UNIVERSITAT POLITÈCNICA DE CATALUNYA

Programa de Máster

AUTOMÁTICA Y ROBÓTICA

Trabajo Final de Máster

PLANIFICACIÓN Y EJECUCIÓN DE TRAYECTORIAS LIBRES DE SINGULARIDADES EN ROBOTS PARALELOS 3-RRR

Alejandro Rajoy Niubó

Directores:

Oriol Bohigas y Lluís Ros

Octubre 2015

Planificación y ejecución de trayectorias libres de singularidades en robots paralelos 3-RRR

por Alejandro Rajoy Niubó

Este trabajo se ha presentado en la Universitat Politècnica de Catalunya para la obtención del título de Máster

Programa de Máster:
Automática y Robótica

Este trabajo se ha realizado en el:
Institut de Robòtica i Informàtica Industrial, CSIC-UPC

Directores:
Oriol Bohigas y Lluís Ros

Tribunal evaluador:
Dr. Federico Thomas Arroyo
Dr. Ramón Costa Castelló
Dr. Lluís Ros Giralt
La última versión de este documento se puede descargar de http://www.iri.upc.edu/people/arajoy.

PLANIFICACIÓN Y EJECUCIÓN DE TRAYECTORIAS
LIBRES DE SINGULARIDADES EN ROBOTS PARALELOS 3-RRR

Alejandro Rajoy Niubó

Resumen

Este proyecto desarrolla un sistema que permite evaluar experimentalmente un método de planificación de movimientos libres de singularidades propuesto por Bohigas y col. en 2013. El sistema incluye (1) un robot paralelo 3-RRR, (2) un planificador que calcula trayectorias libres de singularidades y colisiones entre dos configuraciones dadas del robot, (3) un sistema de control que permite ejecutar las trayectorias con bajo error de posicionado, y (4) una interfaz gráfica que facilita la especificación de los problemas a resolver, así como la simulación, ejecución y análisis de las trayectorias resultantes.

Para calcular las trayectorias definimos un sistema de ecuaciones que integra todas las restricciones de exclusión de singularidades y colisiones, y exploramos el espacio solución de este sistema utilizando métodos numéricos de continuación multidimensional. La integración de todas las restricciones en un único sistema evita la utilización de detectores de colisión, permitiendo el cálculo de las trayectorias en períodos de tiempo generalmente muy breves. El proyecto describe detalladamente los subsistemas de planificación, ejecución y control, e ilustra su funcionamiento mediante un experimento práctico que abarca todo el proceso, desde la definición de las configuraciones inicial y final, hasta la síntesis, acondicionamiento, y ejecución de la trayectoria, así como la captura y análisis de los datos sensoriales del robot a lo largo de todo el movimiento. El proyecto, finalmente, identifica algunos puntos que se podrían considerar en futuras extensiones de este trabajo.

Agradecimientos

Me gustaría agradecer a Lluís Ros su dedicación y soporte, durante la realización del máster, como profesor, y en esta última etapa, como director de este trabajo fin de máster. Sin su aportación, el resultado sería muy diferente al presentado. También quiero dar las gracias a muchas otras personas que, de un modo u otro han contribuido a este trabajo. A Oriol Bohigas, por su codirección y guía en la primera etapa. A Montserrat Manubens, por haber proporcionado la financiación necesaria, y por realizar los pasos iniciales que han permitido la realización de este proyecto. A Patrick Grosch, por el diseño, realización y ensamblaje de la elementos mecánicos, así como por dar solución a los problemas que fueron apareciendo. A Josep Maria Porta por su ayuda en la parte de planificación y filtrado de caminos. A Jordi Riera por sus clarificaciones y aportaciones en la parte de control. Y por último, pero no menos importante, a mi familia y amigos por su comprensión y apoyo.

Índice general

Lista de figuras

Lista de tablas

1 Introducción

1.1 Motivación

La Robótica se encuentra en una etapa de gran expansión. La aparición de mecanismos innovadores es continua, y es previsible que su uso se incremente notablemente en múltiples ámbitos en un futuro próximo. Ya sea en laboratorios de investigación, en medicina, o en la industria, encontramos robots paralelos, dispositivos hápticos, vehículos aéreos no tripulados, manos antropomórficas, humanoides, y otros dispositivos robóticos, realizando funciones que difícilmente podríamos haber imaginado hace unas décadas. La capacidad de ejecutar movimientos complejos de forma precisa y robusta es un aspecto clave en estas máquinas, pero el análisis y la planificación de tales movimientos no es un proceso trivial. Requiere una comprensión profunda de unas configuraciones, denominadas *singulares*, en las que el rendimiento cinetostático del sistema se reduce drásticamente. Cerca de estas configuraciones, el robot puede sufrir pérdidas de destreza o controlabilidad, y puede no ser capaz de equilibrar o regular las fuerzas que ejerce sobre el entorno. Un buen planificador de movimientos, debe mantener al robot alejado de tales configuraciones, evitando especialmente aquellas que comprometan la propia integridad del mecanismo.

Hasta hace poco, la literatura robótica no ofrecía métodos satisfactorios para planificar movimientos libres de singularidades, pero resultados recientes del grupo de Cinemática y Diseño de Robots del IRI permiten ahora tratar este problema con gran generalidad [1]. Este grupo ha desarrollado un método capaz de obtener caminos libres de singularidades en mecanismos robóticos de geometría arbitraria. Dadas dos configuraciones no singulares del mecanismo, el método permite encontrar un camino que las conecta, si éste existe, manteniendo un margen de seguridad con respecto a las singularidades en todo momento. El desarrollo teórico de este método está consolidado, pero hasta el momento sólo se ha probado en simulación. La motivación principal de este proyecto es evaluar su funcionamiento también experimentalmente.

1.2 Objetivo

El objetivo de este proyecto es construir un sistema robótico que permita evaluar experimentalmente el método de planificación de movimientos libres de singularidades propuesto en [1]. Para satisfacer este objetivo desarrollaremos:

1. Un robot físico que sirva para validar el método de planificación.

2. Un planificador capaz de calcular trayectorias libres de singularidades y colisiones entre dos configuraciones dadas del robot.

3. Un sistema de control que permita ejecutar las trayectorias sobre el robot, incurriendo en el menor error de trazado posible.

4. Una interfaz gráfica que facilite la especificación de los problemas de planificación a resolver, la ejecución, y el análisis de las trayectorias resultantes.

1.3 Alcance y asunciones

A continuación detallamos el alcance del proyecto, y las hipótesis de trabajo que se han tenido en cuenta en cada uno de los puntos anteriores.

1.3.1. El robot físico

Se ha optado por construir un robot 3-RRR porque su estructura mecánica es lo suficientemente sencilla (Fