1.2}$$



(a) Perfil Triangular.



(b) Perfil Trapezoidal.

Figura 1.6: Opciones de perfiles al recalcular T y τ.

Cabe destacar que si se tiene que coordinar varios actuadores, se computan los parámetros T y τ de cada una de las articulaciones y para el espacio de tarea o espacio articular, eligiéndose el máximo de estos y computando una sola ley de tiempo para todo el movimiento.

Por cuestiones de simplicidad, en nuestra generación de trayectorias se optó por solo computar los parámetros del espacio de tarea, que en muchas ocasiones suele ser el movimiento más largo y/o más lento. Luego, se debe inspeccionar visualmente las curvas de las articulaciones verificando que no se sobrepase ningún límite.

Las ventajas de usar una ley de tiempo con perfil de velocidad trapezoidal son la limitación de los picos de aceleración, la posibilidad de implementar perfiles tales que los actuadores trabajen en su zona de mayor rendimiento y menor exigencia, y ser ampliamente utilizados en la industria por ser muy práctica su implementación. Su principal desventaja es que, a pesar de que el jerk se ve mejorado en comparación a un perfil de velocidad escalonado, siguen existiendo picos en cada cambio de etapa, que pueden provocar, por ejemplo, vibración en el sistema. Sin embargo, se asume que el controlador del robot posee las características suficientes para cumplir los perfiles planteados.

Una alternativa a los perfiles de velocidades trapezoidales son los perfiles polinomiales. Estos generan curvas mas suaves, entiéndase por suavidad que las derivadas primera, segunda y aveces, la tercera del polinomio, siguen siendo continuas. Esta técnica consiste en generar un polinomio de cierto grado en función del tiempo. Siendo un buen candidato para implementar en nuestro robot un polinomio de grado 5. A continuación, en la ecuación 1.3, se puede observar el polinomio de grado 5 y sus dos siguientes derivadas respecto al tiempo. Estas dos ultimas siguen siendo suaves.

$$\begin{aligned} s(t) &= a_5 * t^5 + a_4 * t^4 + a_3 * t^4 + a_2 * t^2 + a_1 * t + a_0 \\ \dot{s}(t) &= 5 * a_5 * t^4 + 4 * a_4 * t^3 + 3 * a_3 * t^2 + 2 * a_2 * t^1 + a_1 \\ \ddot{s}(t) &= 20 * a_5 * t^3 + 12 * a_4 * t^2 + 6 * a_3 * t + 2 * a_2 \end{aligned} \tag{1.3}$$

Donde $t \in [0, T]$.

Luego, para resolver el sistema de ecuaciones, es necesarios definir 6 condiciones de contorno, 2 para cada polinomio. En general las condiciones de bordes para la velocidad y aceleración son 0. De esta manera nuestra trayectoria queda completamente definida por los valores de posición, velocidad y aceleración inicial y final:

$$\begin{cases} s(0) = s_0 \\ s(T) = s_T \\ \dot{s}(0) = \dot{s}_0 \\ \dot{s}(T) = \dot{s}_T \\ \ddot{s}(0) = \ddot{s}_0 \\ \ddot{s}(T) = \ddot{s}_T \end{cases} \tag{1.4}$$

En forma matricial:

$$\begin{bmatrix} s(0) \\ \dot{s}(0) \\ \ddot{s}(0) \\ s(T) \\ \dot{s}(T) \\ \ddot{s}(T) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & T & T^2 & T^3 & T^4 & T^5 \\ 0 & 1 & 2T & 3T^2 & 4T^3 & 5T^4 \\ 0 & 0 & 2 & 6T & 12T^2 & 20T^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} \tag{1.5}$$

En la Fig. 1.7 se pueden ver las trayectorias generadas.

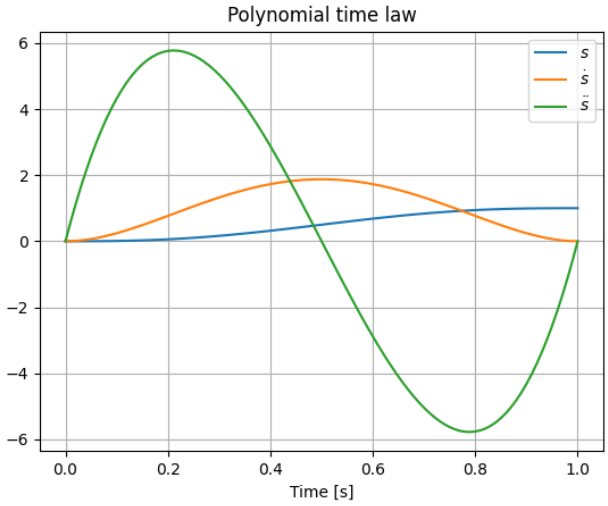


Figura 1.7: Ley de tiempo polinomio de grado 5.

La principal desventaja de estas trayectorias es que se requiere más tiempo para lograr el mismo desplazamiento que con perfiles trapezoidales. Otra desventaja es que para limitar la velocidad y aceleración máxima y la aceleración y velocidad al principio y fin de la trayectoria al mismo tiempo es necesario definir polinomios de mayor orden.

Cabe destacar la existencia de muchos perfiles de velocidad, un punto intermedio entre los dos expuestos en esta sección y muy usado en la industria es la interpolación lineal con ajuste cúbico o también llamada doble S, este perfil disminuye el jerk mediante la generación de un perfil de aceleración lineal. Para una lista exhaustiva de perfiles se puede dirigir a Biagiotti y Melchiorri [4].

1.5.2.2. Movimiento punto a punto en espacio articular

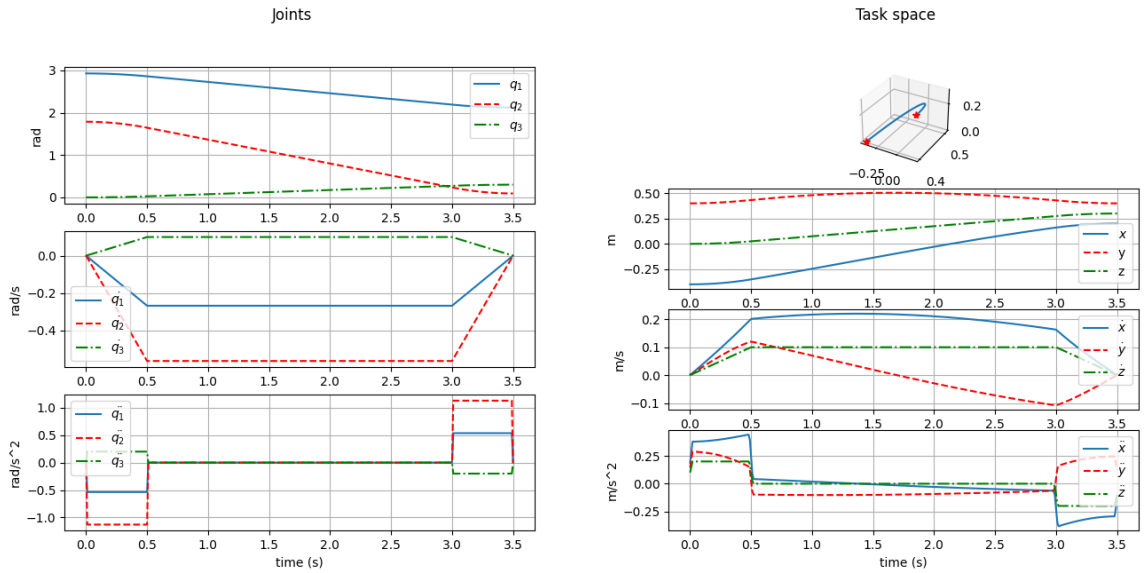
Como se dijo anteriormente, todos los movimientos del robot serán síncronos, es decir, se usará la misma ley de tiempo para todas las articulaciones.

Para generar la ley de tiempo se calculan los parámetros T y τ para todas las articulaciones. Luego, se buscan las articulaciones que cinemáticamente esté más exigida, es decir, se seleccionan los mayores valores para T y τ, esto nos asegura que siempre todas las articulaciones estén por debajo de su velocidad y aceleración máxima. Con estos valores se obtiene nuestra ley $s(t)$, esta controla qué tan rápido el camino va a ser seguido.

Luego, mediante la ecuación 1.6 se obtienen los puntos entre el punto objetivo y el inicial de cada articulación.

$$q_i(t) = q_i^{inicio} + s(t)(q_i^{objetivo} - q_i^{inicio}) \quad \forall i = 1, 2, 3 \tag{1.6}$$

En las Fig. 1.8a se muestran los resultados para una trayectoria desde $q^{inicio} = \begin{bmatrix} -0{,}4 & 0{,}4 & 0 \end{bmatrix}$ hasta $q^{objetivo} = \begin{bmatrix} 0{,}2 & 0{,}4 & 0{,}3 \end{bmatrix}$. Y por cinemática directa se obtiene la Fig. 1.8b.



(a) Espacio articular.

(b) Espacio de tarea.

Figura 1.8: Movimiento P2P.

1.5.2.3. Movimiento en espacio cartesiano

Para un movimiento entre nodos consecutivos en el espacio cartesiano, existen varias maneras de realizar dicho movimiento. Una de ellas es definir puntos de inicio y objetivos y proponer velocidad cero en cada nodo del camino geométrico, esto hace ineficiente la tarea si el punto objetivo es solo un punto de paso y no se requiere que el robot se detenga obligadamente por ese punto. Para mejorar esto, se puede utilizar otra técnica que le da continuidad al movimiento, por ejemplo la aproximación. Esto evade el problema de estar frenándose en cada nodo, entonces mediante el redondeo de las esquinas y la anticipación al cambio de nodo, se logran trayectorias más continuas. A continuación se ven estas dos maneras de realizar dichas trayectorias.

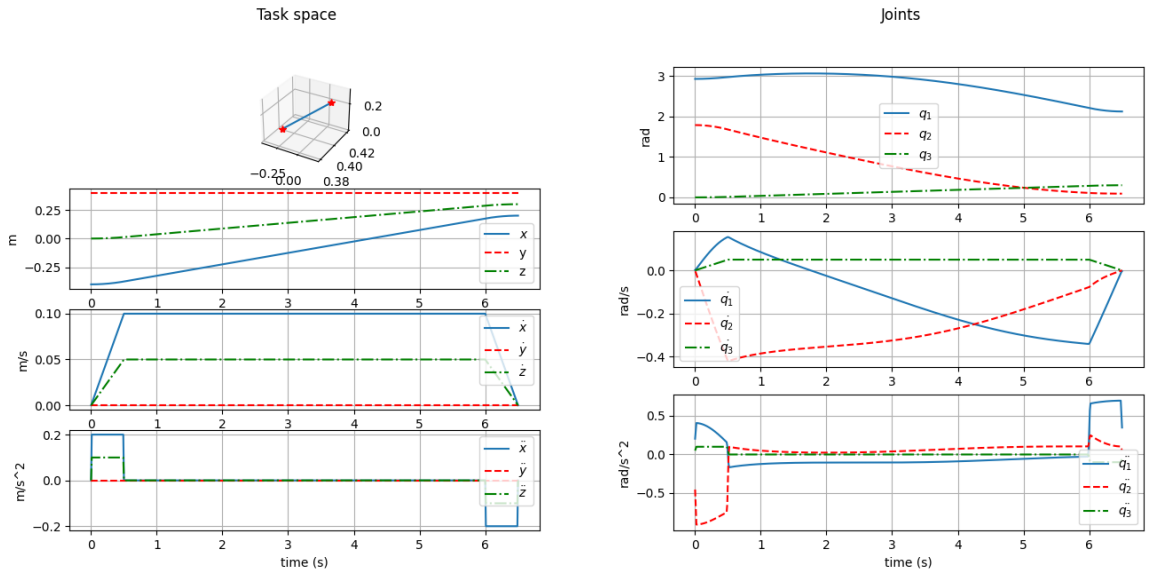
1.5.2.4. Movimiento en línea recta

Se utiliza una ley de tiempo para una v_{max}, a_{max} y Δp dadas en el espacio de tarea. Luego, se calculan los parámetros T y τ y se obtiene nuestra ley $s(t)$.

Una vez obtenida la ley de tiempo y mediante la ecuación 1.7 se obtienen los puntos x, y y z entre la posición objetivo y la inicial de cada variable cartesiana.

$$P_i(t) = P_i^{inicio} + s(t)(P_i^{objetivo} - P_i^{inicio}) \quad \forall i = x, y, z \tag{1.7}$$

En las Fig. 1.9a se muestran los resultados para una trayectoria desde $P^{inicio} = \begin{bmatrix} -0{,}4 & 0{,}4 & 0 \end{bmatrix}$ hasta $P^{objetivo} = \begin{bmatrix} 0{,}2 & 0{,}4 & 0{,}3 \end{bmatrix}$. Y por cinemática inversa se obtiene la Fig. 1.9b.



(a) Espacio de tarea.

(b) Espacio articular.

Figura 1.9: Movimiento línea rectar.

1.5.2.5. Movimiento continuo

En este movimiento se necesitan al menos 3 puntos, P^{inicio}, $P^{siguiente}$ y $P^{objetivo}$, siendo el punto intermedio el punto de paso, el cual se aproxima.

Para realizar esta trayectoria se necesita computar al menos 2 leyes de tiempo (una para cada tramo), $s(t)$ y $s'(t)$, con diferentes parámetros cada una (T y τ para la primera ley y T' y τ' para la segunda).

Luego, mediante la ecuación 1.8 se obtienen los puntos x, y y z entre la posición objetivo y la inicial. Teniendo expresiones similares para el espacio articular. Desde un punto de vista geométrico es la superposición de los perfiles de velocidad de la trayectoria $s(t)$ y $s'(t)$ (ver imagen 1.10).

$$\begin{cases} P(t) = P^{inicio} + s(t)(P^{siguiente} - P^{inicio}) & \text{si} \quad t < t_s \\ P(t) = P^{inicio} + s(t)(P^{siguiente} - P^{inicio}) + s'(t - t_s)(P^{objetivo} - P^{siguiente}) & \text{si} \quad t \geq t_s \end{cases} \tag{1.8}$$

Donde t_s es el tiempo de cambio y puede ser elegido debido a diferentes criterios, estos pueden ser basados en tiempo, posición o velocidad. Una de las más simples es la basada en el tiempo $t_s = T$.

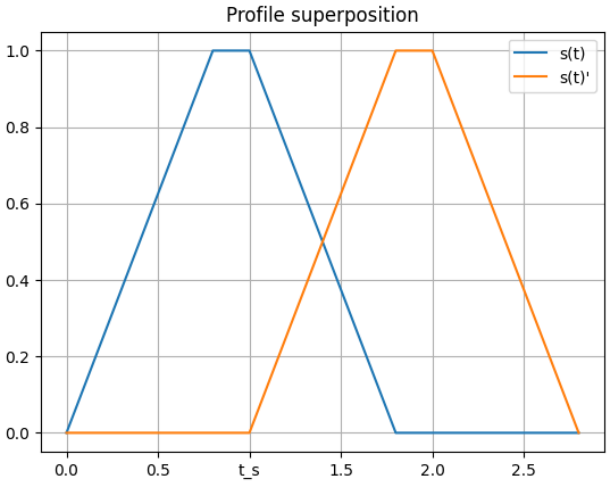
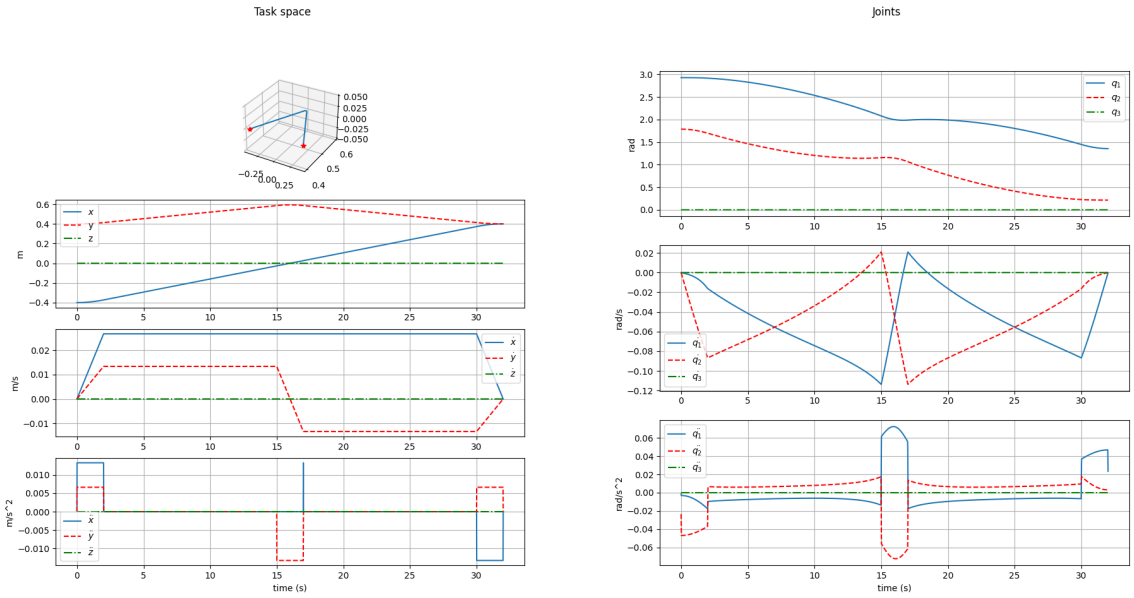


Figura 1.10: Superposición de los perfiles de velocidad.

En las Fig. 1.11a se muestran los resultados para una trayectoria desde $P^{inicio} = \begin{bmatrix}-0{,}4 & 0{,}4 & 0\end{bmatrix}$, pasa por el punto $P^{siguiente} = \begin{bmatrix}0{,}0 & 0{,}6 & 0\end{bmatrix}$ y finaliza en el punto $P^{objetivo} = \begin{bmatrix}0{,}4 & 0{,}4 & 0\end{bmatrix}$. En las curvas se puede observar que el robot no se detiene ni pasa por el punto $P^{siguiente}$ (ver Fig. 1.12). Luego, la Fig. 1.11b es obtenida por cinemática inversa.



(a) Espacio de tarea. (b) Espacio articular.

Figura 1.11: Movimiento línea recta con punto de paso.

Finalmente, la ecuación 1.8 se extiende a m puntos.

$$\begin{cases} P(t) = P^n + s(t-t_n)(P^{n+1} - P^n) \quad \text{si} \quad t < t_{sn} \\ P(t) = P^n + s(t-t_n)(P^{n+1} - P^n) + s'(t-t_{sn})(P^{n+2} - P^{n+1}) \quad \text{si} \quad t \geq t_{sn} \end{cases} \tag{1.9}$$

Para $n = 1, 2, ..., m-2$, $t_n = T_{n-1}$ y $t_{sn} = T_n$ con $T_{n-1} = 0$.

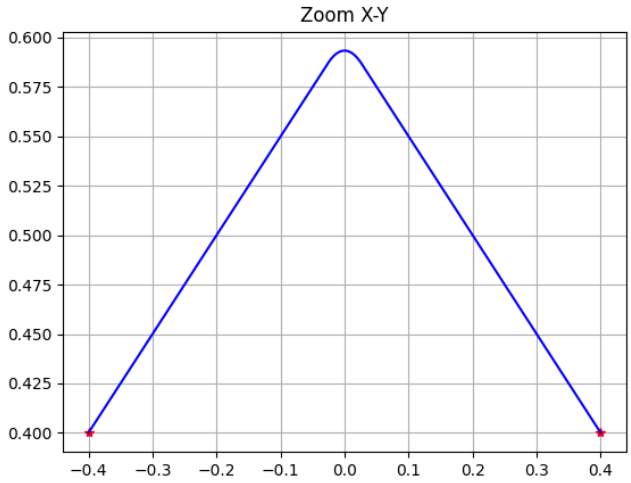


Figura 1.12: Zoom en el plano xy donde se aprecia la aproximación.

1.5.2.6. Movimiento en trayecto circular

Para generar una trayectoria circular se utiliza una matriz de transformación homogénea ${}^0T_C$, para definir el centro del círculo, y se utiliza la representación en coordenadas polares referidas al sistema del centro y parametrizadas por la longitud de arco ψ. Además, se selecciona un punto P^{inicio} para comenzar la trayectoria. A continuación se muestran las ecuaciones paramétricas del círculo.

$$P'(\psi) = \begin{bmatrix} \rho \cos \frac{\psi}{\rho} \\ \rho \sin \frac{\psi}{\rho} \\ 0 \end{bmatrix}$$

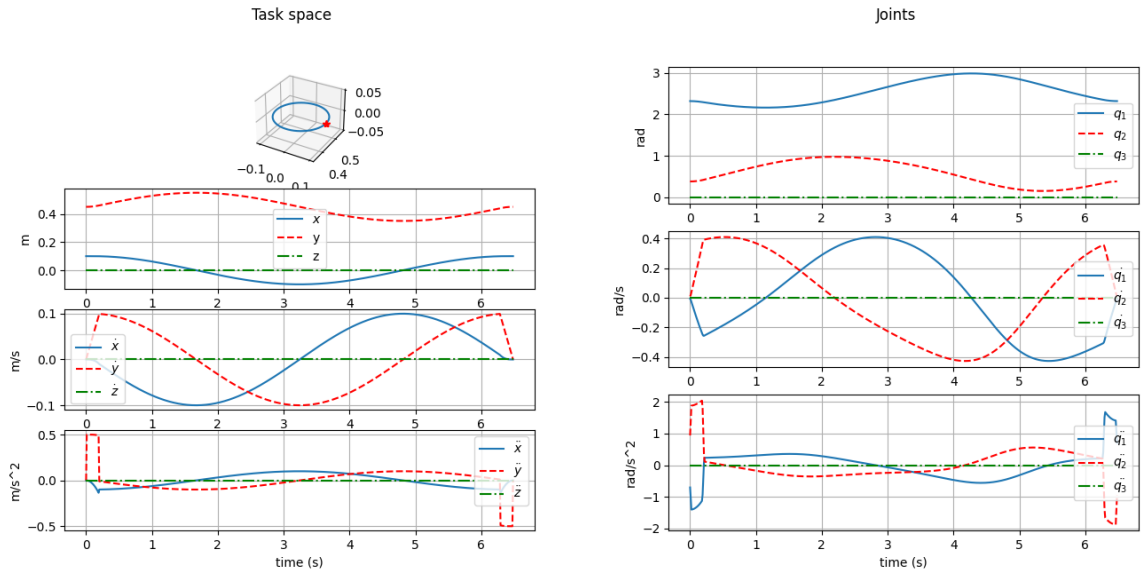
Donde $\rho = \sqrt{\sum (P_i^{inicial} - C_i)} \quad \forall i = x, y, z$, siendo C_i igual a la traslación de ${}^0T_C$.

Ahora, para un sistema de referencia distinto, el camino se convierte en $P(\psi) = C + {}^0R_C P'(\psi)$, donde ${}^0R_C$ es igual a la rotación de ${}^0T_C$. Así, para interpolar nuestra trayectoria queda solamente definir la evolución de $\psi(t)$ en base a nuestra ley de tiempo (ecuación 1.10).

$$\psi(t) = \psi^{inicial} + s(t)(\psi^{objetivo} - \psi^{inicial}) \tag{1.10}$$

En el caso de querer un círculo completo, se debe cumplir que $\psi^{objetivo} = \psi^{inicial} + 2\pi$, para el caso de un círculo incompleto, bastará con encontrar un $\psi^{objetivo}$ apropiado. Ejemplo de este caso sería un arco de circunferencia dado 3 puntos no lineales.

En las Fig. 1.13a se muestran los resultados para una trayectoria circular con centro en $C = \begin{bmatrix} 0 & 0{,}45 & 0 \end{bmatrix}$ y punto inicial en $P^{inicial} = \begin{bmatrix} 0{,}1 & 0{,}45 & 0 \end{bmatrix}$. Luego, por cinemática inversa se obtiene la Fig. 1.13b



(a) Espacio de tarea.

(b) Espacio articular.

Figura 1.13: Movimiento circular.

1.5.3. Comunicaciones

En esta sección se detallarán los protocolos de comunicación utilizados en el sistema de control: CAN y UART. Se realizará una breve reseña del primero junto con una descripción de la estructura de mensajes utilizada, cuyas líneas generales fueron tomadas de CANopen. Para eso se estudiaron y compilaron los trabajos de Di Natale et al. [9], Lawrenz (ed.) [10] y Pfeiffer, Ayre y Keydel [11]. Estos libros y, particularmente el de Pfeiffer *et al.* son un excelente punto de partida para interiorizarse en CAN y CANopen y utilizar como bibliografía de referencia.

Finalmente, al final de la sección se listarán los comandos para la comunicación UART y se introducirá la técnica de control de flujo, utilizada en comunicaciones donde uno de los nodos puede generar información más rápido que lo que el otro nodo puede consumir.

1.5.3.1. CAN

El bus CAN (*Controller Area Network*) es un protocolo de comunicación que se puede ubicar en las capas física y de enlace (1 y 2) dentro del modelo OSI. Fue inventado a mediados de los años '80 por *Bosch* para el intercambio de información en automóviles. Es eficiente, flexible y robusto ante interferencia electromagnética.

CANopen, por otro lado, es un protocolo de comunicaciones que implementa la capa de red y las capas superiores. Los perfiles de comunicaciones básicos están dados en la especificación CiA301 estandarizada por la organización CAN in Automation (CiA) y cuya implementación se puede consultar en Schneider Electric [12]. Perfiles para dispositivos especializados están dados en las especificaciones 4xx, por ejemplo, CiA402 para control de movimiento. Los manuales de campo de Schneider Electric [13] y Elmo Motion Control [14] describen implementaciones de dicho estándar.

A continuación se describen algunos aspectos clave de CAN y CANopen.

Controller Area Network Luego de la presentación de CAN en 1986, se realizaron actualizaciones a la especificación, naciendo de esta forma CAN 2.0A y 2.0B. La diferencia entre estos es la longitud de los identificadores de mensaje CAN. CAN 2.0A también es llamada trama base y utiliza identificadores de 11 bits, mientras que CAN 2.0B se denomina trama extendida y utiliza 29 bits. CANopen está diagramado para funcionar normalmente con trama base, por lo que esta es la que más se utiliza, aunque CANopen permita identificadores de 29 bits.

Las tramas de mensajes CAN fueron diseñadas para enviar datos pequeños frecuentemente. Por esto es que una trama puede contener hasta 8 bytes de datos. El *overhead* de cada trama incluye el identificador

de 11 bits y un cálculo de redundancia cíclica (CRC) de 15 bits. Una trama CAN puede tener hasta un 50 % de *overhead* en cada trama y esto lo hace un protocolo muy seguro y confiable, ya que todos los nodos en la red tienen que verificar y confirmar el CRC.[5]

El medio físico más común es un par de cables trenzados y la longitud del bus depende del *bitrate*. A 1Mbps, la distancia máxima es de 40m, aproximadamente. A medida que se requieren mayores distancias de bus, las velocidades deben bajar.

CAN es una red multimaestro, por lo que cada nodo puede enviar sus datos en cualquier momento. Las colisiones se resuelven por prioridad. El mensaje con el identificador de mensaje más bajo gana el arbitraje y es transmitido. Esta colisión no es destructiva. Se puede considerar que todos los mensajes son enviados como *broadcast*, y cada nodo individual debe verificar si el mensaje transmitido es para él o no. Para evitar que todos los nodos tengan que verificar esto constantemente, los controladores CAN cuentan con filtros implementados en hardware.

Estados de la señal La capa física está basada en la norma ISO 11898 y especifica el uso de transceptores CAN de alta velocidad. Estos transceptores se colocan entre el controlador CAN y el medio físico: el bus en cuestión.

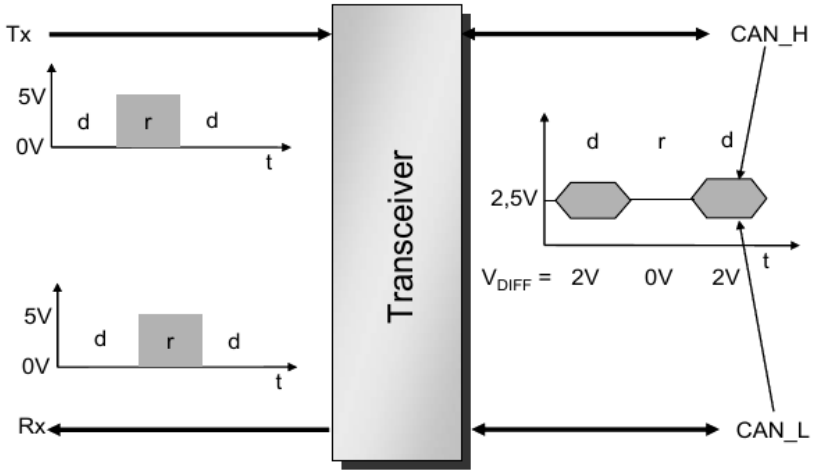


Figura 1.14: Transceptor CAN.

Las señales en el bus tienen dos estados: dominante (d) y recesivo (r). Un 1 lógico es un estado recesivo y se representa por una diferencia de cero entre CANH y CANL. Un 0 lógico es un estado dominante y se representa por una diferencia de 2V entre CANH y CANL. Esto se puede observar en la Fig. 1.14. En estado recesivo, CANH y CANL están a 2.5V; en estado dominante, CANH sube 1V a 3.5V y CANL baja 1V a 1.5V, por lo que la diferencia entre ambos es de 2V.

En CAN, una señal dominante sobreescribe una señal recesiva. Si varios nodos escriben en el bus a la vez, este va a estar en estado dominante si uno solo de ellos escribe una señal dominante. Este mecanismo se utiliza para detectar colisiones. Si un nodo escribe un nivel recesivo y lee un nivel dominante, significa que hubo una colisión y perdió el arbitraje. Este comportamiento implica que CAN se comporta como una AND cableada: solo basta que un solo nodo escriba un cero (dominante) para que el bus esté en estado dominante. Solo si todos los nodos escriben un uno (recesivo), el bus se encuentra en estado recesivo.[6]

El par CANH y CANL forman un par diferencial de señales, cuya mayor ventaja es la inmunidad a interferencia electromagnética (EMI). Ambos cables se trenzan para proveer más inmunidad aún.

Durante un tiempo "muerto" o cuando el bus está inactivo, el nivel del bus es el estado recesivo.

Para codificar los bits, CAN utiliza no retorno a cero (NRZ), por lo que no hay transiciones o flancos durante la validez de un bit. Dentro de las tramas de datos, CAN utiliza una técnica llamada bit-stuffing, en la cual si se envían 5 niveles lógicos idénticos seguidos, el siguiente será del nivel opuesto. Este bit extra se elimina por hardware. Esto se realiza para propósitos de sincronización entre los nodos; los flancos son necesarios para ajustar los clocks locales, ya que no hay una línea dedicada de reloj.

[5] CAN tiene una distancia de Hamming teórica de 6, por lo que puede detectar errores en hasta 5 bits.

[6] Un comportamiento análogo se observa en el arbitraje que se da en el protocolo I2C cuando varios dispositivos intentan transmitir a la vez.

Cableado y topología Se recomienda utilizar cables trenzados para CANH y CANL y, en ambientes muy ruidosos eléctricamente, se recomienda utilizar un cable mallado. Además, una GND común se recomienda para una operación confiable. Muchas aplicaciones industriales utilizan dos pares de cables trenzados; un par conteniendo CANH y CANL y el otro para una GND común y tensión de alimentación.

El cableado y los conectores no son partes de las especificaciones, pero la CiA en la DRP303 propone algunas disposiciones para conectores comunes como DB9, RJ10 o RJ45.

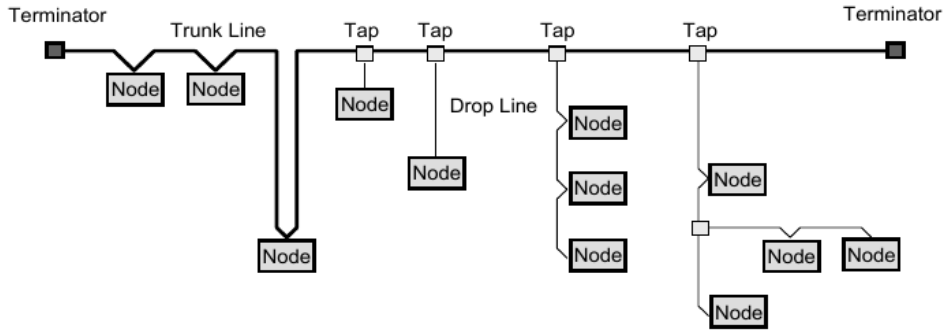


Figura 1.15: Topología de bus lineal. Se observan varios nodos "colgados"del bus y los elementos de terminación.

La topología de la red es generalmente la de un bus lineal que consiste en las líneas CANH y CANL terminadas en los extremos por resistencias de 120Ω (para buses a 1Mbps).[7] Esto se observa en la Fig. 1.15.

Trama base CAN La trama base CAN, como se observa en la Fig. 1.16, comienza con bit de start dominante. Luego sigue el identificador de mensaje CAN (de 11 bits usando CAN2.0A). Este campo es parte del arbitraje, por lo que cada identificador de mensaje debe ser único en cada red. En CANopen, los tres bits que siguen deben ser considerados reservados y dejados en cero. DLC es la longitud de datos a enviar, de 0 a 8 bytes, los cuales ocupan el campo inmediatamente posterior. Luego se envía el CRC de 15 bits y se deja un tiempo para que los nodos calculen el CRC y hagan el ACK. La trama de datos finaliza con una secuencia de 7 bits recesivos consecutivos.

Colisión y arbitraje El proceso de arbitraje es una de las características más importantes de CAN. Asegura que, de ocurrir una colisión, esta es resuelta por prioridad y el mensaje con mayor prioridad ganará el arbitraje. Este proceso se suele denominar CSMA/CA (Carrier Sense, Multiple Access with Collision Avoidance). Estrictamente, en CAN no ocurren colisiones, ya que uno de los nodos gana el arbitraje y sigue transmitiendo la trama; no se debe interrumpir la transmisión como sucede en otros protocolos.

El arbitraje se realiza en el campo de identificador de mensaje. Los nodos escriben con el bit más significativo primero. Hay que tener en cuenta que los mensajes son únicos en una red, que la escritura de un 0 sobreescribe a un 1 y que cada nodo cuando escribe un bit también lo lee. De esta forma, los nodos que envían un 1 y leen un 0 han perdido el arbitraje. La prioridad está implícita en el identificador de mensaje; identificadores con valores numéricos más bajos tienen mayor prioridad.

1.5.3.2. Trama basada en CANopen

CANopen es un protocolo de comunicación utilizado en sistemas embebidos empleados en automatización. CANopen implementa las capas de red y superiores en el modelo OSI, aunque usualmente se lo toma como únicamente un protocolo de aplicación. CANopen está apoyado directamente sobre CAN (capas 1 y 2) aunque también se puede utilizar sobre Ethernet o EtherCAT.

En los siguientes apartados se mencionan algunas características importantes del protocolo CANopen. No fue el objetivo del trabajo hacer un resumen exhaustivo del protocolo y tampoco lo fue implementar una comunicación CANopen *compliant*, por lo que ciertas características, críticas para un nodo CANopen (como la máquina de estados de comunicación, controlada mediante NMT), se han obviado. Esto se debe

[7] Este valor está dado por la impedancia característica de la línea de transmisión que forma el bus.

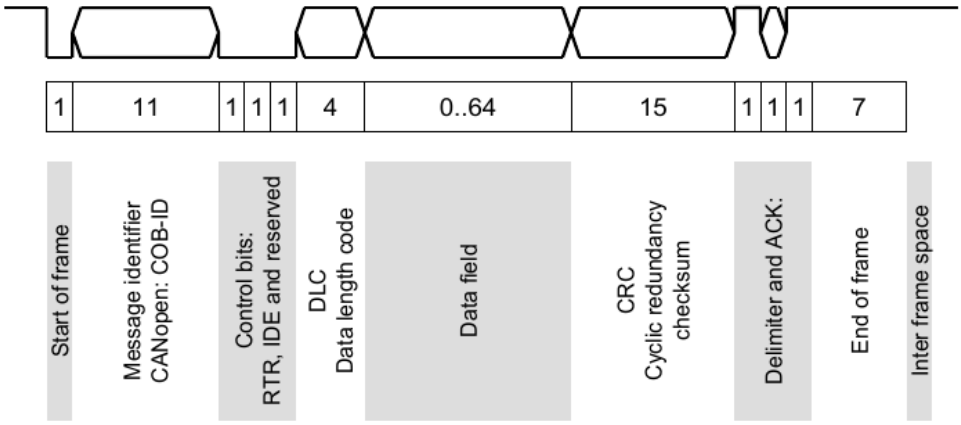


Figura 1.16: Trama base CAN.

mayormente a que en este trabajo se ha tenido que trabajar en implementar las funciones de CANopen deseadas tanto en el nodo orquestrador (o maestro, según CiA301) como en el extremo (o esclavo). En aplicaciones típicas industriales el equipo de desarrollo compra dispositivos CANopen tales como drivers de motores, encoders, joysticks, etc. a alguna compañía especializada que ha testeado y certificado sus dispositivos ante CiA. Es muy poco común que una empresa dedique su tiempo a implementar la DS402 en algún desarrollo si no va a venderlo en masa, sobre todo teniendo en cuenta que ese estándar es cerrado para miembros pagos de CiA. Lo que sí es más usual es desarrollar las funcionalidades para comandar los nodos esclavos; es decir, implementar un nodo maestro. Para eso se pueden utilizar librerías pagas que son MISRA C *compliant* o tienen otras certificaciones; o librerías *FOSS*[8] como CANopenNode. Por esta razón las funcionalidades de CANopen que se implementaron en este proyecto son muy limitadas.

Diccionario de Objetos (OD) El corazón de cada nodo CANopen es su diccionario de objetos (OD), una tabla con un índice de 16 bits y subíndices de 8 bits. Esto permite hasta 256 subentradas por cada índice. Toda la información y los datos relacionados con el proceso y la comunicación se almacena como entradas en posiciones predefinidas en el diccionario de objetos. Las entradas sin utilizar no se implementan. Para saber qué entrada del OD se utiliza para qué fin, se han creado diversos perfiles de dispositivo, los cuales describen los parámetros de comunicación y las entradas del OD soportadas por un cierto tipo de módulo CANopen. Existen perfiles de dispositivos para encoders, módulos de entrada/salida, controladores de movimiento y otro más.

Al escribir ciertas entradas al OD podemos instruir un nodo a realizar cierta tarea; o leyendo alguna otra entrada, se puede conocer información del nodo.

El OD contiene diferentes tipos de datos. Estos tipos pueden estar estandarizados o pueden ser custom. Los tipos de datos que consisten en múltiples bytes son enviados usando el formato little-endian, es decir, el byte menos significativo primero (LSB first).

Acceso al OD mediante SDO Cada nodo CANopen tiene implementado su diccionario de objetos, en el cual se guardan todas las variables relacionadas con el proceso e información del nodo en sí. Debe existir, entonces, una forma de poder acceder a las entradas del OD de ese nodo y poder leerlas o escribirlas. Esa forma es mediante objetos de dato de servicio o SDO. Con este método de comunicación se puede acceder a cualquier entrada del diccionario. Se destacó anteriormente que cada identificador CAN (o CAN_ID) es único en cada red, por lo que el acceso a una entrada del OD de diferentes nodos debe tener distintos CAN_ID. Es por esto que cada nodo cuenta con su NODE_ID, un valor entre 1 y 127; lo que permite hasta 127 nodos simultáneamente en un mismo bus.

Para hacer una *request* a un nodo para acceder a su OD, en SDO se utiliza el CAN_ID = 600h + NODE_ID; el nodo responderá con un CAN_ID = 580h + NODE_ID. Dentro de la zona de datos en la trama CAN (que puede almacenar de 0 a 8 bytes) se agregará el índice y subíndice del objeto que se quiere recibir o enviar.

[8] *Free and open source*

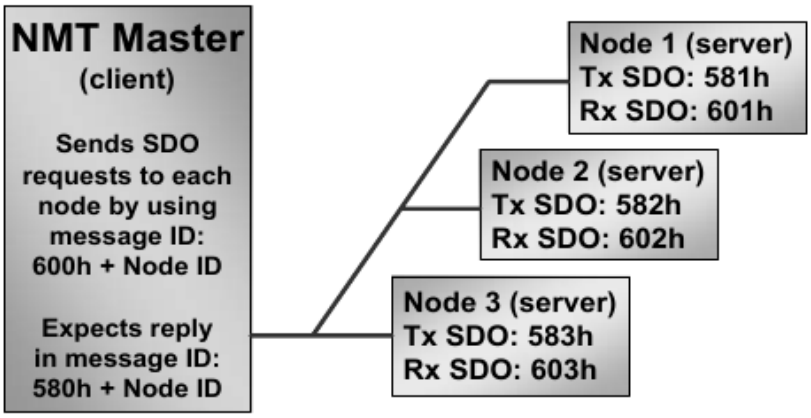


Figura 1.17: Comunicación utilizando SDO.

Los SDO proveen un mecanismo para lectura/escritura genérica al OD para cada nodo de la red. Dado que las transferencias SDO pueden acceder a todas las entradas del diccionario, se podrían transferir variables de proceso usando este método. Sin embargo este nunca fue el objetivo de los SDO y esta forma de comunicación no es muy eficiente. Esto se debe a que, en una SDO, de los 8 bytes máximos que se pueden transmitir como payload en la trama, 1 byte está reservado para configuración, los 2 siguientes para indicar el índice del objeto, el siguiente para el subíndice y únicamente quedan 4 bytes para el contenido del objeto en sí.

Las SDO presentan una gran flexibilidad pero a la vez un overhead muy grande. Por ejemplo, para transmitir una palabra de control de 16 bits y una consigna de 32 bits, se requerirían 2 SDOs. Para mensajes ocasionales de configuración esto no es problema; pero para mensajes que se envían con mucha frecuencia como las consignas a un interpolador de posición es preferible utilizar otro esquema y evitar la congestión en la red. Para eso se utilizan los PDOs.

Acceso al OD mediante PDO Los nodos CAN deben poder enviar datos cuando sea que quieran y poder colocar múltiples variables de proceso en un solo mensaje CAN. Para esto se utilizan objetos de datos de proceso o PDOs. Distinguimos entre TPDOs (transmisión) y RPDOs (recepción). Esta terminología indica si el PDO es producido o consumido por el nodo.

Las PDO relacionan un CAN_ID con un payload determinado. Por ejemplo, en nodos CANopen DS402, la PDO3 utiliza CAN_ID = 0x400 + NODE_ID[9] y los bytes de datos están compuestos de la palabra de control (entrada 6040h del OD) y la palabra de posición objetivo (entrada 607Ah del OD). Los nodos de la red interpretan el contenido de las PDO en función de los perfiles de dispositivo. Existe además la opción de configurar el contenido de cada PDO a mano.

Hay dos parámetros de configuración de un PDO: los parámetros de comunicación indican qué mensaje CAN es utilizado por el PDO y los parámetros de mapeo indican qué entradas del OD están contenidas en el PDO. Esto no se explora en la aplicación desarrollada.

Existen distintas opciones para lanzar una TPDO, estas son: por evento (cambio de estado), por temporización (cada cierto período de tiempo), por polling individual, o sincronizada (por polling grupal). Nos detenemos brevemente en esta última ya que es de importancia para este proyecto.

Si se quiere controlar un sistema con múltiples ejes que necesitan ser sincronizados, como robots de varios grados de libertad, es deseable evitar efectos de jitter y tener un movimiento suave y preciso. Para eso es necesario capturar todas las señales de los dispositivos de entrada en el mismo instante de tiempo y aplicar todas las salidas a los actuadores a la vez. Para eso, en CANopen se utiliza una señal SYNC, que es un tipo de mensaje que se utiliza únicamente para propósitos de sincronización. Esto se utiliza por ejemplo, para obtener mediciones sincronizadas de sensores; estos miden continuamente la variable y al recibir el SYNC guardan el valor y lo transmiten. También se pueden sincronizar salidas; los actuadores reciben una consigna y la guardan en un buffer pero sin aplicarla. Esperan a recibir la próxima señal

[9]Este valor es para la RPDO, es decir, el nodo que recibe la consigna. Si este mismo nodo reportara el contenido de la PDO, utilizaría el identificador correspondiente a la TPDO3, que es 0x380 + NODE_ID

de SYNC para, en ese instante, aplicar sus salidas. De esta forma todos los actuadores se mueven de forma sincronizada y en paralelo. La calidad de la sincronización dependerá de la velocidad del bus, de su longitud y del tiempo que tarden los nodos en reaccionar[10]. En los mejores casos, se pueden lograr diferencias de tan solo 100us. Si se desean sincronizaciones mejores (con menor latencia), se puede proveer una línea extra en el cableado de la red para enviar pulsos de sincronización directamente y no a través del bus CAN. De esa forma se pueden llevar las diferencias en sincronización al orden de menos de una decena de microsegundos.

Para la red, un PDO no es más que un mensaje con un identificador y hasta 8 bytes de datos. El contenido enviado dependerá del perfil de dispositivo en cuestión.

Se pueden ver los identificadores para distintos PDOs y el SDO en la Fig. 1.18. Es valioso recordar que, en caso de colisión al intentar transmitir varios mensajes a la vez, aquel mensaje cuyo CAN_ID es más bajo gana el arbitraje. Es decir, un mensaje SYNC tiene más prioridad que un PDO; y cualquier PDO tiene más prioridad que un SDO.

CAN ID			
From	To	Communication Objects	Comment
0h	--	NMT Service	From NMT Master
80h	--	SYNC Message	From SYNC Producer
81h	FFh	Emergency Message	From nodes 1 to 127
100h	--	Time Stamp Message	From timestamp producer
181h	1FFh	1st Transmit PDO	From nodes 1 to 127
201h	27Fh	1st Receive PDO	For nodes 1 to 127
281h	2FFh	2nd Transmit PDO	From nodes 1 to 127
301h	37Fh	2nd Receive PDO	For nodes 1 to 127
381h	3FFh	3rd Transmit PDO	From nodes 1 to 127
401h	47Fh	3rd Receive PDO	For nodes 1 to 127
481h	4FFh	4th Transmit PDO	From nodes 1 to 127
501h	57Fh	4th Receive PDO	For nodes 1 to 127
581h	5FFh	Transmit SDO	From nodes 1 to 127
601h	67Fh	Receive SDO	For nodes 1 to 127
701h	77Fh	NMT Error Control	From nodes 1 to 127

Figura 1.18: Conjunto predefinido de CAN_ID definidos en CANopen para el NODE_ID = 1.

Aplicación. Perfiles de dispositivo En el conjunto de estándares DS4xx, la organización CiA especifica diversos perfiles de dispositivo, que le otorgan significado a las entradas del diccionario de objetos. Es decir, vinculan un número de índice con una variable física o del proceso. El perfil para controladores de movimiento es el DS402 en el que, se indica, que se debe guardar la consigna de posición target_position en el índice 607Ah. En otros perfiles de dispositivo esa entrada puede tener otro significado, o no estar definida.

En esa especificación se establece que el nodo debe funcionar con una máquina de estados, cuyas transiciones son manejadas utilizando la palabra de control (6040h). Esta se observa en la Fig. 1.19. El estado del dispositivo se guarda en la palabra de estado (6041h). Los contenidos normales de cada RPDO incluyen en los primeros dos bytes la palabra de control, y en todos los TPDO se incluyen en los primeros dos bytes la palabra de estado.

[10]Cuando todos los nodos son iguales, como en una aplicación de control de movimiento coordinado como la de este proyecto, esta última variable pierde mucho peso.

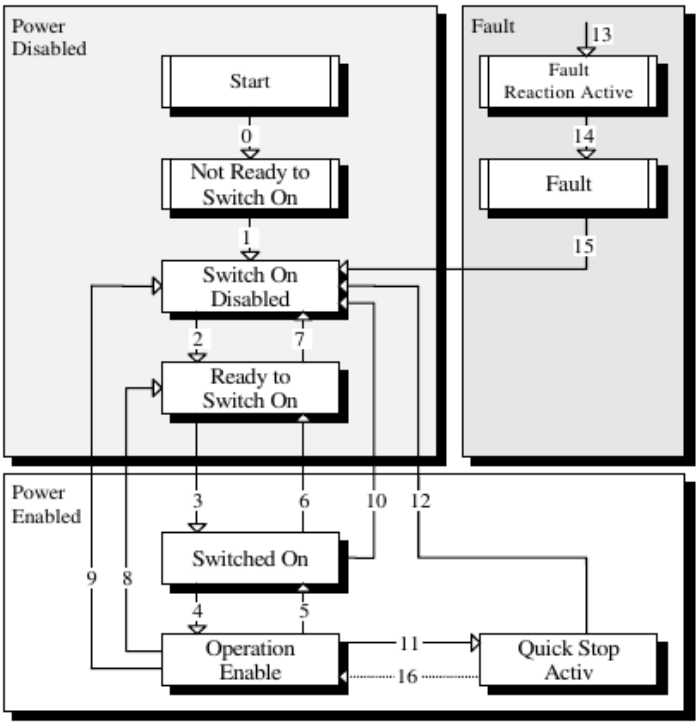


Figura 1.19: Máquina de estados definida en DS402.

En el DS402 se han establecido 6 modos de operación, los cuales se seleccionan usando el objeto 6060h. Los modos de operación son: Homing (H), Profile Position (PP), Interpolated Position (IP), Profile velocity (PV), Torque(T) y Velocity mode (V). Los valores que puede tomar el objeto 6060h se detallan en la Fig. 1.20.

Value	**Description**
-128…-2	Reserved
-1	No mode
0	Reserved
1	Profile position mode
2	Velocity (not supported)
3	Profiled velocity mode
4	Torque profiled mode
5	Reserved
6	Homing mode
7	Interpolated position mode
8…127	Reserved

Figura 1.20: Valores de 6060h según el modo de operación del nodo.

Los TPDO y RPDO fueron predefinidos por el DS402 y se observan algunos de ellos en las Fig. 1.21 y 1.22.

PDO No.	Mapping Object Index	Mapping Object Name	M/O	Comment
1	6040_h	controlword	M	controls the state machine
2	6040_h 6060_h	controlword modes_of_operation	O	controls the state machine and mode of operation
3	6040_h $607A_h$	controlword target_position	O	controls the state machine and the target position (pp)

Figura 1.21: Algunas RPDOs para un dispositivo conforme a la CiA402.

PDO No.	Mapping Object Index	Mapping Object Name	M/O	Comment
1	6041_h	statusword	M	shows status
2	6041_h 6061_h	statusword modes_of_operation_display	O	shows status and the actual mode of operation
3	6041_h 6064_h	statusword position_actual_value	O	shows the status and the actual position (pp)

Figura 1.22: Algunos TPDOs para un dispositivo conforme a la CiA402.

1.6. Metodología de trabajo y organización del informe

Por cuestiones de organización, el desarrollo del proyecto se dividió en tres etapas básicas:

- Diseño e implementación de la celda: análisis cinemático y dinámico previo y estudio mecánico con herramienta CAD. Simulación para verificar esfuerzos físicos y prototipado con fabricación aditiva (impresión 3D).
- Diseño electrónico y de control: diseño e implementación de lazo de control abierto de los actuadores y control supervisor del robot. Desarrollo de la electrónica de potencia asociada, software de control y su implementación en un sistema embebido, con un prototipo inicial que se corrigió en sucesivas iteraciones con los respectivos análisis mediante mediciones eléctricas correspondientes.
- Generación y planificación de trayectorias, integración y pruebas del prototipo terminado.

Todo el desarrollo será presentado en este informe, el cual está organizado de la siguiente manera. En el capítulo 2, se describe el modelo cinemático y dinámico del robot. Luego, en el capitulo 3, se muestra el diseño mecánico del robot en detalle. En el capítulo 4 se describe la arquitectura, el firmware y software utilizados para controlar el robot. Más tarde, en el capítulo 5, se muestran los resultados de la experimentación con el robot. Finalmente, en el capítulo 6, se detallan las conclusiones del trabajo y los trabajos futuros sugeridos.

Capítulo 2

Modelo matemático y simulación

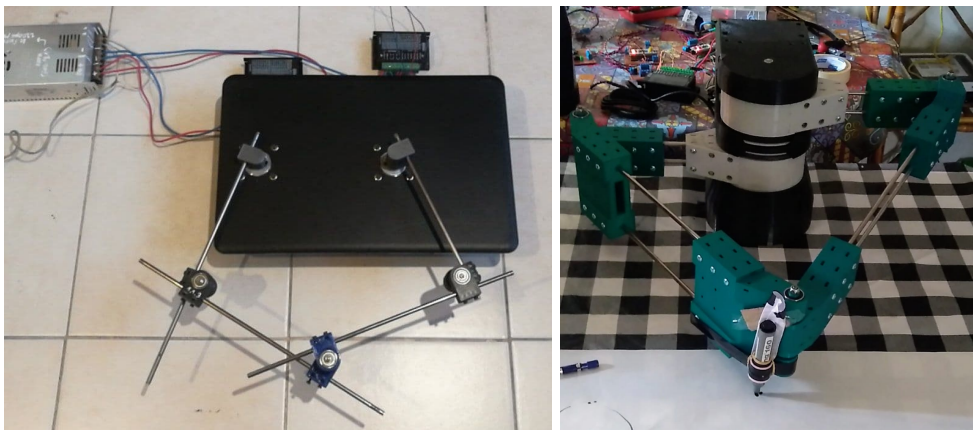
En este capítulo se describe el modelo completo del robot, este se divide en modelo dinámico y cinemático. En la sección 2.1 se describe geométricamente el robot utilizando los parámetros MDH. Luego, en la sección 2.2 se desarrolla la cinemática directa e inversa del robot. Más tarde, en la sección 2.3 se ven los modelos geométricos de primer y segundo orden, útiles para el modelo dinámico. También se analizan las singularidades y el espacio de trabajo del robot. En la sección 2.4 se describe el modelo dinámico. Finalmente, en la sección 2.5 se muestran la simulación dinámica y cinemática.

2.1. Descripción general

El robot propuesto tiene 3 grados de libertad, pero en este capítulo se analizarán los 2 principales, los que permiten posicionar el extremo del mecanismo en el plano, en una determinada coordenada xy que se encuentre dentro del espacio de trabajo del robot. Si luego se agrega un extremo operativo con posibilidad de posicionamiento en el eje vertical z, se logra la operación en el espacio xyz.

Por lo tanto, para simplificar su estudio, la estructura se divide en dos: el mecanismo de cinco barras propiamente dicho, y el eje z que permite subir y bajar el efector final.

En el transcurso del proyecto se estudiaron dos robots. El primero, similar al robot *Dextar*, de ahora en adelante **primer prototipo** (Fig. 2.1a), donde el mecanismo de cinco barras tiene las siguientes dimensiones: **205mm** para todos los eslabones ($\mathscr{B}_{11}$, $\mathscr{B}_{21}$, $\mathscr{B}_{12}$ y $\mathscr{B}_{22}$) y una distancia de **250mm** de separación entre las articulaciones actuadas q_{11} y q_{21}. Y el segundo, similar al robot *RP-5AH* fabricado por Mitsubishi Electric, de ahora en adelante **segundo prototipo** (Fig. 2.1b), cuyas dimensiones del mecanismo de cinco barras son las siguientes: **250mm** para los eslabones $\mathscr{B}_{11}$ y $\mathscr{B}_{21}$, **380mm** para los eslabones $\mathscr{B}_{12}$ y $\mathscr{B}_{22}$ y una distancia de **0mm** de separación entre las articulaciones actuadas q_{11} y q_{21}. Se describirán con más detalle en el capítulo 3.



(a) Primer prototipo construido. (b) Segundo prototipo construido.

Figura 2.1: Robot de cinco barras simulado y construido.

Los robots paralelos, también llamados manipuladores paralelos o máquinas cinemáticas paralelas (PKM), son definidas en Khalil y Briot [2] como robots que controlan el movimiento del efector final a través de al menos dos cadenas cinemáticas que van desde el efector final hasta la base fija.

Entonces a partir de esta definición se pueden distinguir los siguientes elementos constitutivos de un robot paralelo:

- La base fija o soporte.
- La plataforma móvil en la que usualmente se monta el efector final.
- Las cadenas cinemáticas que vinculan la base con la plataforma.

Como se mencionó previamente, para su análisis matemático, el robot paralelo diseñado se divide en dos:

- El mecanismo de cinco barras que trabaja sobre el plano x_0y_0.
- El eje Z sobre el extremo del mecanismo anterior, que opera por un eje que se mantiene siempre paralelo al eje z_0.

Dicha división es posible porque el eje Z es, en esencia, un eslabón en serie con la cadena cinemática cerrada que conforma el mecanismo de cinco barras, y el actuador en dicho eje Z no modificará de ninguna forma la configuración del mecanismo.

A continuación, se describe el análisis matemático y físico realizado sobre la estructura cinemática del robot en estudio. Se divide el estudio cinemático en análisis geométrico y análisis cinemático. En el primero se obtiene el modelo geométrico directo e inverso que relaciona la configuración articular del robot con la posición del extremo, y en el segundo se obtienen diferentes matrices jacobianas que relacionan velocidades articulares con velocidad cartesiana del extremo operativo (también haciendo un análisis específico para articulaciones activas y pasivas) y lo mismo para las aceleraciones.

2.2. Análisis geométrico del sistema

El modelo geométrico permite establecer la relación entre el espacio articular del mecanismo, es decir, su configuración física, y las coordenadas cartesianas de su extremo.

Para el análisis geométrico del mecanismo de 5 barras, por ser una cadena cinemática cerrada, se procede abriendo virtualmente la cadena por la articulación A_{13}, como se observa en la Fig. 2.2, obteniendo:

- La cadena cinemática 1: Un robot 3R serie planar compuesto por los eslabones $\mathscr{B}_{11}$, $\mathscr{B}_{12}$ y $\mathscr{B}_{13}$ y 3 articulaciones rotacionales cuyos ejes son paralelos a lo largo de la dirección z_0 y están ubicadas en los puntos A_{11}, A_{12} y A_{13}, la articulación ubicada en el punto A_{11} es actuada..
- La cadena cinemática 2: Un robot 2R serie planar compuesto por los eslabones $\mathscr{B}_{21}$ y $\mathscr{B}_{22}$ y 2 articulaciones rotacionales cuyos ejes son paralelos a lo largo de la dirección z_0 y están ubicadas en los puntos A_{21} y A_{22}, la articulación ubicada en el punto A_{21} es actuada.

Ambas cadenas fijas al suelo o soporte $\mathscr{B}_0$.

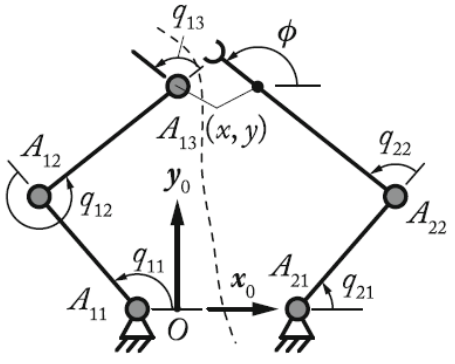


Figura 2.2: División de las dos cadenas cinemáticas. Extraído de Khalil y Briot [2].

Todas las articulaciones pasivas son agrupadas en un vector $q_d^T = \begin{bmatrix} q_{12} & q_{13} & q_{22} \end{bmatrix}$. El vector de coordenadas actuadas es $q_a^T = \begin{bmatrix} q_{11} & q_{21} \end{bmatrix}$. El efector final está posicionado en el punto A_{13} y sus coordenadas a lo largo de x_0 e y_0 son llamadas x e y respectivamente.

El método de Denavit-Hartenberg estándar fue desarrollado para estructuras en serie. Es por esto que para describir de forma consistente el robot, se utilizó la notación de Khalil y Kleinfinger [15], que extiende la descripción de Denavit-Hartenberg a robots paralelos. Los parámetros de esta descripción también se denominan parámetros de Denavit-Hartenberg modificados (MDH por sus siglas en ingles).

A continuación, en la tabla 2.1 se observan los parámetros utilizados para describir la geometría del primer prototipo. Y en la tabla 2.2, los parámetros del segundo prototipo. En estas se observan los siguientes elementos:

- ij denomina el cuerpo o eslabón $\mathscr{B}_{ij}$,
- $a(ij)$ define el antecedente del eslabón ij; por ejemplo: si el cuerpo $\mathscr{B}_{11}$ es el antecedente del cuerpo $\mathscr{B}_{12}$, entonces $11 = a(12)$.
- μ_{ij} define si la articulación es activa o pasiva (0 = pasiva, 1 = activa).
- σ_{ij} define si la articulación es rotacional (0), lineal (1) o virtual (2).
- γ_{ij}, b_{ij}, α_{ij}, d_{ij}, θ_{ij}, r_{ij} son los 6 parámetros MDH.

ij	$a(ij)$	μ_{ij}	σ_{ij}	γ_{ij}	b_{ij}	α_{ij}	d_{ij}	θ_{ij}	r_{ij}
11	0	1	0	0	0	0	$d_{11} = -0{,}125m$	q_{11}	0
12	11	0	0	0	0	0	$d_{12} = l_{A_{11}A_{12}} = 0{,}205m$	q_{12}	0
13	12	0	0	0	0	0	$d_{13} = l_{A_{12}A_{13}} = 0{,}205m$	q_{13}	0
21	0	1	0	0	0	0	$d_{21} = 0{,}125m$	q_{21}	0
22	21	0	0	0	0	0	$d_{22} = l_{A_{21}A_{22}} = 0{,}205m$	q_{22}	0
23	22	0	2	0	0	0	$d_{23} = l_{A_{22}A_{13}} = 0{,}205m$	0	0

Cuadro 2.1: Parámetros MDH para el primer prototipo.

ij	$a(ij)$	μ_{ij}	σ_{ij}	γ_{ij}	b_{ij}	α_{ij}	d_{ij}	θ_{ij}	r_{ij}
11	0	1	0	0	0	0	$d_{11} = 0m$	q_{11}	0
12	11	0	0	0	0	0	$d_{12} = l_{A_{11}A_{12}} = 0{,}25m$	q_{12}	0
13	12	0	0	0	0	0	$d_{13} = l_{A_{12}A_{13}} = 0{,}38m$	q_{13}	0
21	0	1	0	0	0	0	$d_{21} = 0m$	q_{21}	0
22	21	0	0	0	0	0	$d_{22} = l_{A_{21}A_{22}} = 0{,}25m$	q_{22}	0
23	22	0	2	0	0	0	$d_{23} = l_{A_{22}A_{13}} = 0{,}38m$	0	0

Cuadro 2.2: Parámetros MDH para el segundo prototipo.

En las tablas 2.1 y 2.2 se puede destacar que en su descripción mediante parámetros MDH ambos prototipos difieren únicamente en la columna de d_{ij}. Una primera observación es que en el segundo prototipo d_{11} y d_{21} son nulos. Esto se debe a que las primeras articulaciones, es decir, las articulaciones actuadas, son co-axiales y por lo tanto la distancia entre sus ejes es cero. El resto de los valores son diferentes dado que la longitud de los eslabones son distintas: en el primer prototipo todos los eslabones tienen igual longitud de $0{,}205m$, mientras que en el segundo prototipo ambas cadenas son iguales pero el primer eslabón de ellas mide $0{,}25m$ y el segundo $0{,}38m$.

2.2.1. Modelo geométrico inverso del mecanismo de 5 barras

En general, el modelo geométrico inverso de un sistema permite la determinación del vector de coordenadas articulares $\bar{q}$ dado el vector de coordenadas $\bar{x_n}$ del eslabón final $\mathscr{B}_n$ respecto al sistema de referencia $\mathscr{F}_0$.

Para el caso particular de este robot, el modelo geométrico inverso permite la determinación del vector de coordenadas articulares $q^T = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{21} & q_{22} \end{bmatrix}$ dada una posición del efector final respecto al sistema de referencia F_0 fijo en la base $\mathbf{x}^T = \begin{bmatrix} x & y \end{bmatrix}$.

$$q = f(\mathbf{x})$$

El método más utilizado para resolver este problema consiste en abrir virtualmente la cadena cinemática en el efector final, Khalil y Briot [2]. Esto permite calcular el vector $\mathbf{x}$ desde la base hacia el efector final como si fuera una cadena cinemática abierta. Obteniendo así relaciones implícitas entre las coordenadas q de la plataforma y $\mathbf{x}$. Para el caso en estudio, el método resulta en la siguiente ecuación matricial:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} d_{i1} + d_{i3}\cos(q_{i1} + q_{i2}) + d_{i2}\cos(q_{i1}) \\ d_{i3}\sin(q_{i1} + q_{i2}) + d_{i2}\sin(q_{i1}) \end{bmatrix} \tag{2.1}$$

Para $i = 1, 2$, cada una de las cadenas cinemáticas obtenidas al abrir virtualmente el manipulador.

Se puede reducir la complejidad de las ecuaciones obtenidas eliminando las articulaciones pasivas $q_d^T = \begin{bmatrix} q_{12} & q_{13} & q_{22} \end{bmatrix}$, de esta forma las coordenadas activas del manipulador $q_a^T = \begin{bmatrix} q_{11} & q_{21} \end{bmatrix}$ quedan directamente vinculadas con las de la plataforma móvil $\mathbf{x}$ usando una relación de la forma: $h_p(x, q_a) = 0$.

Trabajando algebraicamente con la ecuación 2.1, separando las funciones trigonométricas, elevando al cuadrado, igualando a cero y simplificando:

$$h_p(\mathbf{x}, q_a) = \begin{bmatrix} (x - d_{11} - d_{12}\cos(q_{11}))^2 + (y - d_{12}\sin(q_{11}))^2 - d_{13}^2 \\ (x - d_{21} - d_{22}\cos(q_{21}))^2 + (y - d_{22}\sin(q_{21}))^2 - d_{23}^2 \end{bmatrix} = 0 \tag{2.2}$$

Como puede observarse en la Fig. 2.3, desde un punto de vista geométrico, resolver el sistema de ecuaciones 2.2 es equivalente a encontrar la intersección entre dos circunferencias:

- Circulo C_{i1} centrado en A_{i1} de radio d_{i2}
- Circulo C_{i2} centrado en A_{i3} (considerado como fijo) de radio d_{i3}

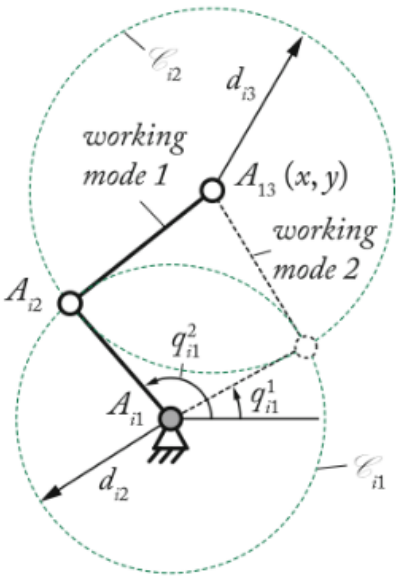


Figura 2.3: Dos modos de trabajo de la pierna i. Extraído de Khalil y Briot [2].

Resolviendo el sistema de ecuaciones 2.2 se obtiene finalmente el **modelo geométrico inverso**, con el cual se puede determinar el valor de las articulaciones activas:

$$\boxed{q_{i1} = 2\arctan\frac{-b_i \pm \sqrt{b_i^2 - c_i^2 + a_i^2}}{c_i - a_i}} \tag{2.3}$$

Donde: $a_i = -2d_{i2}(x - d_{i1})$, $b_i = -2d_{i2}y$, $c_i = (x - d_{i1})^2 + y_2 + d_{i2}^2 - d_{i3}^2$.

Una vez obtenidos los valores de las articulaciones activas, se pueden obtener las articulaciones pasivas mediante la ecuación 2.1, lo que resulta en la siguiente expresión:

$$q_{i2} = \text{atan2}\,\frac{y - d_{i2}\sin(q_{i1})}{x - d_{i1} - d_{i2}\cos(q_{i1})} - q_{i1} \tag{2.4}$$

Para $i = 1, 2$.

Luego, el valor de q_{13} puede ser obtenido directamente al introducir (2.3) y (2.4) dentro de las ecuaciones (2.2)

$$q_{13} = q_{21} + q_{22} - q_{11} - q_{12} \tag{2.5}$$

En la Fig. 2.3 pueden observarse gráficamente las dos soluciones que provee el modelo para cada una de las cadenas serie, estas soluciones se denominarán **modos de trabajo** del robot. Esto es análogo al clásico caso 'codo arriba y codo abajo' en el problema inverso de robots tipo serie.

Finalmente, en la Fig. 2.4 se observan los 4 modos de trabajo del robot, soluciones del modelo geométrico inverso, para una configuración similar a la planteada en el primer prototipo del trabajo. Se puede observar cómo la solución 1 debe ser descartada por presentar una colisión.

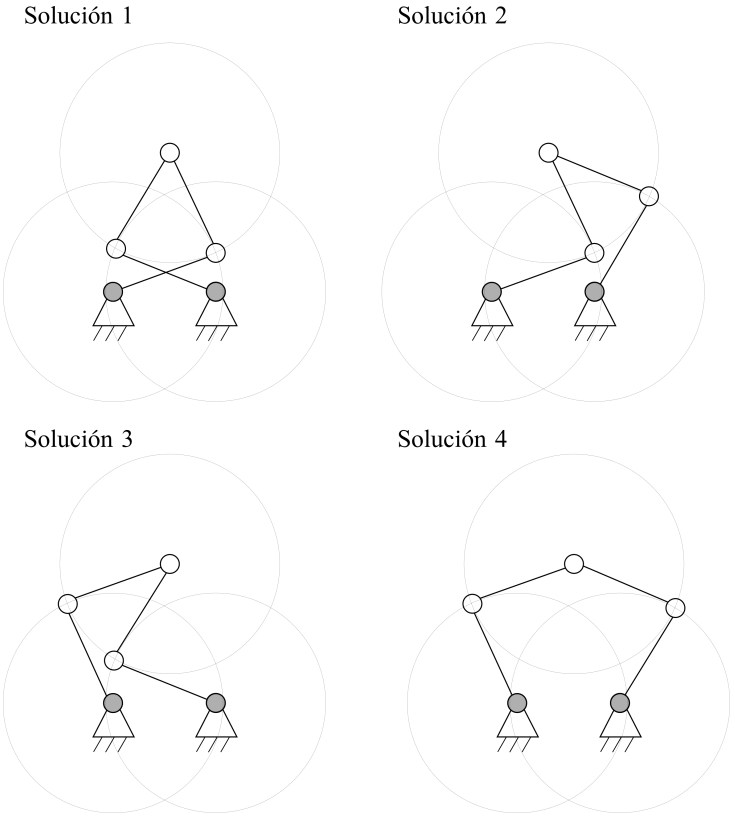


Figura 2.4: Modos de trabajo del robot.

2.2.2. Modelo geométrico directo del mecanismo de 5 barras

El modelo geométrico directo de un sistema permite la determinación del vector $\mathbf{x}$ del efector final como una función de las articulaciones activas q_a.

$$\mathbf{x} = f^{-1}(q_a)$$

Para el caso en estudio, partiendo de la ecuación 2.2, se puede operar algebraicamente para obtener la siguiente expresión:

$$h_p(\bar{x}, \bar{q_a}) = \begin{bmatrix} x^2 + y^2 + a_1 x + b_1 y + c_1 \\ x^2 + y^2 + a_2 x + b_2 y + c_1 \end{bmatrix} = 0 \tag{2.6}$$

Donde $a_i = -2(d_{i1} + d_{i2}\cos(q_{i1}))$, $b_i = -2d_{i2}\sin(q_{i1})$ y $c_i = (d_{i1} + d_{i2}\cos(q_{i1}))^2 + d_{i2}^2\sin^2(q_{i1}) - d_{i3}^2$.
Nuevamente puede destacarse que dichas ecuaciones corresponden a la intersección de dos círculos, como se puede observar en la Fig. 2.5, estos son:

- El círculo C_1 centrado en A_{12} de radio d_{13}.
- El círculo C_2 centrado en A_{22} de radio d_{23}.

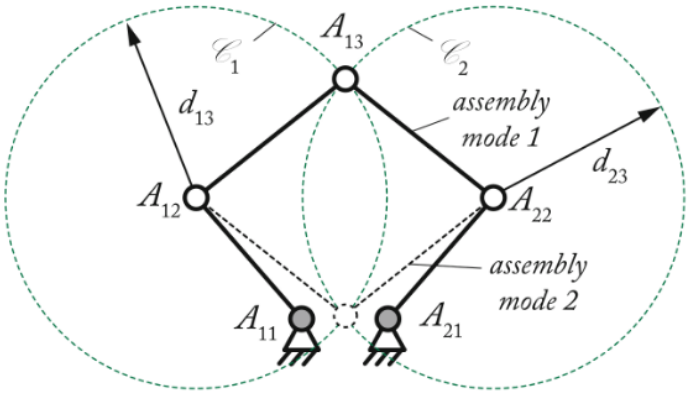


Figura 2.5: Dos modos de ensamblaje del robot. Extraído de Khalil y Briot [2].

Así, las dos soluciones de la cinemática directa (modos de ensamble) corresponden a los puntos de intersección entre los círculos C_1 y C_2.
Resolviendo la ecuación 2.6 se obtiene el **modelo geométrico directo**. Siendo $b_i = -2d_{i2}\sin(q_{i1})$, para el caso en que $b_1 \neq b_2$:

$$\boxed{x = -\frac{-f_2 \pm \sqrt{f_2^2 - 4f_1 f_3}}{2f_1} \quad ; \quad y = e_1 x + e_2} \tag{2.7}$$

Donde:

$$\begin{aligned} f_1 &= 1 + e_1^2 \\ f_2 &= 2e_1 e_2 + a_1 + b_1 e_1 \\ e_1 &= -d_1/d_2 \\ e_2 &= -d_3/d_2 \\ d_1 &= a_2 - a_1 \\ d_2 &= b_2 - b_1 \\ d_3 &= c_2 - c_1 \end{aligned}$$

Si, por el contrario, se trata de $b_1 = b_2$, será de la forma:

$$\boxed{x = -\frac{c_2 - c_1}{a_2 - a_1} \quad ; \quad y = \frac{-b_1 \pm \sqrt{b_1^2 - 4(x^2 + a_1 x + c_1)}}{2}} \tag{2.8}$$

Una vez obtenidos los valores de x e y, el valor de las articulaciones pasivas puede obtenerse mediante las ecuación 2.4 y 2.5.
Así como en el modelo geométrico inverso se pueden clasificar las soluciones en modos de trabajos, las soluciones del modelo geométrico directo se pueden clasificar en **modos de ensamble**. Para cada

modo de ensamble, las coordenadas articulares q_a se mantienen constante y se obtienen dos posiciones diferentes del efector final xy.

En la Fig. 2.6 se exponen los dos posibles resultados del modelo geométrico directo del robot para un q_a ejemplo. Similar a lo sucedido en la sección anterior, la solución 2 debería ser descartada si la estructura mecánica presentara colisiones.

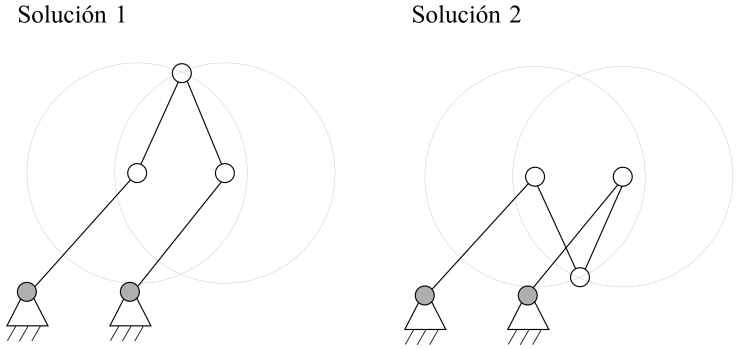


Figura 2.6: Modos de ensambles.

2.3. Análisis cinemático del sistema

El modelo cinemático permite establecer la relación entre las velocidades de las articulaciones activas del mecanismo de cinco barras y las velocidades del extremo operativo. También se establece y determina la velocidad de las articulaciones pasivas o subactuadas.

Estas relaciones podrían obtenerse derivando respecto al tiempo los modelos geométricos ya obtenidos, sin embargo, no es computacionalmente eficiente. En su lugar, Khalil y Briot [2] propone un planteo vectorial que facilita la utilización de algoritmos recursivos. A continuación, se aplicará dicha propuesta.

2.3.1. Modelo cinemático de primer orden

El modelo cinemático de primer orden expresa las relaciones de velocidades entre las coordenadas independientes del efector final, en este caso ${}^0\dot{t}_r^T = \begin{bmatrix}\dot{x} & \dot{y}\end{bmatrix}$, y las coordenadas articulares, que en este caso son $\dot{q_a}^T = \begin{bmatrix}\dot{q_{11}} & \dot{q_{21}}\end{bmatrix}$. El desarrollo en detalle del modelo cinemático de primer orden es extenso y tedioso, por eso, solo se muestran las matrices y ecuaciones principales que servirán para el desarrollo de la dinámica del robot. Para un análisis detallado del modelo de primer orden, por favor, refiérase al apéndice A (Modelo cinemático de primer orden) y apéndice B (Cálculo de velocidad de articulaciones pasivas).

El modelo cinemático de primer orden directo es:

$$ {}^0\dot{t}_r = -A_r^{-1} B \dot{q_a} = J \dot{q_a} \tag{2.9} $$

Donde:

$$ B = -\begin{bmatrix} d_{12}\sin(q_{12}) & 0 \\ 0 & d_{22}\sin(q_{22}) \end{bmatrix} \tag{2.10} $$

$$ A_r = \begin{bmatrix} \cos(q_{11}+q_{12}) & \sin(q_{11}+q_{12}) \\ \cos(q_{21}+q_{22}) & \sin(q_{21}+q_{22}) \end{bmatrix} \tag{2.11} $$

En la que $J = -A_r^{-1}B$ es la matriz Jacobiana cinemática del robot. La expresión 2.9 es válida en cuanto A_r no sea singular.

Y el modelo cinemático de primer orden inverso está dado por:

$$ \dot{q_a} = -B^{-1} A_r \, {}^0\dot{t}_r = J_{inv} \, {}^0\dot{t}_r \tag{2.12} $$

Además, $J_{inv} = -B^{-1}A_r$ es la inversa de la matriz Jacobiana cinemática del robot. En este caso, un robot sin redundancia, $J_{inv} = J^{-1}$. La expresión 2.12 es válida en cuanto B no sea singular.

Es importante notar que para el caso en estudio la matriz Jacobiana cinemática y la matriz Jacobiana son las mismas debido a que nuestro robot solo posee dos grados de libertad, es decir, las coordenadas

independientes son $\mathbf{x} = \begin{bmatrix} x & y \end{bmatrix}^T$. Esto no es para todos los robot paralelos igual ya que la matriz Jacobiana relaciona las velocidades $\dot{q_a}$ con las derivadas respecto del tiempo de las coordenadas del efector final (denotadas como $\dot{x}$) y la matriz Jacobiana cinemática lo hace respecto al *twist* del efector final ${}^0t_r$. Para ver la relación de estas matrices, diríjase al capitulo 7.3.2.5 del libro, Khalil y Briot [2].

Una vez encontrada la relación entre las velocidades de las coordenadas independientes del efector final ${}^0t_r$ y las velocidades de las articulaciones activas $\dot{q_a}$, se pueden calcular las velocidades de las articulaciones pasivas $\dot{q_d}^T = \begin{bmatrix} \dot{q_{12}} & \dot{q_{13}} & \dot{q_{22}} \end{bmatrix}$:

$$\dot{q_d} = J_{td}^{-1}(J_t {}^0t_r - J_{ta}\dot{q_a}) \tag{2.13}$$

Donde:

$$J_t = \begin{bmatrix} -\sin(q_{11}+q_{12}) & \cos(q_{11}+q_{12}) \\ (a_x + a_q j_{inv}^{21} & (a_y + a_q j_{inv}^{22} d_{22}\sin q_{22} \\ -\sin(q_{21}+q_{22}) & \cos(q_{21}+q_{22}) \end{bmatrix} \tag{2.14}$$

$$J_{ta} = \begin{bmatrix} d_{21}\cos(q_{21}) + d_{13} & 0 \\ 1 & 0 \\ 0 & d_{22}\cos(q_{22}) + d_{23} \end{bmatrix} \tag{2.15}$$

$$J_{td} = \begin{bmatrix} d_{13} & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & d_{23} \end{bmatrix} \tag{2.16}$$

Siendo:

$$\begin{aligned} a_x &= \tfrac{\cos^2(\phi)}{(x - d_{22}\cos(q_{21} - d_{21})^2)} \\ a_y &= \tfrac{\cos^2(\phi)}{x - d_{22}\cos(q_{21}) - d_{21}} \\ a_q &= -d_{22}\tfrac{(x-d_{21})\cos(q_{21}) + y\sin(q_{21}) - d_{22}}{(x - d_{22}\cos(q_{21}) - d_{21})^2}\cos^2(\phi) \end{aligned}$$

Para simplificar los términos dentro de las matrices anteriores, los símbolos $j_{inv}^{21} = J_{inv}[2,1]$ y $j_{inv}^{22} = J_{inv}[2,2]$ corresponden a el primer y segundo valor de la segunda fila de la matriz J_{inv} respectivamente.

Es necesario mencionar que las matrices J, J_t, J_{ta} y J_{td} son cruciales para el modelo dinámico del robot.

2.3.2. Modelo cinemático de segundo orden

El modelo cinemático de segundo orden expresa las relaciones de aceleraciones entre las coordenadas independientes del efector final, en este caso ${}^0\dot{t}_r^T = \begin{bmatrix} \ddot{x} & \ddot{y} \end{bmatrix}$, y las coordenadas articulares, en este caso $\ddot{q_a}^T = \begin{bmatrix} \ddot{q_{11}} & \ddot{q_{21}} \end{bmatrix}$. Al igual que en el modelo cinemático de primer orden, solo se mostrarán las ecuaciones y matrices importantes. Para un extensivo desarrollo del modelo, por favor, diríjase a el capitulo 7 de Khalil y Briot [2].

El modelo cinemático directo de segundo orden es:

$${}^0\dot{t}_r = A_r^{-1}({}^0\bar{b}_p - B\ddot{q_a}) \tag{2.17}$$

Y el modelo cinemático inverso de segundo orden es:

$$\ddot{q_a} = B^{-1}({}^0\bar{b}_p - A_r^0\dot{t}_r) \tag{2.18}$$

Donde las matrices A_r y B están definidas en las ecuaciones 2.11 y 2.10 respectivamente, y

$${}^0\bar{b}_p = \begin{bmatrix} \zeta_1^T {}^0\bar{b}_{p1} \\ \zeta_2^T {}^0\bar{b}_{p2} \end{bmatrix}$$

Siendo ζ_1 y ζ_2 *wrenches* recíprocos a los *twists* de las articulaciones pasivas pero no a los de las articulaciones activas[1] y

[1] Un *wrench* ζ es recíproco a un *twits* $\mathbb{S}$ si su producto es nulo, por ejemplo, $\mathbb{S}^T\zeta = \zeta^T\mathbb{S} = 0$. Esto significa que el poder desarrollado por el *wrench* a lo largo del movimiento definido por el *twist* es nulo (ver apéndice C del libro Khalil y Briot [2]).

$$
{}^0\bar{\dot{b}}_{pi} = \begin{bmatrix} -d_{i2}\dot{q_{i1}}\cos(q_{i1}) - d_{i3}\cos(q_{i1}+q_{i2})(\dot{q_{i1}}+\dot{q_{i2}}) \\ -d_{i2}\dot{q_{i1}}\sin(q_{i1}) - d_{i3}\sin(q_{i1}+q_{i2})(\dot{q_{i1}}+\dot{q_{i2}}) \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \dot{q_{i1}} + \begin{bmatrix} -d_{i3}\cos(q_{i1}+q_{i2}) \\ -d_{i3}\sin(q_{i1}+q_{i2}) \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \dot{q_{i2}}(\dot{q_{i1}}+\dot{q_{i2}})
$$

Para $i = 1, 2$

Ahora se pueden obtener las aceleraciones de las articulaciones pasivas $\ddot{q_d}^T = \begin{bmatrix} \ddot{q_{12}} & \ddot{q_{13}} & \ddot{q_{22}} \end{bmatrix}$:

$$
\ddot{q_d} = J_{td}^{-1}(J_t\,{}^0\dot{t}_r - J_{ta}\ddot{q_a} + d_c) \tag{2.19}
$$

donde J_t, J_{ta} y J_{td} están dadas en las ecuaciones 2.14, 2.15 y 2.16 respectivamente y

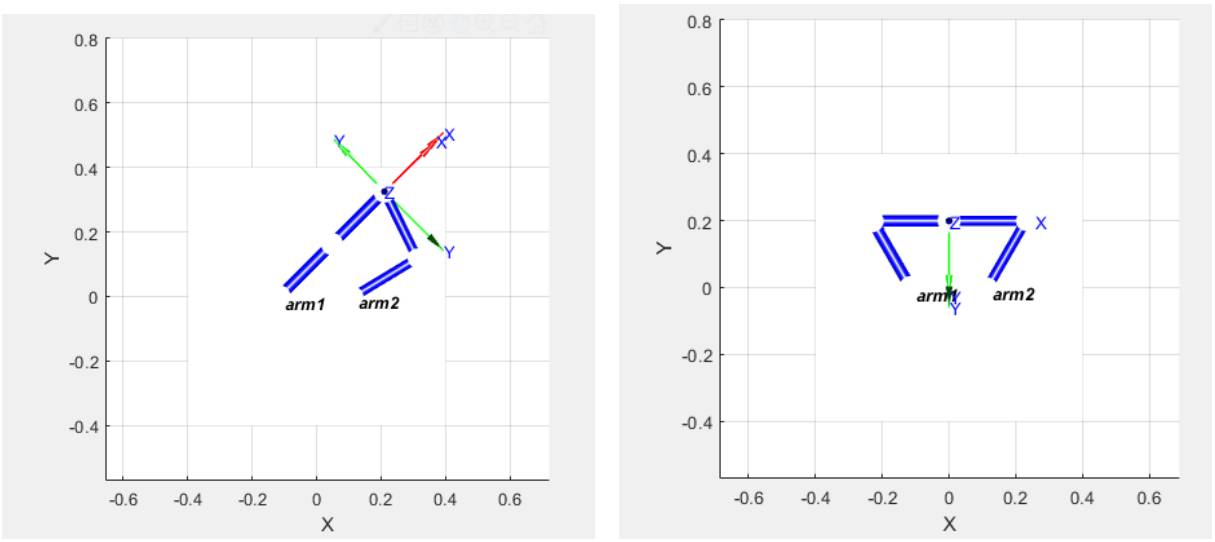
$$
d_c = \begin{bmatrix} d_1^c \\ d_2^c \end{bmatrix}
$$

con $d_i^c = \Psi_{ti}(J_{ti}\dot{\Psi}_t\,{}^0t_r + d_i - {}^0\bar{b}_{pi})$, donde $d_i = 0_{6x1}$ (debido a que la última articulación de la pierna i y el efector final coinciden), Ψ_t y Ψ_{ti} son matrices que se encuentran cuando se desarrolla el modelo de primer orden y $\dot{\Psi}_t$ es la derivada respecto del tiempo de la matriz Ψ_t.

2.3.3. Singularidades

Las singularidades de un robot paralelo se pueden dividir en 2 tipos.

- Singularidades tipo 1: en este tipo de singularidades el robot pierde la habilidad de moverse a lo largo de una o más direcciones en el espacio de trabajo. Se dan cuando la matriz B es deficiente en rango. Para el caso en estudio aparecen cuando $q_{i2} = 0 \vee \pi$. Esto sucede cuando las articulaciones están completamente extendidas o solapadas. En la Fig. 2.7a se puede observar la imposibilidad de continuar el movimiento a lo largo del eje x.
- Singularidades tipo 2: en este tipo de singularidades el robot tiene uno o más movimientos incontrolables, este se vuelve inestable. Se dan cuando la matriz A_r es deficiente en rango. Para el caso en estudio aparecen cuando $q_{11} + q_{12} - q_{21} - q_{22} = 0 \vee \pi$. Esto sucede cuando el punto A_{12}, A_{13} y A_{22} están alineados. En nuestro caso se pierde el movimiento a lo largo de la dirección perpendicular a la línea que pasa por los 3 puntos anteriores. En la Fig. 2.7b se puede observar la perdida de control a lo largo del eje y



(a) Tipo 1. (b) Tipo 2.

Figura 2.7: Singularidades tipo 1 y 2.

El espacio de trabajo de los robots paralelos típicamente se ve segmentado por las singularidades existentes. Estrategias comunes de control, como el control PID, no toman en cuenta la perdida de control sobre un grado de libertad, esto produce una discontinuidad en el torque al momento de atravesar una singularidad de tipo 2 debido a que el determinante de la matriz A_r es muy pequeño en las vecindades de las singularidades. Además, en ese tipo de controles no se tiene en cuenta el posible rebote del efector final al no poder controlar el robot en la dirección en la cual queremos dirigirnos. Para atravesar estas singularidades se requieren técnicas de control más avanzadas como las propuestas en el capítulo 9 de Khalil y Briot [2] y en el artículo Koessler y col. [16]. Debido a la complejidad que implica atravesar singularidades del tipo 2, en el robot solo se utilizará un modo de ensamble, dado en la solución 1 de la Fig. 2.5. Esto implica que solo habrá una solución de la cinemática directa y simplificará tanto el diseño mecánico del robot como su control y uso.

2.3.4. Espacio de trabajo del primer prototipo

En la Fig. 2.8 se puede ver representada una vista del plano xy del espacio de trabajo del primer prototipo, que puede observarse en la Fig. 2.1a. Este cuenta con una distancia entre las bases de los actuadores igual a $0{,}25m$ y todos los eslabones iguales a $0{,}205m$.

El espacio de trabajo del robot está delimitado por líneas negras punteadas (singularidades de tipo 1). Las líneas continuas, rojas y azules, corresponden a singularidades de tipo 2 que se presentan según sea el modo de trabajo en el que el robot se encuentre.

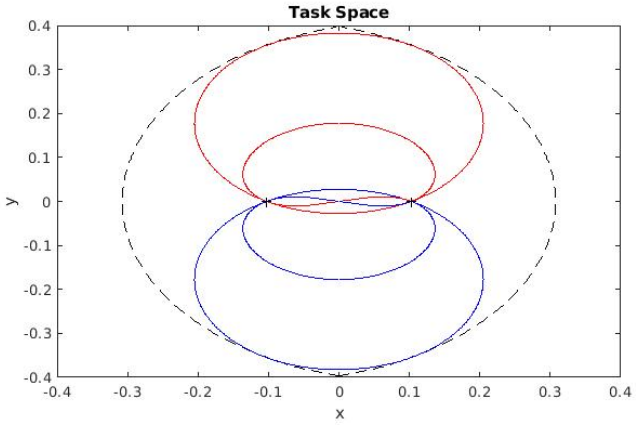


Figura 2.8: Vista superior del espacio de trabajo del primer prototipo en el plano cartesiano xy.

Como se vio anteriormente, el espacio de trabajo de este prototipo se ve segmentado por singularidades de tipo 2. Para hacer un uso completo del espacio de trabajo, es necesario evitarlas, para esto se decidió trabajar en cualquiera de los 4 modos de trabajo, sabiendo que para cambiar entre ellos es necesario atravesar singularidades de tipo 1, es decir, uno de los brazos se estirará completamente para cambiar de configuración. Esto trae como resultado un incremento en el espacio útil del robot.

En la Fig. 2.9 se puede observar en más detalle la segmentación del espacio de trabajo. Se expone la vista superior del robot para los distintos modo de trabajo y el primer modo de ensamble (solución 1 de la Fig. 2.6). Estos están delimitados por los círculos C1 y C2 y una ecuación de sexto grado S, Figielski, Bonev y Bigras [17].

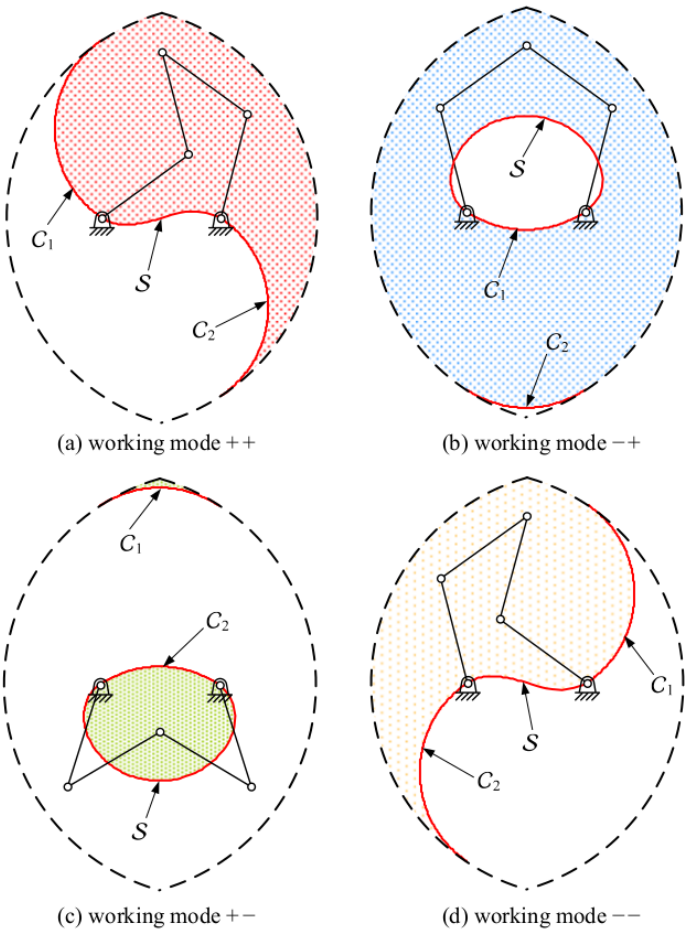


Figura 2.9: Vista superior del espacio de trabajo accesible en diferentes modos de trabajo. Primer prototipo. Extraído de Figielski, Bonev y Bigras [17].

Finalmente, para definir el espacio de trabajo tridimensional solo basta con prolongar la Fig. 2.8 a lo largo del eje z la distancia de $50mm$.

En la Fig. 2.10 se puede observar el espacio articular de nuestro robot generado a partir de únicamente el primer modo de ensamble. Las elipses blancas son valores articulares que solo se pueden acceder atravesando singularidades de tipo 2.

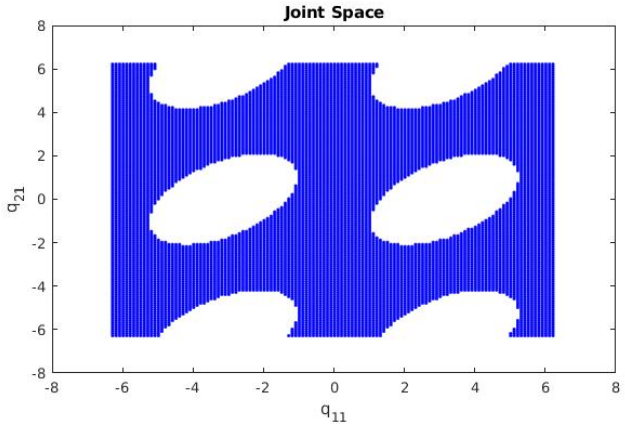


Figura 2.10: Vista superior del espacio articular del primer prototipo.

2.3.5. Espacio de trabajo del segundo prototipo

El espacio de trabajo anterior tiene como principal desventaja la presencia de singularidades de tipo 2 y la necesidad de cambiar de modo de trabajo para hacer un amplio uso del mismo. Para eliminar las singularidades de tipo 2 es necesario cambiar la geometría del robot, para esto se hace que las longitudes de los eslabones más cercanos a la base del robot sean menores que la de los más lejanos, es decir, $d_{i2} < d_{i3}$ para $i = 1, 2$. Esto trae como consecuencia una disminución del espacio de trabajo del robot. Para compensar esta disminución y eliminar la necesidad de tener que cambiar entre modos de trabajos, se decidió hacer co-axiales las articulación A_{11} y A_{12}, o lo que es lo mismo, $d = 0$.

En la Fig. 2.11 se puede ver el espacio de trabajo del segundo prototipo. En este se tuvieron en cuenta límites articulares y la interferencia mecánica, aspectos que se verán con detalle más adelante en la descripción del diseño mecánico.

Los límites del espacio de trabajo están marcados con línea negra punteada, esto son tres círculos: $C1$ con centro en A_{12} y de radio d_{13}, $C2$ con centro en A_{22} y de radio d_{23} y $C3$ con centro en el origen de coordenadas y de radio $d_{12} + d_{13}$ o $d_{22} + d_{23}$. Sobre este último círculo el robot se encuentra en una singularidad de tipo 1. Se puede observar ver que mientras menor sea la diferencia entre la longitud de los eslabones del robot, mayor será el espacio útil. El máximo teórico se da cuando los eslabones tienen la misma longitud, en ese caso, el espacio útil es un círculo centrado en el origen de radio $2l$ siendo l la longitud de los eslabones. El espacio de trabajo útil que se obtiene, luego de contemplar límites articulares e interferencia mecánica, es de aproximadamente $0{,}3396m^2$, siendo este 5.4 veces más grande que el espacio de trabajo del robot industrial RP-5AH fabricado por *Mitsubishi Electric*, cuyo espacio útil es de $0{,}06237m^2$ y sus dimensiones son $d = 85mm$ (distancia entre centros de las articulaciones activas), $d_{12} = d_{22} = 200mm$ y $d_{13} = d_{23} = 260mm$, similares a las del prototipo en desarrollo.

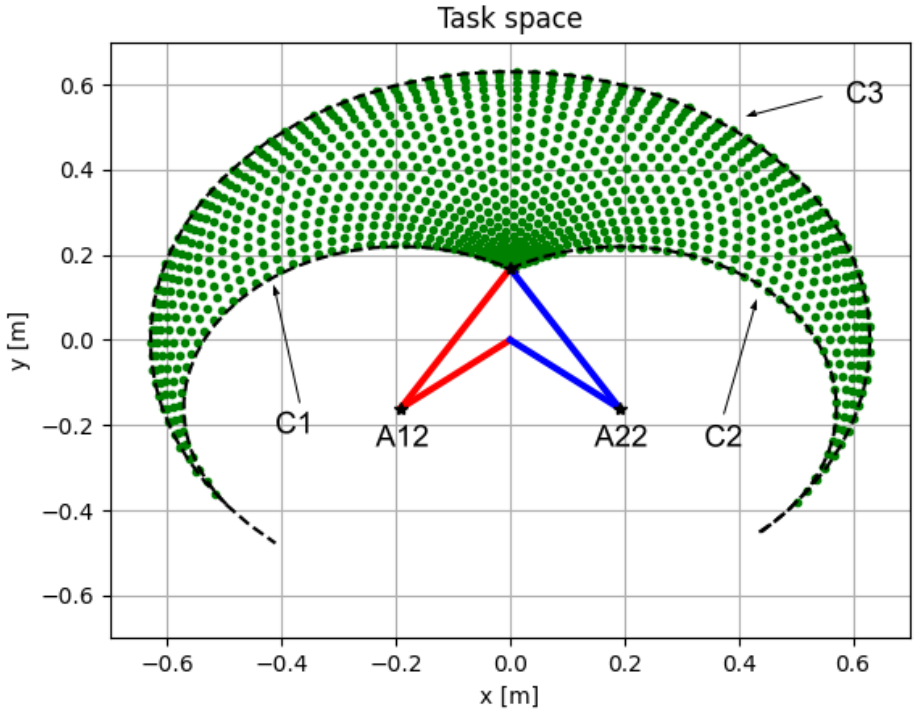


Figura 2.11: Vista superior del espacio de trabajo del segundo prototipo.

2.4. Análisis dinámico del sistema

El modelo dinámico inverso y directo de un robot pueden jugar un rol importante en el diseño mecánico y del sistema de control del robot, logrando resultados más precisos y eficientes. El modelo inverso puede ser usado, por ejemplo, para seleccionar los actuadores y determinar sus esfuerzos ante determinadas trayectorias del robot; mientras que el modelo directo es empleado para llevar acabo simulaciones que

evalúen el desempeño del mismo. Dichas simulaciones también pueden ser utilizadas para desarrollar esquemas de control avanzados y ponerlos a pruebas en un entorno virtual.

En esta sección se verán los modelos matemáticos presentados en el capítulo 8 del libro Khalil y Briot [2]. A diferencia del libro, en nuestro desarrollo se consideró la aceleración de la gravedad a lo largo del eje y. Esto, si bien no aplica al robot en desarrollo, se utilizó para poder simular el colapso de la estructura debido a la gravedad y verificar nuestros modelos. Para que las ecuaciones presentadas en este capítulo apliquen al caso del robot en estudio es necesario igualar a cero el término escalar de la gravedad g. En el apéndice C pueden observarse más detalladamente los aspectos teóricos sobre cómo obtener el modelo dinámico inverso de un robot paralelo mediante un enfoque de Lagrange.

2.4.1. Modelo dinámico inverso

El modelo dinámico inverso (MDI) provee el torque en los actuadores en términos de las posiciones, velocidades y aceleraciones de las articulaciones activas. Es descripto por:

$$T = idm(\ddot{q_a}, \dot{q_a}, q_a, w_e) \tag{2.20}$$

Donde T es el torque en los actuadores y w_e es un sistema de *wrenches*[2] aplicados por el robot en el entorno.

Para computar el MDI, el problema se descompone en dos pasos:

1. Primero: todos los bucles cerrados son virtualmente abiertos para desensamblar la plataforma virtualmente del resto de la estructura; además, cada articulación es virtualmente considerada actuada (incluso para articulaciones pasivas) así el robot se convierte en uno con estructura de árbol y un cuerpo libre. Los modelos dinámicos de estos son calculados haciendo uso del principio de Newton-Euler que hace posible obtener la siguiente ecuación:

$$\begin{aligned} T_t &= idm_t(\ddot{q_t}, \dot{q_t}, q_t, w_t) \\ w_p &= idm_p(\ddot{t_p}, \dot{t_p}, x_p, w_e) \end{aligned} \tag{2.21}$$

   Donde idm_t representa el MDI de la estructura en árbol, idm_p el MDI de la plataforma, q_t las coordenadas articulares de la estructura en árbol y t_p, x_p son el *twist* y la posición de la plataforma, w_t y w_p son sistemas de *wrenches* aplicados por la estructura y la plataforma en el ambiente respectivamente.

2. Luego, los bucles son cerrados usando las ecuaciones de cierre que relacionan t_p y $\dot{q_t}$ con $\dot{q_a}$

El MDI de la estructura en árbol, luego de aplicar el procedimiento descripto para la cadena 1, queda:

$$\begin{aligned} T_{t_{11}} = (&my_{12}sin(q_{11}+q_{12}) - mx_{12}cos(q_{11}+q_{12}) \quad ... \\ &- mx_{11}cos(q_{11}) + my_{11}sin(q_{11}) - d_{12}m_{12}cos(q_{11}))g \quad ... \\ &+ mx_{12}(-d_{12}sin(q_{12})\dot{q_{12}}^2 - 2d_{12}\dot{q_{11}}sin(q_{12})\dot{q_{12}} \quad ... \\ &+ 2d_{12}\ddot{q_{11}}cos(q_{12}) + d_{12}\ddot{q_{12}}cos(q_{12})) \quad ... \\ &- my_{12}(d_{12}cos(q_{12})\dot{q_{12}}^2 + 2d_{12}\dot{q_{11}}cos(q_{12})\dot{q_{12}} \quad ... \\ &+ 2d_{12}\ddot{q_{11}}sin(q_{12}) + d_{12}\ddot{q_{12}}sin(q_{12})) \quad ... \\ &+ \ddot{q_{11}}zz_{11} + \ddot{q_{11}}zz_{12} + \ddot{q_{12}}zz_{12} + m_{12}\ddot{q_{11}}d_{12}^2 \quad ... \\ &+ I_{a_{11}}\ddot{q_{11}} + f_{v_{11}}\dot{q_{11}} + f_{s_{11}}sign(\dot{q_{11}}) \end{aligned} \tag{2.22}$$

$$\begin{aligned} T_{t_{12}} = (&my_{12}sin(q_{11}+q_{12}) - mx_{12}cos(q_{11}+q_{12}))g \quad ... \\ &+ mx_{12}(d_{12}sin(q_{12})\dot{q_{11}}^2 + d_{12}\ddot{q_{11}}cos(q_{12})) \quad ... \\ &- my_{12}(-d_{12}cos(q_{12})\dot{q_{11}}^2 + d_{12}\ddot{q_{11}}sin(q_{12})) \quad ... \\ &+ zz_{12}(\ddot{q_{11}} + \ddot{q_{12}}) + f_{v_{12}}\dot{q_{12}} + f_{s_{12}}sign(\dot{q_{12}}) \end{aligned} \tag{2.23}$$

[2] $w = \begin{bmatrix} f^T & m^T \end{bmatrix}^T$ donde f es fuerza y m momento (ver capítulo 3.3 del libro Khalil y Briot [2])

$$T_{t_{13}} = f_{s_{13}} sign(\dot{q_{13}}) + f_{v_{13}} \dot{q_{13}} \tag{2.24}$$

Para la cadena 2:

$$\begin{aligned} T_{t_{21}} = (&my_{22} sin(q_{21} + q_{22}) - mx_{22} cos(q_{21} + q_{22}) \quad ... \\ &- mx_{21} cos(q_{21}) + my_{21} sin(q_{21}) - d_{22} m_{22} cos(q_{21}))g \quad ... \\ &+ mx_{22}(-d_{22} sin(q_{22}) \dot{q_{22}}^2 - 2d_{22} \dot{q_{21}} sin(q_{22}) \dot{q_{22}} \quad ... \\ &+ 2d_{22} \ddot{q_{21}} cos(q_{22}) + d_{22} \ddot{q_{22}} cos(q_{22})) \quad ... \\ &- my_{22}(d_{22} cos(q_{22}) \dot{q_{22}}^2 + 2d_{22} \dot{q_{21}} cos(q_{22}) \dot{q_{22}} \quad ... \\ &+ 2d_{22} \ddot{q_{21}} sin(q_{22}) + d_{22} \ddot{q_{22}} sin(q_{22})) \quad ... \\ &+ \ddot{q_{21}} zz_{21} + \ddot{q_{21}} zz_{22} + \ddot{q_{22}} zz_{22} + m_{22} \ddot{q_{21}} d_{22}^2 \quad ... \\ &+ I_{a_{21}} \ddot{q_{21}} + f_{v_{21}} \dot{q_{21}} + f_{s_{21}} sign(\dot{q_{21}}) \end{aligned} \tag{2.25}$$

$$\begin{aligned} T_{t_{22}} = (&my_{22} sin(q_{21} + q_{22}) - mx_{22} cos(q_{21} + q_{22}))g \quad ... \\ &+ zz_{22}(\ddot{q_{21}} + \ddot{q_{22}}) + f_{v_{22}} \dot{q_{22}} + f_{s_{22}} sign(\dot{q_{22}}) \quad ... \\ &+ mx_{22}(d_{22} sin(q_{22}) \dot{q_{21}}^2 + d_{22} \ddot{q_{21}} cos(q_{22})) \quad ... \\ &- my_{22}(-d_{22} cos(q_{22}) \dot{q_{21}}^2 + d_{22} \ddot{q_{21}} sin(q_{22})) \end{aligned} \tag{2.26}$$

Donde:

- d_{ij} son los parámetros MDH d dados en la tabla 2.1 y 2.2.
- q_{ij}, $\dot{q_{ij}}$ y $\ddot{q_{ij}}$ son las posiciones, velocidades y aceleraciones articulares del robot.
- zz_{ij} es el momento axial de inercia alrededor del eje $z_i j$, para el eslabón ij, expresado al origen del marco local $F_i j$.
- m_{ij} masa del eslabón ij.
- mx_{ij} y my_{ij} son las componentes x e y del vector primer momento de inercia del eslabón ij respecto al marco F_{ij}.
- $I_{a_{1j}}$ es la inercia del actuador de la articulación activa $1j$ referida al eje rápido.
- $f_{v_{ij}}$ amortiguamiento viscoso de la articulación ij referida al eje rápido.
- $f_{s_{ij}}$ fricción seca de la articulación ij referida al eje rápido.
- g es la gravedad actuando a lo largo del eje y.

Para el efector final:

$$^0 w_r = m_p \begin{bmatrix} \ddot{x} \\ \ddot{y} - g \end{bmatrix} \tag{2.27}$$

Donde m_p es la masa de la plataforma y $\ddot{x}$ e $\ddot{y}$ son las aceleraciones de la plataforma.
Finalmente, luego de aplicar las ecuaciones de cierre se tiene:

$$\boxed{T = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = T_{ta} + J^{T\,0} w_r + J_d^T T_{td}} \tag{2.28}$$

Donde:

- $^0 w_r$ está dado en 2.27.
- $T_{ta} = \begin{bmatrix} T_{t_{11}} & T_{t_{21}} \end{bmatrix}^T$ están dadas en las ecuaciones 2.22 y 2.25.
- $T_{td} = \begin{bmatrix} T_{t_{12}} & T_{t_{13}} & T_{t_{22}} \end{bmatrix}^T$ están dadas en las ecuaciones 2.23, 2.24 y 2.26.
- $J_d = J_{td}^{-1}(J_t J - J_{ta})$.

2.4.2. Modelo dinámico directo

El modelo dinámico directo (MDD) provee las aceleraciones de las articulaciones activas como función de los esfuerzos entrantes y de las posiciones y velocidades de las articulaciones activas:

$$\ddot{q_a} = ddm(\dot{q_a}, q_a, T, w_e) \tag{2.29}$$

Analizando las ecuaciones (2.22 a 2.26) anteriores, se puede mostrar que el MDI de una estructura de árbol puede ser escrita en la siguiente forma matricial:

$$T_t = M_t(q_t)\ddot{q}_t + c_t(q_t, \dot{q}_t) \tag{2.30}$$

Donde $M_t(q_t)$ es la matriz de inercia y $c_t(q_t, \dot{q}_t)$ es el vector de los esfuerzos de Coriolis, centrífugos, gravitacionales, de fricción y de fuerzas externas. Para el robot en estudio:

$$M_t(q_t) = \begin{bmatrix} M_{11} & M_{12} & 0 & 0 & 0 \\ M_{21} & zz_{12} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & M_{44} & M_{45} \\ 0 & 0 & 0 & M_{54} & zz_{22} \end{bmatrix} \tag{2.31}$$

con

$$M_{11} = zz_{11_r} + zz_{12} + 2d_{12}mx_{12}cos(q_{12}) - 2d_{12}my_{12}sin(q_{12}) \tag{2.32}$$

$$zz_{11_r} = zz_{11} + I_{a_{11}} + d_{12}^2 m_{12}$$

$$M_{12} = M_{21} = zz_{12} + d_{12}mx_{12}cos(q_{12}) - d_{12}my_{12}sin(q_{12}) \tag{2.33}$$

$$M_{44} = zz_{21_r} + zz_{22} + 2d_{21}mx_{22}cos(q_{22}) - 2d_{21}my_{22}sin(q_{22}) \tag{2.34}$$

$$zz_{21_r} = zz_{21} + I_{a_{21}} + d_{22}^2 m_{22}$$

$$M_{45} = M_{54} = zz_{22} + d_{21}mx_{22}cos(q_{22}) - d_{21}my_{22}sin(q_{22}) \tag{2.35}$$

Y $c_t(q_t, \dot{q}_t) = \begin{bmatrix} c_{t1} & c_{t2} & c_{t3} & c_{t4} & c_{t5} \end{bmatrix}^T$ con:

$$\begin{aligned} c_{t1} = &-d_{12}mx_{12}\dot{q_{12}}(2\dot{q_{11}} + \dot{q_{12}})sin(q_{12}) - d_{12}my_{12}\dot{q_{12}}(2\dot{q_{11}} + \dot{q_{12}})cos(q_{12}) \quad ... \\ &+ (my_{12}sin(q_{11} + q_{12}) - mx_{12}cos(q_{11} + q_{12}) \quad ... \\ &- mx_{11}cos(q_{11}) + my_{11}sin(q_{11}) - d_{12}m12cos(q_{11}))g \quad ... \\ &+ fs1(1)sign(\dot{q_{11}}) + fv1(1)\dot{q_{11}} \end{aligned} \tag{2.36}$$

$$\begin{aligned} c_{t2} = &\, d_{12}mx_{12}\dot{q_{11}}^2 sin(q_{12}) + d_{12}my_{12}\dot{q_{11}}^2 cos(q_{12}) \quad ... \\ &+ (my_{12}sin(q_{11} + q_{12}) - mx_{12}cos(q_{11} + q_{12}))g \quad ... \\ &+ fs1(2)sign(\dot{q_{12}}) + fv1(2)\dot{q_{12}} \end{aligned} \tag{2.37}$$

$$c_{t3} = fs1(3)sign(\dot{q_{13}}) + fv1(3)\dot{q_{13}}; \tag{2.38}$$

$$\begin{aligned} c_{t4} = &-d_{21}mx_{22}\dot{q_{22}}(2\dot{q_{21}} + \dot{q_{22}})sin(q_{22}) - d_{21}my_{22}\dot{q_{22}}(2\dot{q_{21}} + \dot{q_{22}})cos(q_{22}) \quad ... \\ &+ (my_{22}sin(q_{21} + q_{22}) - mx_{22}cos(q_{21} + q_{22}) \quad ... \\ &- mx_{21}cos(q_{21}) + my_{21}sin(q_{21}) - d_{21}m22cos(q_{21}))g \quad ... \\ &+ fs2(1)sign(\dot{q_{21}}) + fv2(1)\dot{q_{21}} \end{aligned} \tag{2.39}$$

$$c_{t5} = d_{21}mx_{22}\dot{q_{21}}^2 sin(q_{22}) + d_{21}my_{22}\dot{q_{21}}^2 cos(q_{22}) \quad ... \\ + (my_{22}sin(q_{21}+q_{22}) - mx_{22}cos(q_{21}+q_{22}))g \quad ... \\ + fs2(2)sign(\dot{q_{22}}) + fv2(2)\dot{q_{22}} \quad ... \tag{2.40}$$

Luego, se descompone el vector T_t en dos subpartes, torques activos T_{ta} y pasivos, T_{td} tal que $T_t = E_\tau \begin{bmatrix} T_{ta} & T_{td} \end{bmatrix}^T$ siendo E_τ una matriz cuadrada de dimensión 5 que se usa para ordenar el vector T_t:

$$E_\tau = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.41}$$

Note que la matriz E_τ también sirve para ordenar las matrices M_t, c_t y los vectores q_t, $\dot{q}_t$, etc., en componentes activas y pasivas.

De manera similar, extendiendo y analizando la ecuación 2.27 del efector final, se obtiene:

$${}^0w_p = {}^0M_p \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ 0_{4x1} \end{bmatrix} + {}^0c_p \tag{2.42}$$

Donde las matrices ${}^0M_p$ y ${}^0c_p$ son:

$${}^0M_p = m_p \begin{bmatrix} I_2 & 0_{2x4} \\ 0_{4x2} & 0_{4x4} \end{bmatrix} \tag{2.43}$$

y

$${}^0c_p = m_p \begin{bmatrix} 0 & -g & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{2.44}$$

Finalmente, luego de aplicar las ecuaciones de cierre, se obtiene:

$$\boxed{\ddot{q_a} = M^{-1}(q_a)(T - c(q_a, \dot{q_a}))} \tag{2.45}$$

Donde

$$\boxed{M(q_a) = \begin{bmatrix} I_2 & J_d^T \end{bmatrix} E_\tau^T M_t E_\tau \begin{bmatrix} I_2 \\ J_d \end{bmatrix} + J^T {}^0M_r J} \tag{2.46}$$

y

$$\boxed{c(q_a, \dot{q_a}) = \begin{bmatrix} I_2 & J_d^T \end{bmatrix} E_\tau^T (c_t + M_t E_\tau \begin{bmatrix} 0_{2x1} \\ a_d \end{bmatrix}) + J^T \Psi_t^T ({}^0c_r + {}^0c_p)} \tag{2.47}$$

con:

- ${}^0M_r = m_p I_2$
- ${}^0c_r = {}^0M_p(\Psi_t a_t + \dot{\Psi_t}\,{}^0t_r)$
- $a_d = J_{td}^{-1}(d_c + J_t(A_r^{-1}{}^0b_p))$

2.4.3. Limitaciones del modelo

En el análisis previo solo se modelaron los principales efectos del robot, no contemplando varios efectos como:

- Colisiones entre eslabones.
- Dinámica completa de los actuadores.
- Drivers.
- Reductores.
- Rigidez de los eslabones.

Para el análisis de la fricción y alguno de los fenómenos antes mencionados se recomienda la siguiente bibliografía: Khalil y Briot [2] y Sanchez [18].

2.5. Simulación

Una de las aplicaciones de los modelos es la generación de simulaciones. Mediante éstas, y alimentándolas con diferentes entradas, se puede realizar una validación de los modelos desarrollados y, además, familiarizarse aún más con el comportamiento del sistema al poder observar cómo responde a ciertos estímulos para los que se esperan determinados movimientos. A continuación, una descripción de cómo se llevaron acabo las simulaciones dinámica y cinemática.

2.5.1. Simulación dinámica

Para validar el modelo y familiarizarse con el comportamiento del sistema, acompañando las decisiones tomadas en el diseño mecánico del primer prototipo, se crearon dos simulaciones utilizando *Matlab/Simulink*: la primera utilizando *Simscape* (herramienta de caja negra) y la segunda utilizando *Simulink*.

El modelo en *Simscape*, que se observa en la Fig. 2.12, consta de 5 cuerpos unidos por 5 articulaciones, 2 actuadas y 3 pasivas. Este se utilizó para contrastar con el modelo simulado en *Simulink*.

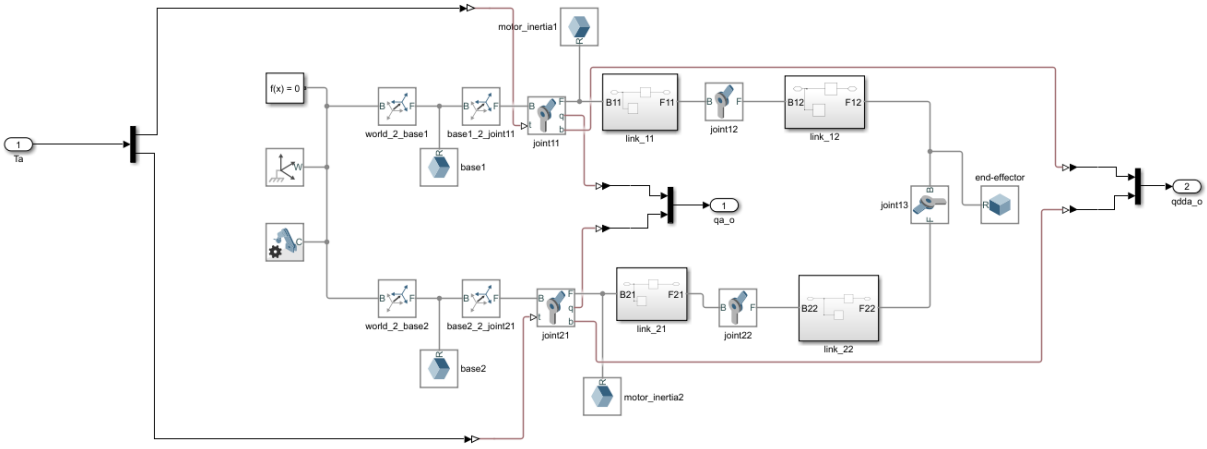


Figura 2.12: Modelo en *Simscape*.

Para modelar el robot en *Simulink* se implementaron funciones de *Matlab* de los siguientes modelos descriptos anteriormente: modelo geométrico directo e inverso, modelo cinemático de primer y segundo orden, cálculo de velocidades y aceleraciones pasivas y dinámica directa e inversa. Luego, en el entorno *Simulink* se los unió con bloques integradores de señales para poder simularlos.

En la Fig. 2.13 puede observarse el modelo implementado en *Simulink*, en esta se pueden ver bloques utilizados para calcular la dinámica directa, la cinemática directa, las velocidades de las articulaciones pasivas y el modelo inverso de segundo orden.

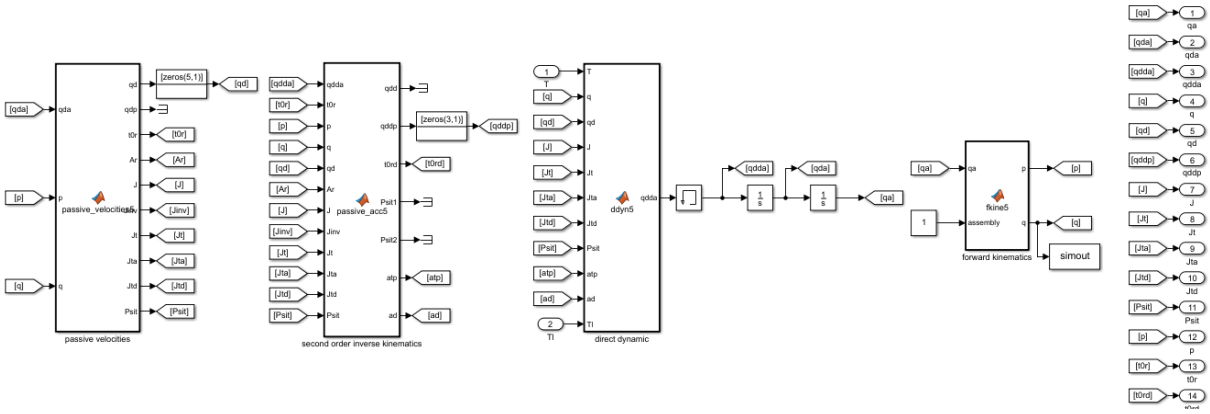


Figura 2.13: Modelo en *Simulink*.

En la Fig. 2.14 se puede observar el primer prototipo en el entorno gráfico 3D de la simulación con *Simscape*.

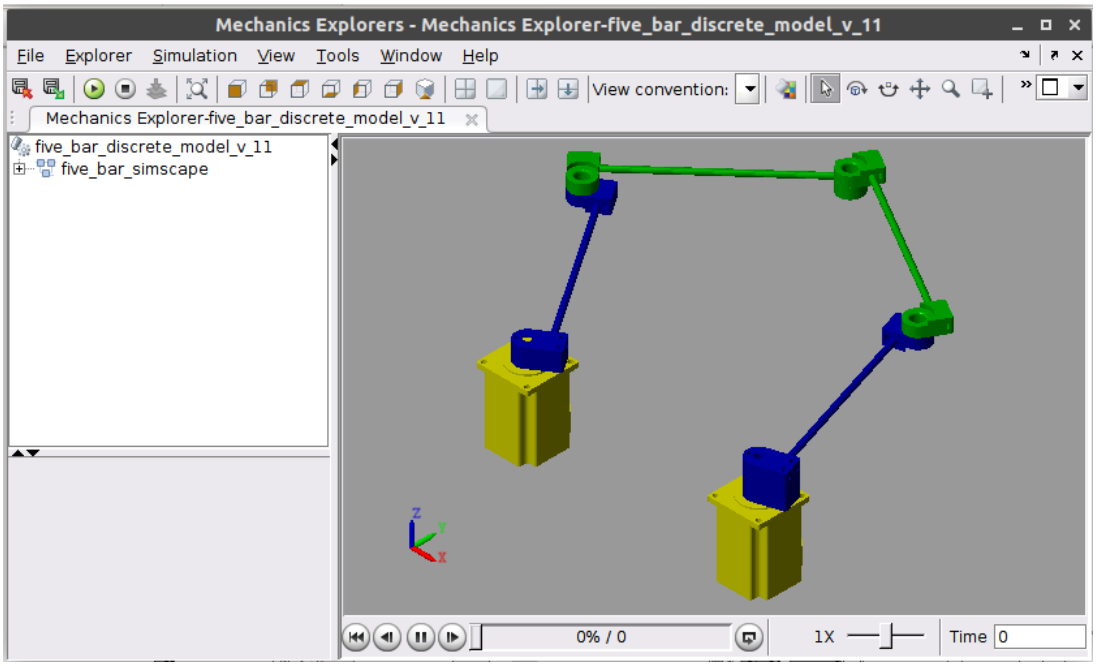


Figura 2.14: Primer prototipo modelado en *Simscape*.

Teniendo en cuenta que en el modelo implementado con *Simulink* se incorporan los modelos descriptos en las secciones previas de forma explícita, y que el modelo en *Simscape* es de tipo "caja negra", la validación de los modelos obtenidos se realiza mediante la comparación de los resultados obtenidos en *Simulink* contra los resultados obtenidos en *Simscape*, para mismas consignas de entrada y estados iniciales.

Al comparar los modelos entre sí se obtuvieron diferencias de posición que fueron ignoradas ya que, en general, eran mínimas y es probable que sean originadas por errores numéricos arrastrados en los algoritmos o debido a diferencias de efectos modelados. Se deja como trabajo futuro indagar más en las posibles fuentes de errores.

Estas simulaciones se utilizaron para probar 2 algoritmos de control. Un controlador proporcional, integral y derivativo (PID) descentralizado para cada grado de libertad; y otro llamado control dinámico inverso o control de torque. Este último consiste en utilizar la dinámica inversa del robot para linealizar y desacoplar las ecuaciones de movimiento del robot, Demichelis [19].

No se continuó con esta línea de desarrollo debido a la falta de presupuesto para conseguir motores a los cuales se les pudiera aplicar un control de torque, pero quedaron asentadas todas las bases para seguir desarrollando esta línea cuando el proyecto pase a una etapa posterior.

2.5.2. Simulación cinemática

Para el segundo prototipo, y debido a que este posee motores paso a paso como actuadores, se decidió generar una simulación meramente cinemática. Para esto se decidió hacer uso del simulador *CoppeliaSim* en su versión educativa. Algunas de sus principales características son:

- *Cross-Plattform*: se puede usar en *Windows*, *Linux* y *MacOs*.
- Dinámica: tiene 4 motores de física que permiten realizar cálculos dinámicos en los modelos.
- Cinemática: soporta cálculos de cinemática para cualquier tipo de mecanismo.
- *Remote API*: se pude controlar la simulación desde *Python*, *Lua*, *Java* y *C/C++*.

La simulación se usó para validar el modelo cinemático y la planificación de la trayectorias.

Se muestra el modelo generado en *CoppeliaSim* en la Fig. 2.15. A la izquierda de la figura se puede ver el árbol de objetos y a la derecha el robot.

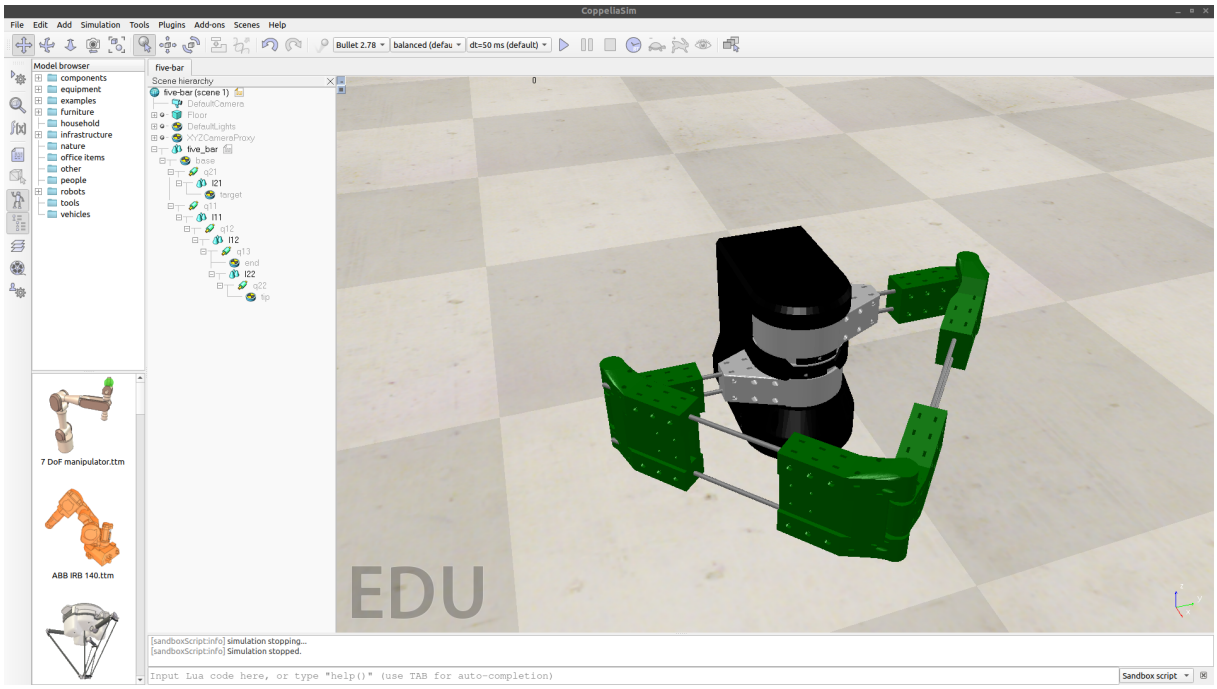


Figura 2.15: Modelo en *CoppeliaSim*.

Para probar el funcionamiento del planificador de trayectorias se decidió generar la trayectoria de la Fig. 2.16. Esta contempla líneas rectas en el plano, arcos de circunferencia y círculos. Además, fue generada usando una ley de tiempo trapezoidal y sincronizando el movimiento de las articulaciones. Para el envío de las consignas se utilizó *ZeroMQ remote API*[3] para *Python*. Esta API nos permite controlar la simulación, obtener y enviar datos al simulador.

[3] `https://zeromq.org/`

En la Fig. 2.17a se pueden observar las posiciones (x, y) deseadas y las obtenidas en la simulación. Mientras que en la Fig. 2.17b se pueden ver las posiciones articulares deseadas y las obtenidas. Analizando la gráfica en el espacio cartesiano, se obtuvo que el valor medio de la diferencia entre la posición deseada y la obtenida fue de $0{,}00045m$. Por otra parte, analizando los errores medios de las coordenadas articulares q_1 y q_2 ($-4{,}805e-10m$ y $-2{,}029e-10m$ respectivamente), podemos decir que estos son despreciables. Por lo tanto, se puede considerar válido nuestro modelo de generación de trayectorias. Posteriormente se probó y validó la generación trayectorias en el prototipo construido, se hablará de esto en el capítulo 5.

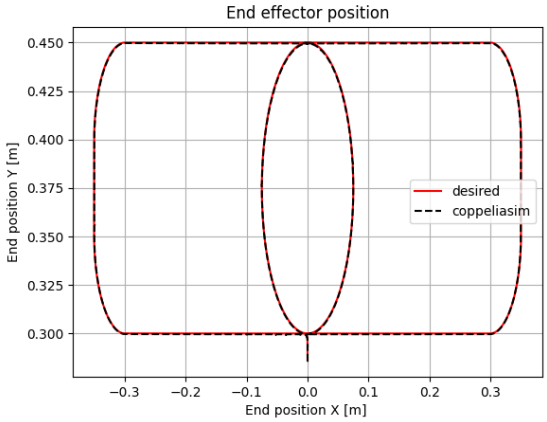
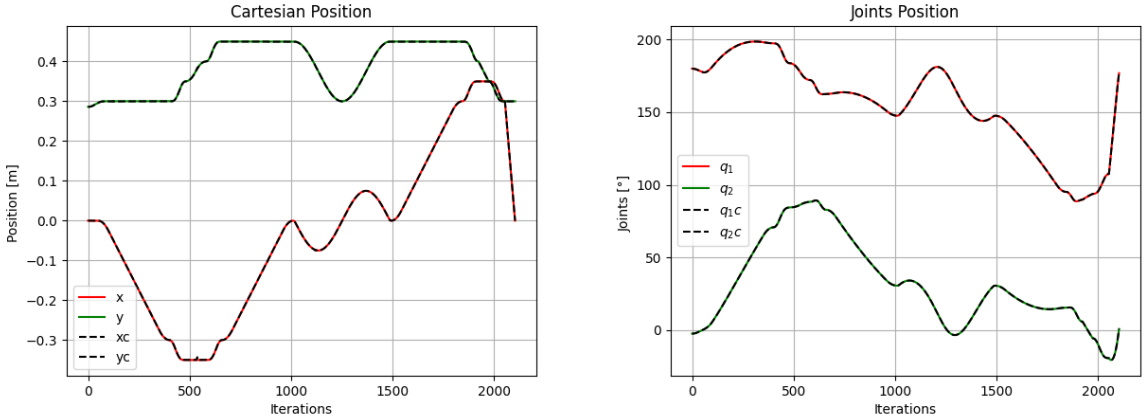


Figura 2.16: Trayectoria generada en el plano xy.



(a) Espacio de tarea.

(b) Espacio articular.

Figura 2.17: Trayectoria simulada con *CoppeliaSim*.

Por último, en la Fig. 2.18 se puede ver el error de posición a lo largo de la trayectoria. Este alcanza un valor máximo de $0{,}007m$ y uno mínimo de $0{,}0003m$

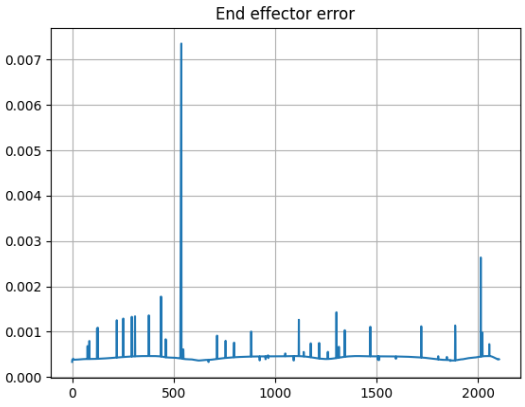


Figura 2.18: Error de posición.

Capítulo 3

Diseño mecánico y fabricación

Desarrollados los modelos matemáticos del robot deseado y conociendo sus características, se prosiguió con el diseño mecánico y fabricación del mismo. En el presente capítulo se realiza una breve descripción del primer prototipo y qué aspectos fueron considerados en su diseño; luego una descripción más detallada del proceso de diseño y fabricación del segundo prototipo y todos los elementos que lo conforman, y por último una descripción del mecanismo asociado al eje z o movimiento vertical del robot.

3.1. Primer prototipo

Una vez obtenidos los modelos matemáticos del robot y ejecutada su simulación, pudo discutirse en base a los resultados la estructura mecánica a plantear. En dicha discusión, lo más importante a definir fue la libertad de movimiento de las articulaciones activas y la longitud de los eslabones, lo que determina el espacio de trabajo del robot.

Contempladas las distintas configuraciones posibles, se optó por aquella que permite revolución completa de ambas articulaciones activas. El fundamento detrás de la selección fue la cantidad de características que quedan disponibles para explotar a la hora de controlar el mecanismo: se puede utilizar el robot en los diferentes modos de ensamble y de trabajo, el estudio de las singularidades presentes, el espacio de trabajo disponible y cómo utilizarlo de forma efectiva, qué aplicaciones se benefician de ello, y más. En la Fig. 3.1 se puede observar, en una vista de la versión preliminar del robot a realizar, la configuración elegida.

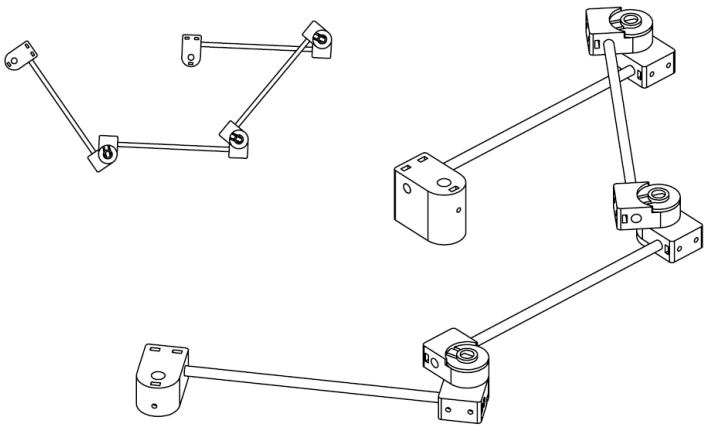


Figura 3.1: Vista de la primera versión del robot propuesta en la configuración elegida.

Definida la configuración del robot a construir, se procedió al diseño en CAD de cada componente necesario. La ejecución del proceso de diseño, que se llevó a cabo de forma remota, fue la siguiente:

1. Reunión remota para la discusión de ideas sobre la pieza o conjunto de piezas en cuestión.
2. Primer etapa de diseño en software CAD.

3. Revisión del diseño por quien posee la impresora 3D, devolución con correcciones.

4. Segunda etapa de diseño en software CAD con la obtención de una nueva versión de la pieza o piezas involucradas.

5. Iteración de los pasos 3 y 4.

6. Impresión de la pieza, análisis de resultados y corrección de errores que no fueron contemplados en el diseño CAD. Propuesta de mejoras.

7. Iteración de pasos 5 y 6 hasta obtener resultado final.

En esencia, las piezas a diseñar fueron el acople del eje motor con el eslabón correspondiente (diferentes para cada eje por variación de alturas para evitar interferencia mecánica), las articulaciones pasivas y el mecanismo que conforma el eje z del robot. Para cada una de las piezas mecánicas mencionadas fue necesaria más de una iteración para obtener el modelo final.

El acople y las articulaciones pasivas se diseñaron de tal forma que le permitan al usuario del robot una reconfiguración de la longitud de sus eslabones. El cambio de dichas longitudes impacta directamente en las características cinemáticas y dinámicas del robot, y resultó interesante que pueda estudiarse este aspecto. Además, como se mencionó previamente, al elegir una configuración que permite la revolución completa de las articulaciones activas fue necesario diseñar las piezas de tal forma que no haya interferencia mecánica. Es por esto, por ejemplo, que los acoples son diferentes en altura.

En la articulación pasiva se evaluó el uso de rodamientos para disminuir fricción, y se obtuvieron excelentes resultados con el uso de 4 rodamientos de bolas 608-2Z. Como eje de la articulación se utilizó una varilla roscada de 8mm cortada a medida, y las piezas se unen mediante arandelas y tuercas auto frenantes.

Como eslabones se utilizó una barra rectificada de 6mm de diámetro, seleccionada debido a su disponibilidad y rigidez. Todo el mecanismo se ensambla con tornillos allen M3 y M4 de distintas longitudes.

En la Fig. 3.2 se puede observar una fotografía del primer prototipo fabricado con el que se hicieron las primeras pruebas experimentales.

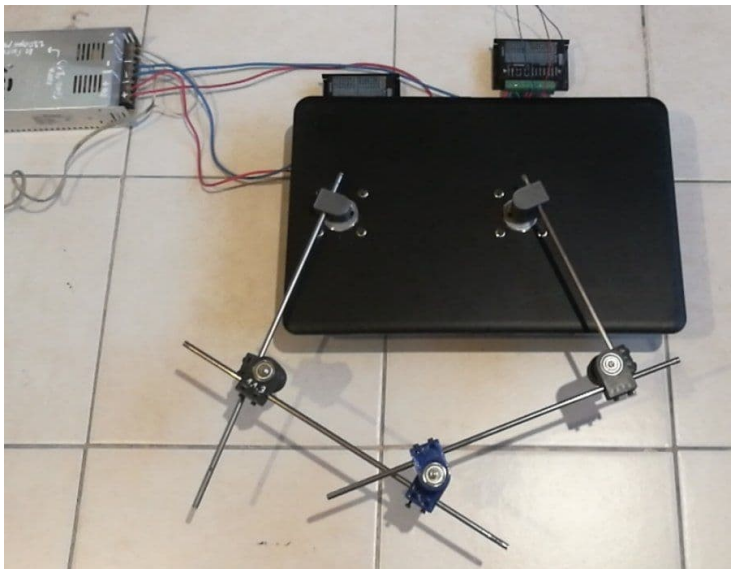


Figura 3.2: Fotografía de primer prototipo construido.

3.2. Segundo prototipo

Hechas las pruebas con el primer prototipo se continuó con un segundo diseño desde cero, buscando obtener un producto más cercano a lo que podría ser un robot comercial, siempre acotando los materiales y herramientas seleccionados para su fabricación al presupuesto disponible.

En la Fig. 3.3 se puede observar una propuesta de diseño más avanzada, siguiendo la misma geometría planteada en el primer prototipo, pero incorporando el movimiento vertical en Z.

El movimiento vertical mediante un mecanismo de tornillo en la base reduce la masa en el extremo, ya que solo poseería la herramienta a utilizar. Sin embargo, al discutir la propuesta, se optó por trasladar el mecanismo de movimiento en Z al extremo para evitar el problema de coordinar los ejes en la base (que se actúan con un actuador en cada columna).

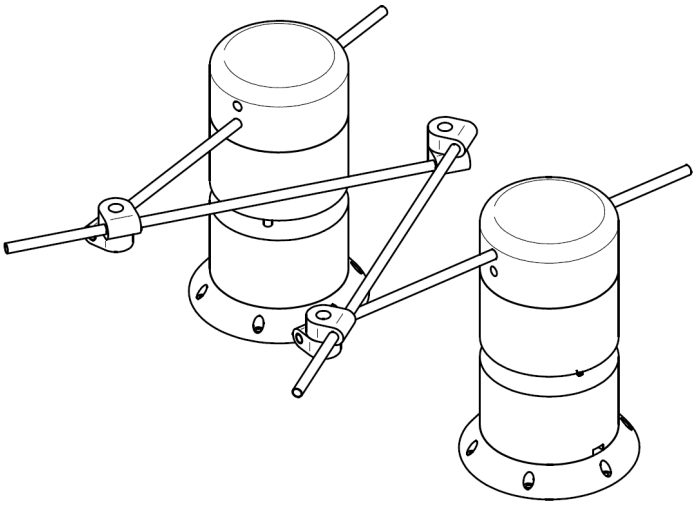


Figura 3.3: Propuesta de segundo prototipo con articulaciones activas no coaxiales.

En la propuesta de diseño final para el segundo prototipo se decidió realizar cambios en la geometría del robot. A pesar de que la propuesta en el primer diseño era muy interesante desde un punto de vista académico con el análisis de singularidades, el movimiento entre modos de trabajo y ensamble y demás aspectos mencionados previamente; es complejo lograr un diseño que en toda su libertad de movimiento no tenga interferencia mecánica y además logre un desempeño adecuado.

Por todo lo expuesto, se llegó a las siguientes conclusiones de diseño: que las articulaciones activas se encuentren en un mismo eje, que exista una relación de transmisión entre la articulación y el motor, y que los eslabones próximos a las articulaciones activas sean de menor longitud que aquellos que por su extremo forman el extremo operativo. La nueva configuración puede observarse en la Fig. 3.4.

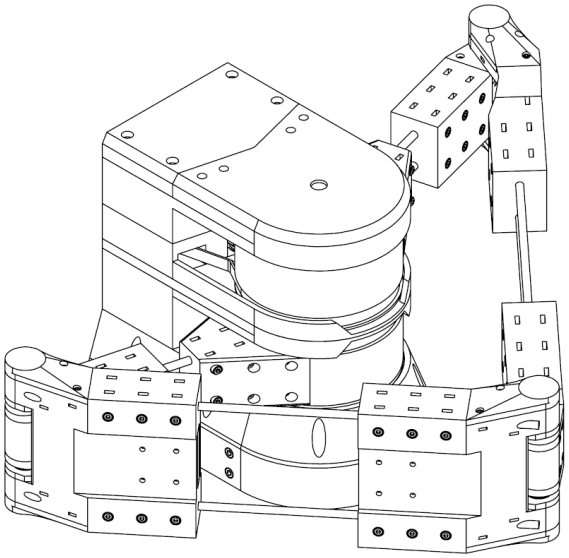


Figura 3.4: Propuesta de segundo prototipo con articulaciones activas coaxiales.

A continuación se describe el proceso de diseño y fabricación del segundo prototipo, y se presentan los resultados como última versión del trabajo.

3.2.1. Relación de transmisión

El diseño mecánico del segundo prototipo se inició visualizando la relación de transmisión deseada entre los motores y las articulaciones A_{11} y A_{21}. El objetivo de la relación de transmisión no solo es aumentar el torque disponible, sino disminuir la velocidad de la articulación permitiendo que el actuador trabaje en un rango de velocidades mayor y así lograr un movimiento más suave.

Se propuso utilizar una relación de transmisión de 1:3, pero se estableció que el diseño se realizara de tal forma que sea factible cambiar su valor luego de hacer pruebas experimentales.

Por practicidad en la realización del prototipo, se definió construir las poleas necesarias con fabricación aditiva (impresión 3D). Basándose en diseños como los del brazo manipulador MOVEO de BCN3D, y también contemplando las características de tracción necesarias, se decidió utilizar poleas dentadas o sincrónicas. Además, este tipo de mecanismo permite tener una relación de transmisión constante, un movimiento preciso (no tiene resbalamiento), tiene un rendimiento elevado y la tensión necesaria para su funcionamiento es menor que en una correa plana.

A diferencia del brazo MOVEO donde se utilizan correas dentadas de la serie HTD-5M, se utilizaron correas de la serie GT2 por ser las de mayor disponibilidad en el mercado local. La desventaja es el tamaño reducido de los dientes en comparación a otras: paso de 2mm y una altura de diente de aproximadamente 0.75mm, frente a los 5mm de paso y 2.1mm de altura de las HTD-5M. En la Fig. 3.5 se puede observar el perfil acotado de las correas de la serie HTD-5M, y en la Fig. 3.6 el perfil de las correas de la serie GT2.

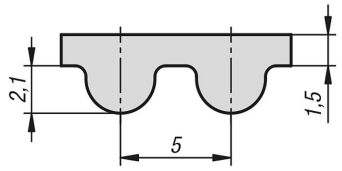


Figura 3.5: Dimensiones de correas serie HTD-5M.

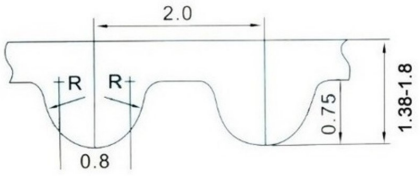


Figura 3.6: Dimensiones de correas serie GT2. Patrón utilizado en diseño de poleas dentadas.

En el proceso de diseño de las poleas sobre el software CAD, se realizó una extrusión del negativo del perfil de la correa, se lo extendió a la mitad de la cantidad de dientes correspondientes, se lo flexionó 180° y luego de una simetría se obtuvo las piezas de la Fig. 3.7. Todas las operaciones mencionadas fueron parametrizadas, por lo que se obtuvo un diseño final donde ingresando como parámetros la cantidad de dientes y el ancho, automáticamente se obtiene la polea deseada.

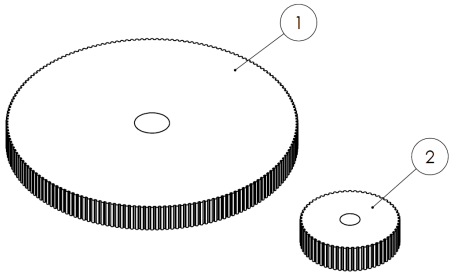


Figura 3.7: Poleas dentadas GT2 (1 conducida y 2 conductora) diseñadas en software CAD para el mecanismo de transmisión.

Siendo la ecuación de poleas dentadas que relaciona la cantidad de dientes Z, el diámetro D y el paso p la siguiente:

$$\pi D = Zp \tag{3.1}$$

Y siendo la relación de reducción deseada 1:3, se elige una polea conducida de 180 dientes (pieza 1 en Fig. 3.7), y una polea conductora de 60 dientes (pieza 2 en Fig. 3.7). Aplicando la ecuación 3.1 resulta en diámetros primitivos de aproximadamente 114.65mm y 38.22mm respectivamente (que verifican con las piezas obtenidas en el software).

Una vez definido el tamaño de las poleas, se procedió a plantear la distribución espacial de los componentes de ambos sistemas de transmisión, intentando aprovechar al máximo el espacio disponible y finalizar con un producto compacto.

Como en el modelo geométrico planteado para este segundo prototipo las dos primeras articulaciones, es decir, las articulaciones activas, comparten el mismo eje de rotación, las poleas conducidas coinciden axialmente una sobre otra. Luego se colocan los motores uno al lado del otro con un offset en altura debido al cuerpo de la articulación.

Los motores utilizados fueron paso a paso Nema 23 con torque de 2.5 Nm y ejes de 8mm y 6.35mm. Ambos motores son iguales, el diámetro original del eje de los motores es de 8mm, pero uno de ellos ha sido mecanizado a 6.35mm previo a su adquisición y es por esto que en las poleas conductoras se parametrizó la dimensión del eje, para adaptarse al motor. Las dimensiones generales de los actuadores pueden observarse en la Fig. 3.8.

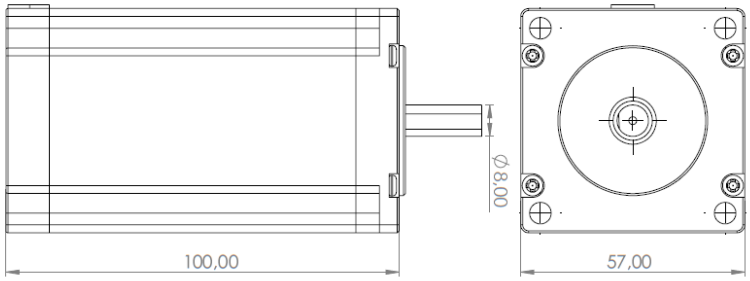


Figura 3.8: Vista de motores Nema 23 utilizados en el segundo prototipo.

Para lograr un correcto movimiento de las correas se colocó un eje con poleas de guía, que tienen como objetivo dirigir el camino de la correa sin que impacten con las demás piezas de la base (ver sección 3.2.3). Todo lo anterior puede observarse en la Fig. 3.9.

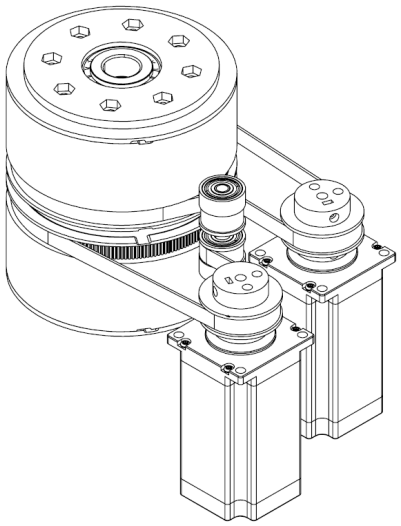


Figura 3.9: Sistemas de transmisión resultante para construcción de articulaciones activas

Teniendo presente las limitaciones de las técnicas y herramientas disponibles para la construcción del prototipo, se optó por la utilización de correas abiertas, que en la práctica facilitan el montaje, son más accesibles en el mercado local (con correas cerradas uno debería diseñar la relación de transmisión en base a las longitudes presentes en el mercado, lo que es un gran limitante) y además son más flexibles en esta etapa de diseño ya que es fácil modificar su longitud.

En la Fig. 3.10 se encuentra la vista superior de uno de los sistemas de transmisión. Se puede observar como la polea guía incrementa el arco de contacto de la correa con la polea conducida, algo muy favorable siendo que el mayor limitante del rango articular es el hecho de que los extremos de la correa deben permanecer en contacto con la polea de forma permanente. Es decir, la polea guía, además de colaborar con el objetivo de obtener un diseño más compacto, incrementa el rango articular.

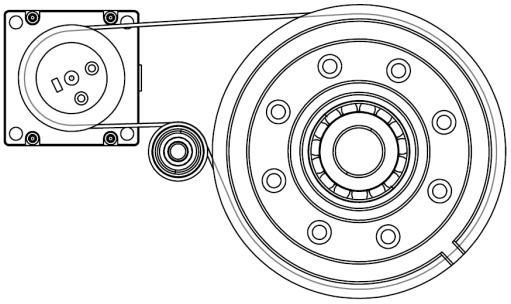


Figura 3.10: Vista superior del sistema de transmisión resultante para una de las articulaciones activas.

3.2.2. Articulaciones activas A_{11} y A_{21}

Previo al diseño de la base del prototipo se realizó el diseño de las articulaciones activas A_{11} y A_{21}. Uno de los objetivos en el diseño de estas articulaciones era permitir el montaje y desmontaje de los correspondientes brazos de una forma relativamente sencilla (sin perjudicar la rigidez del sistema). El motivo era facilitar modificaciones futuras en el diseño, modularizando la base con los motores, sistema de transmisión, sensores, cableados, etc., de los brazos del prototipo.

Como se mencionó previamente, el sistema de transmisión diseñado contempla el uso de correa abierta por lo que fue necesario idear un método de fijación de los extremos sobre la polea conducida. La forma propuesta fue a través de un pin impreso en 3D que al atornillarse comprime los extremos de la correa. En la Fig. 3.11 se puede observar el pin mencionado (pieza 3) que es fijado a la polea conducida (pieza 1) a través del tornillo M3 (pieza 4) y la tuerca M3 (pieza 2) que se inserta dentro de la pieza 1 en la ranura pertinente. En un principio se supuso que este método sería suficiente no solo para resolver la fijación de los extremos de la correa, sino también el tensado de la misma.

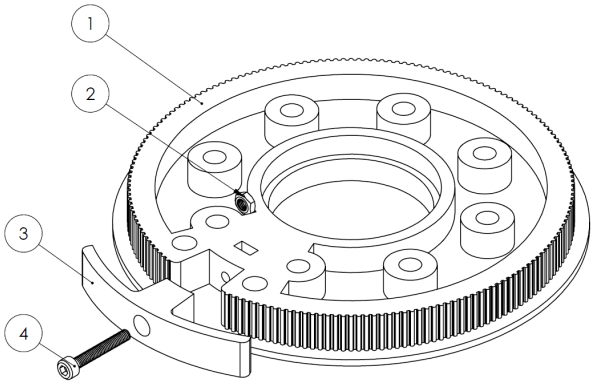


Figura 3.11: Vista explosionada de polea conducida con sistema de fijación de extremos de correa.

Sin embargo, el pin, a pesar de ser una pieza pequeña y relativamente sencilla, es muy importante en

el funcionamiento final del sistema. Siendo su objetivo fijar los extremos de la correa, es una pieza que queda sometida a esfuerzos relativamente grandes, y la rigidez en una pieza tan pequeña construida en plástico como PLA o PETG resultó ser un problema. Su diseño requirió varias iteraciones, incluyendo en el bucle la correspondiente fabricación y evaluación del funcionamiento en el prototipo.

Para lograr la rotación de la articulación con el menor rozamiento posible, y manteniendo la rigidez deseada, se estudió la incorporación de rodamientos. Analizando los esfuerzos presentes sobre la articulación en los puntos de rotación, que son tanto axiales como radiales, se seleccionó rodamientos del tipo cónico de rodillos.

Los rodamientos cónicos de rodillos cuentan con un conjunto de copa y cono. La copa está formada por el aro exterior y el conjunto de cono por el aro interior, los rodillos y la jaula. Esta estructura de rodamiento soporta cargas combinadas y proporciona una baja fricción durante el funcionamiento. En la Fig. 3.12 se puede observar 3 formas de montar en pareja los rodamientos cónicos de rodillos: a la izquierda tipo "X" o cara contra cara, en el centro tipo "O" o espalda contra espalda, y a la derecha la disposición en tándem.

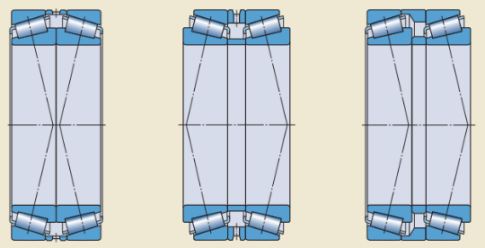


Figura 3.12: Tipos de montaje de rodamientos cónicos de rodillos.

El diseño se debe realizar contemplando una pareja de rodamientos colocados de forma invertida y bajo precarga para obtener una solución rígida. La disposición seleccionada, de tal forma que se facilite el montaje, fue tipo "O". En la Fig. 3.13 se puede observar un corte de la articulación diseñada para exponer la disposición de los rodamientos cónicos de rodillos.

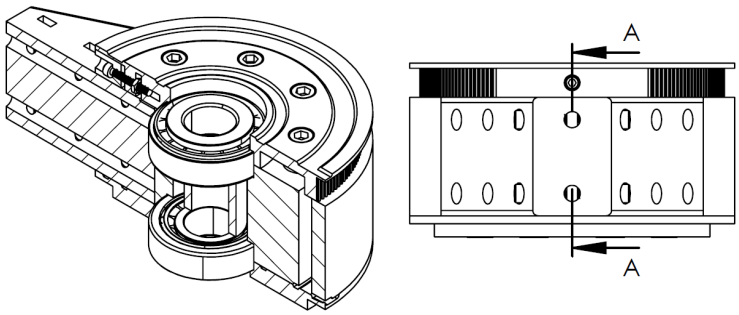


Figura 3.13: Corte de articulación activa A_{11} o A_{21}.

La selección del modelo en específico se vio acotada a aquellos disponibles en el mercado local y que estuvieran dentro del rango de precios aceptables en el presupuesto del proyecto. Es así como se definió el uso del rodamiento cónico de rodillos 30204, principalmente por sus dimensiones que se adecúan a las de la articulación planteada: esto es 47mm de diámetro exterior y 20mm de diámetro interior. En la Fig. 3.14 se puede observar una vista explosionada de un modelo simplificado del rodamiento cónico de rodillos 30204 proveído por SKF.

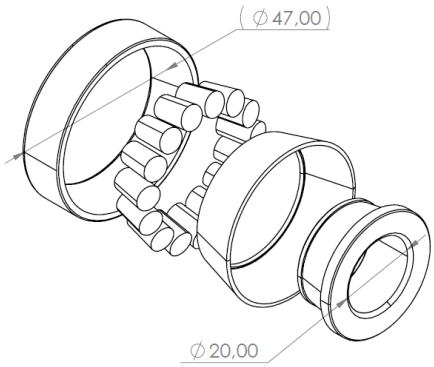


Figura 3.14: Vista explosionada de modelo simplificado de rodamiento SKF 30204.

No se realizó un estudio detallado de la magnitud de los esfuerzos a los que se encuentra sometido el rodamiento por razones de tiempo y porque se trata de valores muy por debajo de los admisibles por el elemento en cuestión.

Una característica importante añadida en el proceso de diseño al cuerpo de las articulaciones activas, fue un agujero donde al ensamblar se puede colocar un imán de neodimio que permitirá marcar el límite de movimiento de la articulación, haciendo uso de un sensor de efecto hall que se describirá más adelante.

Por disponibilidad de material, y buscando buena rigidez en el sistema, para el eje de rotación de las articulaciones A_{11} y A_{21} se utilizó una barra rectificada de acero de 12mm de diámetro.

En la Fig. 3.15 se puede observar el diseño realizado. Para el montaje del cuerpo de la articulación se optó por la utilización de 8 tornillos allen M6 con sus correspondientes tuercas.

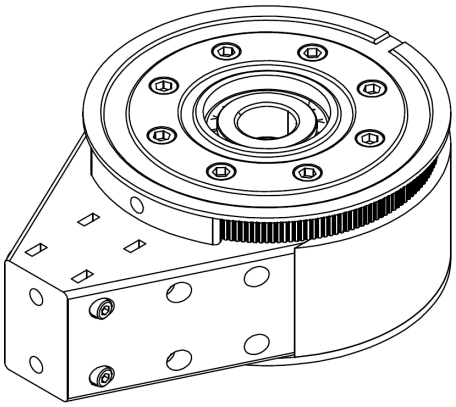


Figura 3.15: Vista del diseño de la articulación A_{11} y A_{21}.

Mecánicamente, A_{11} y A_{21} son iguales y se montan sobre la base una sobre otra de forma invertida. Su disposición se puede observar en la Fig. 3.16.

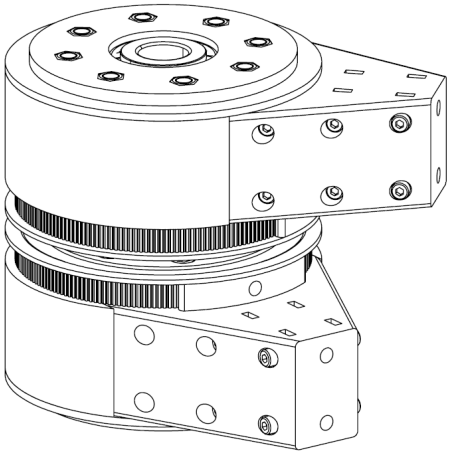


Figura 3.16: Disposición espacial de articulaciones A_{11} y A_{21}.

3.2.3. Poleas guía del sistema de transmisión

En el diseño de las poleas guía se utilizaron rodamientos 608-2Z. En una revisión del prototipo se puede contemplar su reemplazo por un modelo más económico y de menores dimensiones, los 608-2Z se utilizaron por disponibilidad pero se encuentran sobredimensionados.

Nuevamente, por disponibilidad de material, para el eje de rotación de las poleas guía se utilizó una barra rectificada de acero de 6mm de diámetro, y espaciadores impresos en 3D.

En la Fig. 3.17 se puede ver un corte de la polea guía descripta.

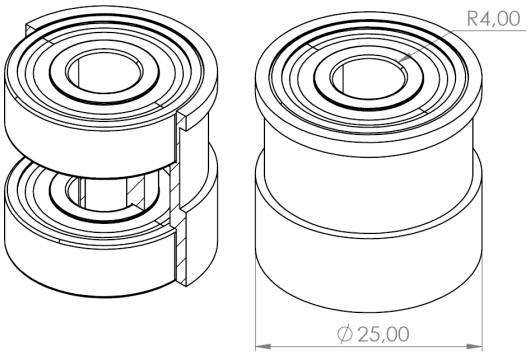


Figura 3.17: Corte de polea guía.

En el corte se puede observar que la canaleta de la polea guía se encuentra descentrada, es decir, la polea es asimétrica. El motivo puede observarse en la vista lateral de la Fig. 3.18. El proceso de diseño fue dar primero una distancia adecuada a los dos rodamientos, y luego adaptar la canaleta de la polea inferior a la altura entre la base donde se apoya y la correspondiente correa. Una vez que se obtuvo la primera polea, la segunda es idéntica pero se monta de forma invertida. Luego, solo restó diseñar el separador con las dimensiones para que la polea superior quede a la altura adecuada.

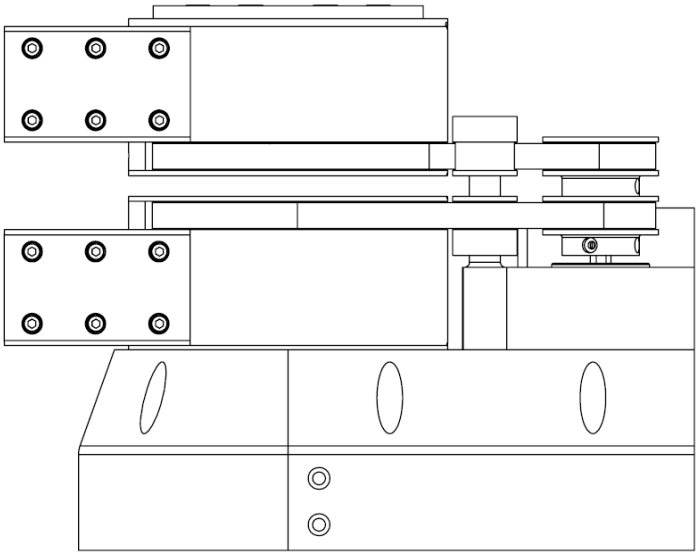


Figura 3.18: Vista lateral de poleas guía sobre base en interacción con A_{11} y A_{21}.

En un futuro se puede realizar una revisión de estas poleas para que resulten en un cuerpo simétrico. También se puede estudiar incorporar un mecanismo de tensión de la correa haciendo que su eje sea móvil.

3.2.4. Base del prototipo

Para el diseño de la base se procedió de abajo hacia arriba, dividiendo el elemento en diferentes niveles correspondientes a la forma en que se ensambla el prototipo. La división es de la siguiente forma: un primer nivel donde se alojan los motores y reposa la articulación activa A_{11}, un segundo nivel que separa las dos articulaciones activas, y un tercer nivel donde se realiza un cierre de la base.

Se comenzó con la pieza que aloja los motores Nema 23, para lo que se tuvo en cuenta la incorporación de una ranura que permita guiar sus cables (y también del resto de componentes electrónicos, que son los sensores de efecto hall y los encoder incrementales) hacia una abertura en la parte posterior del prototipo. Luego se diseñó la base de la articulación A_{11}, donde se coloca el eje de 12mm de diámetro. También se encuentran los agujeros para poder fijar el prototipo a la superficie donde operará y el agujero donde se aloja la barra rectificada de 6mm que hace de eje a las poleas guía. El primer nivel de la base puede observarse en la Fig. 3.19.

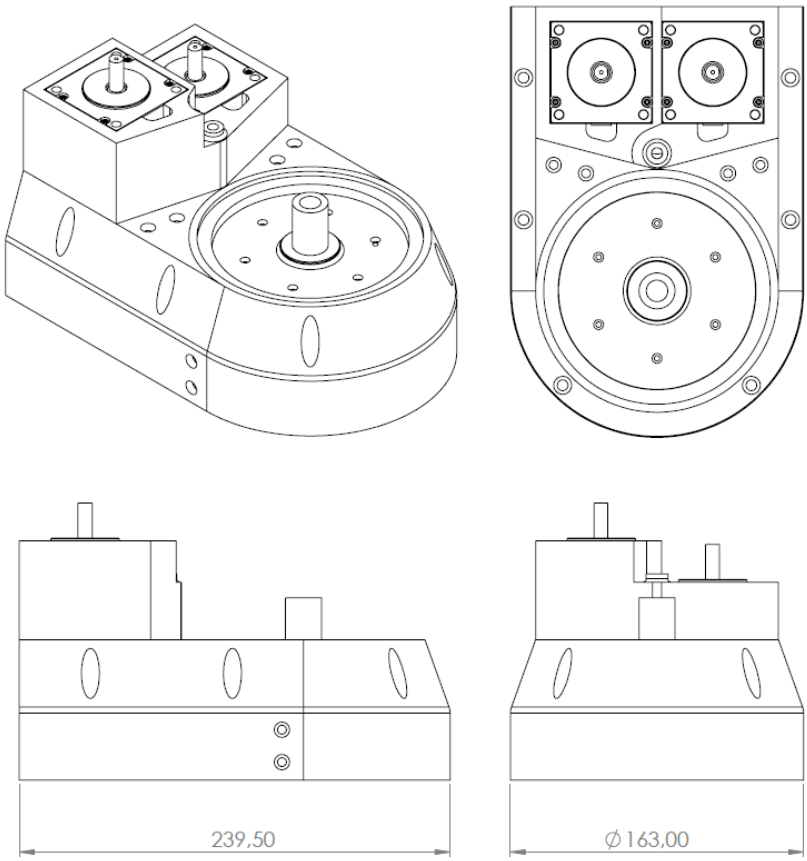


Figura 3.19: Vista del primer nivel de la base.

En primera instancia se diseñó lo anterior como una única pieza sólida y luego se procedió a dividirla para adecuarse al espacio de trabajo de la impresora 3D disponible, mediante operaciones de corte y agregando los correspondientes mecanismos de fijación a través de tuerca-tornillo. El anillo interno y superficie sobre la que apoya el cono del rodamiento cónico de rodillos también se diseñó como una pieza separada que se monta sobre la base mediante 6 tornillos e insertos para impresión 3D M3. Dicha división se puede observar en la vista explosionada del primer nivel en la Fig. 3.20.

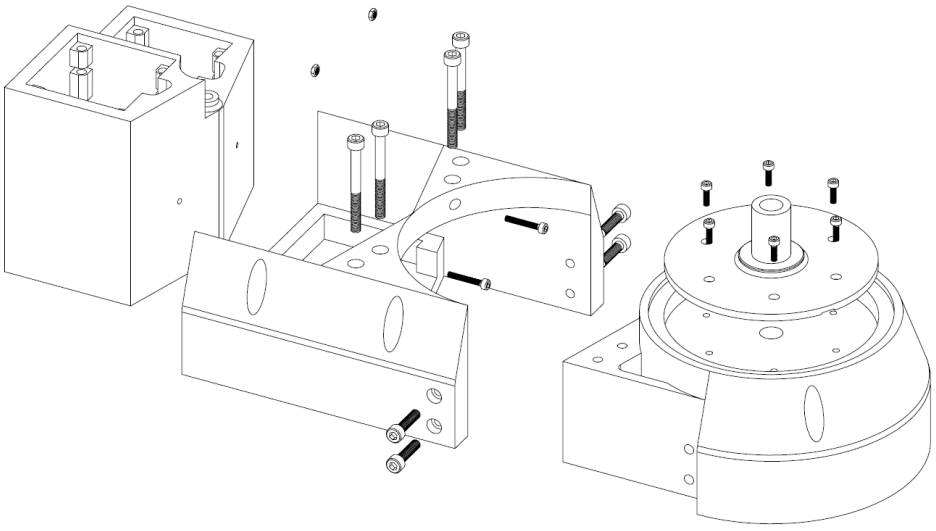


Figura 3.20: Vista explosionada del primer nivel de la base.

Se continuó con el diseño de las piezas que se observan en la Fig. 3.21, que separan las articulaciones activas. Se diseñó de tal forma que sean dos piezas iguales que se ajustan mediante tornillos y tuercas M3 de forma invertida. Entre ambas se conforma una ranura donde se colocan dos sensores de efecto hall en los ángulos adecuados, que permitirán sensar el límite de ambas articulaciones activas. Además se debió tener en cuenta el espacio para las poleas conductoras, el movimiento de la correa, y las poleas guía, permitiendo su ensamblaje con facilidad.

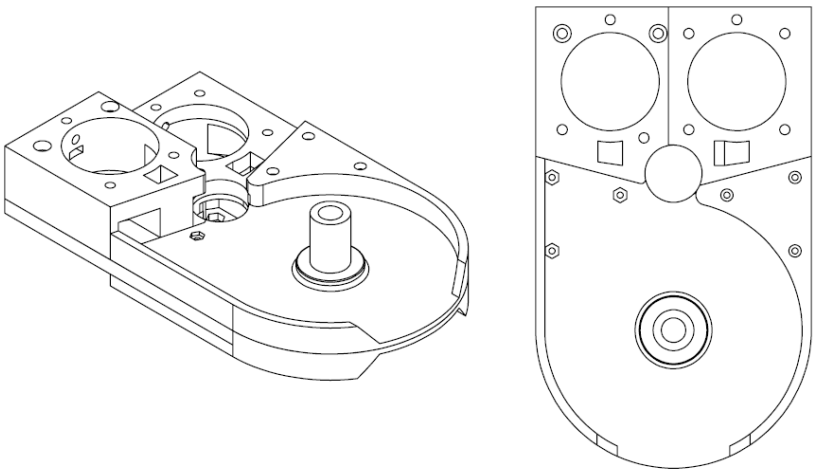


Figura 3.21: Vista del segundo nivel de la base.

El tercer nivel se observa en la Fig. 3.22 y da un cierre a la base. Tiene como función el alojamiento del encoder rotativo incremental de cada eje motor, la sujeción del extremo libre del eje de las articulaciones activas (barra rectificada de 12mm) y el eje de las poleas guías. Además posee la superficie y anillo interior donde reposa el cono del rodamiento cónico de rodillos superior de la segunda articulación activa.

Para dar una mejor apariencia estética al prototipo, también se diseñó una tapa superior que se ajusta con tornillos y tuerca M4.

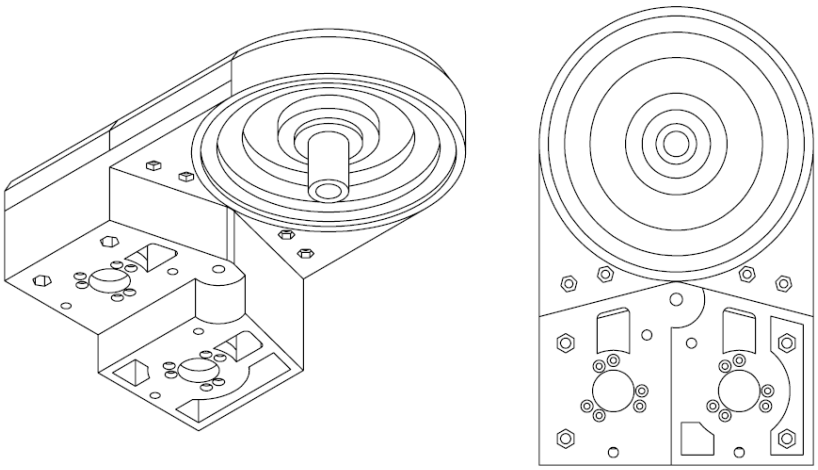


Figura 3.22: Vista del tercer nivel de la base.

Todas las piezas de la base fueron diseñadas contemplando un comportamiento simétrico del robot, considerando una apertura máxima de ambos brazos de aproximadamente 260° como se observa en la Fig. 3.23.

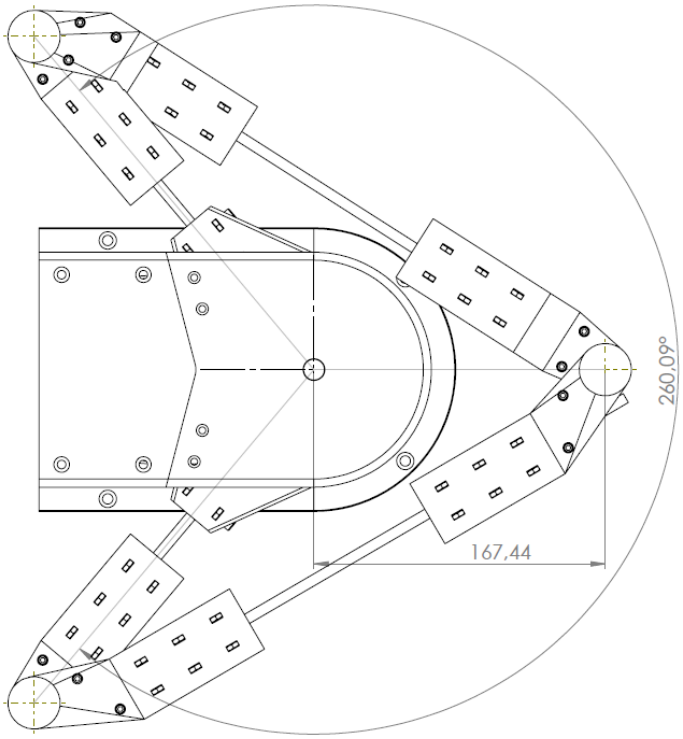


Figura 3.23: Vista del prototipo en máxima retracción del extremo hacia la base.

Por motivos de interferencia mecánica los brazos en su movimiento nunca se cruzan, es decir, el brazo inferior siempre permanece a la derecha del brazo superior. Como se observa en la Fig. 3.24, cuando se encuentran en máxima extensión (posición con el extremo más lejos de la base) pueden formar un ángulo de aproximadamente 28° ubicando al extremo a unos 618mm de la base.

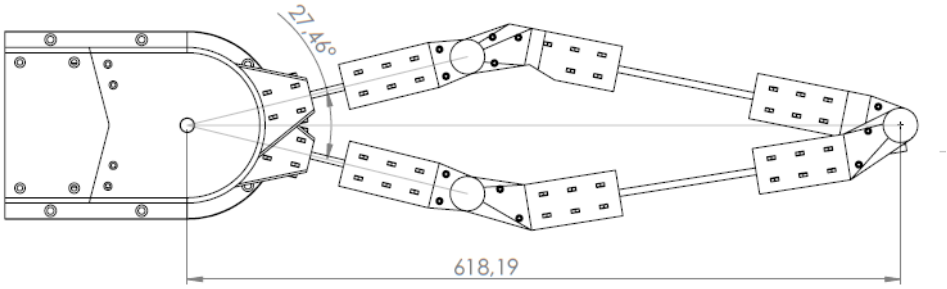


Figura 3.24: Vista del prototipo en máxima extensión del extremo.

3.2.5. Articulaciones pasivas A_{12} y A_{22}

El inicio del diseño de las articulaciones pasivas A_{12} y A_{22} fue con el cuerpo que se observa en la Fig. 3.25, asociado a los eslabones 11 y 21. Es un cuerpo, impreso en 3D, que permite deslizar las barras rectificadas de 6mm que conforman el cuerpo del eslabón y ajustarlas con tornillos y tuercas M4. El eje de la articulación consiste en una varilla roscada M6 que se enrosca sobre el cuerpo, quedando fija mediante tuercas colocadas previamente dentro del cuerpo en su montaje (ver apéndice D, Fig. D.5).

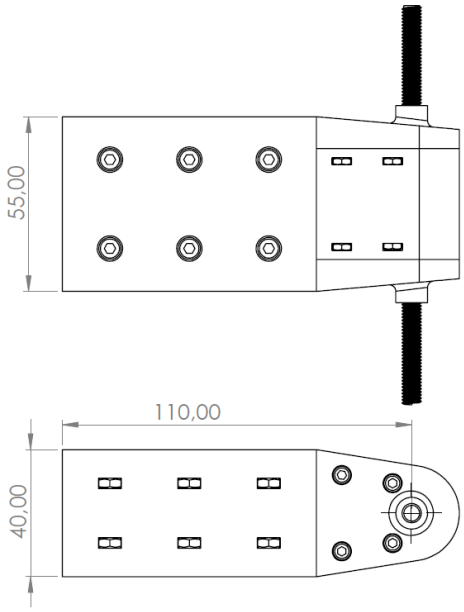


Figura 3.25: Primer cuerpo de articulaciones pasivas A_{12} y A_{22}.

Los cuerpos descriptos son iguales para ambas articulaciones, con lo que se puede resaltar que los primeros dos eslabones son completamente idénticos: polea conducida y su respectivo cuerpo, las barras rectificadas que conforman el eslabón y el primer cuerpo de la articulación pasiva, como se expone en la Fig. 3.26. Ambos se ensamblan para formar eslabones de una longitud entre ejes de 250mm.

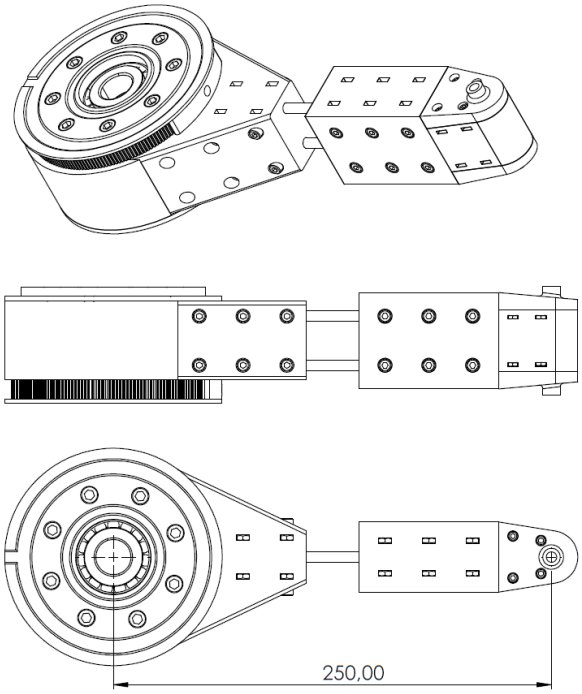


Figura 3.26: Vista de eslabones 11 y 21.

El segundo cuerpo de la articulación A_{12} sí difiere del de la articulación A_{22}, pero conceptualmente siguen el mismo principio que es el de ser una horquilla que rota sobre la varilla roscada descripta previamente. Esa rotación se realiza sobre rodamientos 608-2Z, con los separadores impresos en 3D similares a los utilizados en las polea guía de la base. El mecanismo de sujeción de las barras rectificadas

del segundo eslabón es el mismo, se deslizan sobre agujeros en el cuerpo de la articulación y se fijan mediante tornillos y tuercas M4.

El segundo cuerpo de la articulación A_{12} se diseña de tal forma que las barras del eslabón 12 queden paralelas al eslabón 11, es decir perpendiculares al eje de rotación como se observa en la Fig. 3.27. Además, también se considera que la articulación completamente ensamblada debe poder extenderse hasta un ángulo de 180° y flexionarse hasta, por lo menos, 15°, como se observa en la Fig. 3.28.

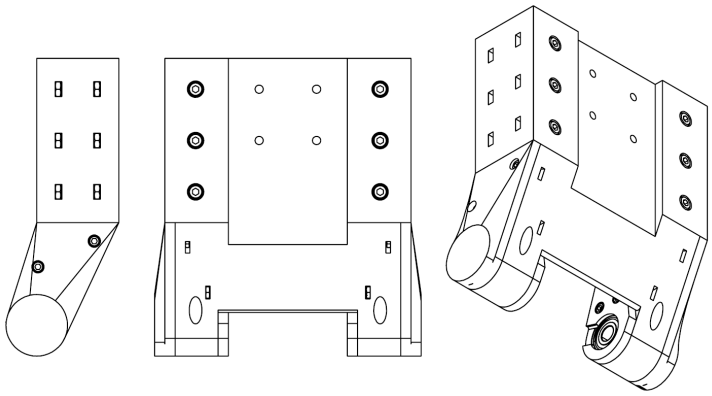


Figura 3.27: Vista de segundo cuerpo de articulación A_{12}.

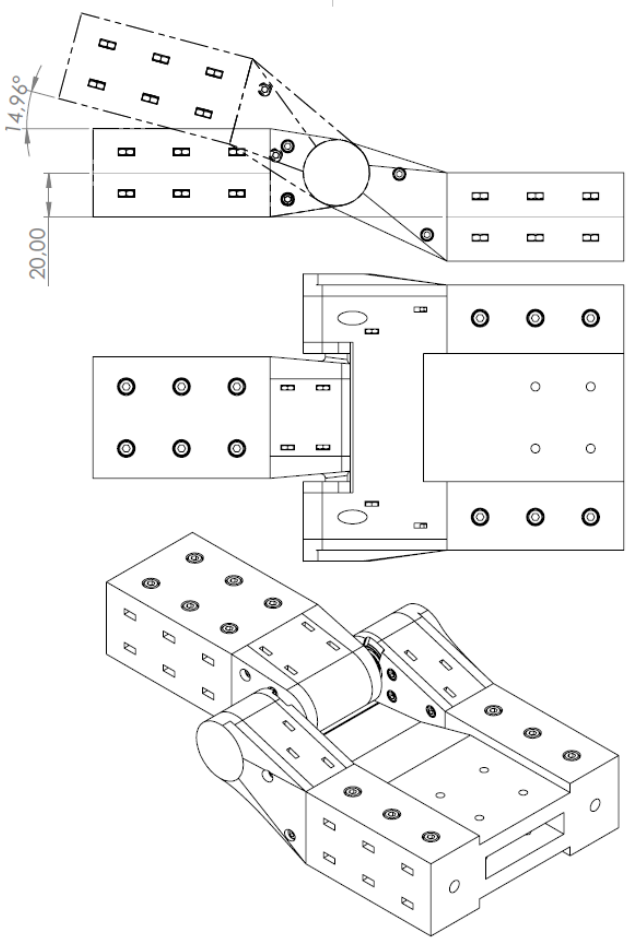


Figura 3.28: Vista de articulación A_{12} completamente extendida y vista en trazo de máxima flexión.

El segundo cuerpo de la articulación A_{22} se diseña de tal forma que las barras del segundo eslabón

queden a un ángulo de aproximadamente 15.77°(ver Fig. 3.29), que permita el cierre del mecanismo de 5 barras mediante la articulación A_{23} con un eslabón de 380mm. También se consideró en el diseño del segundo cuerpo, que la articulación A_{22} completamente ensamblada debe poder extenderse hasta un ángulo de 180° y flexionarse hasta un ángulo de, por lo menos, 21°, como se observa en la Fig. 3.30.

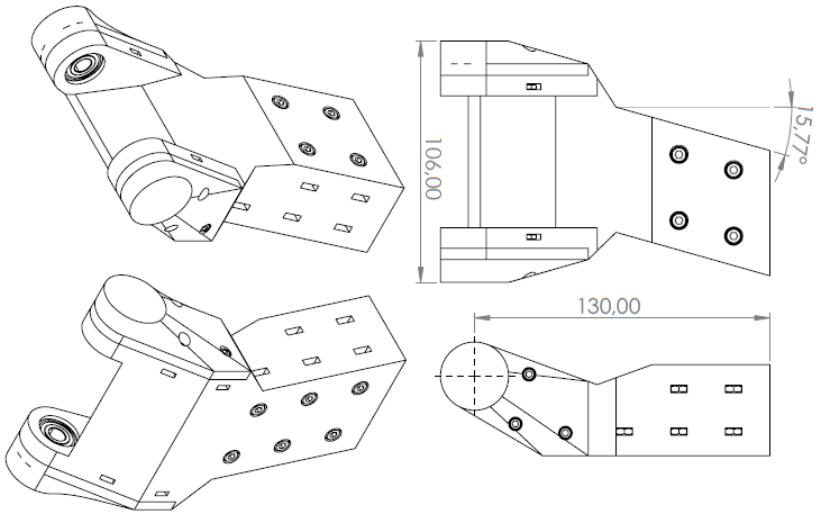


Figura 3.29: Vista de segundo cuerpo de articulación A_{22}.

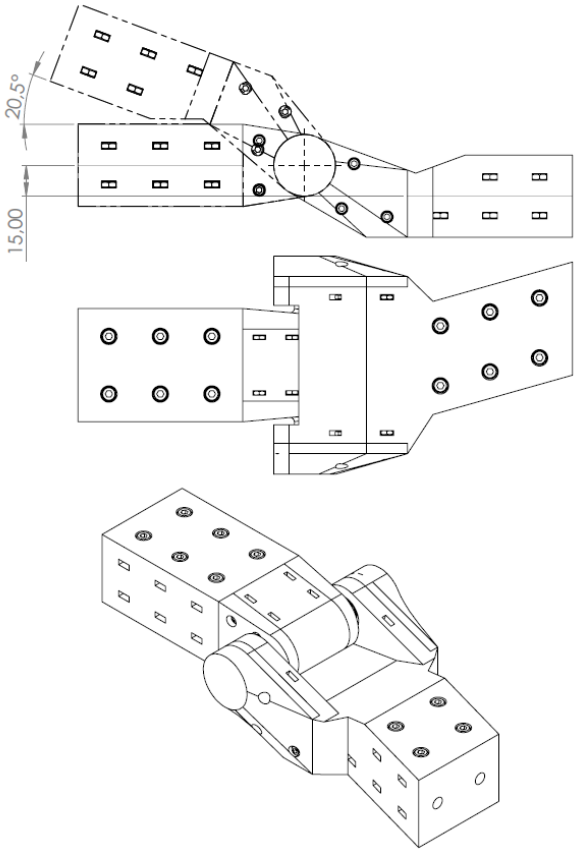


Figura 3.30: Vista de articulación A_{22} completamente extendida y vista en trazo de máxima flexión.

3.2.6. Articulación pasiva de cierre de cadena de eslabones

El primer cuerpo de la articulación A_{23}, vinculado al eslabón 12, es exactamente igual al segundo cuerpo de la articulación A_{12}, que corresponde a la Fig. 3.27. Ambos cuerpos quedan ensamblados como se observa en la Fig. 3.31, formando el eslabón 12 de 380mm de longitud.

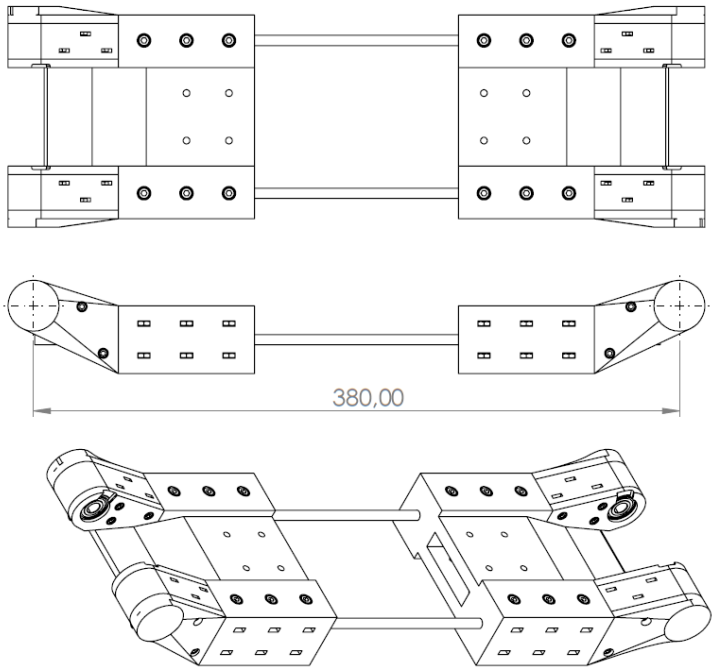


Figura 3.31: Vista de eslabón 12.

Se lo diseñó para que su longitud pueda ajustarse de forma sencilla, deslizando uno de los cuerpos sobre la barra y ajustando los tornillos sobre ellas a la longitud deseada. En el modelo se consideró una longitud de 380mm.

El segundo cuerpo de la articulación A_{23} se diseña siguiendo los mismos principios que el cuerpo de la Fig. 3.25, pero como se mencionó para el segundo cuerpo de la articulación A_{22}, se modifica el ángulo del cuerpo para que forme un ángulo de 15.77° con la horizontal. De esta forma con el cuerpo de la Fig. 3.29 y el cuerpo de la Fig. 3.32 se puede ensamblar el eslabón 22, que entre ejes tiene una longitud de 380mm pero también baja 79mm de forma diagonal, como se observa en la Fig. 3.33, permitiendo el cierre con el eslabón 12 mediante la articulación A_{23}.

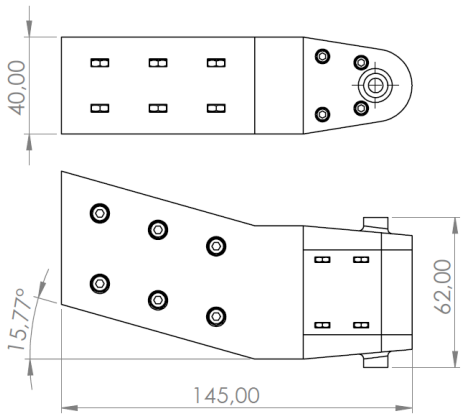


Figura 3.32: Vista de segundo cuerpo de articulación A_{23}.

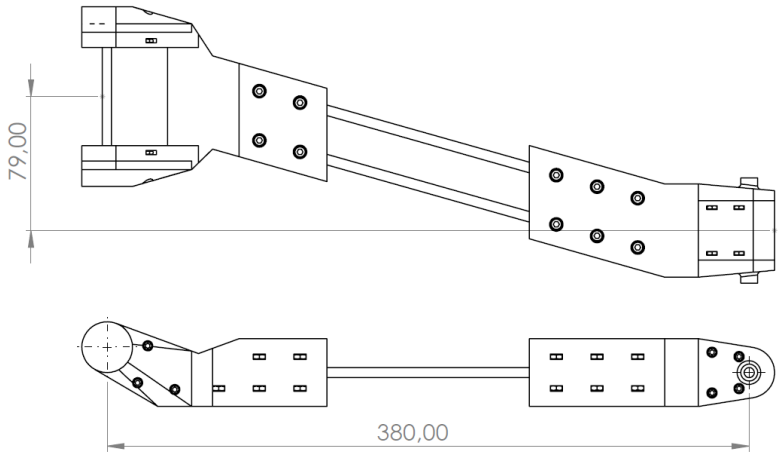


Figura 3.33: Vista de eslabón 22.

La articulación de cierre A_{23} puede observarse en la Fig. 3.34, en su posición de máxima extensión (formando un ángulo de 180°) y máxima flexión (permitiendo un ángulo de hasta 20°).

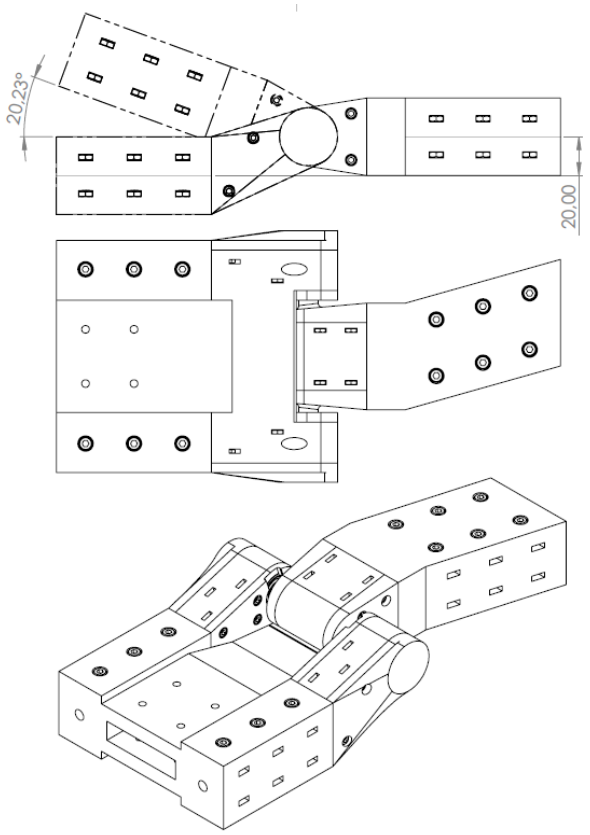


Figura 3.34: Vista de articulación A_{23} completamente extendida y vista en trazo de máxima flexión.

En la Fig. 3.35 se puede observar el ensamble final obtenido, segundo y último prototipo diseñado, y en la Fig. 3.36 se puede observar una fotografía del mismo ya fabricado.

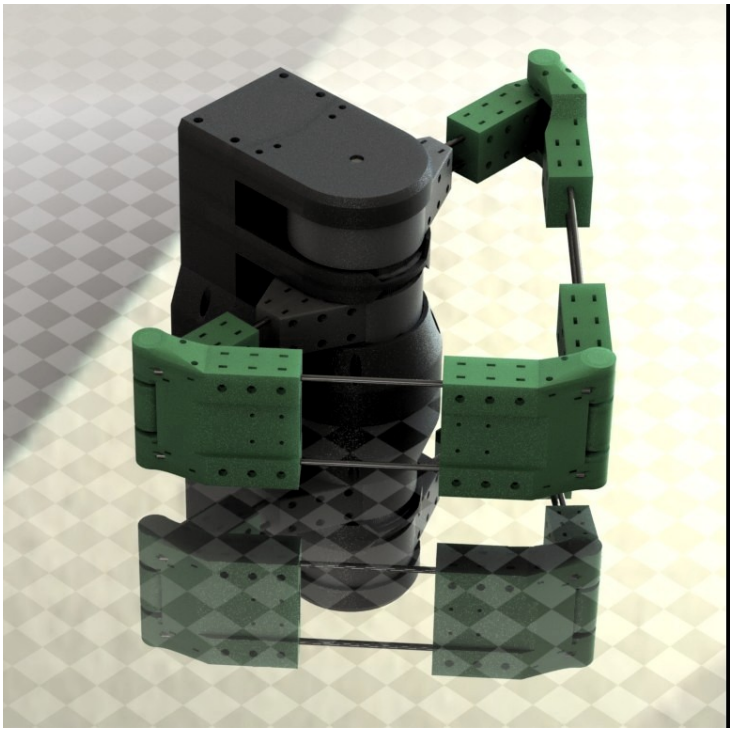


Figura 3.35: Render del segundo prototipo.

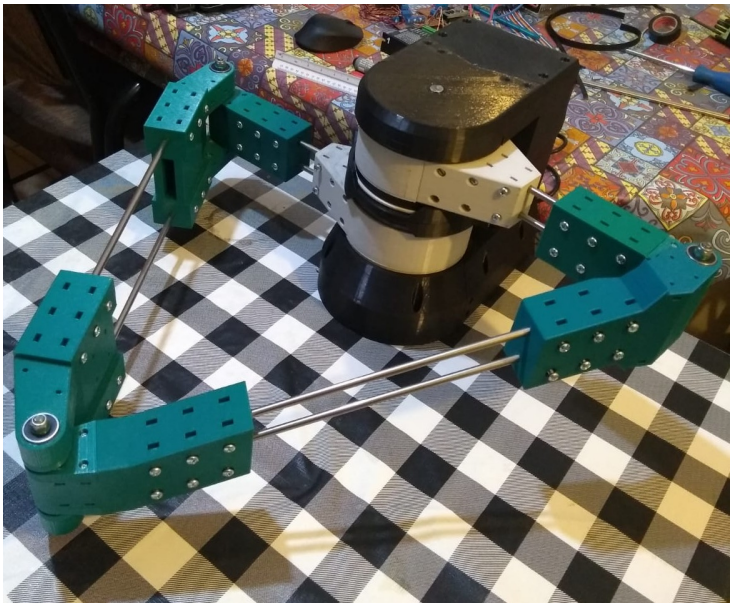


Figura 3.36: Fotografía de segundo prototipo construido.

3.2.7. Tensor de correas

Como se mencionó en la sección 3.2.2, en la elección del uso de correa abierta y el diseño del mecanismo de fijación de sus extremos sobre la polea conducida se supuso que no haría falta un mecanismo de tensión. La suposición fue errónea ya que en el ensamblaje del prototipo no hay forma de que, al fijar los extremos con el pin diseñado, la correa quede con la tensión adecuada.

Buscando solucionar el problema sin realizar grandes cambios en la estructura básica del robot, se obtuvo el mecanismo de tensión que se observa en la Fig. 3.37.

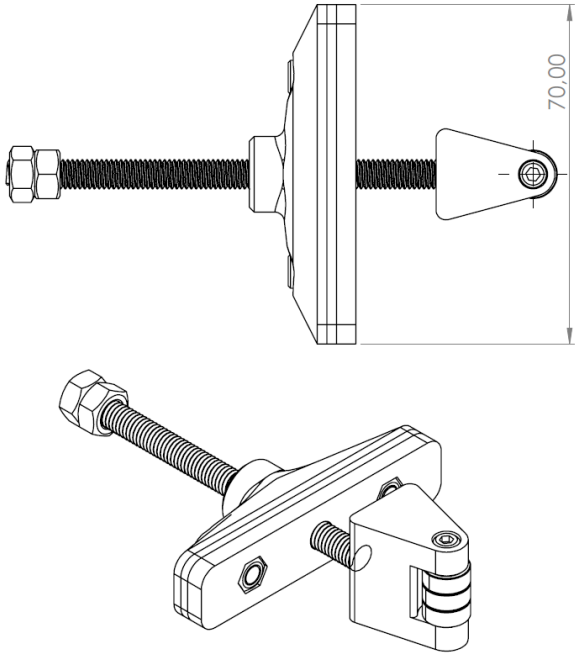


Figura 3.37: Tensor de correas.

Consiste en un cuerpo de varias láminas impresas en 3D que mediante tornillos y tuercas M4 se fija sobre la base del prototipo, y una varilla roscada M6 que rosca sobre una tuerca M6 dentro del cuerpo anterior, para desplazarse de forma perpendicular a la pared sobre la que se fija el mecanismo. La varilla en su extremo mueve un cuerpo también impreso en 3D que tiene como objetivo empujar, en mayor o menor medida, la correa a tensar.

Para disminuir la fricción entre la correa en movimiento y el cuerpo que la empuja, se colocaron 3 rodamientos 623-2Z con libertad de movimiento sobre un tornillo M3 que hace de eje.

En la Fig. 3.38 se puede observar el mecanismo de tensión de correa diseñado, montado sobre el prototipo completo.

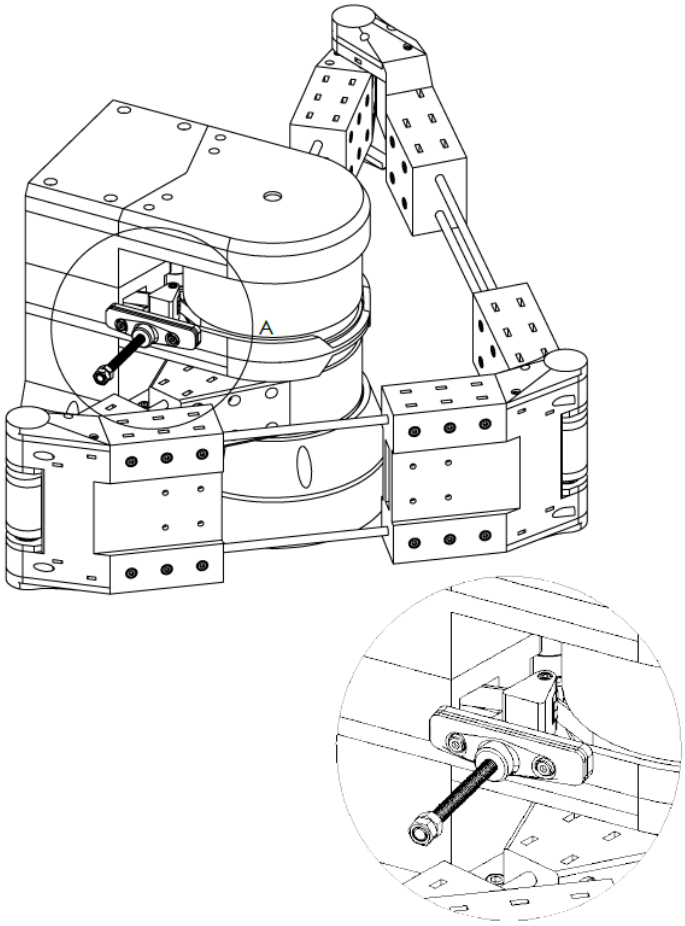


Figura 3.38: Tensor de correas montado sobre el ensamble del mecanismo completo.

3.3. Diseño del mecanismo asociado al eje Z

Como se mencionó en la sección 3.2, en el diseño del eje Z del robot, encargado de subir y bajar el extremo operativo, inicialmente se consideraron dos opciones: colocar el eje y mecanismo asociado en el extremo operativo, lo que conlleva a una mayor inercia en el extremo y posibles interferencias mecánicas; o darle movimiento a la base del robot, lo que tiene como desventaja que se debe mover toda la estructura.

Luego de contemplar y proponer diferentes diseños, como el que se mostró en la Fig. 3.3, se optó por colocar el eje Z en el extremo del mecanismo de 5 barras. Para esto se lo trató de implementar de tal forma que tenga la menor interferencia mecánica posible con el mecanismo base. Se propusieron dos soluciones:

La primera fue la utilización de un mecanismo de tornillo sin fin para el movimiento del eje, como se observa en la Fig. 3.39. Las principales desventajas de este mecanismo son su complejidad constructiva, el tamaño y peso, la selección de motores adecuados y su cableado.

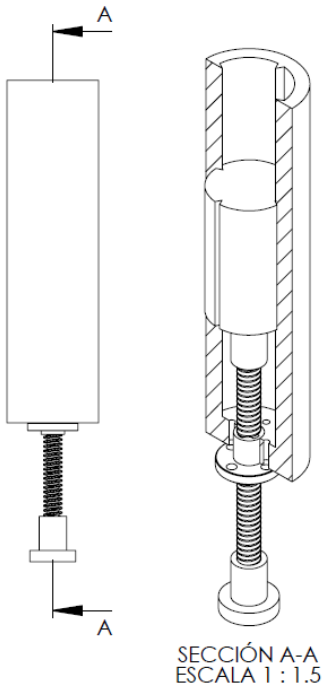


Figura 3.39: Propuesta de eje Z mediante tornillo.

También se propuso desarrollar una solución como la expuesta en la Fig. 3.40. Este consta de un mecanismo de cuatro barras actuado por un motor de corriente continua. Permite el ascenso y descenso de la carga con baja interferencia mecánica, pero su construcción es relativamente compleja por contar con tantas piezas móviles diferentes.

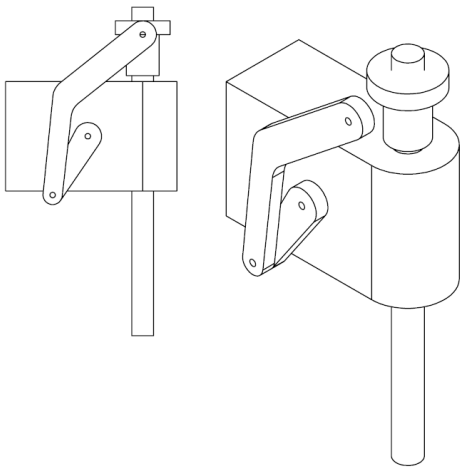


Figura 3.40: Propuesta de eje Z mediante mecanismo de 4 barras.

En última instancia, se optó por una solución mecánicamente más sencilla como se observa en la Fig. 3.41. Trata de un mecanismo con un engranaje de dientes rectos impulsado por un servomotor, que da movimiento de traslación a una cremallera de dientes rectos unida a dos barras de acero rectificadas de 6mm de diámetro que en el extremo poseen el extremo operativo.

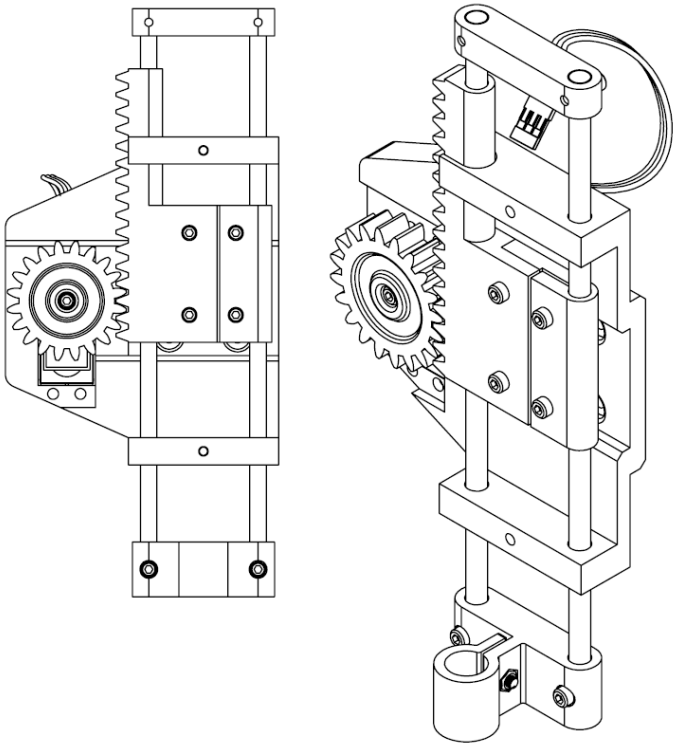


Figura 3.41: Eje Z diseñado para segundo prototipo con extremo para sostener lápiz.

El engranaje es de 20 dientes rectos, con módulo 2 (lo que resulta en un diámetro primitivo de 40mm) y un ancho de 10mm, fabricado con impresión 3D y montado al eje del servomotor a presión con un tornillo M3 de 10mm de longitud.

La cremallera es de 100mm de longitud y módulo 2, también fabricada con impresión 3D. La mitad de la cremallera puede desplazarse fuera del encapsulado del mecanismo, lo que resulta en un total de 50mm de amplitud en el movimiento del eje.

El servomotor utilizado es el modelo Tower Pro MG996R, con un torque de 9.4 Kgcm y alimentado con 5V.

Para finalizar el capítulo de diseño y fabricación del prototipo en estudio, en la Fig. 3.42 se puede observar una fotografía del resultado de la fabricación del eje Z descripto previamente, y en la Fig. 3.43 se puede observar el eje Z diseñado, montado sobre la estructura completa del prototipo.

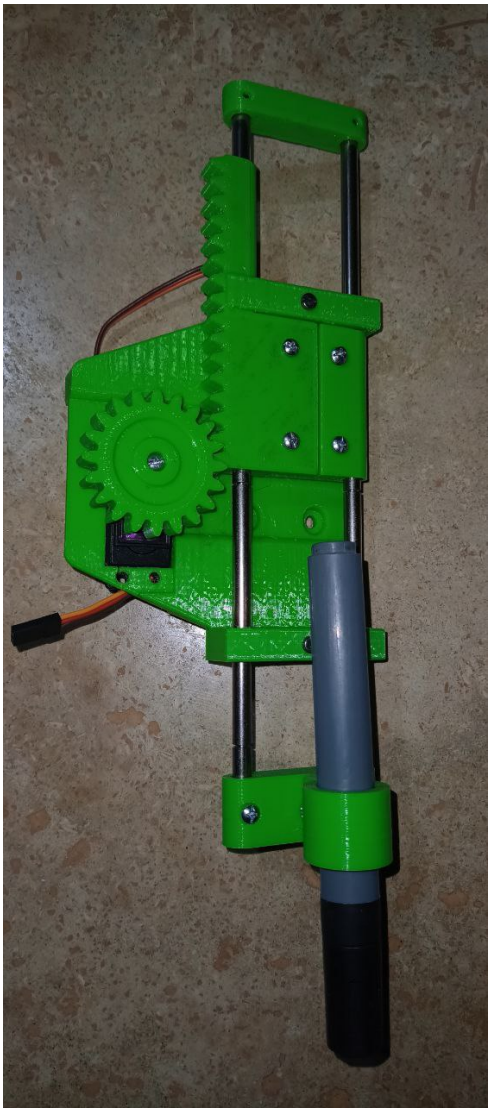


Figura 3.42: Fotografía de eje Z para segundo prototipo fabricado.

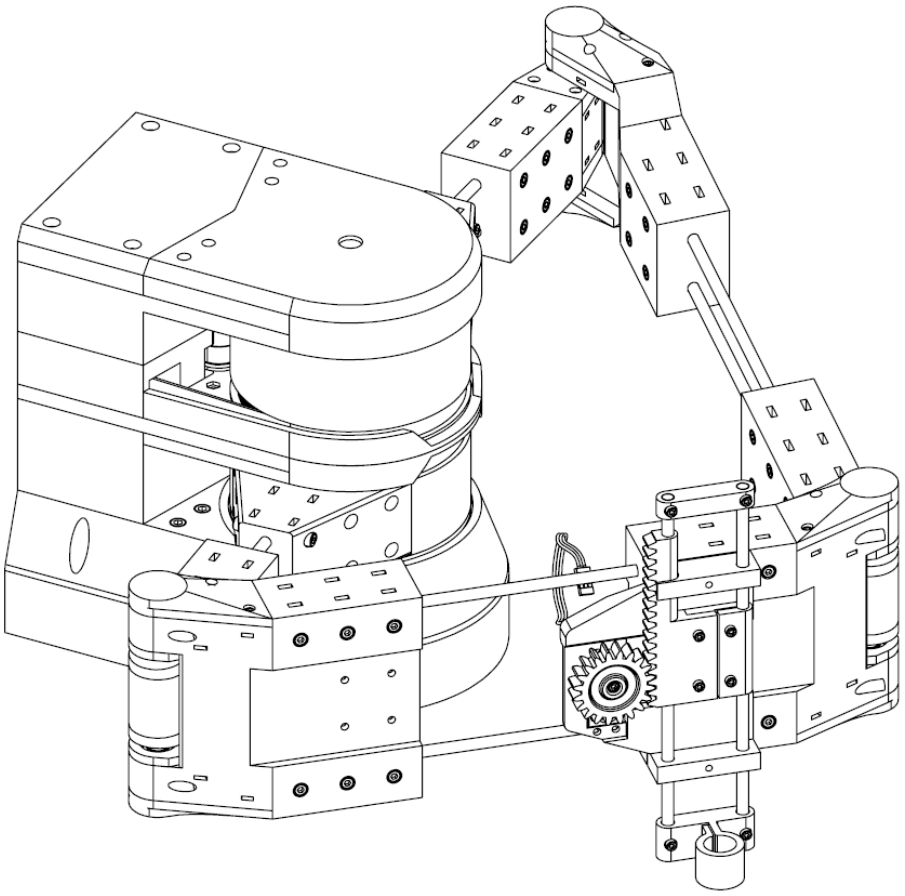


Figura 3.43: Vista del eje Z final montado sobre el ensamble del mecanismo completo.

Capítulo 4

Arquitectura de control, firmware y comunicaciones

En este capítulo se hablará de la arquitectura de control que se definió para el robot, junto con una descripción del software que corre en los nodos embebidos y en la computadora principal.

Primero se describirá el hardware utilizado, la arquitectura elegida, el proceso y la metodología de diseño del firmware, el software y las comunicaciones utilizadas. Luego, se explicará en detalle el funcionamiento del firmware y los modos de operación del robot.

4.1. Descripción general

La definición de la arquitectura del sistema fue uno de los primeros pasos que se realizó durante la etapa de diseño. Como en esa etapa el robot estaba aún en proceso de definición, se ideó una estructura que cumpliera la mayor cantidad de las siguientes características:

- Generalizable: el hardware y firmware del sistema de control deben estar lo menos acoplado posible con la mecánica y la disposición particular del robot de este proyecto.
- Independiente del software superior: se debe establecer una interfaz clara entre el hardware y firmware de control, y el software de planificación de trayectoria. Esta interfaz debe abstraer los detalles de la implementación de bajo nivel y del hardware.
- Jerarquizada o estructura en capas: debe haber distintas capas de abstracción y, mientras que la capa inferior se encarga de la interacción con el hardware, la capa superior debe enviar comandos. Las capas intermedias interpretan y adaptan la información para que sea consumida por la capa receptora.
- Online: operación en vivo, sin tiempos muertos.
- Extensible: facilidad para agregar nuevas funciones, sobre todo con el uso de capas superiores. Además el firmware debe ser capaz de ser reutilizado en robots con una cantidad arbitraria de grados de libertad, con cambios mínimos de firmware.
- Un nodo genérico por grado de libertad y un nodo orquestrador: relacionado con el punto anterior, para facilitar la extensibilidad y el uso en otro tipo de robots, se debe utilizar un nodo genérico por grado de libertad. Si se quiere agregar un GDL, simplemente hay que añadir un nodo al bus.

Entonces, se propuso una arquitectura de control jerárquica y distribuida que permitiera cumplir con las características antes mencionadas y facilitara la paralelización del desarrollo, es decir, se puede trabajar simultáneamente en el firmware y el software.

En la Fig. 4.1 se pueden observar los diferentes actores en el control y el flujo de la información. La capa de más bajo nivel es la que transforma energía eléctrica e información en energía mecánica; está integrada por los motores, los drivers y, potencialmente, encoders para realimentación de posición. La capa inmediatamente superior está integrada por los nodos extremo q_i, que envían señales de pulso y dirección en función de lo indicado por el actor de la capa superior. El nodo orquestrador envía consignas y señales

de sincronización a todos los nodos extremo por medio de un bus CAN. A su vez, el orquestrador se comunica con el software de interfaz UART, que, precisamente, envía mediante una interfaz USB-UART los lotes de mensajes que el orquestador debe repartir apropiadamente. Este software de interfaz UART toma la secuencia de comandos de un archivo .csv, que se obtiene como salida de la capa más superior. El software de planificación es utilizado por el usuario, quien elige la trayectoria que debe seguir el efector final del robot. Esta capa implementa los comandos del usuario, genera la trayectoria necesaria en espacio cartesiano, calcula la cinemática inversa, discretiza la curva y guarda las consignas a cada actuador en el archivo .csv que utiliza la capa inferior.

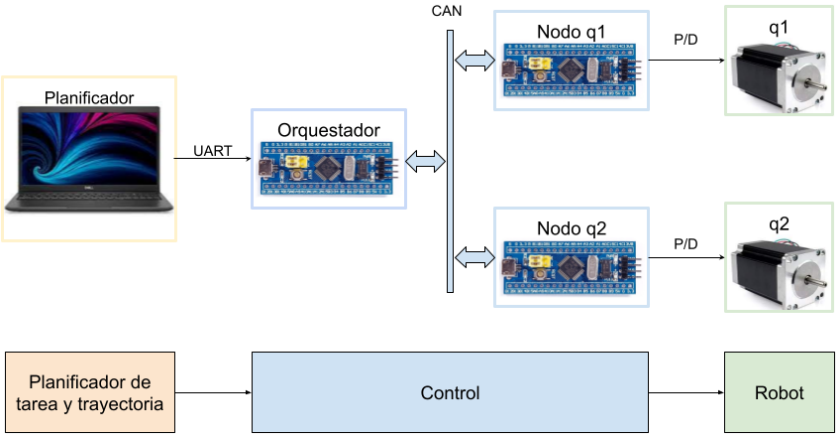


Figura 4.1: Actores en la arquitectura de control.

4.2. Hardware

Dentro de los elementos de hardware que componen el sistema de control podemos distinguir elementos de cómputo, elementos de potencia y elementos de señales y comunicaciones.

Dentro de los elementos de cómputo se encuentra lo siguiente:

- Planificador: computadora principal para cálculo de trayectorias. Es una computadora portátil convencional, corre alguna distribución de Linux[1] y se encarga principalmente de realizar el procesamiento más intensivo del pipeline: cinemática directa e inversa, generación de trayectoria y, eventualmente, modelo dinámico. Conforma la capa superior en la jerarquía del sistema.
- Nodo orquestador: microcontrolador *STM32*, encargado de comunicarse mediante puerto serie con la computadora principal, interpretar, coordinar y enviar las trayectorias recibidas a los nodos extremos por bus CAN.
- Nodo extremo q1: microcontrolador *STM32*, encargado de recibir las consignas enviadas por CAN y enviar las señales correspondientes al driver del motor.
- Nodo extremo q2: ídem pero para el motor de q2.

En la Fig. 4.1 se observan los bloques correspondientes a cada elemento junto con sus señales e interfaces.

Además se cuenta con componentes adicionales que controlan potencia y proveen interfaces de señal y comunicaciones. Estos son:

- Puente USB-UART. Basado en el circuito integrado CP2102 de *Silicon Labs*, permite el intercambio de mensajes entre el nodo orquestador y el planificador de trayectorias utilizando la UART.
- Placa genérica "CAN node". Es una suerte de *carrier board* para la placa comercial *bluepill* que contiene el microcontrolador STM32F103C8T6. Fue realizada utilizando PCB experimentales perforadas[2] y contiene un socket para enchufar la *bluepill*, un transceptor CAN de *Microchip* MCP2561,

[1] En este caso se utilizó Arch Linux y Ubuntu
[2] Queda como deuda técnica el diseño y manufactura de una PCB

un *level shifter* utilizando un integrado SN74HCT32 de compuertas OR, borneras para conectar el cableado y un *jumper* para utilizar o no la resistencia de terminación del bus CAN. Se puede observar una foto de la placa junto con sus componentes en la Fig. 4.2. Esta placa fue rediseñada y ampliada en base a la desarrollada en Avanzini [20].

- Driver de motor paso a paso con interfaz pulso-dirección. Se utilizaron los drivers 67S109.
- Fuente de alimentación de 24V. Para alimentar los drivers de los motores. Idealmente se iba a utilizar una fuente con tensión más alta, pero debido a cuestiones de presupuesto y disponibilidad, se empleó una fuente con ese nivel de tensión.
- Fuente de alimentación de 5V: Para alimentar las placas genéricas "CAN node". Si bien el microcontrolador funciona con 3V3, la *bluepill* cuenta con un regulador dedicado del tipo LDO[3]. El resto de los elementos del sistema funcionan con 5V, por lo que fue más conveniente proveer este riel de tensión.
- Placa de conversión de señal digital de tensión a corriente. Las señales de pulso y dirección salen directamente de un pin del microcontrolador pero las entradas de los drivers están optoacopladas y requieren entre 10 y 20mA y una tensión de 5V para asegurar un funcionamiento correcto. Como ninguna de esas dos condiciones es posible y/o sana para la integridad del microcontrolador se hizo una pequeña placa que utilizando unos transistores NPN realizan la adaptación.

La Fig. 4.3 muestra un diagrama de bloques y señales de todos los elementos de hardware que componen el subsistema de control. En la Fig. 4.4 se observa el detalle de las conexiones entre el nodo orquestador y uno de los nodos extremo. Se omitió el segundo extremo por claridad.

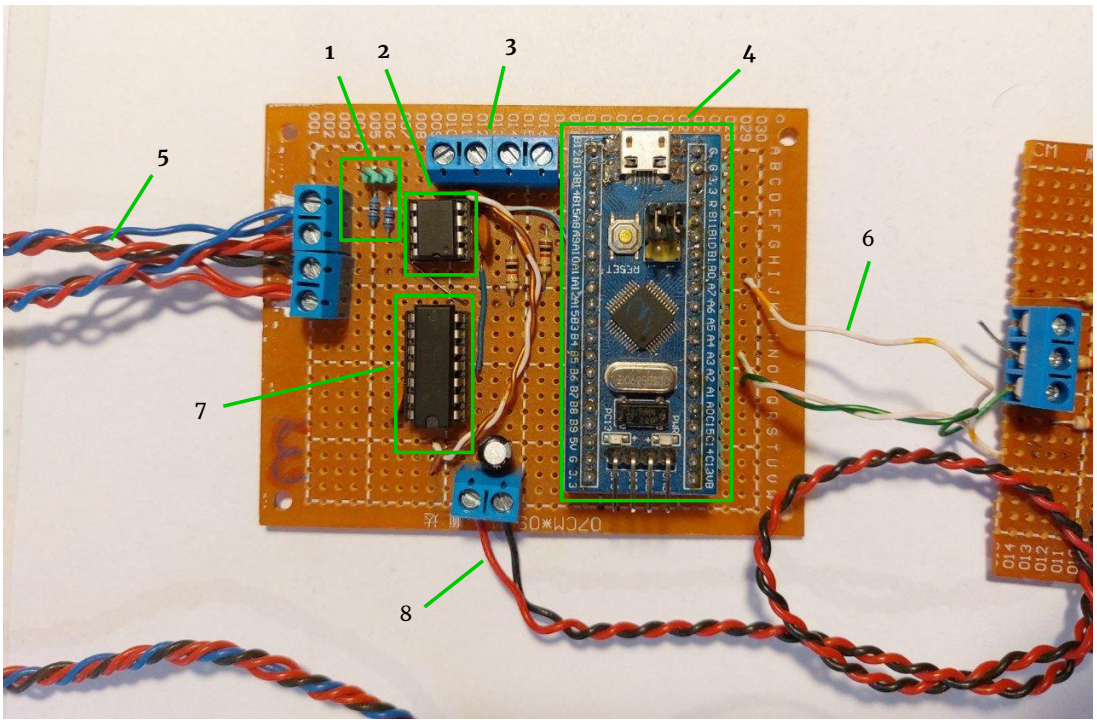


Figura 4.2: Foto de la placa "CAN node" con sus componentes: 1) Jumper para selección de la resistencia de terminación del bus. 2) Transceptor CAN MCP2561. 3) Bornera para futura integración de un encoder incremental. 4) Bluepill. 5) Cableado de alimentación y CAN hacia otras placas. 6) Cableado hacia la placa de conversión tensión-corriente. 7) *Level shifter* 8) Alimentación de 5V para la placa de conversión tensión-corriente.

[3]Low-dropout

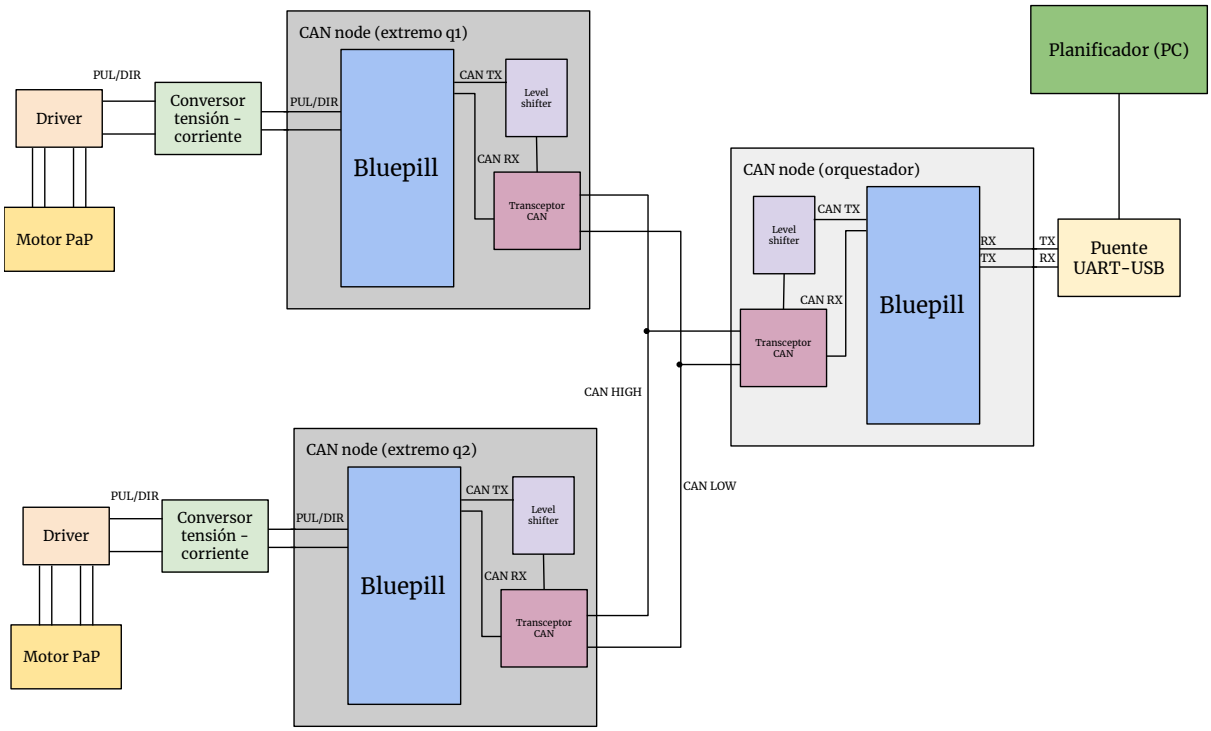


Figura 4.3: Diagrama lógico de los componentes de hardware del sistema. Se omiten las líneas de alimentación.

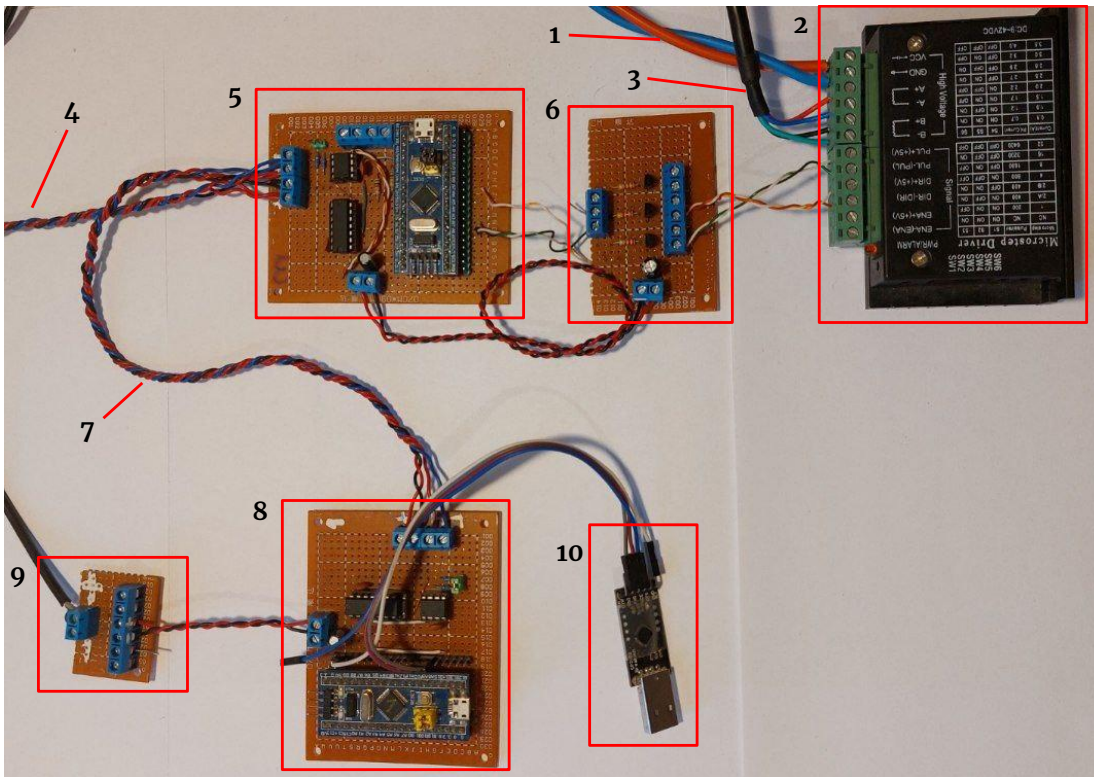


Figura 4.4: Foto de los elementos del sistema de electrónica de control y potencia: 1) Alimentación de 24V para el driver de los motores. 2) Driver de motor paso a paso. 3) Cable del motor. 4) Cable de alimentación y CAN para el nodo extremo q1, fuera de la figura. 5) Nodo extremo q2. 6) Placa de adaptación tensión-corriente. 7) Cable de alimentación y CAN para el nodo orquestador. 8) Nodo orquestador. 9) Bornera de alimentación de 5V. 10) Puente UART-USB.

4.3. Arquitectura

Como se mencionó anteriormente, se diseñó la arquitectura de control teniendo en mente la flexibilidad, la extensibilidad y la separación en capas o jerarquización de las distintas funciones. En la figura 4.5 se observan las distintas capas de la arquitectura del sistema junto con las interfaces entre las mismas.

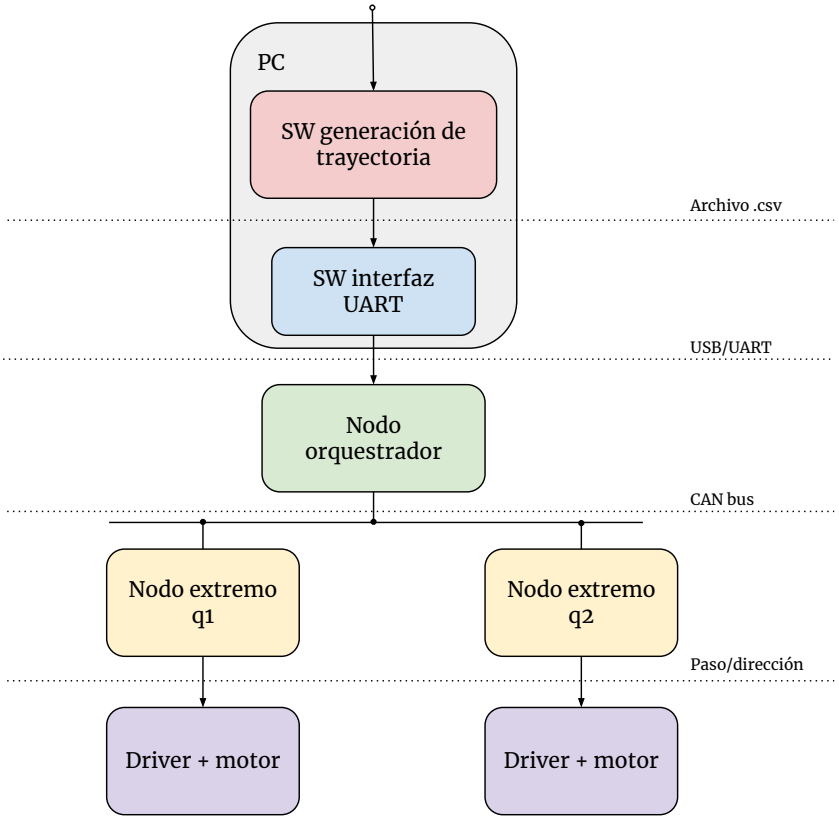


Figura 4.5: Capas y jerarquía de la arquitectura del sistema.

Haciendo una descripción desde el hardware hacia el software o *bottom-up*, lo que interactúa con el robot físico es el conjunto motor+driver. Cada motor mueve un grado de libertad y es comandado por un conjunto de placa "CAN node" más su firmware que denominamos **nodo extremo**. Se decidió tener un microcontrolador por motor con el objetivo de facilitar la interoperabilidad con otros tipos de robots y tener la posibilidad de, con el mismo hardware y un mínimo cambio de firmware[4], reutilizar el desarrollo para N nodos extremo. Es decir, el firmware es totalmente agnóstico al hecho de que el robot sea serie o paralelo, a la cantidad de grados de libertad o a qué dispositivo le envía consignas.

Todos los nodos extremo están conectados mediante un bus CAN en el cual también se encuentra el **nodo orquestador**, que se encarga de dos funciones principales. Primero, interpreta, almacena y redirige las consignas que recibe por puerto serie a través del bus CAN, a cada uno de los nodos correspondientes. Este es el único punto de contacto con la capa superior. La segunda función es la de sincronizar el movimiento de todos los nodos extremos. Como el bus es compartido, para asegurar la ejecución en tiempo real y el movimiento sincronizado se utiliza un mensaje especial de sincronismo. Primero, durante el intervalo entre dos instantes $t = t_0$ y $t = t + \Delta t$ el orquestador envía a todos los nodos la posición en la que se deberán encontrar en un instante $t = t + 2\Delta t$. En $t = t + \Delta t$ se envía el mensaje SYNC, instante en el cual todos los nodos extremo deben comenzar a mover los motores (a la velocidad adecuada) de forma tal de alcanzar en $t = t + 2\Delta t$ la posición deseada. Este modo se describirá en mayor detalle en la sección 4.7.2.3.

Luego, en un nivel superior se encuentra la capa de interfaz pc-microcontrolador, un programa escrito en *Python* cuya función principal es controlar el flujo de datos que se envía a través del puerto serie. El protocolo de comunicación elegido fue una UART, aunque en la comunicación con la computadora

[4]Solamente cambiando la dirección CAN del nodo

principal se usa un puente USB-UART. De esta forma, es posible utilizar como nivel superior de control cualquier dispositivo que pueda ejecutar código *Python* y tenga un puerto serie o bien un puerto USB.

Finalmente, como capa de aplicación, se tiene el planificador de trayectorias. Es un programa escrito en *Python* que genera la secuencia de puntos que deben seguir los actuadores en función del plan de trayectoria solicitado por el usuario. Para la comunicación con la aplicación de comunicación por UART, se utilizó una interfaz utilizando un archivo .csv, que si bien no es muy sofisticada, cumple con su objetivo.

La elección de esta jerarquía en la estructura de control supuso un *tradeoff*. La opción elegida es mejorable pero funcional. La elección de utilizar una computadora de propósito general convencional significó tener que agregar un nodo orquestrador que se encargue del *timing* de los mensajes y la ejecución de los movimientos. Un sistema operativo como Linux no cumple requerimientos de *hard real-time*, por lo que por sí solo no sería la mejor opción. Lo cierto es que hay alternativas, como el parche PREEMPT_RT al kernel de *Linux*, que le da propiedades de tiempo real. De haber ido por esta ruta, se podría haber trabajado con una SBC (*single board computer*) del estilo *Raspberry Pi* o, mejor aún, una *BeagleBone black (BBB)*. Junto con un dispositivo del tipo USB2CAN o utilizando el controlador CAN del microcontrolador integrado en la BBB nos habríamos ahorrado una interfaz, aquella entre la computadora y el nodo orquestrador (UART).

Quizás esta solución habría sido más elegante y en cambio posiblemente se cometió el error de optimizar antes de tiempo [5]. Sin embargo, trabajar con texto plano y una interfaz UART fue muy beneficioso para la velocidad de desarrollo y la corrección de los errores en el código. También es más directa la cadena de procesamiento cuando lo único que se intercambia es un archivo .csv. Por el contrario, el uso de *SocketCAN* en *Linux* habría implicado un mayor esfuerzo de programación debido a su mayor complejidad. Así, la estructura elegida es funcional y más extensible (el programador de alto nivel puede ignorar todos los detalles relacionados al bus CAN, por ejemplo), por lo que consideramos que fue la opción correcta para lograr un MVP[6]. En futuras iteraciones estas decisiones pueden ser revisitadas.

4.4. Firmware

La programación del nodo orquestrador y los nodos secundarios se realizó con el lenguaje de programación C y usando la capa de abstracción de hardware (HAL) del fabricante de los microcontroladores, STMicroelectronics. Además, se utilizó el entorno de desarrollo STM32CubeIDE, que cuenta con la herramienta de configuración Cube con la que se especificaron los parámetros de los periféricos. Para poder utilizar como debugger el ST-LINK/V2 (clon) hubo que recurrir a la versión 1.6 del STM32CubeIDE, ya que STMicroelectronics decidió deprecar de forma lamentable el uso de debuggers no originales en las versiones posteriores del IDE.

En lo que refiere a software embebido, en este proyecto existen dos bases de código: para el nodo orquestrador, y para los nodos extremo (inicialmente llamados primario y secundario, respectivamente). El objetivo fue desarrollar un codebase que sea extensible, flexible y genérico. Para ello se orientó el desarrollo utilizando *event-driven development* y *hierarchical state machines*, explicadas en la sección 4.5. El conjunto de estas dos técnicas permite modularizar y facilitar el desarrollo de software.

Al comenzar el proyecto se planteó la posibilidad de, además, utilizar un sistema operativo de tiempo real (RTOS) tal como *FreeRTOS*. Se decidió no utilizarlo ya que, el software del nodo orquestrador, que consideramos sería el más beneficiado con el uso de un RTOS, era lo suficientemente acotado como para utilizar una solución *bare-metal*. Analizándolo a posteriori, no habría sido mala la inclusión de este tipo de sistema operativo ya que habría permitido una mayor agilidad en la programación. Aún así, la ausencia de un RTOS no fue un impedimento para alcanzar los requerimientos de tiempo real del sistema.

4.5. Marco de desarrollo

Dados los lineamientos nombrados en la sección 4.1 que indicaban que se debía desarrollar el firmware intentando que sea lo más generalizable y extensible posible, se buscó un marco de desarrollo de software embebido acorde a esos criterios. Nos basamos fuertemente en el trabajo de Miro Samek, quien en su libro Samek [21] desarrolla los conceptos de *event-driven development* y *Hierarchical State Machines* (HSM). Ambas técnicas fueron exploradas y adaptadas al desarrollo de este proyecto; las siguientes subsecciones

[5] "The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming". Donald Knuth

[6] *Minimum viable product*

describen cómo. Además del libro mencionado, se obtuvieron recursos del curso de programación moderna de sistemas embebidos del mismo autor, Samek [22].

4.5.1. Event driven development and architecture

La arquitectura y desarrollo basado en eventos o *Event-driven development* por su nombre en inglés es un paradigma de programación en el cual el flujo del programa está determinado por eventos tales como acciones de un usuario, salidas de un sensor o mensajes generados por otros hilos o tareas. Luego de reconocer el evento, el programa reacciona ejecutando un acción que puede ser realizar un cálculo, manipular algún dispositivo de hardware o generar eventos por software que a su vez disparan otros componentes internos de software. Por esta razón los sistemas controlados por eventos también son llamados sistemas reactivos.

Aunque es muy utilizado en aplicaciones como interfaces gráficas de usuario (GUI), también tiene su lugar en sistemas embebidos ya que estos son, por naturaleza, predominantemente reactivos.

El objetivo en la programación de sistemas de este tipo es ejecutar una acción como reacción a un evento. Las acciones están determinadas por dos factores: la naturaleza del evento y el contexto (o estado) actual. Podemos considerar que existen cuatro capas dentro de un sistema con una arquitectura basada en eventos:

- Productor de eventos: actores que generan eventos, como lazos dentro de un estado en una FSM, cambios de hardware que disparan una rutina de servicio de interrupción o un nodo externo que envía un mensaje mediante un canal de comunicaciones.
- Canal de eventos: medio por el cual los eventos son almacenados y transmitidos. En nuestro caso se implementó una cola (FIFO).
- Procesador de eventos (o *event handler*): es una función que se encarga de identificar los eventos y ejecutar la reacción adecuada.
- Acción: es la capa en la que se ejecutan las acciones que fueron disparadas por los eventos.

En una aplicación que utiliza *event-driven development* hay generalmente un lazo principal con un *event handler* que escucha eventos y lanza funciones de *callback* cuando uno de esos eventos es encontrado. En sistemas embebidos también se puede hacer uso de interrupciones para generar eventos y lanzar acciones.

4.5.2. Máquina de estados finitos (FSM) y jerárquicas (HSM)

Una máquina de estados finitos (FSM) es una muy buena forma de implementar un sistema con una arquitectura basada en eventos. Las técnicas más populares para implementar una máquina de estados en el lenguaje C son las siguientes:

- Una estructura utilizando `switch-case`: es una técnica simple pero es difícil de mantener y no escala.
- Una tabla de estados que contiene un arreglo de transiciones por cada estado. Cada celda en la tabla tiene un puntero a una función y el estado siguiente.
- Un diseño orientado a objetos que representa la máquina de estados como un árbol cuyos estados y transiciones son recorridos por un intérprete.

Sin embargo, existe otra técnica que se basa casi enteramente en punteros a funciones. De hecho, el "estado" está implementado directamente como un puntero a función. Esta es la técnica elegida y está fuertemente basada en la arquitectura descripta en Samek [21] y Samek [22]. La ventaja de esta solución respecto a las enumeradas es que en esta solución la máquina de estados no está contenida en una estructura de datos sino que está implementada en el código, tal como sucede en un `switch-case`. Pero a diferencia de esa, el resultado es mucho más limpio y mantenible. Además es muy sencillo pasar de una descripción en UML o un *statechart* al código.

Existe otra ventaja que es la posibilidad de extender las capacidades de las máquinas de estados finitos convencionales agregando jerarquía. Esto se conoce como máquinas de estado jerárquicas o *hierarchical*

state machines (HSM) y se basan en la posibilidad de tener máquinas de estados anidadas con estados internos. De esta forma un sistema puede tener una mayor complejidad agrupando varios estados en un supraestado que describe un comportamiento compartido.

Utilizar esta forma de implementar máquinas de estado, mientras se escriban las funciones adecuadas, tiene el beneficio de que agregar máquinas de estado jerárquicas (internas) y en paralelo es trivial. Sin embargo una de las desventajas es que la implementación actual no toma en consideración la prioridad de varias máquinas de estado en paralelo, por lo que su ejecución es siempre secuencial. Esto se podría solucionar con el uso de un RTOS, como se comentó anteriormente.

A continuación se encuentra un extracto del contenido de un archivo de cabecera (.h) que declara una máquina de estados genérica.

```
/*
 * fsm.h
 */

#ifndef INC_FSM_H_
#define INC_FSM_H_

// Enumeracion de los tipos de eventos
typedef enum Event_enum {
    NO_EVENT = 0,
    // ...
    NEW_CAN_MESSAGE = 100
} Event_enum;

typedef struct Event {
    Event_enum e;
} Event;

// Declaracion del function pointer generico
typedef void (*StateHandler)(Fsm * me, Event * event);

// Objeto maquina de estado. Contiene el function pointer del estado actual
typedef struct Fsm {
    StateHandler State;
} Fsm;

// Declaracion de la cola de eventos
QUEUE_DECLARATION(EventQueue, struct Event, 16);
struct EventQueue EventQueue;

// Protipos de las funciones que implementan cada estado
/* FSM: Operation */
void Operation_Start    (Fsm * me, Event * ev);
// ...
void OpEnable_PP        (Fsm * me, Event * ev);
void OpEnable_IP        (Fsm * me, Event * ev);

#endif /* INC_FSM_H_ */
```

Existe una estructura `Fsm`, que como único miembro contiene un puntero a función con la forma de `StateHandler`. Ese puntero apuntará a la función que implemente el estado actual. A esa función se le pasan como parámetros un puntero al objeto máquina de estado `Fsm` (para que el estado pueda realizar una transición y que el *function pointer* apunte a otro estado) y un puntero al evento que disparó la acción. Además se declara una cola de eventos, donde estos son almacenados. Por último se declaran algunos prototipos de las funciones que implementarán los estados.

Dentro del lazo principal del programa, como se muestra en el extracto siguiente, primero se declaran e inicializan la cola de eventos, el objeto máquina de estados y un objeto de eventos. Dentro del lazo principal se espera un *flag* de sincronización que puede provenir de un *timer* o del *SysTick* y a continuación verifica si la cola de eventos contiene algo. De ser así, desencola el dato, teniendo cuidado de deshabilitar momentáneamente las interrupciones para evitar una *race-condition*[7]. Finalmente se llama a la función

[7]Los servicios de interrupción pueden agregar eventos a la cola. Podría suceder que la ISR intentara encolar un evento mientras el lazo principal está desencolando otro, lo que generaría un comporamiento indefinido y no determinístico. Aunque es poco probable, el hecho de que las operaciones sobre la cola no sean atómicas implica que se debe tener cuidado al acceder y modificar un recurso compartido como lo es la cola de eventos. Es suficiente con desabilitar momentáneamente las interrupciones ya que el flujo del programa no es interrumpible por otras tareas. Si se usara un RTOS esas dos instrucciones se reemplazarían por un semáforo o un MUTEX.

correspondiente al estado actual, que se accede con `Fsm_Operation.State` y se le pasan como parámetros la dirección de memoria del objeto máquina de estado y del último evento encolado, que se guarda en una variable intermedia.

```
/*
 * main.c
 */

#include "fsm.h"

int main(void) {
    // ...
    EventQueue_init(&EventQueue);
    Fsm Fsm_Operation;
    Event event;
    Fsm_Operation.State = Operation_Start;
    event.e = NO_EVENT;
    EventQueue_enqueue(&EventQueue, &event);

    (Fsm_Operation.State)(&Fsm_Operation, &event);
    while (1) {
        if(sync_flag){
            if(!EventQueue_is_empty(&EventQueue)){
                __disable_irq();
                EventQueue_dequeue(&EventQueue, &event);
                __enable_irq();
            } else {
                event.e = NO_EVENT;
            }
            // Ejecucion de la funcion del estado
            (Fsm_Operation.State)(&Fsm_Operation, &event);
            sync_flag = 0;
        }
    }
}
```

La implementación de un estado tomaría una forma similar a la del extracto que se encuentra a continuación. Se cuenta con la posibilidad de ejecutar una acción al entrar al estado, ejecutar una acción siempre que corra la función y actuar en reacción a distintos eventos. Estos eventos pueden generar una reacción interna, disparar un cálculo, generar otro evento, realizar un cambio de estado, etc. Adicionalmente, en el cuerpo de la función y antes de evaluar el valor del evento se puede declarar y correr otra máquina de estados interna con sus propios eventos locales o heredados de la máquina de estados superior. Tanto la posibilidad de correr una FSM interna como la de tener acciones de entrada (y acciones de salida, de ser implementado) se debe al *keyword* `static` que asegura la persistencia en memoria de la variable más allá de cada llamada a la función, pero manteniendo el *scope* local.

```

/*
 * fsm.c
 */

// ...

void OpEnable_PP(Fsm * me, Event * ev){
    // FSM interna
    // static Fsm internalFSM;
    // static Event * localEvent;

    static uint16_t EntryFlag = 1;
    if (EntryFlag == 1){
        // Execute something on first entry ONLY
        // Must be returned to 1 when transitioned to another state
        internalFSM.State = StateFunction;
        EntryFlag = 0;
    }

    switch (ev->e) {
        case NEW_TARGET_POSITION:
            // ...
            break;

```

```
        case //...:
            // ...
            break;

        case OPERATION_MODE_CHANGED:
            // ...
            me->State = AnotherState;
            EntryFlag = 1;
            break;

        default:
        break;
    }

    // Llamado a la FSM interna
    // (internalFSM.State)(&internalFSM, localEvent);
}
```

4.6. Software

Se decidió posteriormente generar una aplicación capaz de generar diversas trayectorias y poder enviarlas al robot o al simulador. Para esto, se utilizó *Python* como lenguaje de programación. Este nos permite generar una aplicación que corra en diferentes sistemas operativos. Además, por su sintaxis cercana al lenguaje humano, se decidió obviar la interfaz de usuario, dejando esta como trabajo futuro.

La aplicación desarrollada contempla los casos de uso mostrados en la Fig. 4.6.

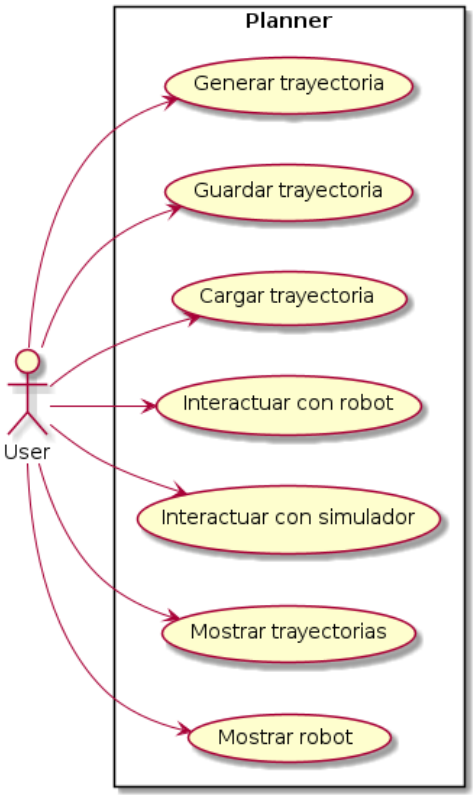


Figura 4.6: Casos de uso del software.

A lo largo del desarrollo se decidió utilizar la librería *Pytest* para *testing* unitario de algunas funciones, *Sphinx* para la documentación automática y *Github* como herramienta de control de versión. Además, se generó un paquete *Python* que se puede instalar con la herramienta *Pip*. Finalmente, se desarrollaron *scripts* de ejemplo para facilitar el uso de la aplicación.

El software se encuentra disponible en su repositorio *Github*: `https://github.com/gonzafernan/five-bar-robot`

El desarrollo del software se dividió en 2 partes descritas a continuación.

4.6.1. Planificador de trayectorias

Esta parte del software se encarga de generar las trayectorias y resolver la cinemática del robot. Para su implementación, se decidió hacer que el programa corra *off-line*, es decir, se descartó el requerimiento de generar una trayectoria en tiempo real. Este solo ofrece una interacción mediante su API y se le facilita al usuario en forma de librería.

En cuanto a sus dependencias, se destacan las siguientes:

- *Matplotlib* 3.4: utilizada para graficar.
- *Numpy* 1.21: utilizada para el uso de matrices y vectores.

Para el desarrollo del planificador, se decidió seguir el paradigma de programación orientada a objetos, lo cual llevó a organizar la librería en las siguientes clases:

- **FiveBar**: en esta clase se encuentra el modelo de nuestro robot. Es decir, la solución de la cinemática directa (CD) e inversa (CI)[8], los parámetros geométricos, los limites articulares del robot, la interferencia mecánica y métodos para graficar el robot y su espacio de trabajo.
- **TimeLaw**: esta clase se implementan las leyes de tiempo trapezoidal y polinómica de grado 5 como se explicó en la sección 1.5.2.1.
- **Path**: en esta clase se implementan los caminos geométricos, estos pueden ser: líneas, círculos, arcos de circunferencia, movimientos relativos al efector final y movimientos en el espacio articular, como se vio en la sección 1.5.2. También provee una función para graficar las trayectorias.

En la Fig. 4.7 se muestra el diseño de clases del planificador de trayectorias.

[8] El método elegido para resolver CD y CI fue el propuesto en Bourbonnais, Bigras y Bonev [1] debido a que no hay necesidad de obtener la posición, velocidad ni aceleración de las articulaciones pasivas.

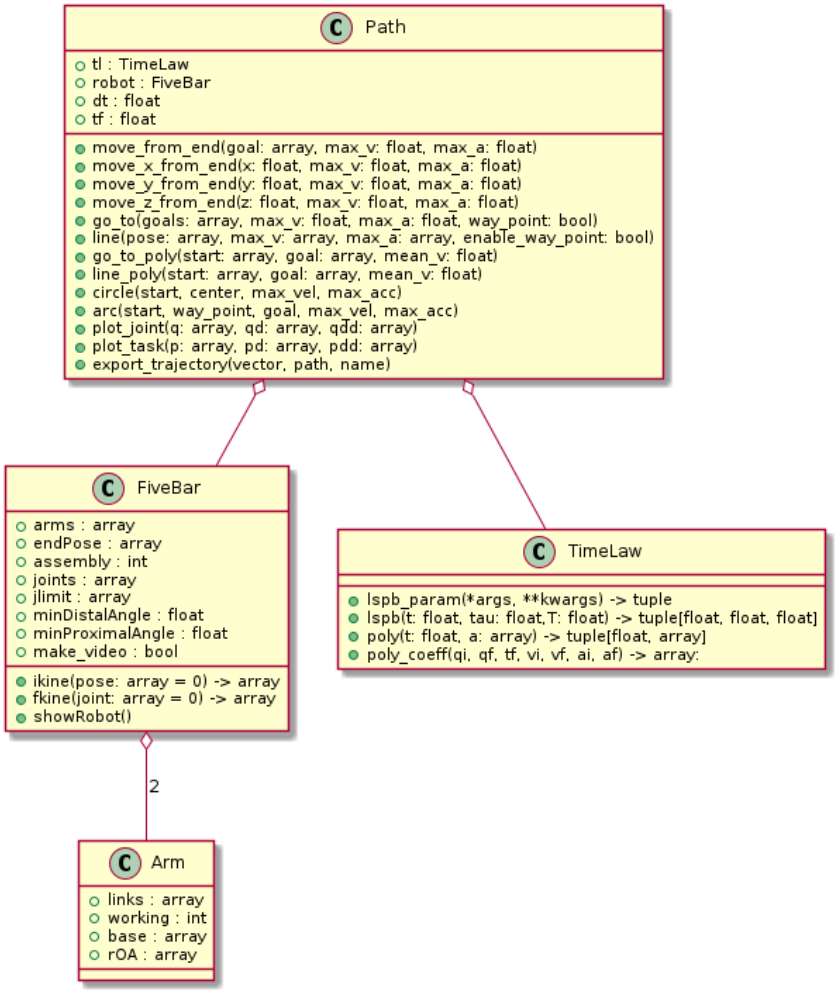


Figura 4.7: Diagrama de clases planificador de trayectorias.

En el anexo F, se puede ver la documentación del planificador en inglés.

4.6.2. Utilidades

Para brindar la capacidad de guardar las trayectorias y comunicarse con el robot y la simulación, se desarrollaron *scripts*. Estos se dividieron en los siguientes archivos:

- utilities.py: en este archivo se encuentran las funciones para normalizar, tratar, exportar y leer trayectorias.
- coppelia-sim-task.py: en este archivo se encuentra la comunicación con la simulación.
- uart-interface.py: este script se utiliza como interfaz entre el planificador (computadora principal) y el nodo orquestrador (microcontrolador). Recibe el archivo .csv con la trayectoria y lo va enviando a medida que haya espacio en el buffer del orquestrador. La interfaz es una comunicación UART con control de flujo por software (XON/XOFF).

En cuanto a sus dependencias, se destacan las siguientes:

- *Pandas* 1.4: lectura y escritura de archivos .csv.
- *Pyserial* 3.5: comunicación con el nodo orquestrador.
- *Pyzmq* 22.3: comunicación con la simulación.

4.6.3. CAN - Aplicación

En la aplicación final, se utilizaron solo algunas de las características que presenta CANopen y que fueron descriptas anteriormente. El nodo extremo es el dispositivo que realiza el control de movimiento. Como tal, se planteó una máquina de estados basada en la que es estándar para controladores de movimiento, que se observa en la Fig. 1.19, pero con 3 estados únicamente. Comienza con un estado STARTED y dentro del propio flujo de programa pasa a READY. Este estado es simil al SWITCHED_ON que se observa en el gráfico. Luego, el nodo orquestrador realiza un cambio en la palabra de control (según DS402) mediante un RPDO y el nodo secundario para a modo OPERATIONAL_ENABLE. Esto se realiza automáticamente y sin la intervención del usuario.

Para poder enviar consignas de posición se debe pasar a modo PP. Para esto se envía nuevamente un RPDO2, pero con el objeto 6060h igual a 1 (modo PP). Esto lo realiza el usuario con un mensaje UART específico. En este momento es posible enviar consignas al eje. Se fijan nuevos setpoints con el comando `:S`, teniendo en cuenta que las consignas están en pasos del motor PaP y que indican posiciones absolutas. Esto se envía al nodo secundario mediante el RPDO3, el cual envía la palabra de control (6040h) y el objeto target position (607Ah). Para encontrar una descripción extensiva de las entradas del diccionario de objetos correspondiente al perfil de dispositivo CiA402 se puede acudir a Elmo Motion Control [14].

El sistema de mensajes empleado tiene ciertas cosas en común con CANopen y el DS402 pero no es CANopen compliant ni intenta serlo.

4.6.4. UART - Control de Flujo

El otro protocolo de comunicación utilizado es la UART (universal asynchronous receiver-transmitter), que se utilizó para establecer el enlace entre la computadora y el nodo orquestador.

La configuración utilizada fue de 921600 baudios, 8 bits por palabra, sin bit de paridad, 1 bit de stop y control de flujo por software. La conexión con la computadora fue por USB, utlizando un puente USB-UART que utiliza el integrado CP2102. En cuanto al software utilizado, se utilizó la librería de *Python Pyserial*. El microcontrolador del nodo orquestador se configuró para que la recepción de datos por la UART sea por DMA (direct access memory) con el objetivo de aliviar la carga del procesador. Por último, se implementó la técnica de control de flujo por software, que será explicada en detalle más adelante.

Trama UART Se diseñó una trama para enviar los comandos desde la computadora principal a los nodos embebidos. Estos comandos pueden ser de cambio de modo de trabajo, de envío de consignas en modo PP, de consignas en modo IP y de cambio de velocidad máxima para el modo PP. Si bien esta lista de comandos es acotada, es suficiente para la realización de pruebas que demuestren la funcionalidad del robot. Nuevos comandos pueden ser agregados con facilidad.

La trama estándar es ASCII y está compuesta por $2+(GDL-1)$ campos, el primero es el identificador del comando y los siguientes son campos de datos, uno por cada grado de libertad que se desea controlar. Además, todas las tramas comienzan con un caracter `:` y finalizan con los caracteres de retorno de carro y nueva línea `\r\n`. De esa forma, los comandos tienen la siguiente forma:

```
:X,YYYYYYYY,ZZZZZZZZ\r\n
```

Donde `X` es el identificador de comando y los dos campos siguientes corresponden al *payload*, uno para cada nodo extremo. Los identificadores implementados se observan en la tabla 4.1.

En este ejemplo el campo de datos contiene dos miembros, `YYYYYYYY` y `ZZZZZZZZ`. Estos corresponden a cada uno de los grados de libertad que hay en el sistema. La trama puede extenderse tanto como sea necesario para enviar consignas a múltiples nodos. Consecuentemente la longitud de la trama está predeterminada y esta es función de la cantidad de grados de libertad, de forma que $l_{trama} = 4+9*GDL$.

Cada miembro del campo de datos tiene 8 caracteres ASCII, correspondientes a la codificación hexadecimal de números enteros con signo de 32 bits `int32_t`. Esto se eligió de esa forma por dos razones. Primero, debido a que el método de control de flujo que se explicará en la siguiente subsección utiliza símbolos determinados (`0x11` y `0x13`) y estos pueden aparecer en cualquier instante de la comunicación, la trama de mensajes no puede ser binaria. Por ende se implementó una trama compuesta con caracteres ASCII alfanuméricos, y, para economizar caracteres, los campos numéricos están codificados en hexadecimal (base 16) (MSB first). La segunda razón para esta elección es que a utilización de longitudes de trama

Identificador	Significado	Campo de datos	Ejemplo
M	Cambio de modo de trabajo	Modo de trabajo para cada nodo, siguiendo la convención de la entrada ModesOfOperation (6060h)del OD `int32_t`	:M,00000007, 00000007\r\n
P	Nueva consigna en modo de posicionamiento por perfil (PP)	Consigna de posición absoluta a cada nodo, en incrementos del motor. Implementa la entrada NewTargetPosition (607Ah) del OD. `int32_t`	:P,00000AA8,FFFFF060\r\n
S	Nueva punto de interpolación en el modo de posición interpolada (IP)	Consigna de posición interpolada a cada nodo, en incrementos del motor. Implementa la entrada InterpolationDataRecord del OD `int32_t`	:S,FFFFFFBA0,00000D4A\r\n
V	Seteo de velocidad máxima para el modo de posicionamiento por perfil	Valor de velocidad máxima en el perfil de velocidad del modo PP. Se expresa en pulsos por segundo. `int32_t`	:V,000005DC,00000321\r\n

Cuadro 4.1: Tipos de identificadores en la trama UART

fijas facilita el procesamiento en el nodo orquestador, que utiliza el periférico DMA para la recepción por UART. Además la gran mayoría de las entradas del diccionario de objetos son enteros de 32 bits y las que no, pueden ser fácilmente convertidas a ese tipo de dato. Esta combinación de características es un buen balance entre longitud de la trama, esfuerzo de implementación y flexibilidad para futuras *features*.

Por ejemplo, una trama de posición interpolada con consignas de posición absoluta de $q1 = -22$ y $q2 = 2361$ se vería de esta forma:

```
:S,FFFFFFEA,00000939\r\n
```

Teniendo en cuanta que los valores `0xFFFFFFEA` y `0x00000939` son la representación en hexadecimal de los valores de la consigna, utilizando números enteros con signo de 32 bits (`int32_t`).

Control de Flujo La comunicación se entabla entre dos nodos: un emisor "rápido" (computadora) y un receptor "lento" (microcontrolador). Como la computadora puede enviar datos mucho más rápido que lo que el microcontrolador puede procesar y enviar por CAN, y como el nodo "lento" tiene un buffer limitado, se utiliza una técnica llamada control de flujo. Es un mecanismo que le permite al dispositivo receptor lento controlar la tasa de transmisión, de forma que la comunicación se realice sin sobrecargar con datos del nodo transmisor.

En esta implementación se utilizó el control de flujo por software, también conocido como control XON/XOFF. A diferencia del control de flujo por hardware, en que se utilizan pines y cables dedicados, el control de flujo por software utiliza caracteres determinados para indicar que la tranmisión debe pausarse o reanudarse. La comunicación es iniciada por la computadora, que envía periódicamente una trama de datos al nodo al orquestador. Este los almacena en un buffer de entrada hasta que está casi lleno[9], cuando envía un caracter XOFF (`0x13`). La computadora lo recibe y deja de transmitir. El buffer del nodo orquestrador se vacía a medida que envía los comandos a los nodos secundarios por la red CAN. Una vez que ese buffer llega a un límite inferior, por ejemplo el 20 % de capacidad, se emite el caracter XON (`0x11`), por lo que la computadora inicia nuevamente la transmisión. Este proceso puede continuar indefinidamente.

4.7. Funcionamiento

Se implementaron dos modos de operación: modo de posicionamiento por perfil (PP) y modo de posición interpolada (IP). En el primero se envía una posición objetivo que cada uno de los nodos alcanzará

[9]No se espera a llenar completamente el buffer del microcontrolador para dejar espacio por si todavía tienen que llegar paquetes que están en "camino". Esto puede suceder porque la comunicación no se entabla puramente por un puerto serie/UART, sino que existen búfferes intermedios en el sistema operativo y en el conversor USB-UART.

siguiendo un perfil de velocidad trapezoidal a partir de la posición inicial. Este modo no requiere de ningún cálculo ya que el procesamiento es efectuado en los nodos secundarios. Por el contrario, en el modo de posición interpolada los puntos de la trayectoria son precalculados (o calculados online) por el software en la PC y enviados al nodo orquestrador. A su vez, este último envía periódicamente las consignas que deben cumplir los nodos secundarios en el siguiente instante de tiempo junto con un mensaje de sincronización.

4.7.1. Nodo orquestador

Como se describió anteriormente, el nodo primario u orquestador es el encargado de comunicarse con el software de planificación de trayectoria, solicitar el setpoint o la secuencia de puntos que deben cumplir los nodos secundarios y asegurar la sincronización y el cumplimiento en términos de tiempo.

Aunque inicialmente iba a contar con más funcionalidades, se decidió por motivos de tiempo que el nodo orquestrador no realice verificaciones acerca del estado de los nodos extremo. Por esa razón toda la responsabilidad de conocer el estado de los nodos extremo y la validación de los movimientos recae sobre el operador y el software de generación de trayectoria.

Comunicación con PC Tanto en el funcionamiento en modo de posicionamiento por perfil como en el de posición interpolada, el nodo se comunica con la computadora principal mediante una interfaz USB/UART. Esto fue elegido debido a la simplicidad y fácil disponibilidad del hardware necesario. En este caso se utilizó con éxito un puente USB/UART con el integrado CP2102. Queda como trabajo futuro la utilización de otras interfaces que permitan la comunicación entre la PC y el microcontrolador. Se configuró la UART con un baudrate de 921600 baudios, 8 bits por palabra, sin bit de paridad y 1 bit de stop.

Del lado del microcontrolador, se utilizaron las funciones de recepción UART por DMA (direct memory access), permitiendo bajar la carga del procesador del microcontrolador. Esto es particularmente útil para el modo de posición interpolada, en el cual la PC continuamente envía comandos que el orquestador almacena en una FIFO[10] hasta que es llenada. Esta FIFO se va vaciando a medida que se envían los comandos a los nodos extremos. Como se comentó en la sección 4.6.4, el proceso de vaciado y llenado de la FIFO se realiza utilizando control de flujo por software; el nodo orquestrador, utilizando caracteres especiales, indica si puede seguir recibiendo consignas o si la PC debe detener la transmisión.

Sincronización de movimiento Al mismo tiempo que se llena la FIFO por UART, el orquestrador comienza a enviar periódicamente los comandos a los nodos extremo. Primero se envía la primer consigna de la secuencia a los nodos extremos, y a continuación se genera una base de tiempo de sincronización con un periodo que, en nuestra aplicación, es de 10ms. Esa es la frecuencia a la cual se envían los mensajes de SYNC, que indican que todos los nodos colgados al bus deben comenzar el movimiento. En los intervalos entre dos mensajes de sincronización se deben enviar los nuevos grupos de consignas a todos los nodos extremos, con la ventaja de que no tienen requerimientos de tiempo y pueden ser enviados en cualquier orden. Lo único importante es que los nodos extremo hayan recibido la consigna antes del próximo SYNC.

En la Fig. 4.8 se puede observar el comportamiento descrito. El símbolo en la columna central representa el estado de la FIFO: en verde significa que está vacía y lista para recibir comandos; en rojo llena. Los mensajes de control de flujo XON/XOFF son enviados cuando puede o no puede recibir comandos, respectivamente. Las mensajes de la derecha se envían por el bus CAN; en rojo los mensajes de SYNC. Por simplificación solo se muestra un mensaje de consignas CAN, pero en la realidad se envía cada una a continuación de la otra. El mensaje SYNC es recibido por todos los nodos a la vez.

4.7.2. Nodos extremo

Los nodos extremo son aquellos encargados de recibir los comandos y consignas desde el orquestrador, realizar los cálculos necesarios y enviar las señales de paso y dirección al driver.

4.7.2.1. Máquina de estados

Dado que el nodo extremo es el más complejo, se decidió implementar un conjunto de máquinas de estado jerárquicas con la idea de facilitar el diseño conceptual del sistema y, sobre todo, su implementación

[10] *First input, first output*

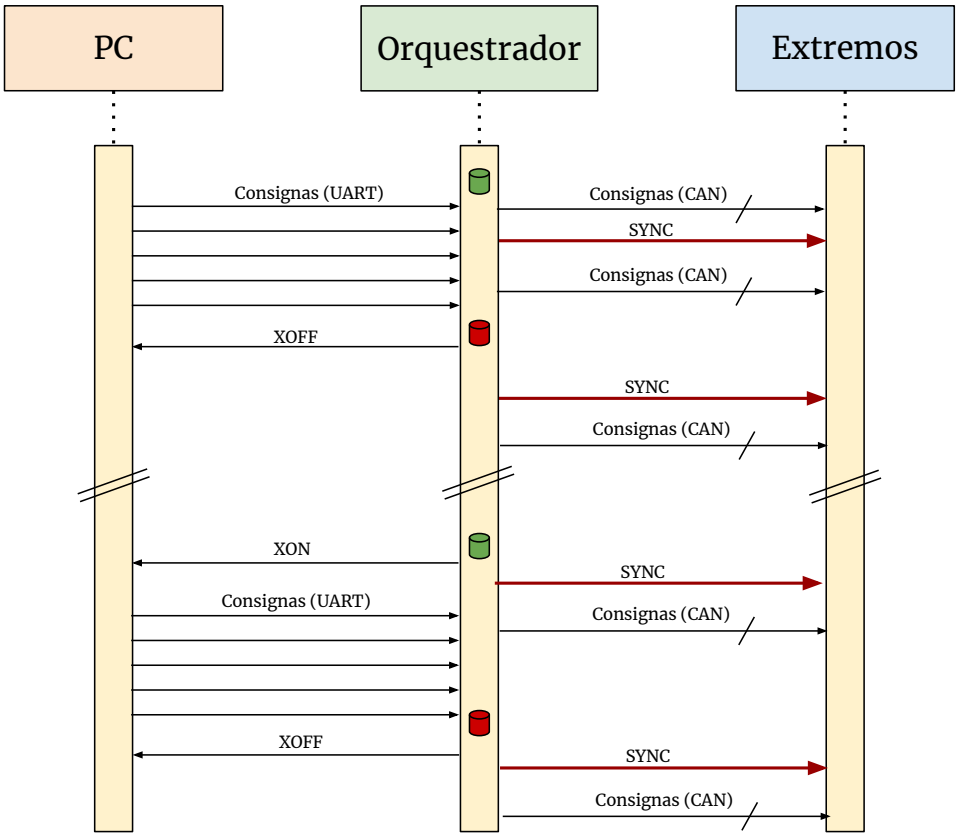


Figura 4.8: Diagrama temporal indicando la secuencia de envío de consignas por UART y el envío de consignas sincronizadas a los extremos.

en C.

De la máquina de estados CANopen correspondiente al perfil de dispositivo DS402 se implementaron los estados Operation ready, start, enable y disable. Operation ready es el modo con el que comienza la máquina de estado, donde automáticamente se establecen algunas variables del proceso y se indica la transición a Operation Start. El nodo espera en este estado hasta recibir un cambio de modo desde el orquestrador, quien a su vez recibió esa orden desde el usuario operando el software de planificación.

Cada uno de los modos de posicionamiento tiene su submáquina de estados propia, como se verá en las siguientes secciones. El diagrama de estados completo se observa en la Fig. 4.9.

4.7.2.2. Modo de posicionamiento por perfil (PP)

El modo de posicionamiento por perfil es el más simple de los dos modos que se implementaron. Se le envía a cada uno de los nodos secundarios una posición final objetivo, en espacio articular. No se realiza movimiento alguno hasta que se recibe un mensaje de SYNC desde el nodo orquestador; momento en el cual comienza el movimiento. Desde la posición inicial, se realiza una trayectoria con un perfil de velocidad trapezoidal con las esquinas recortadas, como se observa en la Fig. 4.10. Esto significa que los motores comienzan y terminan el movimiento con una velocidad no-nula. La razón de ser es que los motores paso a paso presentan un punto de resonancia a bajas velocidades, lo que genera vibraciones y alguna inestabilidad al pasar por ese rango. Dado que el torque a bajas velocidades es relativamente alto, es posible comenzar y terminar el movimiento abruptamente con un cierto valor de velocidad. De esta forma la rampa comienza y termina en ese valor no nulo y se saltea la zona problemática. Este valor de velocidad inicial es configurable, por lo que es posible obtener un perfil puramente trapezoidal simplemente estableciendo la velocidad mínima en 0.

Este modo de posicionamiento tiene tres fases, como se puede observar en la Fig. 4.10.

I. Aceleración constante

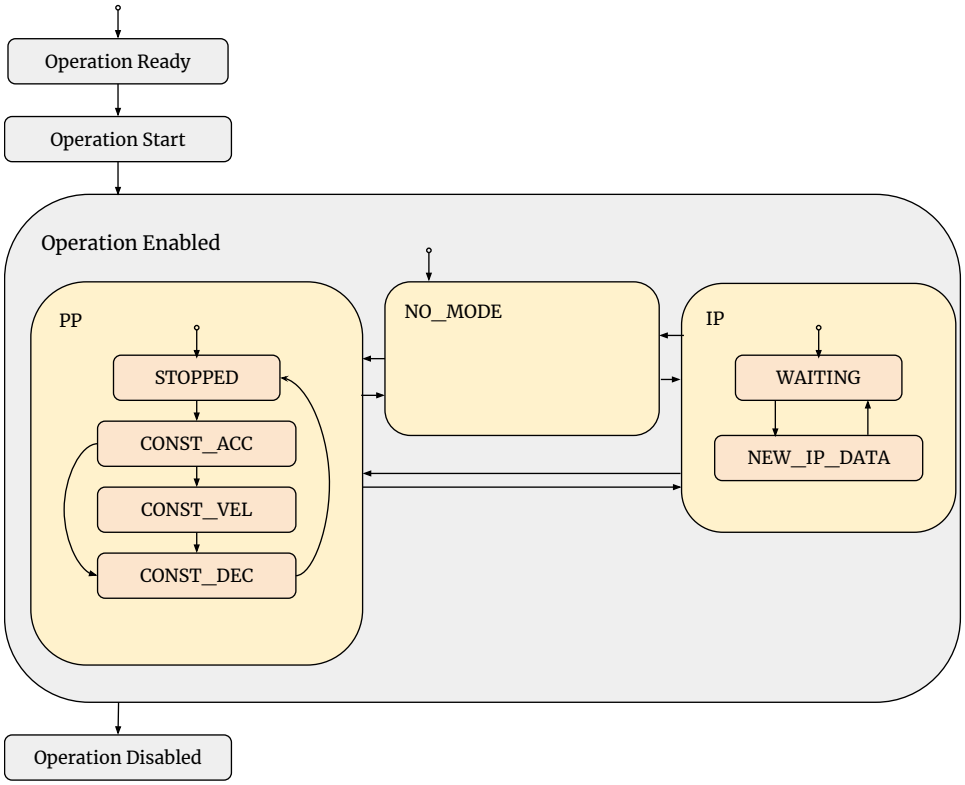


Figura 4.9: Diagrama de estados de un nodo extremo.

II. Velocidad constante

III. Desaceleración constante

En el caso en el que la distancia entre la posición inicial y final no sea suficiente para acelerar y alcanzar la velocidad máxima, se pasa directamente de la fase I a la III. El perfil se vuelve triangular, también con las puntas recortadas, tal como se observa en el perfil de la derecha en la Fig. 4.10.

Un aspecto negativo a destacar es que en este modo los nodos secundarios son ignorantes del estado de los otros nodos y sus posiciones. Esto es particularmente problemático al realizar movimientos punto a punto de forma independiente en cada motor en robots paralelos, ya que, si bien las posiciones iniciales y finales pueden ser cinemáticamente válidas, no se puede asegurar que durante la ejecución del movimiento no se pase por o cerca de un punto singular. Esto lleva a la conclusión de que este tipo de modo no puede ser utilizado a la ligera en robots paralelos. Para asegurar que el movimiento será ejecutado sin problemas es conveniente simular la trayectoria o, al menos, hacer un análisis del movimiento antes de enviar el comando. Esta acotación no es necesaria en robots serie ya que el movimiento en espacio articular es libre y no existe acoplamiento. Al no cruzar por singularidades, el modo PP se puede utilizar sin restricciones cinemáticas en robots serie y solo se deben tener en cuenta los límites articulares, posibles obstáculos en el espacio de trabajo, etc.

Este modo fue pensado para ser utilizado como posicionamiento inicial o para movimientos punto a punto que no requieren sincronización ni seguimiento de trayectoria. Además no se puede asegurar el cumplimiento temporal de las consignas. Por otro lado, el perfil de velocidad generado es trapezoidal, lo que puede generar vibraciones y cierta tosquedad en el movimiento; aunque menores que las que generaría un perfil rectangular. Si esto no es aceptable entonces se debe utilizar el modo IP, que permite realizar perfiles de velocidad arbitrarios.

Los valores de aceleración a_{acc}, desaceleración a_{dec}, velocidad máxima v_{max} y velocidad de inicio y frenada v_{ini} tienen la posibilidad de ser configurables. En la figura 4.11 se observan estos parámetros. Se debe tener en cuenta que $tan(\alpha) = a_{acc}$ y $tan(\beta) = a_{dec}$. Finalmente, el área bajo la curva, pintada de azul, es igual a la distancia recorrida entre la posición inicial y la posición objetivo.

Por razones de tiempo solo se implementó la posibilidad de cambiar el valor de velocidad máxima desde el software de PC. Ya que este parámetro es configurable (pero constante durante la ejecución

del movimiento), esta flexibilidad permite realizar movimientos punto a punto a máxima velocidad, a velocidad reducida, isócronos, etc. Esto requiere esfuerzo adicional de programación y no fue explorado en profundidad por lo que queda como trabajo futuro.

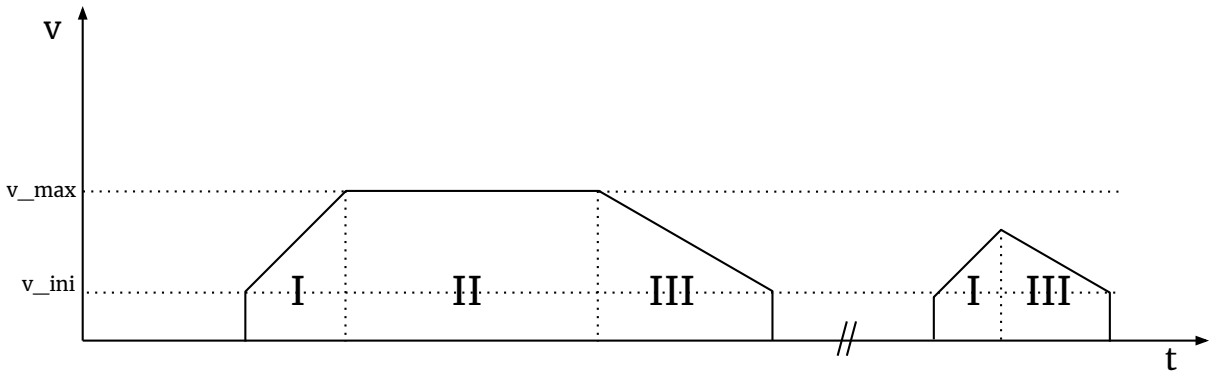


Figura 4.10: Perfil de velocidad generado por el modo PP.

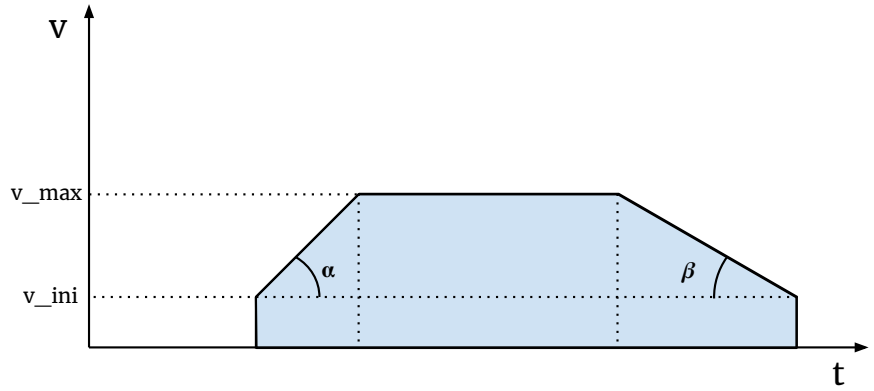


Figura 4.11: Perfil de velocidad típico con los distintos parámetros configurables del modo de posicionamiento por perfil: velocidad máxima, velocidad inicial o mínima y las pendientes de aceleración.

En la Fig. 4.9 se pudo observar la submáquina de estados del modo PP. Se implementa un estado por cada fase del movimiento, explicadas anteriormente, junto con el estado detenido. La primera transición, de STOPPED a CONST_ACC se produce al recibir una nueva consigna. Al entrar a ese modo se calculan los parámetros de la trayectoria: velocidades, aceleraciones y en qué puntos de la trayectoria se debe hacer el cambio de modo. Estos puntos de cambio de modo se denominan milestones, y cuando se alcanzan se genera un evento que dispara la transición entre las fases. Esto se repite en los estados de CONST_VEL Y CONST_DEC hasta llegar a la posición final deseada, momento en el cual se produce la transición hasta STOPPED.

4.7.2.3. Modo de posición interpolada (IP)

El modo de posición interpolada es el modo de posicionamiento más flexible, pero a diferencia del modo PP, donde el poco procesamiento que se debe hacer recae sobre los nodos extremo, en el modo IP se debe utilizar el software de planificación de trayectoria en la PC.

Este modo permite enviarle a cada uno de los nodos secundarios distintos perfiles arbitrarios de posición, sincronizados entre sí. Primero se deben calcular las trayectorias en espacio articular requeridas para cada eje, en base al movimiento deseado. También con el uso del software de planificación se discretiza la trayectoria en puntos cada $10ms$ y se convierten las consignas, de posición angular de la articulación, a posición en incrementos (pasos) del motor. Es importante destacar que cada nodo extremo es ciego de

las consignas que se le envían a los otros. Por lo tanto es responsabilidad del software de planificación generar trayectorias consistentes, válidas y libre de singularidades.

Una vez que se tiene la secuencia de puntos que debe seguir cada nodo extremo, estos son enviados por UART al nodo orquestrador, tal como se explicó anteriormente en la sección 4.6.4. Este último envía cada $10ms$ la consigna correspondiente a cada motor.

En la Fig. 4.12 se observa en negro la curva analítica ideal, muestreada en intervalos de Δt, que consignan los distintos puntos objetivo. La línea roja punteada muestra la interpolación lineal (a velocidad constante), entre los distintos puntos objetivo. Siguiendo la misma figura, la secuencia de pasos es la siguiente:

1. En un tiempo $t = t_0$, el nodo extremo se encuentra estático, y no cambiará su posición, al menos hasta $t = t_0 + \Delta t$.

2. En el intervalo entre $t = t_0$ y $t = t_0 + \Delta t$ recibe una consigna por parte del nodo orquestrador. Esta consigna le indica en qué posición se deberá encontrar en el instante $t = t_0 + 2\Delta t$. Dado que la posición en $t = t_0 + \Delta t$ es conocida y sabiendo que la posición consigna en $t = t_0 + 2\Delta t$ y $\Delta t = 10ms$, se interpola la trayectoria entre los dos puntos utilizando una velocidad constante. Esto es, una interpolación de primer orden.

$$v_{t_0+\Delta t \to t_0+2\Delta t} = \frac{q_{t_0+2\Delta t} - q_{t_0+\Delta t}}{\Delta t} \tag{4.1}$$

3. En el instante $t = t_0 + \Delta t$ el nodo orquestrador emite[11] el mensaje de sincronización SYNC. Todos los nodos lo reciben a la vez y en ese preciso instante cambian su velocidad; de la que tenían en el intervalo anterior, a la calculada con la ecuación 4.1.

4. En el intervalo siguiente, el orquestrador enviará la posición en la que se deberá encontrar el nodo en $t = t_0 + 3\Delta t$. Ya que conoce en qué posición estará en $t = t_0 + 2\Delta t$, puede calcular la velocidad para ese intervalo

5. Este proceso se repite indefinidamente hasta que el orquestrador vacía su buffer y deja de enviar consignas.

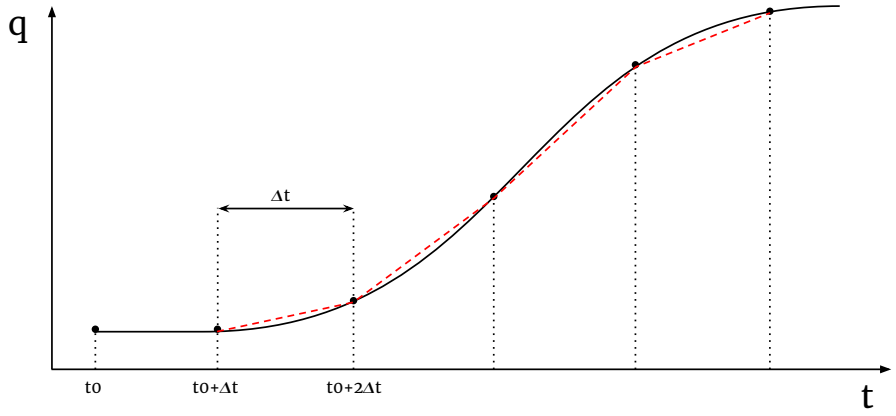


Figura 4.12: Perfil de posición ideal de una trayectoria (en negro) junto con la interpolación lineal (en línea roja punteada).

[11] En forma de *broadcast*

Capítulo 5

Experimentación

El desarrollo de este trabajo concluyó con varias pruebas que se hicieron sobre el robot, las cuales serán presentadas en este capítulo. Dichas pruebas fueron en torno a dos tipos de trayectorias cerradas: líneas rectas y círculos. En la sección 5.1 se presentan los movimientos en línea recta. Más adelante, en la sección 5.2 se presentan los movimientos circulares. Finalmente, en la sección 5.3 se detallan algunas conclusiones de la experimentación.

Para correr distintos ejemplos en el robot se puede seguir el procedimiento indicado en el Apéndice E.

5.1. Línea recta

Para probar los movimientos en línea recta se decidió hacerlo mediante una trayectoria cerrada, esto nos permite repetir la trayectoria una cierta cantidad de veces para poder observar la repetibilidad de nuestro robot.

5.1.1. Definición de la trayectoria

La trayectoria elegida fue un cuadrado[1] y se muestra en la Fig. 5.1. Empieza y termina en el punto $\begin{bmatrix}0 & 0{,}37 & 0\end{bmatrix} m$, pasa por el punto 1 $\begin{bmatrix}0{,}1 & 0{,}37 & 0\end{bmatrix} m$, punto 2 $\begin{bmatrix}0{,}1 & 0{,}47 & 0\end{bmatrix} m$ y punto 3 $\begin{bmatrix}0. & 0{,}47 & 0\end{bmatrix} m$. Esta trayectoria fue generada usando un perfil de velocidad trapezoidal, una velocidad y aceleración máxima de $0{,}3\frac{m}{s}$ y $1\frac{m}{s^2}$ respectivamente (ver Fig. 5.2 y 5.3[2]). La misma se encuentra en el archivo *line.py* ubicado en la carpeta *five-bar-robot/src/motion_planning/examples* del repositorio y se generó utilizando el método *line* utilizando el argumento *enable_way_point=False* (interpolación lineal en el espacio cartesiano, ver apéndice F).

[1] El origen de coordenadas coincide con el origen del robot donde se encuentran las dos articulaciones activas.

[2] Esta no presenta simetría ya que el cuadrado no es simétrico respecto al eje y, eso quiere decir que las dos articulaciones tiene recorridos distintos.

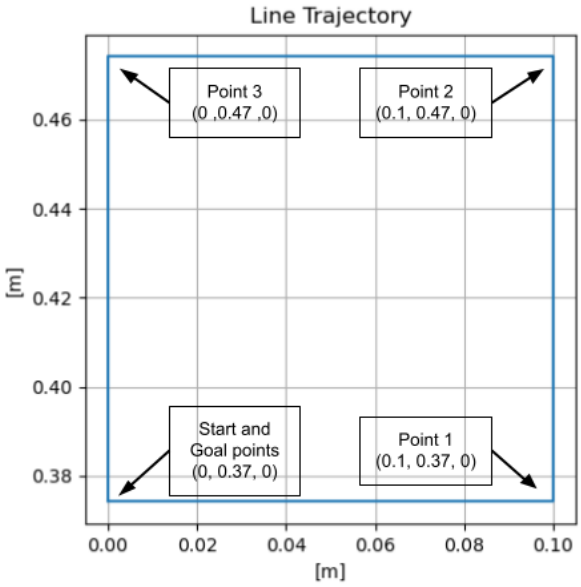


Figura 5.1: Trayectoria cuadrada. Origen de robot coincidente con el origen de coordenadas.

Para verificar la trayectoria de nuestro robot, y debido a que el mismo no cuenta con sensores, se decidió colocar un fibrón en el efector final para poder dibujar en un papel.

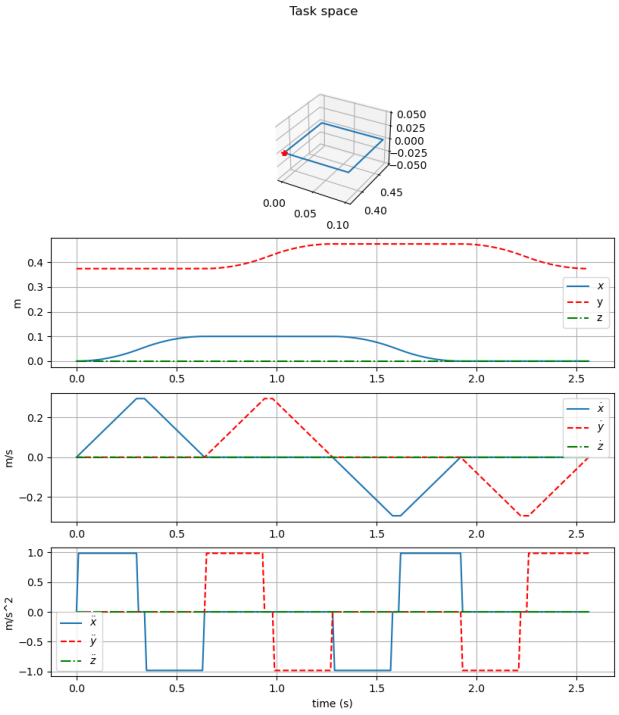


Figura 5.2: Trayectoria cuadrada, espacio de tarea.

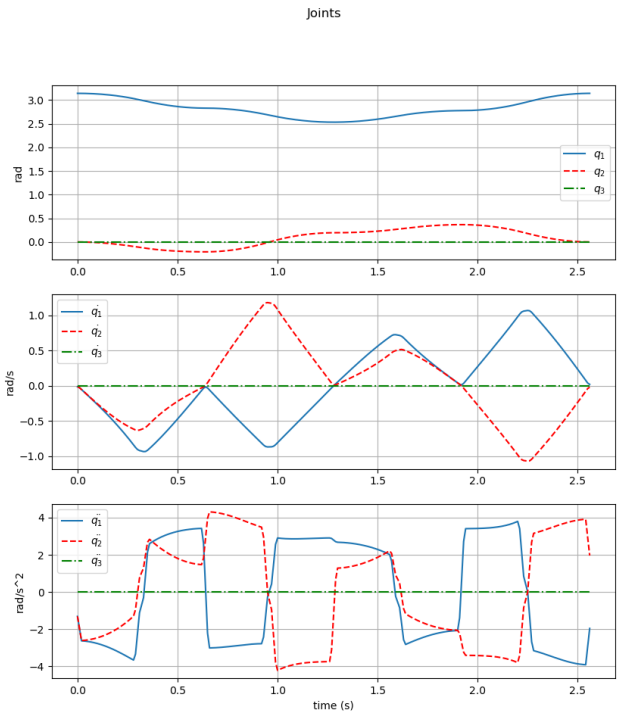


Figura 5.3: Trayectoria cuadrada, espacio articular

5.1.2. Resultados

La trayectoria se ejecutó una vez con los drivers de los motores en micropaso por cuatro.

En la Fig. 5.4 se muestra la trayectoria realizada por el efector final del robot. En esta el eje x es representado por la flecha roja y el y por la flecha verde. Se puede observar que hay una diferencia entre alturas de 0,3cm (eje y) y de anchos de 0,1cm (eje x). A pesar de estos desperfectos, se puede observar también, un cierto paralelismo entre líneas y una pequeña fracción recta en cada línea.

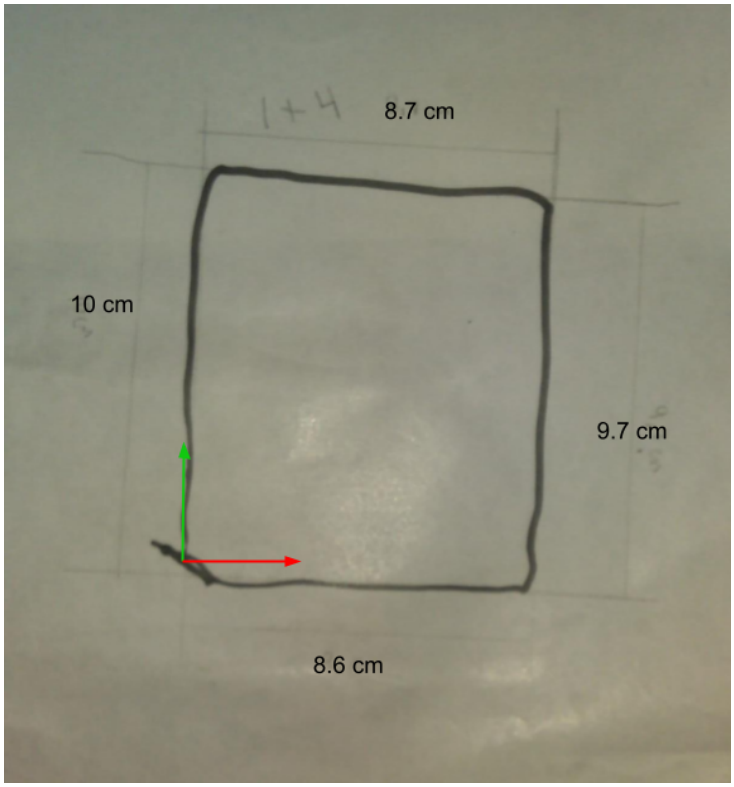


Figura 5.4: Trayectoria cuadrada (micropaso por cuatro y repetida una veces).

Luego, para evaluar la repetibilidad del robot, se realizó la misma trayectoria cinco y diez veces.

En la Fig. 5.5 se muestra la trayectoria realizada cinco veces con los drivers de los motores en micropaso por cuatro. En esta imagen se puede observar que el robot se desplazó a la derecha, esto ocurrió en el segundo ciclo de la trayectoria. La línea que se observa en la esquina inferior derecha fue provocada al llevar el robot al punto de inicio de la trayectoria.

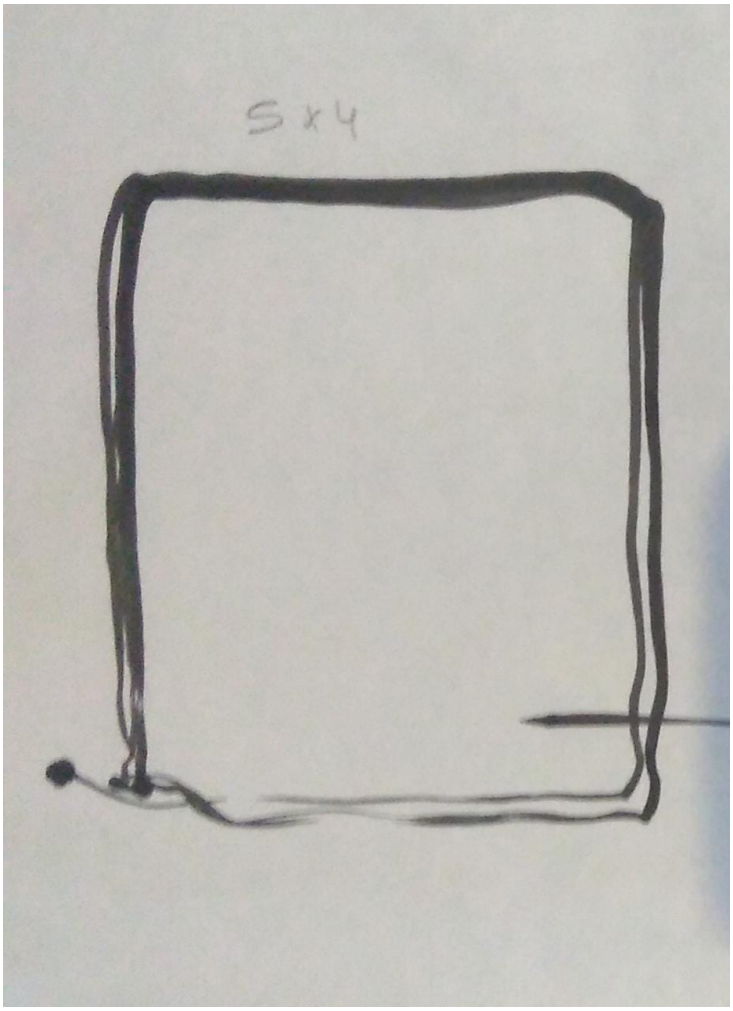


Figura 5.5: Trayectoria cuadrada (micropaso por cuatro y repetida cinco veces).

En la Fig. 5.6 se muestra la trayectoria realizada diez veces con los drivers de los motores en micropaso por cuatro. En este caso se puede ver un corrimiento continuo hacia la derecha luego de una cantidad de ciclos en el mismo lugar.

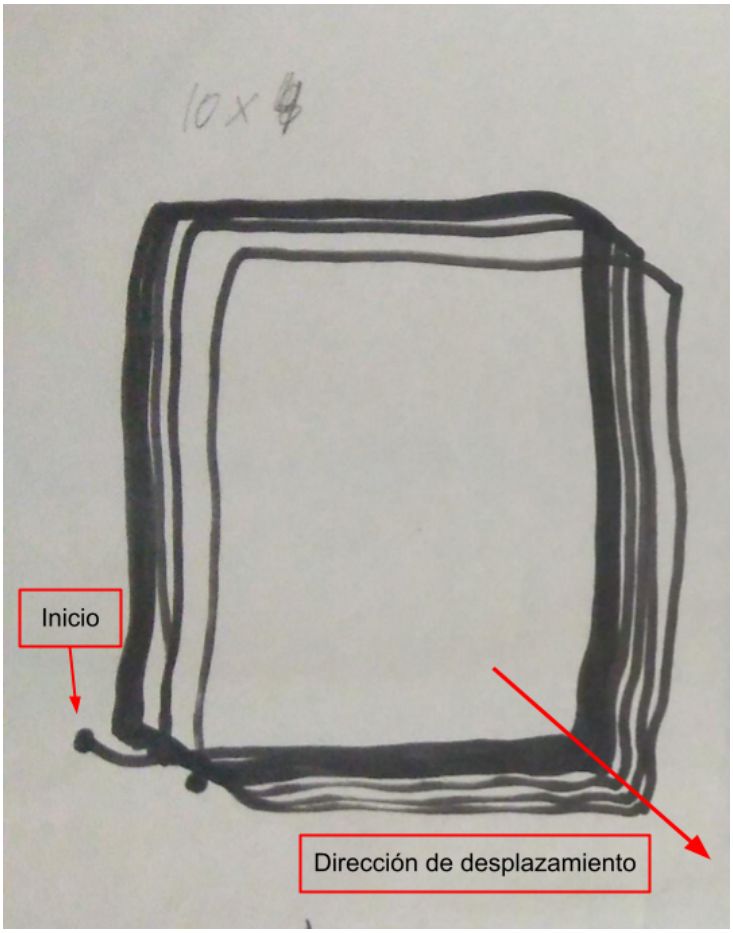


Figura 5.6: Trayectoria cuadrada (micropaso por cuatro y repetida diez veces).

Luego, para tratar de mejorar la precisión y la repetibilidad, se decidió repetir el experimento cambiando el micropaso de los drivers por ocho.

En la Fig. 5.7 se muestran los resultados para la trayectoria, en esta se puede ver que el robot no empieza ni termina en el mismo punto como se programó, también se puede ver que la forma es rectangular y no un cuadrado como se esperaba. En cuanto a la rectitud, se nota una mejora en comparación con la de micropaso por cuatro y un incremento en el paralelismo y en la suavidad de los trazos.

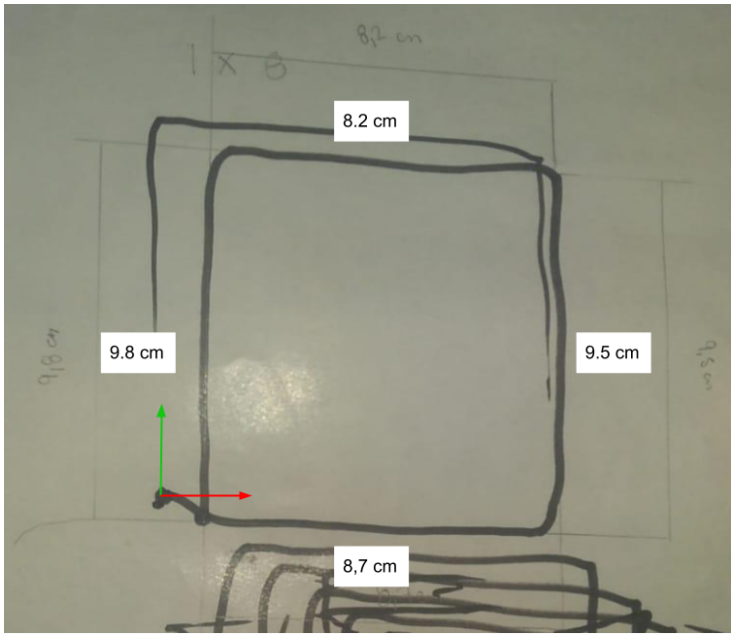


Figura 5.7: Trayectoria cuadrada (micropaso por ocho y repetida una vez).

En la Fig. 5.8 se muestra el resultado de repetir cinco veces seguidas la misma trayectoria con micropaso por ocho. En esta figura se puede observar el mismo patrón de corrimiento hacia la derecha que en el caso de la trayectoria con drivers en micropaso por cuatro, pero esta vez el corrimiento se ve disminuido.

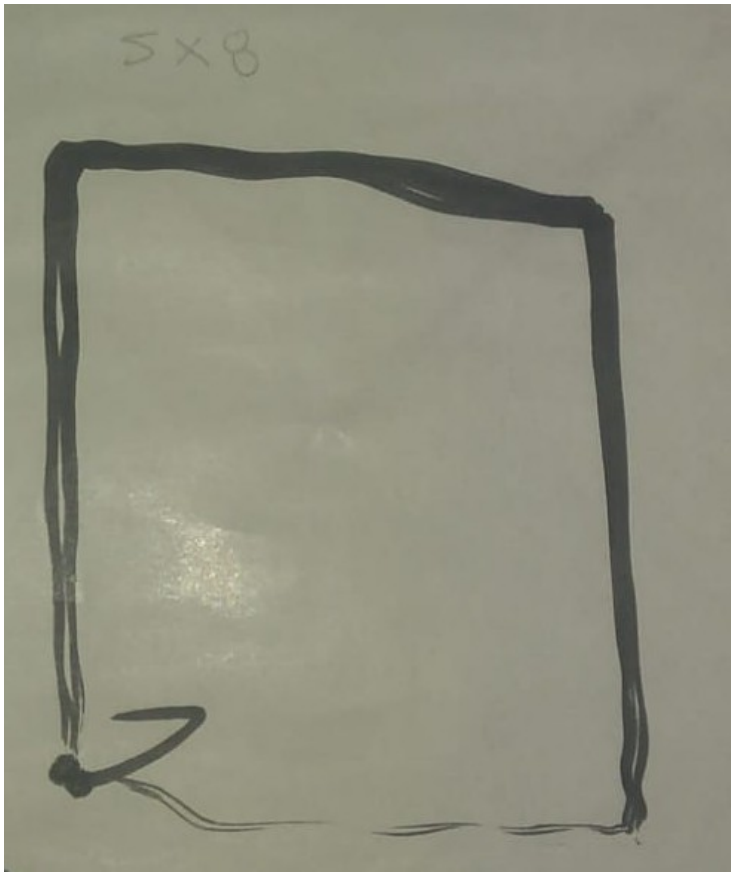


Figura 5.8: Trayectoria cuadrada (micropaso por ocho y repetida cinco veces).

Finalmente en la Fig. 5.9 se muestra el resultado de repetir diez veces seguidas la misma trayectoria con micropaso por ocho. Nuevamente se percibe el corrimiento a la derecha, pero a diferencia de la trayectoria con drivers en micropaso por cuatro, esta se estabiliza después de una cierta cantidad de repeticiones.

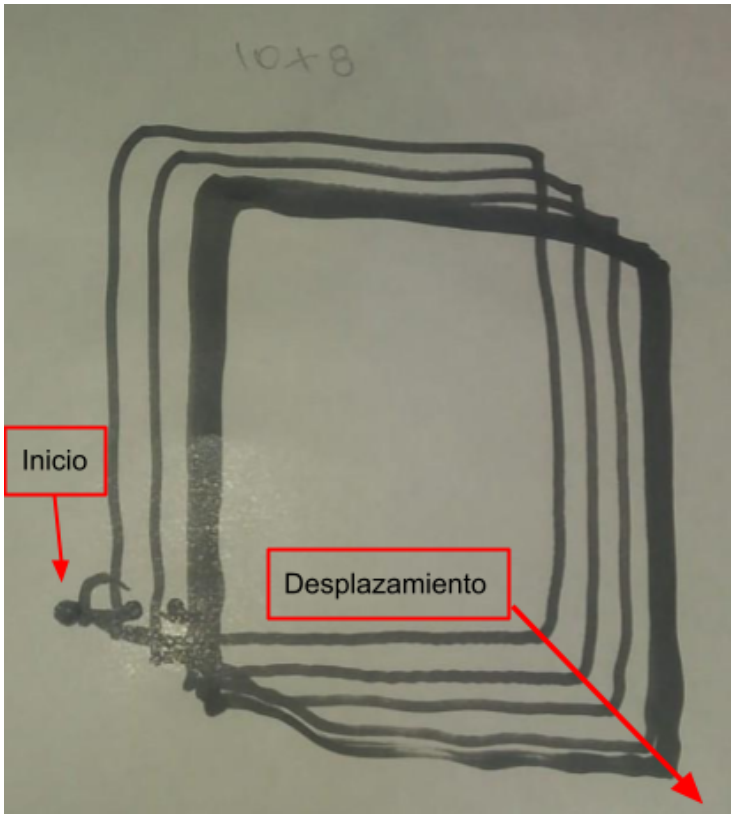


Figura 5.9: Trayectoria cuadrada (micropaso por ocho y repetida diez veces).

Se concluye que el mejor desempeño de los motores se logra con un micropaso de por ocho. En estos experimentos se decidió variar solo ese parámetro (dejando las demás variables constantes) debido a que la variación de los demás parámetros se realizaron previamente a este experimento.

5.2. Círculo

Para probar la capacidad de realizar un círculo de nuestro robot, se realizaron solo los experimentos con el micropaso por ocho. Esto fue debido a que esta configuración de drivers mostró un mejor rendimiento.

5.2.1. Definición de la trayectoria

La trayectoria elegida para estos experimentos se muestra en la Fig. 5.10. Esta empieza y termina en el punto $\begin{bmatrix}0 & 0{,}37 & 0\end{bmatrix}$, tiene como centro el punto $\begin{bmatrix}0. & 0{,}42 & 0\end{bmatrix}$ dejando un radio de $5cm$. Esta trayectoria fue generada usando un perfil de velocidad trapezoidal, una velocidad y aceleración máxima de $0{,}2\frac{m}{s}$ y $1\frac{m}{s^2}$ respectivamente (ver Fig. 5.11 y 5.12). La misma se encuentra en el archivo *circle.py* ubicado en la carpeta *five-bar-robot/src/motion_planning/examples* del repositorio y se generó utilizando el método *circle* (ver apéndice F).

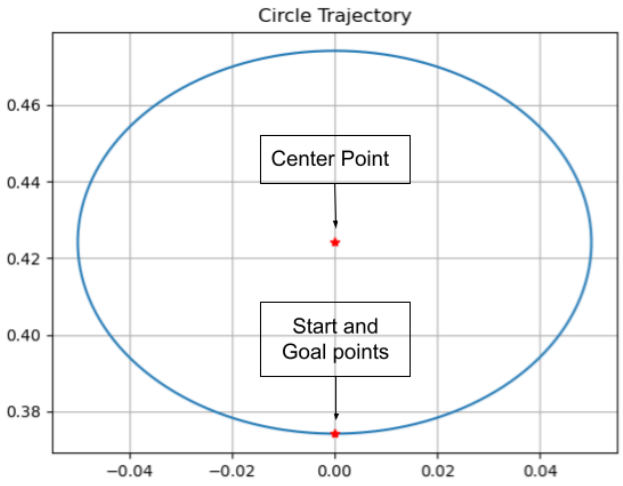


Figura 5.10: Trayectoria circular.

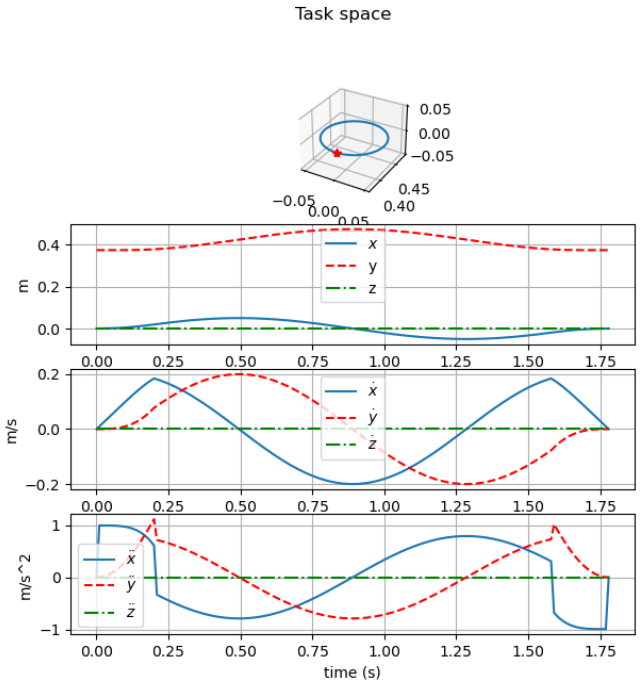


Figura 5.11: Trayectoria circular, espacio de tarea.

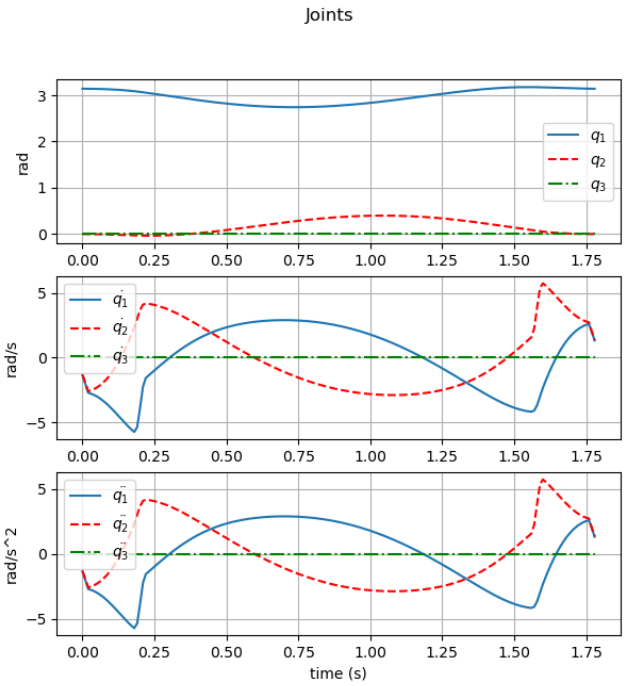


Figura 5.12: Trayectoria circular, espacio articular.

5.2.2. Resultados

Se ejecutó la trayectoria una vez con los drivers de los motores en micropaso por ocho. En la Fig. 5.13 se muestran los resultados de la ejecución. En esta se puede observar que la trayectoria no termina donde empieza y contiene tramos rectos. Esto se puede deber a los mismos problemas identificados en la sección anterior. A pesar de su forma, se pueden distinguir arcos de circunferencia.

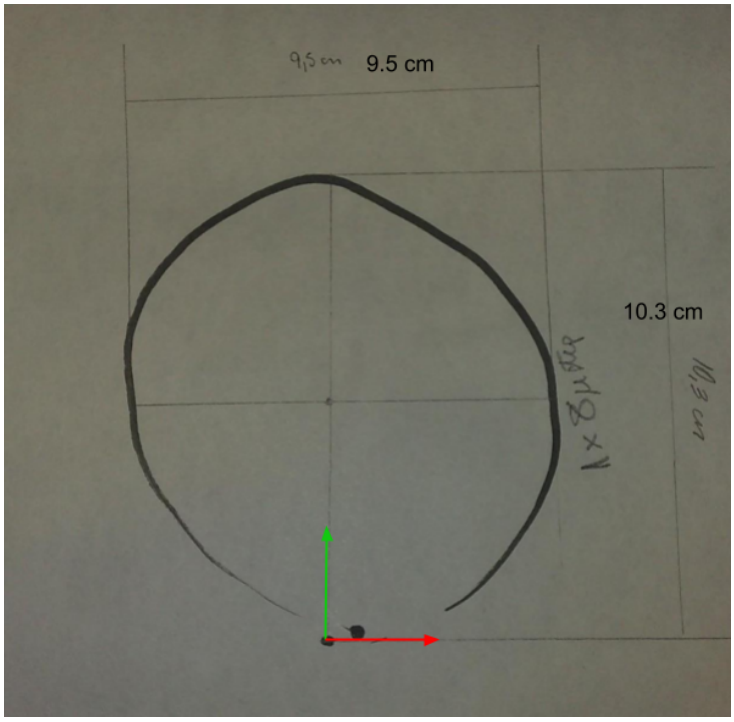


Figura 5.13: Trayectoria circular, micropaso por ocho. Ejecución una vez.

En la Fig. 5.14 se observan los resultados de reproducir la trayectoria anterior cinco veces seguidas. En esta imagen se puede observar que se mantiene la forma de la trayectoria anterior y un cierta repetibilidad en esta.

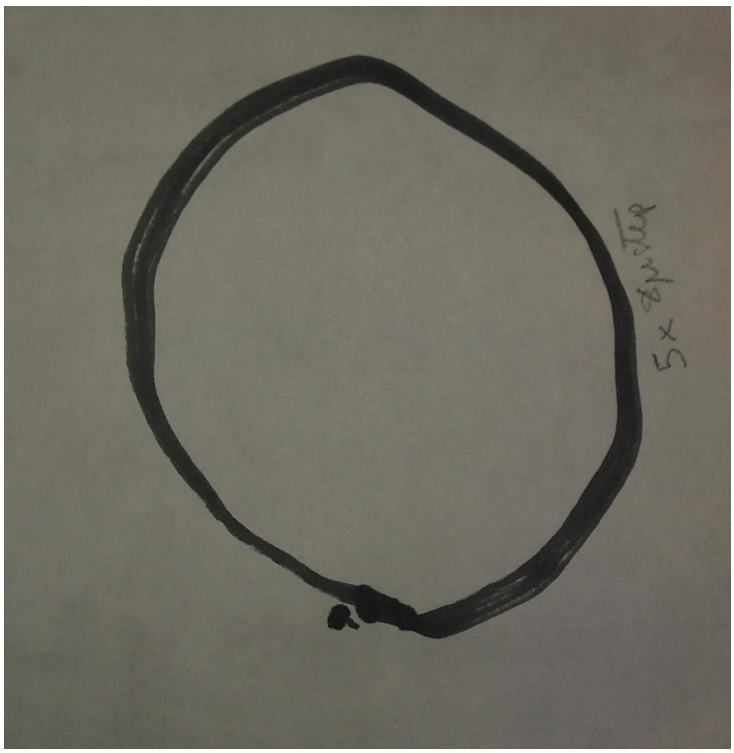


Figura 5.14: Trayectoria circular, micropaso por ocho. Ejecución cinco veces.

En la Fig. 5.15 se observan los resultados de reproducir la trayectoria anterior diez veces seguidas. En esta imagen se puede observar un continuo desplazamiento hacia la derecha con cada nueva repetición.

Figura 5.15: Trayectoria circular, micropaso por ocho. Ejecución diez veces.

A partir de esta experimentación se observa una capacidad de realizar arcos de circunferencia pero no círculos completos.

5.3. Análisis de resultados

De la experimentación de la trayectoria en línea recta (5.1) se obtuvieron las siguientes conclusiones:

- Se validó la aplicación que generan las trayectorias en línea recta.
- Se validó la aplicación que envía las consignas al robot.
- Se validó el firmware.
- Se validó el funcionamiento mecánico del robot.
- Se percibió una cierta repetibilidad en las trayectorias del robot.
- Se percibe un mejor rendimiento del robot al utilizarlo los drivers en micropaso por ocho.

Por otra parte, de la generación de trayectorias circulares, se puede concluir que: si bien el robot no se comporta como era esperado en este tipo de trayectorias, se pude inducir que la generación de arcos funcionaría mejor. Debido a la falta de tiempo, no se experimentó con este tipo de trayectorias aisladas. También, se puede ver una cierta repetibilidad, ya que todas las figuras generadas tienen las mismas imperfecciones.

Finalmente, se presenta una lista de posibles fuentes de error que podrían justificar las faltas en el desempeño de las trayectorias anteriores:

- Posibles errores en los algoritmos de cinemática directa y/o inversa.
- Posibles errores en los algoritmos de planificación de trayectorias.
- Posibles errores al estimar las distancias geométricas de nuestro robot.
- Posible diferencia en los drivers.

- Posible falta de torque en los motores.
- Posible falta de tensión en las correas.
- Posible deformación de las poleas conductoras.
- Posibles errores en la maniobra *homing* (este se realiza a mano por los alumnos).

En la sección de trabajos futuros se darán algunas recomendaciones acerca de estos problemas.

Capítulo 6

Resumen, conclusiones y trabajos futuros

A continuación, se dedica el capítulo a realizar un resumen del trabajo presentado. Se describen las conclusiones generales obtenidas y luego se realiza un detalle sobre cada una de las etapas para llevarlo a cabo. Se reflexiona sobre aciertos y errores en las distintas decisiones tomadas durante el desarrollo, primero sobre el modelado del robot, luego del diseño mecánico y por último sobre la arquitectura de control y el firmware desarrollado. Para finalizar el capítulo se enumeran diferentes tareas que quedaron pendientes como trabajo a futuro, a modo de guía para continuar el desarrollo del robot.

6.1. Conclusiones

A modo de conclusión general, se logró obtener un robot funcional, que dispone de la electrónica necesaria para su operación. Su construcción es económica en comparación a otros robots que realizan tareas similares, y sobre todo es fácil de replicar, característica que se consideró de suma importancia siendo un robot que tiene una finalidad educativa. Además, se deja una base teórica sobre la que los estudiantes podrían apoyarse a la hora de elaborar sus proyectos.

A continuación, se detallan las distintas conclusiones obtenidas en cada etapa del proyecto: primero las conclusiones sobre el modelado del robot, luego sobre el diseño mecánico y por último de la arquitectura de control y el firmware desarrollado.

6.1.1. Modelado del robot

En el capítulo 2 se describió el robot desde un punto de vista cinemático y dinámico. El modelo cinemático allí presentado permite entender las características geométricas del robot, esto resultó ser de gran utilidad al momento de diseñar el robot y su planificador de trayectorias. El modelo dinámico permite la obtención de los torques necesarios para llevar el robot a un cierto estado, esto resulta ser de utilidad al momento de seleccionar los motores.

A pesar de las dos bondades antes mencionadas, ambos modelos resultaron ser extremadamente complejos. Esto, sumado a la falta de presupuesto, impidió su uso en la practica. Una alternativa al modelo geométrico planteado se presenta en Bourbonnais, Bigras y Bonev [1], este resuelve el modelo geométrico mediante la resolución de triángulos simplificando la obtención de las coordenadas articulares activas. Un punto en contra de este desarrollo es que no obtiene las coordenadas pasivas ni los modelos cinemáticos. Se concluyó que un desarrollo más pragmático en el modelo geométrico del robot es preferible cuando se trata de robots de bajo presupuesto con controladores con pulso y dirección. De haber tenido la posibilidad de implementar un control más avanzado, hubiera sido de mayor utilidad el modelo dinámico planteado en el capítulo 2.4.

Para validar los dos modelos, se obtuvieron dos simulaciones, una cinemática, realizada en *CoppeliaSim*, y otra dinámica, realizada en *Simulink/Simscape*. Estas resultaron de gran utilidad para entender el funcionamiento del robot y simular diversas trayectorias. Esto ultimo permitió agilizar el desarrollo de algoritmos. Como conclusión de la utilización de diversas herramientas de simulación, se destaca la

practicidad de *CoppeliaSim* y *Simscape* (herramientas de caja negra) que permiten un desarrollo más rápido y sencillo.

El análisis del espacio de trabajo del robot nos llevó a la conclusión que mientras más pequeña sea la separación entre las articulaciones actuadas q_{i1} y mientras menor sea la diferencia entre los eslabones $\mathscr{B}_{i1}$ y $\mathscr{B}_{i2}$ para $i = 1, 2$, mayor será nuestro espacio de trabajo. Como resultado de este análisis, se definió la estructura geométrica de nuestro robot.

A modo de conclusión general, si bien el desarrollo de modelos es arduo y complejo, estos nos permiten entender completamente el sistema en cuestión.

6.1.2. Diseño mecánico

En el capítulo 3 se describió el diseño de un primer y segundo prototipo del robot.

El primer prototipo fue un diseño simple que permitía de forma sencilla cambios en la geometría fundamental del robot: esto es la longitud de los eslabones y el rango articular. El motivo de su simpleza fue la necesidad, en esa instancia del proyecto, de ver la viabilidad de la construcción del mecanismo y qué aspectos eran los fundamentales en un diseño mecánico avanzado para lograr obtener un movimiento suave y preciso. Sin embargo, como se mencionó previamente, a pesar de que la propuesta era muy interesante desde un punto de vista académico, lograr un diseño que en toda su libertad de movimiento no tenga interferencia mecánica y además logre un desempeño adecuado es complejo. El objetivo de este trabajo en particular era otro, pero sería interesante continuar con el desarrollo de un prototipo con una configuración similar para hacer frente a los desafíos que presenta en su control y planificación de trayectorias. Sería interesante utilizar los actuadores y la electrónica necesaria para implementar sobre este tipo de configuración un control dinámico.

El segundo prototipo se inició con ánimos de obtener un diseño que se acercara a un producto comercial. A la hora de realizar pruebas se reconoció que fue un error, y que hubiera sido necesario una instancia previa para experimentación y luego proceder en una tercera o cuarta versión al diseño de un prototipo más estético. La decisión desde un principio fue impulsada por las ansias del equipo de poder finalizar con un robot cercano a un producto final, pero esto provocó retrasos en la experimentación al no tener en cuenta que en esa etapa del proyecto es fundamental la facilidad de ensamble y desensamble del prototipo a la hora de corregir, mejorar o reparar componentes internos.

En la sección 3.2.1 se menciona que a la hora de plantear la relación de transmisión se proponía lograr un diseño que permitiera sustituir fácilmente poleas de diferentes dimensiones y así experimentar con distintos valores de relación de transmisión. Esto en la práctica no se logró por ser demasiado complejo desarmar el prototipo hasta lograr acceder a las poleas. Es por esto que no se pudo observar los movimientos resultantes con diferentes relaciones de transmisión.

En la misma sección se menciona la elección de utilizar correa abierta en lugar de cerrada para no estar limitados y realizar el diseño alrededor de la longitud disponible en el mercado. En la etapa de experimentación se concluyó que no es la mejor decisión de diseño: como ya se mencionó, la relación de transmisión quedó fija, el mecanismo de fijación de los extremos de la correa sobre la polea conducida fue complejo, y de una u otra forma es necesario la incorporación de un mecanismo de tensión de la correa. La solución planteada se describió en la sección 3.2.7, sin embargo, en una versión posterior del robot sería recomendable un rediseño de la base considerando incluir el mecanismo de tensión como un elemento fundamental para el correcto funcionamiento del prototipo. En la sección 3.2.3 se menciona que en una posible revisión se puede corregir el cuerpo de las poleas guía para que sean simétricos, y además contemplar la posibilidad de hacer que su eje sea móvil para convertirlas en poleas de tensión, además de guía de la correa.

En todos los diseños de piezas que se fabricaron con impresión 3D se tuvo en cuenta el proceso de impresión, y se observa, por ejemplo, relieves sobre la superficie de contacto con la cama caliente para facilitar su extracción, o también huecos dentro del cuerpo de la pieza para disminuir la cantidad de material necesario. También se adaptaron las piezas de mayor tamaño al espacio de trabajo de la impresora 3D disponible, que consistió en una Artillery Genius que permite impresiones dentro de un cubo de $220mm \times 220mm \times 250mm$. Las piezas del proyecto pueden dividirse en dos grupos: aquellas piezas de tamaño pequeño, como los separadores para rodamientos o con perfiles detallados como las poleas, que se recomienda imprimir con un pico extrusor de por lo menos 0.4mm de diámetro; y luego piezas de mayor tamaño como las que conforman la base y los cuerpos de las diferentes articulaciones, para las que se recomienda utilizar en su impresión un pico extrusor de 0.8mm. Esta última recomendación es porque no tienen detalles que se pierdan con ese tamaño de pico, y el tiempo de impresión se ve realmente

acortado (con un pico extrusor de 0.4mm, y a velocidades normales de impresión, el tiempo a veces supera por mucho las 24 horas).

Los diferentes diseños en el software CAD se encuentran ordenados, de forma jerárquica, con lo que si un tercero desea realizar modificaciones es sencillo e intuitivo llegar a la pieza o ensamble deseado. Además, los recursos que se reutilizan con frecuencia como tornillos, tuercas, barras rectificadas, poleas, etc. se encuentran parametrizados y no es necesario diseñarlos desde cero, solo basta con añadir la nueva configuración con los parámetros deseados en la planilla de cálculo correspondiente.

6.1.3. Arquitectura de control, firmware y comunicaciones

En el capítulo 4 se describió la arquitectura de control, el firmware utilizado, y la estructura de comunicaciones.

Primero, se describió la arquitectura general de nuestro sistema de control. Esta fue dieñada con una estructura en capas, la cual permitió un desarrollo modular del sistema completo. Segundo, se desarrolló el firmware utilizado para el control de bajo nivel. Tercero, se desarrolló el software de control de alto nivel. Este consta de una librería para generar las trayectorias y un módulo para enviarlas al robot. Finalmente, se describió la estructura de comunicaciones.

Al comienzo del capítulo se planteó una serie de objetivos hacia a donde se deseaba guiar el desarrollo. Siempre considerando que eran lineamientos y no metas rígidas, se quería contar con un sistema que sea generalizable, independiente del software superior, jerarquizado y con una estructura en capas, operable *online*, extensible y con un nodo por grado de libertad y un nodo orquestrador.

Podemos afirmar que el resultado del subsistema de control cumple en gran medida todos los objetivos planteados, con la excepción de la operación del robot de forma *online*.

Se desarrolló una placa "CAN node" que, junto con el firmware, conforma un nodo extremo, el cual puede ser utilizado para el control de movimiento sincronizado de un sistema de N grados de libertad. Incluye dos modos de operación: posicionamiento por perfil y posición interpolada. Ambos modos están totalmente desacoplados del sistema mecánico subyacente, por lo que la reutilización del subsistema electrónico y de control de bajo nivel en robots con otras configuraciones es ciertamente posible. El proyecto concluyó con el nodo extremo trabajando a lazo abierto, aunque las placas ya tenían reservadas borneras para la entrada de un encoder en cuadratura; queda como trabajo futuro la adaptación del firmware para trabajar a lazo cerrado.

El nodo orquestrador también corre en una placa "CAN node" y sus funciones son las de comunicarse con el software de planificación de trayectoria, solicitar el setpoint o la secuencia de puntos que deben cumplir los nodos secundarios y asegurar la sincronización y el cumplimiento en términos de tiempo. Inicialmente se deseaba que el orquestrador llevara a cabo un seguimiento del comportamiento de todos los nodos extremo. Se implementarían máquinas de estado análogas a las que corren en los nodos extremo, de forma tal que nunca se enviaran comandos erróneos o potencialmente peligrosos (por ejemplo, enviar comandos PP en modo IP o realizar un movimiento a una posición fuera del espacio de trabajo). Por falta de tiempo esta idea se dejó de lado y por ende toda la responsabilidad de conocer el estado de los nodos extremo y la validación de los movimientos recae sobre el operador y el software de generación de trayectoria.

La arquitectura en capas fue, en nuestra consideración, un gran acierto. Haber definido previamente, aun a alto nivel, los distintos niveles de responsabilidades, de interacción con el hardware y las interfaces entre capas permitió un desarrollo ágil. Esto es a pesar de que todos los miembros nos encontrábamos geográficamente distanciados y a que la mayor parte del desarrollo fue durante una crisis sanitaria. Como consecuencia nos vimos obligados a testear fuertemente el comportamiento individual de las capas y la etapa de integración se llevó a cabo sin demasiadas complicaciones.

Como se mencionó en la sección 4.3, el diseño de la arquitectura fue de las primeras decisiones que se tomaron respecto al subsistema de electrónica y control. Por razones de simplicidad se decidió utilizar una computadora de propósito general convencional para el cálculo y envío de trayectorias, con la consecuencia de tener que delegar las tareas de tiempo real y sincronización de movimientos en un nodo embebido más: el orquestador. Por esa misma razón también se agregó un enlace de comunicación utilizando la UART. Existen otras alternativas, que muy probablemente habrían implicado un mayor esfuerzo monetario y sobre todo en tiempo de desarrollo. Por ejemplo, se podría haber utilizado una *single-board computer* corriendo Linux con un parche de tiempo real y un controlador CAN externo que utilice el driver SocketCAN; o una solución del tipo *System-on-Module* con CAN integrado. Estas son opciones muy interesantes y apuntan a la dirección en la que se mueve la industria, por lo que tenemos

en cuenta su exploración en trabajos futuros e incentivamos su utilización en otros proyectos del ámbito educativo. Habiendo dicho eso, es destacable nombrar que la arquitectura que se utilizó cumple con los lineamientos que se habían planteado al comienzo del desarrollo y la solución es funcional y extensible.

La interfaz entre el nodo orquestrador y los nodos extremos fue por medio del bus CAN, con una fuerte inspiración en el protocolo CANopen para la estructura de mensajes. El hecho de tener un estándar en el cual basarse permitió quitar el foco del qué, para ponerlo en el cómo. Haber reutilizado una estructura que ya ha sido pensada y diseñada para dispositivos de control de movimiento facilitó el desarrollo del firmware ya que nos pudimos centrar en la programación y cómo generar código extensible y generalizable. Se debe recordar que solo una muy pequeña parte del estandar fue estudiada.

La contracara de lo comentado anteriormente fue la interfaz UART entre el nodo orquestrador y la computadora que calcula las trayectorias. Se diseñó una trama ad-hoc para la transmisión de mensajes lo cual agrega overhead y algo de complejidad en el desarrollo. Lo positivo es que esta interfaz resultó muy transparente y no agregó limitaciones a la ejecución del pipeline por la utilización de la técnica de control de flujo. Además, al estar basada en texto plano es fácil generar el archivo la trayectoria utilizando otros programas de generación de trayectoria. Aunque se pensó agregar a la trama un campo de detección de errores utilizando CRC se decidió dejar de lado porque habría implicado un mayor esfuerzo de programación y, realmente, en ningún momento nos encontramos con ningún error en el enlace UART.

El software planificador de trayectorias resultó satisfactorio. Este cumplió con todos los casos de uso previstos y proporcionó herramientas de gran utilidad para la verificación y ejecución de trayectorias. La decisión de utilizar *Python* resulto muy beneficiosa. Este nos permitió un flujo de trabajo ágil y simple, simplificando el trabajo matricial. La única manera de interactuar con el software es mediante su *API* (programando). La carencia de una interfaz gráfica dificultó el proceso de generación de trayectorias, pero el no incluirla nos permitió desarrollar en profundidad el planificador.

6.2. Trabajos futuros

El principal propósito del robot es servir para fines educativos, es por eso que se deja una extensa lista de trabajos futuros. Estos podrían ser llevados a cabo durante la realización de otros proyectos finales de estudios o mediante trabajos prácticos en asignaturas de la carrera.

En cuanto al modelado del robot se presentan los siguientes trabajos futuros:

- Análisis cinemático y dinámico del robot, siguiendo con el desarrollo teórico planteado, al agregar el eje z.
- Agregar transformación TCP (*tool center point*) al cálculo de cinemática del robot. Esto se vuelve indispensable si se quiere utilizar el robot para tareas de *pick and place*.
- Estimar los parámetros dinámicos del robot con mayor precisión. Se recomienda seguir los pasos y estrategias presentadas en el capitulo 12 del libro Khalil y Dombre [23].
- Agregar modelo completo de actuadores y reductores al modelo dinámico del robot. Esto modelaría mejor la realidad. Para su realización se recomienda seguir con la estrategia planteada en Sanchez [18].
- Simular el robot dinámicamente en *CoppeliaSim*. Para esto se necesita conocer los parámetros dinámicos y explorar las opciones que el software ofrece.
- Utilizar la simulación cinemática como gemelo virtual del robot. Para esto es necesario enviar la misma consigna a la simulación y al robot al mismo tiempo. También se podría utilizar la simulación para validar visualmente que la trayectoria generada es válida y físicamente realizable por el robot.
- Utilizar la simulación para obtener puntos a enviar al robot. La idea seria colocar el robot en una determinada posición en la simulación y mediante un *script* externo, guardar esos puntos para luego planificar la trayectoria y enviarla al robot.

Hablando del diseño mecánico se plantean los siguientes trabajos:

- Diseño y construcción del eje z.
- Diseño de un efector final.

- Rediseño y mejora del prototipo, apuntando a tener un tercer prototipo funcional.
- Desarrollar un prototipo capaz de cambiar de modos de trabajo. Para esto sería necesario diseñar el robot la menor interferencia mecánica y el mayor rango articular posible.
- Incorporación de un tensor al mecanismo de transmisión por correas.

En lo que respecta al control del robot se proponen los siguientes trabajos a futuro:

- Desarrollo de una interfaz de usuario para el robot. Se podría usar el framework *Qt*.
- Realizar un *refactor* del firmware, agregando funcionalidades y un RTOS.
- Redefinir la arquitectura de control considerando y evaluando el uso de dispositivos *single-board computer* o *system-on-module*.
- Incorporación de encoders, estos se pueden utilizar para cerrar el lazo de control. El diseño del robot contempla la ubicación de estos, simplemente se debería extender el firmware.
- Agregar maniobra de *homing* utilizando los sensores de efecto hall. El diseño del robot contempla la ubicación de estos, simplemente se debería extender el firmware.
- Utilizar drivers de mayor potencia y calidad, junto con una fuente de tensión de mayor voltaje. Alternativamente, se pueden utilizar drivers con control a lazo cerrado integrado como el 2HSS57. Esta solución debería ser *drop-in* ya que la interfaz también es paso-dirección.
- Tener una mayor observabilidad de las variables y los estados del sistema. En la aplicación desarrollada no se tiene acceso a, por ejemplo, los valores de posición de los nodos. Estos datos podrían ser publicados en el bus CAN y capturados por el nodo orquestador, para una posterior evaluación offline en la PC.
- Utilizar una cámara para la estimación del efector final y el punto objetivo. De esta manera, se podría extender el funcionamiento del robot para realizar separación de residuos. Como caso de prueba, se propone separar tapas de diferentes colores y/o tamaños.
- Utilizar los encoders para incorporar el modo *freedrive* y seleccionar los puntos objetivos posicionando el robot en dichos puntos.
- Agregar unión de líneas, círculos y arcos sin detenimiento. Para esto se debería extender la aplicación que genera las trayectorias para soportar perfiles de velocidades asimétricos.

Bibliografía

[1] Bourbonnais, Bigras y Bonev. «Minimum-Time trajectory Planning and Control of a Pick-and-Place Five-Bar parallel robot». En: (2015).

[2] Wisama Khalil y Sébastien Briot. *Dynamics of Parallel Robots*. Springer International, 2015. ISBN: 9783319197883.

[3] G. Avanzini, G. Fernández y J. Pino. «Proyecto de Investigación Tipo C». En: (2021).

[4] Luigi Biagiotti y Claudio Melchiorri. *Trajectory Planning for Automatic Machines and Robots*. 2008.

[5] Antonio Barrientos y col. *Fundamentos de Robótica*. McGraw Hill, 2007. ISBN: 9788448156367.

[6] Bruno Siciliano y col. *Robotics: Modeling, Planning and Control*. Springer, 2010. ISBN: 9781846286414.

[7] P. I. Corke. *Robotics, Vision & Control*. Springer, 2017. ISBN: 9783319544137.

[8] Hyun Joong Yoon y col. «Trapezoidal Motion Profile to Suppress Residual Vibration of Flexible Object Moved by Robot». En: *MDPI Electronics* 1 (2019), págs. 1-17.

[9] Di Natale et al. *Understanding and using the Controller Area Network Communication Protocol*.

[10] Lawrenz (ed.) *CAN System Engineering: From theory to practical applications*.

[11] Pfeiffer, Ayre y Keydel. *Embedded Networking with CAN and CANopen*.

[12] Schneider Electric. *CANopen DS301. Fieldbus manual*.

[13] Schneider Electric. *CANopen DS402. Fieldbus manual*.

[14] Elmo Motion Control. *CANopen DSP 402. Implementation Guide*.

[15] W. Khalil y J. Kleinfinger. «A new geometric notation for open and closed-loop robots». En: *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA1986), San Francisco, CA, USA* (1986), págs. 1174-1180.

[16] Adrien Koessler y col. «Dynamics-based Algorithm for Reliable Assembly Mode Tracking in Parallel Robots». En: *IEEE Transactions on Robotics* 1 (2020), págs. 937-950.

[17] Alexandre Figielski, Ilian Bonev y Pascal Bigras. «Towards development of a 2-DOF planar parallel robot with optimal workspace use». En: nov. de 2007, págs. 1562-1566. DOI: 10.1109/ICSMC.2007.4413840.

[18] Eric Sanchez. «Modelado, simulación y control, mediante plataforma embebida, de robots industriales tipo serie». 2016.

[19] Jeremías Pino Demichelis. «Modelado y control de un robot paralelo de cinco barras». 2021.

[20] Gino Avanzini. «Control de un eje con bus CAN y comunicación basada en CANopen en plataforma ARM». 2020.

[21] Miro Samek. *Practical UML Statecharts in C/C++, Second Edition: Event-Driven Programming for Embedded Systems*. 2.ª ed. Newnes, 2008. ISBN: 0750687061.

[22] Miro Samek. *Modern Embedded Systems Programming Course*. URL: https://www.youtube.com/playlist?list=PLb-MsRpo_wlLW0EWRpAqnbbDsf4kxSI1x.

[23] W. Khalil y E. Dombre. *Modeling, identification and control of robots*. 2004.

Apéndice A

Modelo cinemático de primer orden

Para el análisis de velocidad del mecanismo se introduce el vector de velocidades articulares $\bar{\dot{q}}$, que es la derivada respecto al tiempo del vector de coordenadas articulares $\bar{q}$, y el concepto de *twist* ($\bar{t_n}$), vector donde se agrupa velocidad lineal ($\bar{v_n}$) y angular ($\bar{\omega_n}$). Para el caso del *twist* del eslabón final, el vector será de dimensión m y agrupa todas las velocidades articulares de la cadena cinemática que conecta $\mathscr{B}_0$ con $\mathscr{B}_n$:

$$\bar{t_n} = \begin{bmatrix} \bar{v_n} \\ \bar{\omega_n} \end{bmatrix} = J_n(\bar{q})\bar{\dot{q}}$$

Donde J_n es la matriz jacobiana o jacobiano, $6 \times m$.

Al mover una articulación q_k con velocidad $\dot{q_k}$, se producen velocidades angulares y lineales en el eslabón final, más específicamente en el sistema de referencia $\mathscr{F}_n$. Dichas velocidades se notarán de la forma $\omega_{k,n}$ y $v_{k,n}$ respectivamente, y el *twist* para el caso en que la articulación k es de revolución (todas en el robot en estudio) está dado por la siguiente expresión:

$$\bar{t}_{k,n} = \begin{bmatrix} \bar{v}_{k,n} \\ \bar{\omega}_{k,n} \end{bmatrix} = \begin{bmatrix} \hat{a_k}\dot{q_k} \times \bar{r}_{O_kO_n} \\ \hat{a_k}\dot{q_k} \end{bmatrix} = \begin{bmatrix} \hat{a_k} \times \bar{r}_{O_kO_n} \\ \hat{a_k} \end{bmatrix} \dot{q_k} = \mathbb{S}_k \dot{q_k} \tag{A.1}$$

Donde $\hat{a_k}$ es un versor a lo largo del eje z_k y $\bar{r}_{O_kO_n}$ es el vector posición que conecta el origen O_k con el origen O_n, de los sistemas de referencia $\mathscr{F}_k$ y $\mathscr{F}_n$ vinculados a los eslabones $\mathscr{B}_k$ y $\mathscr{B}_n$ respectivamente. La matriz $\mathbb{S}_k$ se denomina *twist* unitario y representa el desplazamiento del eslabón final cuando solo el actuador k está en movimiento.

Por lo tanto, para la primer cadena serie planar obtenida al abrir el mecanismo virtualmente (3R) se obtiene:

$${}^0\bar{t}_p = \begin{bmatrix} {}^0\mathbb{S}_{11} & {}^0\mathbb{S}_{12} & {}^0\mathbb{S}_{13} \end{bmatrix} \bar{\dot{q}}_1 \tag{A.2}$$

Y para la cadena 2R:

$${}^0\bar{t}_p = \begin{bmatrix} {}^0\mathbb{S}_{21} & {}^0\mathbb{S}_{22} \end{bmatrix} \bar{\dot{q}}_2 \tag{A.3}$$

Donde: ${}^0\bar{t}_p = [\dot{x} \quad \dot{y} \quad 0 \quad 0 \quad 0 \quad \dot{\phi}]^T$, $\bar{\dot{q}}_1 = [\dot{q_{11}} \quad \dot{q_{12}} \quad \dot{q_{13}}]^T$ y $\bar{\dot{q}}_2 = [\dot{q_{21}} \quad \dot{q_{22}}]^T$.

Luego puede reorganizarse la expresiones según correspondan a articulaciones activas o pasivas:

- Velocidades articulares activas: $\dot{\bar{q}}_{a1} = \dot{q}_{11}$, $\dot{\bar{q}}_{a2} = \dot{q}_{21}$
- Velocidades articulares pasivas: $\dot{\bar{q}}_{d1} = [\dot{q}_{12} \quad \dot{q}_{13}]^T$, $\dot{\bar{q}}_{d2} = \dot{q}_{22}$.
- Matriz de twist unitario activa: ${}^0\mathbb{S}_{a1} = {}^0\mathbb{S}_{11}$, ${}^0\mathbb{S}_{a2} = {}^0\mathbb{S}_{21}$.
- Matriz de twist unitario pasiva: ${}^0\mathbb{S}_{d1} = [{}^0\mathbb{S}_{12} \quad {}^0\mathbb{S}_{13}]$, ${}^0\mathbb{S}_{d2} = {}^0\mathbb{S}_{22}$.

De esta forma la ecuación resultante es como sigue:

$${}^0\bar{t}_p = \begin{bmatrix} {}^0\mathbb{S}_{ia} & {}^0\mathbb{S}_{id} \end{bmatrix} \begin{bmatrix} \dot{q}_{ai} \\ \dot{q}_{di} \end{bmatrix} = {}^0\mathbb{S}_{ia}\dot{q}_{ai} + {}^0\mathbb{S}_{id}\dot{q}_{di} \tag{A.4}$$

Para eliminar las velocidades articulares pasivas de la expresión, se las multiplica por un par $\bar{\zeta}_i$ que es recíproco al twist de las articulaciones pasivas pero no al de las activas:

$$\bar{\zeta}_i^T \mathbb{S}_{id} = 0 \quad \wedge \quad \bar{\zeta}_i^T \mathbb{S}_{ia} \neq 0 \tag{A.5}$$

$\bar{\zeta}_i$ es un par que, aplicado al extremo, puede ser resistido utilizando solo los actuadores de la cadena i. Entonces se puede obtener $\bar{\zeta}_1$ y $\bar{\zeta}_2$, asociados a la cadena 3R y 2R respectivamente. Con los $\bar{\zeta}_i$, se obtuvo las matrices $\mathbb{A}$ y $\mathbb{B}$ que son:

$$\mathbb{A} = \begin{bmatrix} \bar{\zeta}_1^T \\ \bar{\zeta}_2^T \end{bmatrix} \quad ; \quad \mathbb{B} = -\begin{bmatrix} \bar{\zeta}_1^T \cdot^0 \mathbb{S}_{1a} & 0 \\ 0 & \bar{\zeta}_2^T \cdot^0 \mathbb{S}_{2a} \end{bmatrix} \tag{A.6}$$

Se obtiene la expresión:

$$\mathbb{A} \cdot^0 \bar{t}_p + \mathbb{B} \cdot \dot{\bar{q}}_a = 0 \tag{A.7}$$

Por último, se puede reducir las expresiones definiendo un vector ${}^0\bar{t}_r$ que agrupa las coordenadas independientes de ${}^0\bar{t}_p$ de tal forma que:

$${}^0\bar{t}_p = \Psi_t \cdot^0 \bar{t}_r \quad \Longleftrightarrow \quad {}^0\bar{t}_r = \Psi_t^{inv} \cdot^0 \bar{t}_p \tag{A.8}$$

Y la ecuación A.7 queda de la forma:

$$A_r \cdot^0 \bar{t}_r + B \cdot \dot{\bar{q}}_a = 0 \tag{A.9}$$

Donde $A_r = A\Psi_t$, y mantiene las mismas propiedades que A.
Resumiendo, el **modelo cinemático directo de primer orden** esta dado por:

$${}^0\dot{\bar{t}}_r = -A_r^{-1} B \cdot \dot{\bar{q}}_a = J \cdot \dot{\bar{q}}_a \tag{A.10}$$

Y el **modelo cinemático inverso de primer orden** esta dado por:

$$\dot{\bar{q}}_a = -B^{-1} A_r \cdot^0 \dot{\bar{t}}_r = J_{inv} \cdot^0 \dot{\bar{t}}_r \tag{A.11}$$

Apéndice B

Cálculo de velocidad de articulaciones pasivas

Al igual que con la velocidad de las articulaciones activas, la velocidad de las articulaciones pasivas podría obtenerse derivando respecto al tiempo, pero se decidió seguir el método propuesto en Khalil y Briot [2]. El twist del extremo referencia a un punto A_{im_i} será:

$$ {}^0t_p^i = \begin{bmatrix} I_3 & -{}^0\hat{r}_{pA_im_i} \\ 0 & I_3 \end{bmatrix} {}^0t_p = J_{t_i} \cdot {}^0\, t_p \tag{B.1} $$

Y a su vez, el twist ${}^0t_p^i$ puede determinarse como en secciones anteriores mediante matrices de twist unitario:

$$ {}^0t_p^i = {}^0J_{im_i} \cdot \dot{\bar{q}}_i = \begin{bmatrix} {}^0\mathbb{S}_{im_i}^{i1} & \dots & {}^0\mathbb{S}_{im_i}^{im_i} \end{bmatrix} \dot{\bar{q}}_i \tag{B.2} $$

Realizando la misma reagrupación que en el análisis previos según articulaciones activas y pasivas, para la estructura cinemática en estudio se obtiene:

$$ {}^0J_{d1} = {}^0\,\mathbb{S}_{d1}; \quad {}^0J_{d2} = {}^0\,\mathbb{S}_{d2}; \quad {}^0J_{a1} = {}^0\,\mathbb{S}_{a1}; \quad {}^0J_{a2} = {}^0\,\mathbb{S}_{a2} \tag{B.3} $$

Proyectando las matrices ${}^0J_{di}$ en los sistemas de referencia de los eslabones $\mathscr{B}_{i2}$, se obtienen las matrices ${}^{12}J_{d1}$ y ${}^{22}J_{d2}$ (cálculo indirecto de matrices Ψ_{ti}).

Luego se procedió a calcular las siguientes matrices J:

$$ J_{ti}^c = \Psi_{t_i} J_{t_i} \Psi_t; \quad J_{tai} = \Psi_{t_i}^0 J_{ai}; \quad J_{tdi} = \Psi_{t_i}^0 J_{di} \tag{B.4} $$

Que se agrupan en las siguientes matrices globales:

$$ J_{td} = \begin{bmatrix} J_{td1} & 0 \\ 0 & J_{td2} \end{bmatrix}; \quad J_t = \begin{bmatrix} J_{t1}^c \\ J_{t2}^c \end{bmatrix}; \quad J_{ta} = \begin{bmatrix} J_{ta1} & 0 \\ 0 & J_{ta2} \end{bmatrix} \tag{B.5} $$

Por último, la expresión para calcular las velocidad articulares pasivas es:

$$ \dot{\bar{q}}_d = J_{td}^{-1} \left(J_t \, {}^0\dot{\bar{t}}_r - J_{ta} \dot{\bar{q}}_a \right) \tag{B.6} $$

Apéndice C

Análisis dinámico

Una forma de obtener las expresiones del modelo dinámico inverso del sistema es mediante un enfoque de Lagrange. La única desventaja que presenta este método es que se realiza a mano e involucra trabajar matemáticamente con expresiones largas, y es por esto que en Khalil y Briot [2] se propone utilizar una forma algorítmica de Newton Euler. Sin embargo, como al dividir virtualmente la cadena cinemática del mecanismo de 5 barras se obtienen dos cadenas serie (3R y 2R), un enfoque langrangiano se puede aplicar en el robot en estudio.

$$L(\bar{q}_t(t), \dot{\bar{q}}_t(t)) = K(\dot{\bar{q}}_t(t), \bar{q}_t(t)) - V(\bar{q}_t(t)) \tag{C.1}$$

Donde K es la energía cinética del sistema y V su energía potencial.

Se obtienen los torques en las articulaciones mediante las ecuaciones de Lagrange:

$$\bar{\tau}_t = \frac{d}{dt}\left(\frac{\partial L_t}{\partial \dot{\bar{q}}_t}\right)^T - \left(\frac{\partial L_t}{\partial \bar{q}_t}\right)^T \tag{C.2}$$

Estas expresiones de torque pueden agruparse según correspondan a articulaciones activas o pasivas en los siguientes vectores:

$$\bar{\tau}_{ta} = [\tau_{11} \quad \tau_{21}]^T \quad ; \quad \bar{\tau}_{td} = [\tau_{12} \quad \tau_{13} \quad \tau_{22}]^T \tag{C.3}$$

Al realizar la apertura virtual de la cadena cinemática, se consideran todas las articulaciones activas, por lo tanto resta pasar de las expresiones de torque en cada articulación anteriores a una expresión de torque real en las articulaciones activas. Siguiendo lo propuesto en Khalil y Briot [2], el **modelo dinámico inverso** queda de la forma:

$$\bar{\tau} = \begin{bmatrix}\tau_1 \\ \tau_2\end{bmatrix} = \bar{\tau_{ta}} + J^T \cdot^0 \bar{w}_r + J_d^T \bar{\tau}_{td} \tag{C.4}$$

Donde puede observarse la matriz jacobiana cinemática J obtenida en el análisis cinemático y también la matriz jacobiana asociada a articulaciones activas J_d.

El eje z se modela como una masa puntual m_4, ya que su accionamiento es perpendicular al plano en el que trabaja el mecanismo de 5 barras, esto permite separar el modelo dinámico. De esta forma, ${}^0\bar{w}_r$ será:

$${}^0\bar{w}_r = \Psi_t^T \cdot^0 \bar{w}_p^T = m_4 \begin{bmatrix}\ddot{x} \\ \ddot{y}\end{bmatrix} \tag{C.5}$$

Siendo Ψ_t la matriz de reducción calculada en el análisis cinemático.

Apéndice D

Ensamblaje del segundo prototipo

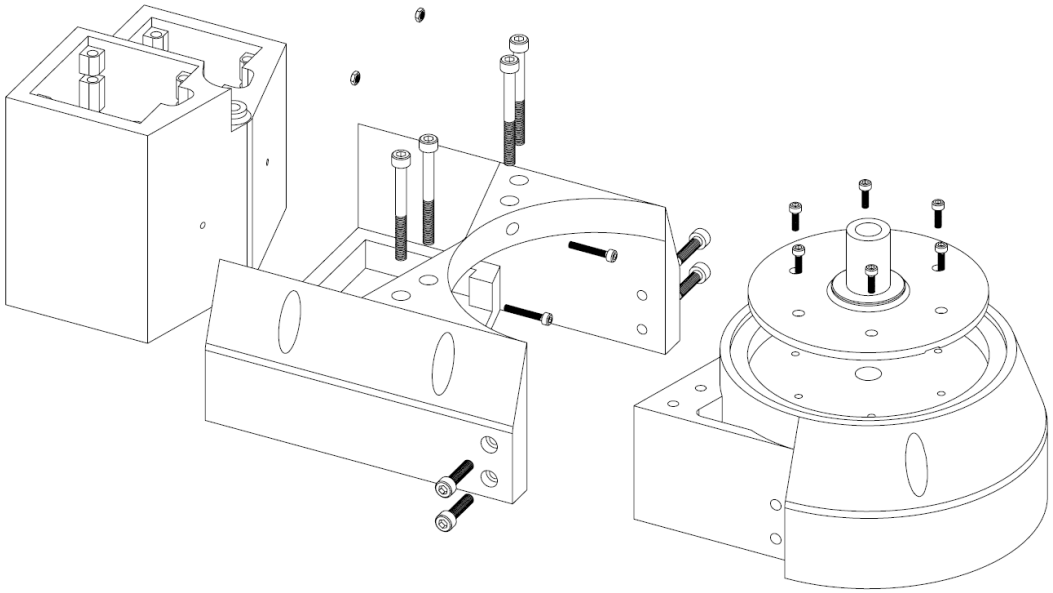


Figura D.1: Vista explosionada del primer nivel de la base.

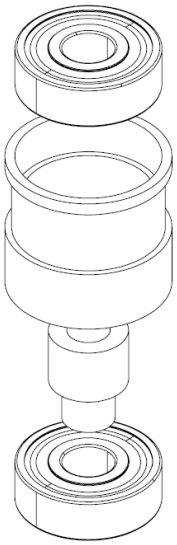


Figura D.2: Vista explosionada de polea guía.

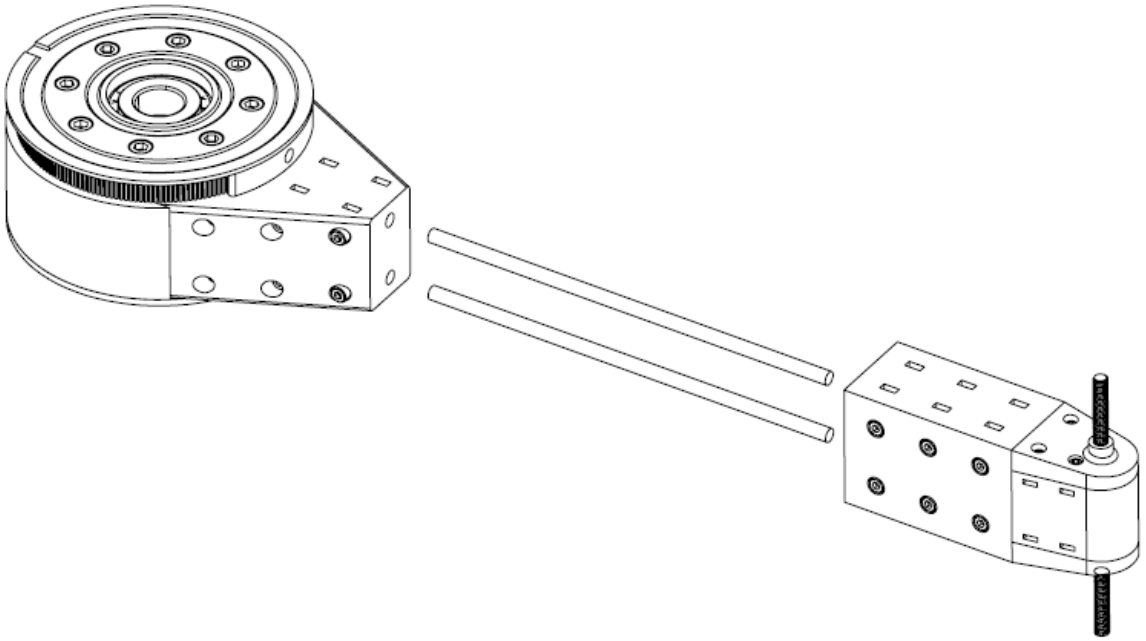


Figura D.3: Vista explosionada de eslabón 11 y 21.

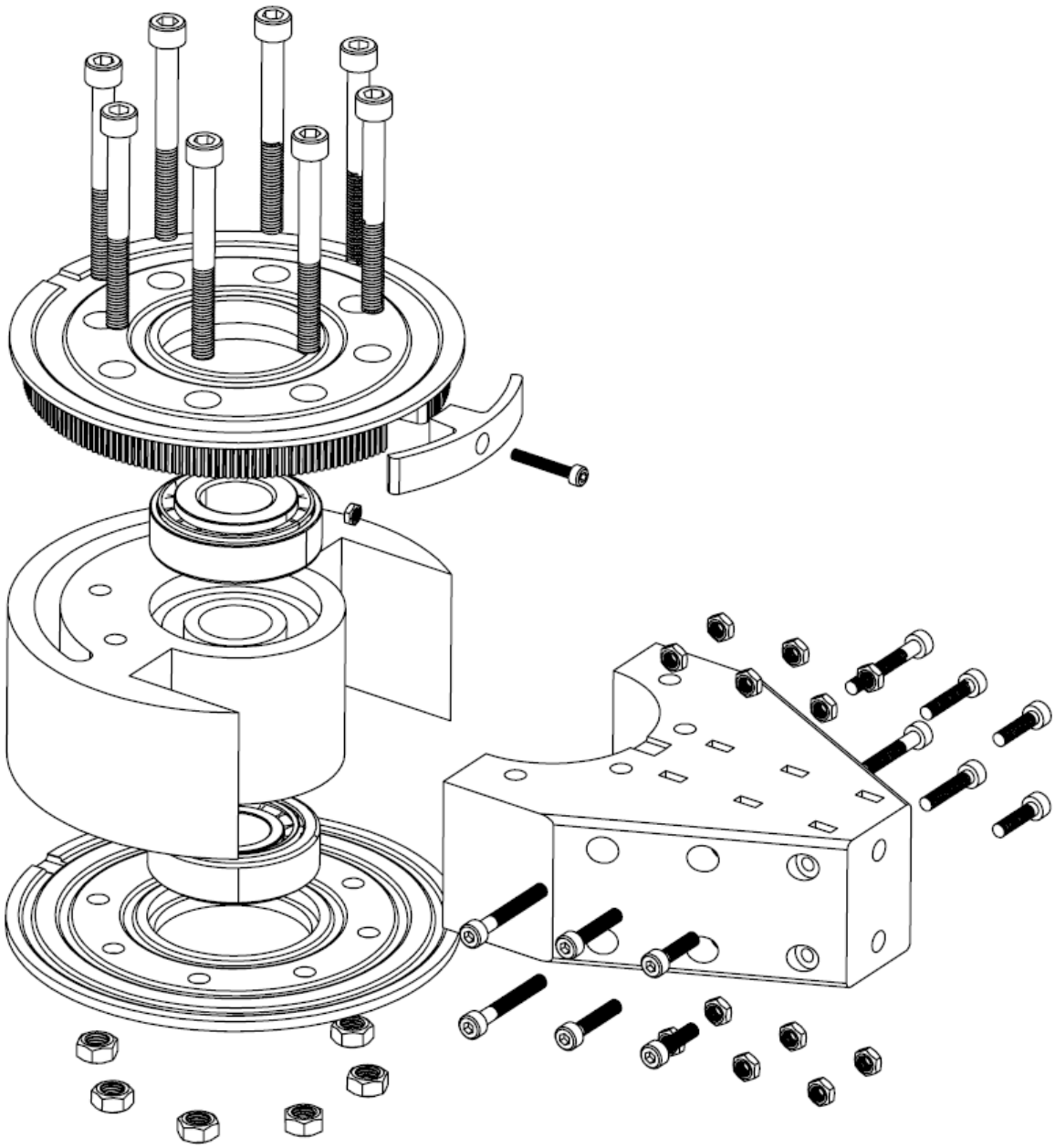


Figura D.4: Vista explosionada del primer cuerpo de los eslabones 11 y 21.

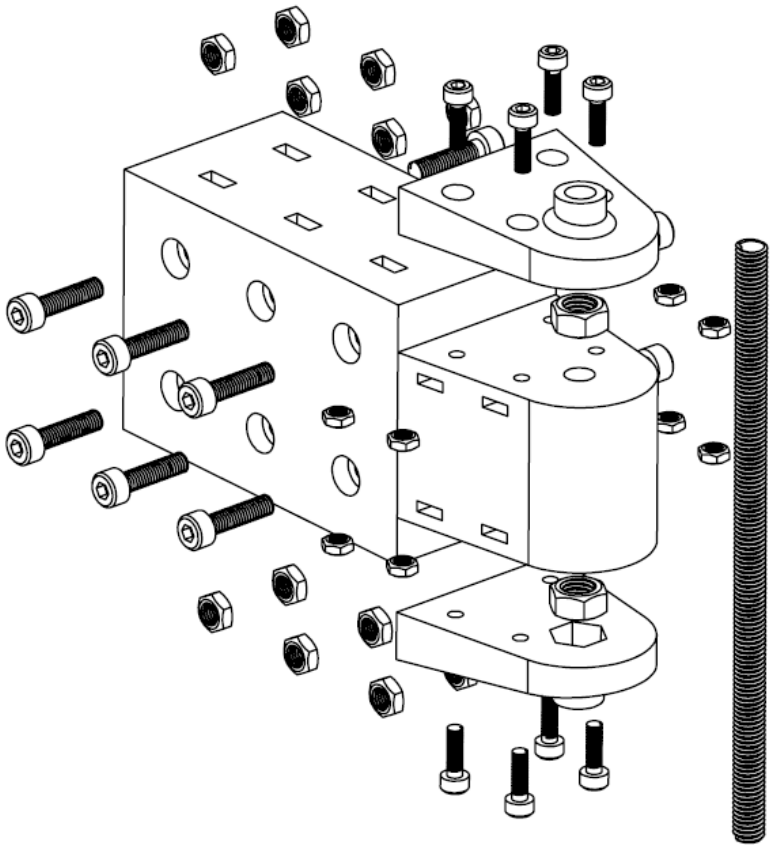


Figura D.5: Vista explosionada del segundo cuerpo de los eslabones 11 y 21.

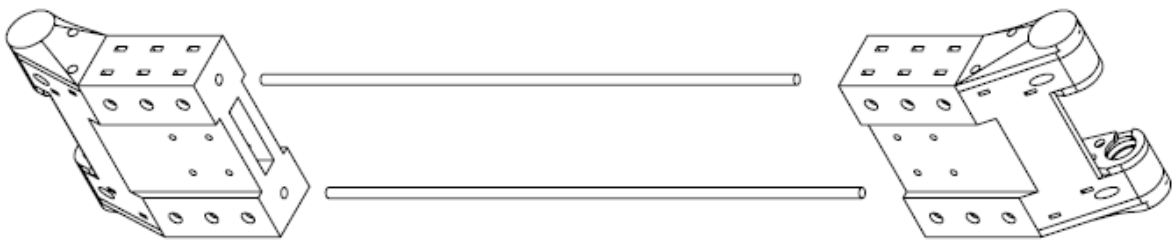


Figura D.6: Vista explosionada de eslabón 12.

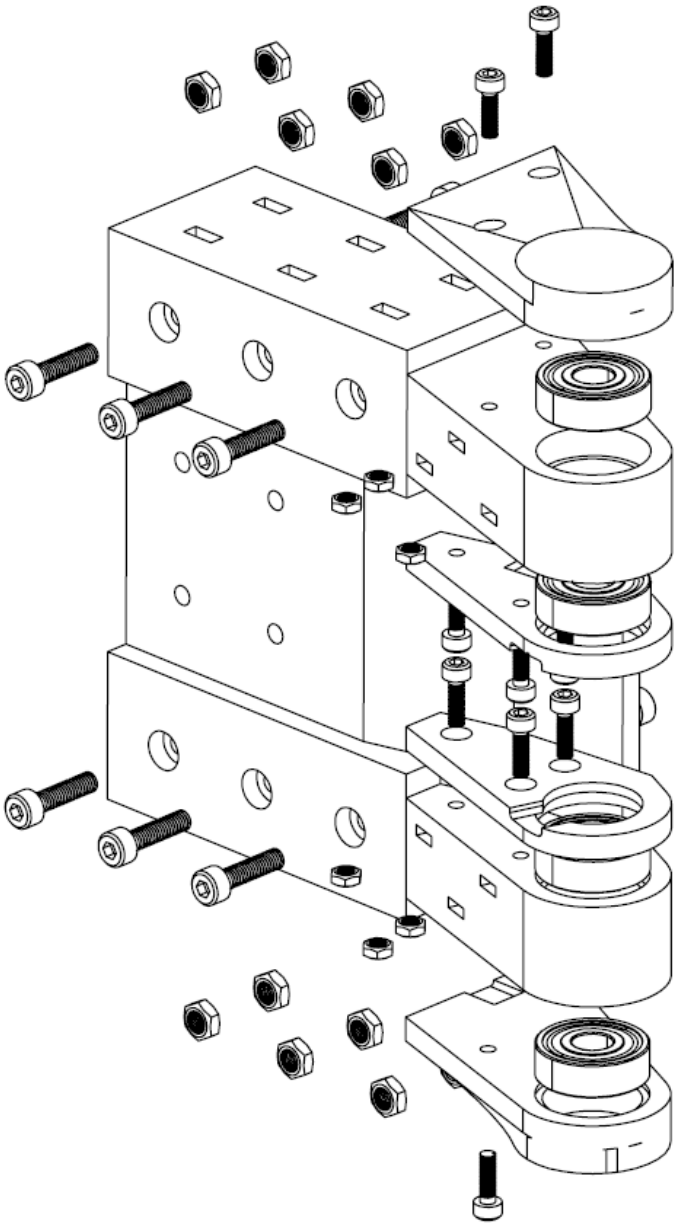


Figura D.7: Vista explosionada de cuerpo de eslabón 12.

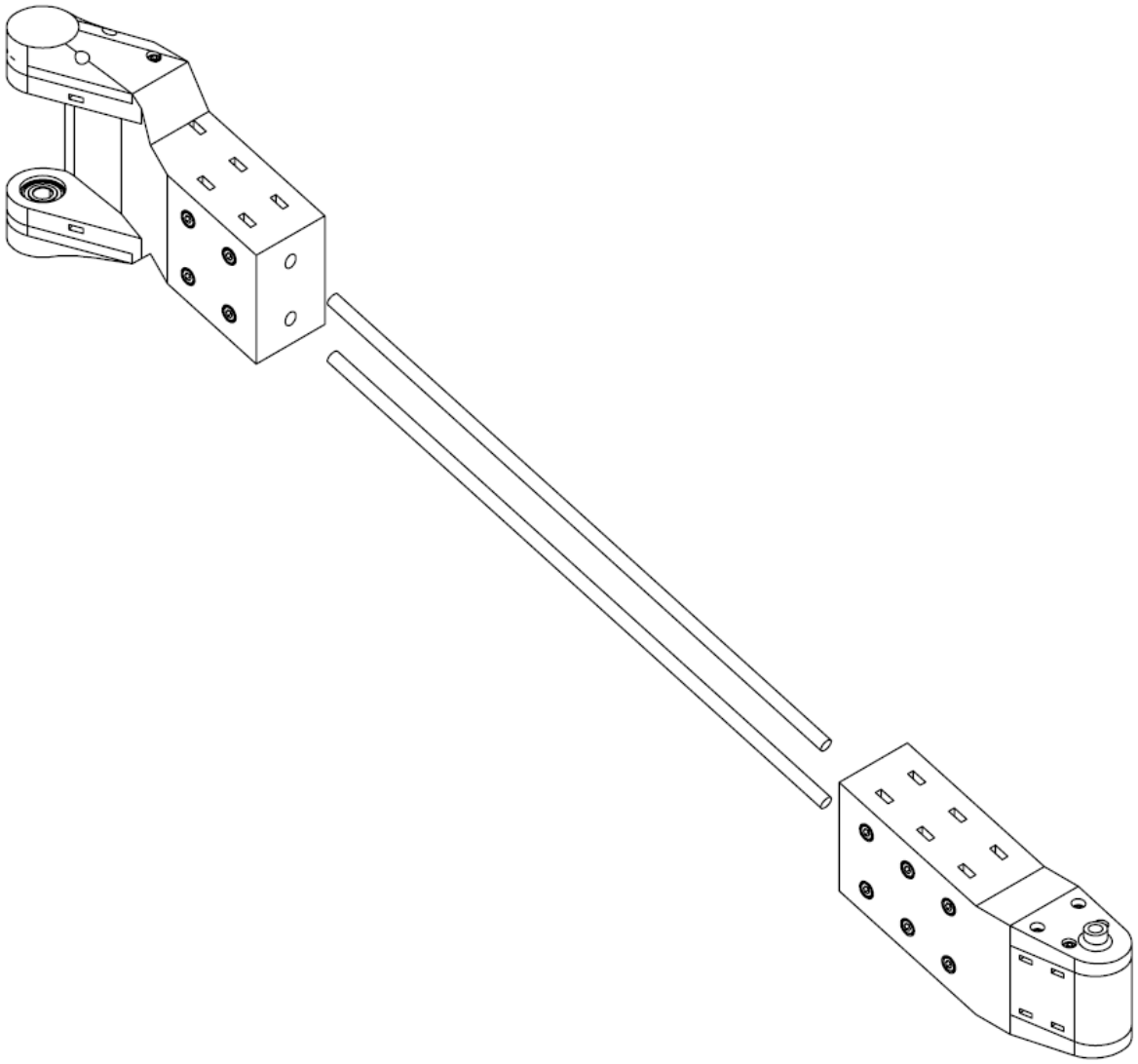


Figura D.8: Vista explosionada de eslabón 22.

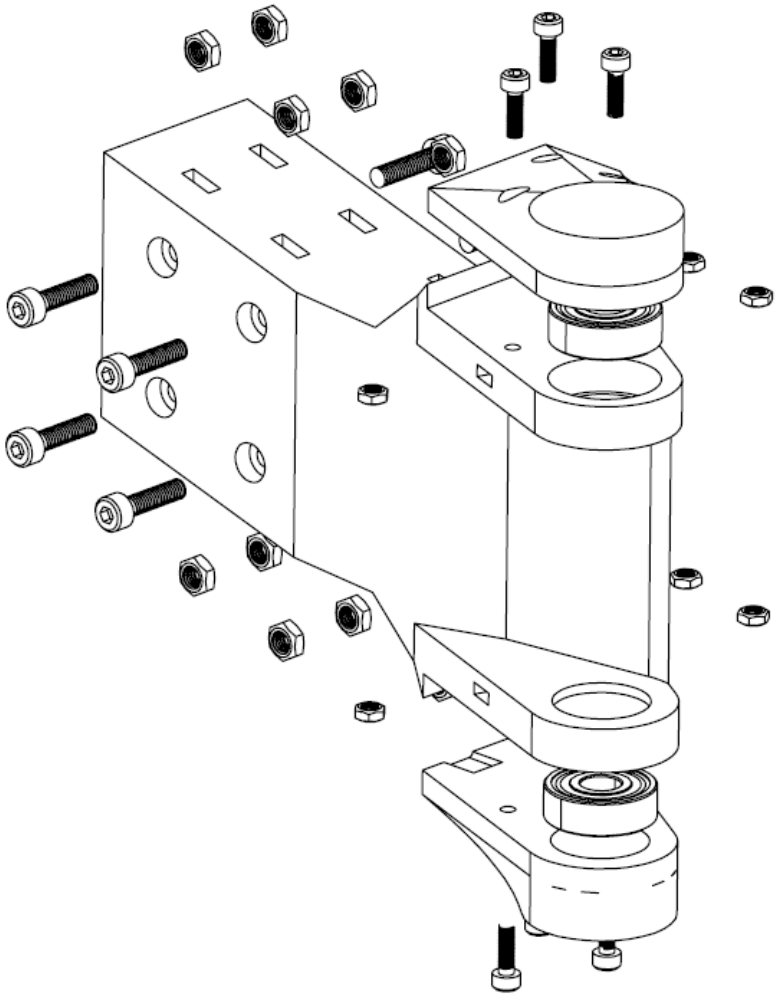


Figura D.9: Vista explosionada del primer cuerpo del eslabón 22.

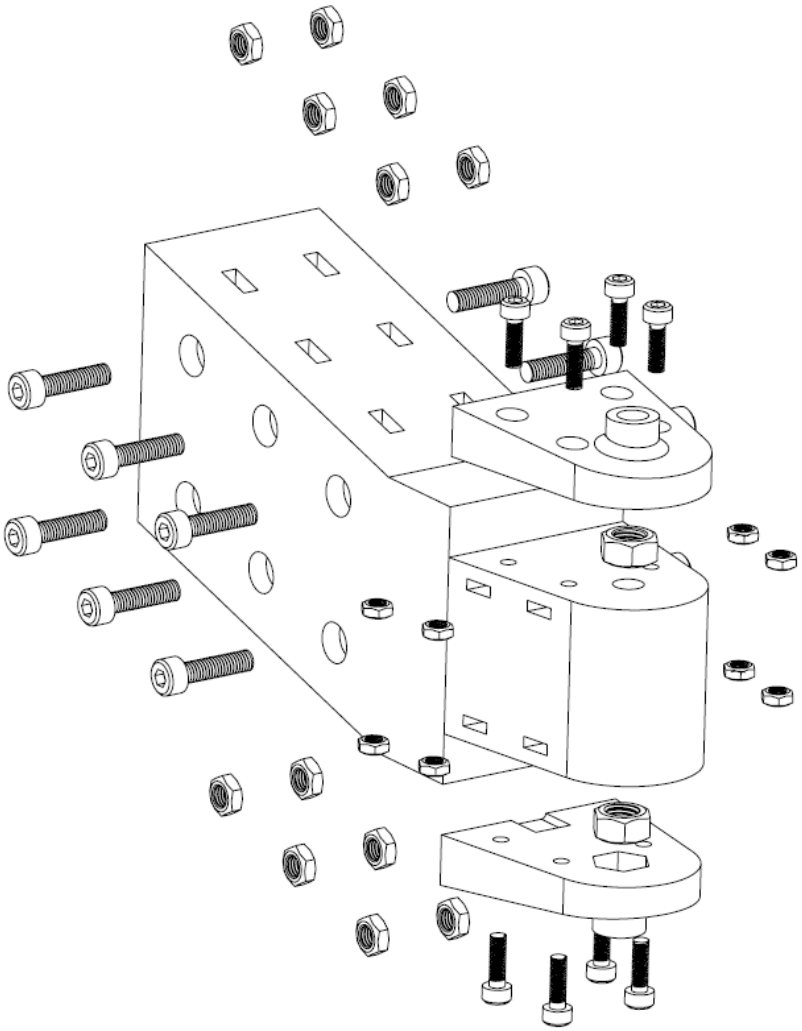


Figura D.10: Vista explosionada del segundo cuerpo del eslabón 22.

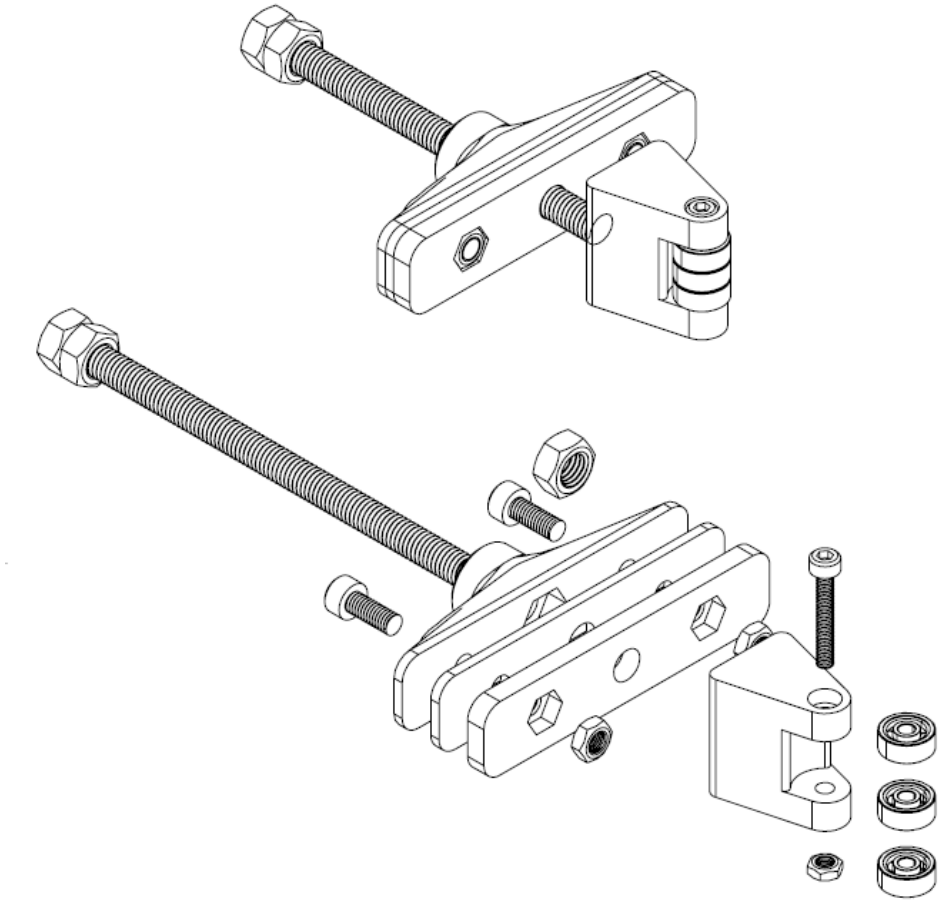


Figura D.11: Vista explosionada de tensor de correa.

Apéndice E

Pasos para ejecutar una trayectoria en el robot

A continuación se describe el procedimiento para correr la aplicación que planifica trayectorias para el robot.

1. Prerrequisitos: antes de comenzar asegúrese de tener instalado los siguientes programas y paquetes *Git*, *Python* y *Pip*, si no los tiene, diríjase a los siguientes links y siga las instrucciones para instalarlos:
   - *Git*: `https://git-scm.com/downloads`
   - *Python*: `https://www.python.org/downloads`
   - *Pip*: `https://pip.pypa.io/en/stable/installation/`
2. Obtener código fuente: este se encuentra en un repositorio *Git*. Para comenzar, clone el repositorio *Git "five-bar-robot"*[1] a una carpeta local. Una vez clonado, diríjase a la carpeta *five-bar-robot/src/motion_planning*. Esto se puede hacer en cualquier terminal con el siguiente comando.

```
$ cd five-bar-robot/src/motion_planning
```

3. Dependencias: una vez en la carpeta, ejecute el siguiente comando en una terminal para instalar las dependencias de nuestro paquete.

```
$ pip install -e .[dev]
```

4. Ejecución del programa: para correr el programa ejecute el siguiente comando en la terminal.

```
$ python3 examples/trajectory.py
```

   Este código generará el archivo *trajectory.csv* con la trayectoria en la carpeta *trajectories*

   Verificar que este conectado el puente *usb-uart*.

5. Enviar consignas al robot: se ejecuta el script *uart-interface.py* el cual es el encargado de leer la trayectoria generada y enviarla al robot.

```
python3 utilities/uart-interface.py
```

[1] `https://github.com/gonzafernan/five-bar-robot`

Apéndice F

Documentación planificador de trayectorias

Five Bar Robot

Release 0.0.x

Gino Avanzini, Gonzalo Fernandez and Jeremías Pino

Jun 15, 2022

CONTENTS:

Five-bar planar parallel robots for pick and place operations.

CHAPTER

ONE

FIVE BAR ROBOT

Five-bar planar parallel robots for pick and place operations.

1.1 Developers information

1.1.1 Install pre-commit

For install pre-commit

```
pip install pre-commit
```

Then after clone this repository and only once.

```
pre-commit install
```

1.1.2 Install

```
cd src/motion_planning
pip install -e .[dev]
```

1.1.3 Create package

```
cd src/motion_planning
python setup.py bdist_wheel sdist
pip install -e .[dev]
```

1.1.4 Documentation

Prerequisites

```
pip install sphinx
pip install sphinx-rtd-theme
pip install myst-parser
pip install sphinxcontrib-spelling
```

Generation

```
cd docs
make clean
make html
```

1.2 Install CoppeliaSim Api

In order to install CoppeliaSim Api follow this steps Then, go to python's install directory <path_to_python>/site-packages. In that directory, create a file name.pth. Finally add the following path <path_to_coppeliasim>/programming/zmqRemoteApi/clients/python into the file.

CHAPTER

TWO

USAGE

2.1 Lines

```
from Path import Path
import numpy as np
import matplotlib.pyplot as plt
from os.path import dirname, abspath
from utilities import normalize_trajectory, export_trajectory

# Create a Path object
path = Path()

# Define some interesting points
st = path.robot.fkine(np.deg2rad([180, 0, 0]))
wp_1 = st + np.array([0.1, 0, 0])
wp_2 = wp_1 + np.array([0, 0.1, 0])
wp_3 = wp_2 + np.array([-0.1, 0., 0])
gl = wp_3 + np.array([0, -0.1, 0])
pose = np.block([[st], [wp_1], [wp_2], [wp_3], [gl]])
max_v = [0.3, 0.3, 0.3, 0.3]
delta_p = np.diff(pose, axis=0)
max_a = [1, 1, 1, 1]

# LINE
q_lin, qd_lin, qdd_lin, p_lin, pd_lin, pdd_lin = path.line(
    pose=pose, max_v=max_v, max_a=max_a, enable_way_point=False)

# Show a trajectory
path.plot_joint(q_lin, qd_lin, qdd_lin)
path.plot_task(p_lin, pd_lin, pdd_lin)
plt.show()

# Normalize desire trajectory
factor = 4 * 300 / np.pi
q = normalize_trajectory(q_lin, factor)

# Save trajectory for send to robot
trj_path = dirname(dirname(abspath(__file__)))
trj_path += '/trajectories'
file_name = 'line.csv'
```

(continues on next page)

(continued from previous page)

```
export_trajectory(q, trj_path, path.total_time, file_name)
```

2.2 Circles

```
from Path import Path
import numpy as np
from os.path import dirname, abspath
from utilities import export_trajectory, normalize_trajectory

# Create a Path object
path = Path()

# CIRCLE
st = path.robot.fkine(np.deg2rad([180, 0, 0]))
c = st + np.array([0., 0.05, 0])

q_cir, qd_cir, qdd_cir, p_cir, pd_cir, pdd_cir = path.circle(start=st,
                                                            center=c,
                                                            max_vel=0.2,
                                                            max_acc=1.)

# Show a trajectory
path.plot_joint(q_cir, qdd_cir, qdd_cir)
path.plot_task(p_cir, pd_cir, pdd_cir)

# Normalize desire trajectory
factor = 8 * 300 / np.pi
q = normalize_trajectory(q_cir, factor)

# Save trajectory for send to robot
trj_path = dirname(dirname(abspath(__file__)))
trj_path += '/trajectories'
file_name = 'circle.csv'

export_trajectory(q, trj_path, path.total_time, file_name)
```

2.3 CoppeliaSim Simulation

```
from FiveBar import FiveBar
import Path as gp
import numpy as np
import math
from zmqRemoteApi import RemoteAPIClient
import time
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
l1 = 0.25
l2 = 0.38
l22 = 0.3878
b = 0.0
five = FiveBar(np.array([-b, l1, l2]), np.array([b, l1, l22]))
path = gp.Path(five)

client = RemoteAPIClient()
sim = client.getObject('sim')
# Get objects
base = sim.getObjectHandle('base')
end = sim.getObjectHandle('end')
joint11 = sim.getObjectHandle('q11')
joint21 = sim.getObjectHandle('q21')

# When simulation is not running, ZMQ message handling could be a bit
# slow, since the idle loop runs at 8 Hz by default. So let's make
# sure that the idle loop runs at full speed for this program:
defaultIdleFps = sim.getInt32Param(sim.intparam_idle_fps)
sim.setInt32Param(sim.intparam_idle_fps, 0)

# Run a simulation in asynchronous mode:
sim.startSimulation()
j1 = sim.getJointPosition(joint11) + math.pi
j2 = sim.getJointPosition(joint21)
# LINE 1
start = path.robot.fkine(np.array([j1, j2, 0]))
point1 = np.array([0., 0.3, 0.])
point2 = np.array([-0.3, 0.3, 0.])
line1 = np.block([[start], [point1], [point2]])
q1, qd1, qdd1, p1, pd1, pdd1 = path.line(pose=line1,
                                         max_v=[0.1, 0.1],
                                         max_a=[0.2, 0.2])
# ARC 1
point3 = np.array([-0.319, 0.304, 0.])
point4 = np.array([-0.35, 0.35, 0.])
q2, qd2, qdd2, p2, pd2, pdd2 = path.arc(start=point2,
                                        way_point=point3,
                                        goal=point4,
                                        max_vel=0.5,
                                        max_acc=1)

# LINE 2
point5 = np.array([-0.35, 0.4, 0.])
line2 = np.block([[point4], [point5]])
q3, qd3, qdd3, p3, pd3, pdd3 = path.line(pose=line2, max_v=0.1, max_a=0.2)

# ARC 2
point6 = np.array([-0.346, 0.419, 0.])
point7 = np.array([-0.3, 0.45, 0.])
q4, qd4, qdd4, p4, pd4, pdd4 = path.arc(start=point5,
                                        way_point=point6,
```

(continues on next page)

(continued from previous page)

```
                                        goal=point7,
                                        max_vel=0.5,
                                        max_acc=1)
# LINE 3
point8 = np.array([0., 0.45, 0.])
line3 = np.block([[point7], [point8]])
q5, qd5, qdd5, p5, pd5, pdd5 = path.line(pose=line3, max_v=0.1, max_a=0.2)

# CIRCLE
c = np.array([0, 0.375, 0])
q6, qd6, qdd6, p6, pd6, pdd6 = path.circle(start=point8,
                                           center=c,
                                           max_vel=0.1,
                                           max_acc=1)

# LINE 4
point9 = np.array([0.3, 0.45, 0.])
line4 = np.block([[point8], [point9]])
q7, qd7, qdd7, p7, pd7, pdd7 = path.line(pose=line4, max_v=0.1, max_a=0.2)

# ARC 3
point10 = np.array([0.319, 0.446, 0.])
point11 = np.array([0.35, 0.4, 0.])
q8, qd8, qdd8, p8, pd8, pdd8 = path.arc(start=point9,
                                        way_point=point10,
                                        goal=point11,
                                        max_vel=0.5,
                                        max_acc=1)

# LINE 5
point12 = np.array([0.35, 0.35, 0.])
line5 = np.block([[point11], [point12]])
q9, qd9, qdd9, p9, pd9, pdd9 = path.line(pose=line5, max_v=0.1, max_a=0.2)

# ARC 4
point13 = np.array([0.346, 0.331, 0.])
point14 = np.array([0.3, 0.3, 0.])
q10, qd10, qdd10, p10, pd10, pdd10 = path.arc(start=point12,
                                              way_point=point13,
                                              goal=point14,
                                              max_vel=0.5,
                                              max_acc=1)

# LINE 6
line6 = np.block([[point14], [point1]])
q11, qd11, qdd11, p11, pd11, pdd11 = path.line(pose=line6, max_v=0.1, max_a=0.2)
p = np.block([[p1], [p2], [p3], [p4], [p5], [p6], [p7], [p8], [p9], [p10],
              [p11]])
q = np.block([[q1], [q2], [q3], [q4], [q5], [q6], [q7], [q8], [q9], [q10],
              [q11]])
n = max(q.shape)
q1_cop = np.zeros(n)
```

(continues on next page)

(continued from previous page)

```
q2_cop = np.zeros(n)
p1_cop = np.zeros(n)
p2_cop = np.zeros(n)

idx = 0
while idx < n:
    sim.setJointPosition(joint11, float(q[idx, 0] - math.pi))
    sim.setJointPosition(joint21, float(q[idx, 1]))
    end_position = sim.getObjectPosition(end, base)
    p1_cop[idx] = end_position[0]
    p2_cop[idx] = end_position[1]
    q1_cop[idx] = sim.getJointPosition(joint11) + math.pi
    q2_cop[idx] = sim.getJointPosition(joint21)
    idx += 1
sim.stopSimulation()
#  We need make sure we really stopped:
while sim.getSimulationState() != sim.simulation_stopped:
    time.sleep(0.1)
#  Restore the original idle loop frequency:
sim.setInt32Param(sim.intparam_idle_fps, defaultIdleFps)

# Show results
plt.figure(1)
plt.plot(np.rad2deg(q[:, 0]), 'r', label='$q_1$')
plt.plot(np.rad2deg(q[:, 1]), 'g', label='$q_2$')
plt.plot(np.rad2deg(q1_cop), 'k--', label='$q_{1}c$')
plt.plot(np.rad2deg(q2_cop), 'k--', label='$q_{2}c$')
plt.ylabel('Joints [°] ')
plt.xlabel('Iterations')
plt.title("Joints Position")
plt.legend()
plt.grid(True)
plt.figure(2)
plt.plot(p[:, 0], 'r', label='x')
plt.plot(p[:, 1], 'g', label='y')
plt.plot(p1_cop, 'k--', label='xc')
plt.plot(p2_cop, 'k--', label='yc')
plt.ylabel('Position [m] ')
plt.xlabel('Iterations')
plt.title("Cartesian Position")
plt.legend()
plt.grid(True)
plt.figure(3)
plt.plot(p[:, 0], p[:, 1], 'r', label='desired')
plt.plot(p1_cop, p2_cop, 'k--', label='coppeliasim')
plt.xlabel('End position X [m] ')
plt.ylabel('End position Y [m]')
plt.title("End effector position")
plt.legend()
plt.grid(True)
plt.show()
```

(continues on next page)

(continued from previous page)

```
print("The averrage error")
pose_x_diff = (p[:, 0] - p1_cop)**2
pose_y_diff = (p[:, 1] - p2_cop)**2
pose_error = np.sqrt(pose_x_diff + pose_y_diff)
q1_diff = q[:, 0] - q1_cop
q2_diff = q[:, 1] - q2_cop

plt.figure(4)
plt.plot(pose_error)
plt.grid(True)
plt.show()

print("Pose mean error:", pose_error.mean())
print("Pose max error:", max(pose_error))
print("Pose min error:", min(pose_error))
print("Q1 mean error:", q1_diff.mean())
print("Q1 max error:", max(q1_diff))
print("Q1 min error:", min(q1_diff))
print("Q2 mean error:", q2_diff.mean())
print("Q2 max error:", max(q2_diff))
print("Q2 min error:", min(q2_diff))
```

CHAPTER

THREE

MOTION_PLANNING

3.1 motion_planning package

3.1.1 Submodules

3.1.2 motion_planning.FiveBar module

class motion_planning.FiveBar.**Arm**(*d: numpy.ndarray*, *wkMode: int*)

Bases: object

This class contain the representation o a 2 dof planar robotic arm

links
2X1 proximal and distal link length.

Type np.ndarray

working
Current working mode of the arm.

Type int

base
distance form the origin to the base of the arm.

Type double

rOA
2x1 point in task space that represent the union between the proximal and distal links.

Type np.ndarray

class motion_planning.FiveBar.**FiveBar**(*d1: numpy.ndarray = array([0., 0.25, 0.45])*, *d2: numpy.ndarray = array([0., 0.25, 0.45])*)

Bases: object

This class implement a model of a five bars robot also know as scara parallel robot. Units are in meters [m] and radians [rad].

arms
2X1 list that contain the 2 dof arms.

Type list(*Arm*)

endEff
3X1 vector that represent the end effector pose [x,y,z].

Type np.ndarray

assembly
Current assembly mode.

Type int

joints
joints position in radians

Type np.ndarray

are_distals_ok()

are_inside_limits(*q1*, *q2*)

are_proximals_ok(*q1*, *q2*)

fkine(*joint: numpy.ndarray = 0*) → numpy.ndarray
Five Bar forward kinematics. If non parameter is passed the current position of the robot

Parameters **q** (*np.ndarray*) – 3X1 vector. Joint position for achieve desire pose [q1, q2, d3]

Returns p 3x1 vector. Desire end effector pose [x, y, z]

Return type np.ndarray

ikine(*pose: numpy.ndarray = 0*) → numpy.ndarray
Five Bar inverse kinematics. If non parameter is passed the current position of the robot

Parameters **pose** (*np.ndarray*) – 3x1 vector. Desire end effector pose [x, y, z]

Returns q 3X1 vector. Joint position for achieve desire pose [q1, q2, d3]

Return type np.ndarray

is_inside_bounds(*q1*, *q2*)

showRobot()
Show robot with numpy

transform_angle(*q1*, *q2*)
Normalize angles q1 and q2

work_space()

motion_planning.FiveBar.**main**()

3.1.3 motion_planning.Path module

class motion_planning.Path.**Path**(*robot: FiveBar.FiveBar = <FiveBar.FiveBar object>*)
Bases: object

Path class

arc(*start*, *way_point*, *goal*, *max_vel*, *max_acc*)
Arc in task space Interpolation method: linear segment with parabolic blend

Parameters

- **start** (*np.ndarray*) – start point 1X3
- **way_point** (*np.ndarray*) – way point 1X3
- **goal** (*np.ndarray*) – goal point 1X3
- **max_v** (*float*) – max velocity

- **max_a** (*float*) – max acceleration

Returns

A tuple containing, respectively

q (np.array) joint position n x 3.

qd (np.array) joint velocity n x 3.

qdd (np.array) joint acceleration n x 3.

p (np.array) task position n x 3.

pd (np.array) task velocity n x 3.

pdd (np.array) task acceleration n x 3.

circle(*start*, *center*, *max_vel*, *max_acc*)

Circle in task space Interpolation method: linear segment with parabolic blend

Parameters

- **start** (*np.ndarray*) – start point 1X3
- **center** (*np.ndarray*) – circle center point 1X3
- **max_v** (*float*) – max velocity
- **max_a** (*float*) – max acceleration nX

Returns

A tuple containing, respectively

q (np.array) joint position n x 3.

qd (np.array) joint velocity n x 3.

qdd (np.array) joint acceleration n x 3.

p (np.array) task position n x 3.

pd (np.array) task velocity n x 3.

pdd (np.array) task acceleration n x 3.

class circle_data(*center: list[float]*, *radius: float*, *start: list[float]*, *delta: list[float]*)

Bases: object

center: list[float]

delta: list[float]

radius: float

start: list[float]

circle_interpolation(*circle_data*, *max_vel*, *max_acc*)

circular_path(*arc_length*, *radius*)

go_to(*goals: numpy.ndarray*, *max_v: float*, *max_a: float*, *way_point: bool = True*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Go to goal with linear segment with parabolic blend in joint space

Parameters

- **goal** (*np.ndarray*) – goal point 3X

- **max_v** (*np.ndarray*) – max velocity
- **max_a** (*np.ndarray*) – max acceleration
- **way_point** (*bool*) – enable use internal point like way points. Default = True

Returns

A tuple containing, respectively

q (np.array) joint position n x 3.

qd (np.array) joint velocity n x 3.

qdd (np.array) joint acceleration n x 3.

p (np.array) task position n x 3.

pd (np.array) task velocity n x 3.

pdd (np.array) task acceleration n x 3.

go_to_poly(*start: numpy.ndarray*, *goal: numpy.ndarray*, *mean_v: float*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Quintic polynomial interpolation in joint space

Parameters

- **start** (*np.array*) – start point task space 3x
- **goal** (*np.array*) – goal point task space 3x
- **mean_v** (*float*) – mean velocity in joint space

Returns

A tuple containing, respectively

q (np.array) joint position n x 3.

qd (np.array) joint velocity n x 3.

qdd (np.array) joint acceleration n x 3.

p (np.array) task position n x 3.

pd (np.array) task velocity n x 3.

pdd (np.array) task acceleration n x 3.

initialize(*n: int*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Initialize 4 matrix nX3 with zeros

Parameters **n** (*int*) – number of elements

Returns A tuple with the 4 matrix nX3

line(*pose: numpy.ndarray*, *max_v: numpy.ndarray*, *max_a: numpy.ndarray*, *enable_way_point: bool = True*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Lineal interpolation in task space Interpolation method: linear segment with parabolic blend

Parameters

- **pose** (*np.ndarray*) – goals positions point nX3
- **max_v** (*np.ndarray*) – max velocity nX
- **max_a** (*np.ndarray*) – max acceleration nX

- **enable_way_point** (*bool*) – Use each intermediate points like a way point. Default = True

Returns

A tuple containing, respectively

q (np.array) joint position n x 3.

qd (np.array) joint velocity n x 3.

qdd (np.array) joint acceleration n x 3.

p (np.array) task position n x 3.

pd (np.array) task velocity n x 3.

pdd (np.array) task acceleration n x 3.

line_poly(*start: numpy.ndarray*, *goal: numpy.ndarray*, *mean_v: float*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Lineal interpolation in task space Interpolation method: quintic polynomial

Parameters

- **start** (*np.array*) – start point 3X1
- **goal** (*np.array*) – goal point 3X1
- **mean_v** (*float*) – mean velocity

Returns

A tuple containing, respectively

q (np.array) joint position n x 3.

qd (np.array) joint velocity n x 3.

qdd (np.array) joint acceleration n x 3.

p (np.array) task position n x 3.

pd (np.array) task velocity n x 3.

pdd (np.array) task acceleration n x 3.

move_from_end(*goal: numpy.ndarray = array([0., 0., 0.])*, *max_v: float = 1*, *max_a: float = 0.5*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Move relative to end effector frame in task space. Linear segment with parabolic blend

Parameters

- **goal** (*np.ndarray*) – goal point 3X
- **max_v** (*np.ndarray*) – max velocity
- **max_a** (*np.ndarray*) – max acceleration

Returns

A tuple containing, respectively

q (np.array) joint position n x 3.

qd (np.array) joint velocity n x 3.

qdd (np.array) joint acceleration n x 3.

p (np.array) task position n x 3.

pd (np.array) task velocity n x 3.

pdd (np.array) task acceleration n x 3.

move_x_from_end(*x: float = 0.05*, *max_v: float = 1*, *max_a: float = 0.5*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Move relative to end effector frame, x direction in task space. Linear segment with parabolic blend

Parameters

- **pose** (*float*) – relative positions to move
- **max_v** (*float*) – max velocity
- **max_a** (*float*) – max acceleration

Returns

A tuple containing, respectively

q (np.array) joint position n x 3.

qd (np.array) joint velocity n x 3.

qdd (np.array) joint acceleration n x 3.

p (np.array) task position n x 3.

pd (np.array) task velocity n x 3.

pdd (np.array) task acceleration n x 3.

move_y_from_end(*y: float = 0.05*, *max_v: float = 1*, *max_a: float = 0.5*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Move relative to end effector frame, y direction in task space. Linear segment with parabolic blend

Parameters

- **pose** (*float*) – relative positions to move
- **max_v** (*float*) – max velocity
- **max_a** (*float*) – max acceleration

Returns

A tuple containing, respectively

q (np.array) joint position n x 3.

qd (np.array) joint velocity n x 3.

qdd (np.array) joint acceleration n x 3.

p (np.array) task position n x 3.

pd (np.array) task velocity n x 3.

pdd (np.array) task acceleration n x 3.

move_z_from_end(*z: float = 0.05*, *max_v: float = 1*, *max_a: float = 0.5*) → tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Move relative to end effector frame, z direction in task space. Linear segment with parabolic blend

Parameters

- **pose** (*float*) – relative positions to move

- **max_v** (*float*) – max velocity
- **max_a** (*float*) – max acceleration

Returns

A tuple containing, respectively

q (np.array) joint position n x 3.

qd (np.array) joint velocity n x 3.

qdd (np.array) joint acceleration n x 3.

p (np.array) task position n x 3.

pd (np.array) task velocity n x 3.

pdd (np.array) task acceleration n x 3.

multi_point_interpolation(*start*, *delta_p*, *t*, *tau*, *T*, *shift*, *enable_way_point*)

Auxiliary method: line interpolation from start upto start + delta_p. Going through many points

Parameters

- **start** – start point
- **delta_p** – distance to interpolate
- **t** – current time
- **tau** – acceleration time
- **T** – deceleration time
- **shift** – time in which the current segment start
- **enable_way_point** – flag for use all the intermediate points as way points

Returns

A tuple containing, respectively

next_point true when the segment interpolation has finished

x pose across the path

xd velocity across the path

xdd acceleration across the path

plot_joint(*q: numpy.ndarray*, *qd: numpy.ndarray*, *qdd: numpy.ndarray*) → None

Plot joints values

Parameters

- **q** (*np.array*) – joints position
- **qd** (*np.array*) – joints velocity
- **qdd** (*np.array*) – joints acceleration

Returns:

Note: total_time must be the sum of all trajectories times

---

plot_task(*p: numpy.ndarray*, *pd: numpy.ndarray*, *pdd: numpy.ndarray*) → None

Plot cartesian values

Parameters

- **p** (*np.array*) – cartesian position
- **pd** (*np.array*) – cartesian velocity
- **pdd** (*np.array*) – cartesian acceleration

Returns:

Note: total_time must be the sum of all trajectories times

---

point_interpolation(*start*, *delta_x*, *s*, *sd*, *sdd*)

Auxiliary method: Line interpolation from start upto start + delta_x. This function return a point, its velocity and its acceleration along the path.

Parameters

- **start** – start point
- **delta_x** – distance to interpolate
- **s** – s variable used for getting the position across the path
- **sd** – s derivative variable used for getting the velocity across the path
- **sdd** – sd derivative used for getting the acceleration across the path

Returns

A tuple containing, respectively

x pose across the path

xd velocity across the path

xdd acceleration across the path

transform_angle(*angle*)

3.1.4 motion_planning.TimeLaw module

class motion_planning.TimeLaw.**TimeLaw**

Bases: object

Class that implement time laws

lspb(*t: float*, *tau: float*, *T: float*) → tuple[float, float, float]

Trapezoidal time law or Linear segment with parabolic blend

Parameters

- **t** (*float*) – current time
- **T** (*float*) – deceleration time
- **tau** (*float*) – acceleration time

Returns

A tuple containing floats, respectively

sd(t) value.

sdd(t) value.

sdd(t) value.

lspb_param(**args*, ***kwargs*) → tuple
Trapezoidal timing law parameters or linear segment with parabolic blend

Parameters

- **delta_q** (*float*) – total displacement.
- **max_v** (*float*) – maximal velocity.
- **max_a** (*float*) – maximal acceleration.
- **dt** (*float, optional*) – step time for round values. Defaults to 0.01

Returns

A tuple containing, respectively

tau (float) acceleration time

T (float) Time to start deceleration

Note: If a trapezoidal timing law can not be reached, with the parameters, this function keep doing a trapezoidal timing law. This approach is achieve by choosing :math:` au = sqrt{Delta q / a_{max}}` triangular law # noqa and $T = v_{max}/a_{max}$. This keep a trapezoidal law

---

lspb_s(*t: float*, *tau: float*, *T: float*) → float
Trapezoidal time law or Linear segment with parabolic blend. Range from [0, 1]

Parameters

- **t** (*float*) – current time
- **T** (*float*) – time to start deceleration
- **tau** (*float*) – acceleration time

Returns float s(t) value.

lspb_sd(*t: float*, *tau: float*, *T: float*) → float
Derivative trapezoidal time law or linear segment with parabolic blend

Parameters

- **t** (*float*) – current time
- **T** (*float*) – deceleration time
- **tau** (*float*) – acceleration time

Returns float sd(t) value.

lspb_sdd(*t: float*, *tau: float*, *T: float*) → float
Second derivative trapezoidal time law or Linear segment with parabolic blend

Parameters

- **t** (*float*) – current time
- **T** (*float*) – deceleration time
- **tau** (*float*) – acceleration time

Returns float sdd(t) value.

poly(*t: float*, *a: numpy.array*) → tuple[float, numpy.array]

Quintic polynomial coefficient A quintic (5th order) polynomial is used with default zero boundary conditions for velocity and acceleration.

$$s(t) = a5 * t^5 + a4 * t^4 + a3 * t^4 + a2 * t^2 + a1 * t + a0$$
$$sd(t) = 5 * a5 * t^4 + 4 * a4 * t^3 + 3 * a3 * t^2 + 2 * a2 * t^1 + a1$$
$$sdd(t) = 20 * a5 * t^3 + 12 * a4 * t^2 + 6 * a3 * t + 2 * a2$$

Parameters

- **t** (*float*) – current time
- **a** (*np.array*) – coefficients

Returns

A tuple containing floats, respectively

s(t) value

sd(t) value

sdd(t) value

poly_coeff(*qi: float*, *qf: float*, *tf: float = 1.0*, *vi: float = 0.0*, *vf: float = 0.0*, *ai: float = 0.0*, *af: float = 0.0*) → numpy.ndarray

Quintic polynomial coefficient. A quintic (5th order) polynomial is used with default zero boundary conditions for velocity and acceleration.

$$s(t) = a5 * t^5 + a4 * t^4 + a3 * t^4 + a2 * t^2 + a1 * t + a0$$

Parameters

- **qi** (*float*) – initial pose
- **qf** (*float*) – final pose
- **tf** (*float*) – final time. Default to 1.
- **vi** (*float, optional*) – initial velocity. Default to 0
- **vf** (*float, optional*) – final velocity. Default to 0
- **ai** (*float, optional*) – initial acceleration. Default to 0
- **af** (*float, optional*) – final acceleration. Default to 0

Returns np.ndarray coefficients [a0, a1, a2, a3, a4, a5]

3.1.5 motion_planning.utilities module

motion_planning.utilities.**encode_trajectory**(*points*)

motion_planning.utilities.**export_trajectory**(*vector*, *path*, *final_time*, *name='trajectory.csv'*)

Export pose to a csv

Parameters

- **pose** (*np.ndarray*) – nx3 with positions, velocities or acceleration
- **path** – path to a directory where the file will be saved
- **name** – name of the file, must contain the extension .csv. Default trajectory.csv

motion_planning.utilities.**main**()

motion_planning.utilities.**normalize_trajectory**(*trajectory*, *factor*)

motion_planning.utilities.**read_csv**(*file*)

3.1.6 Module contents

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

ÍNDICE

N

P

R

S

T

W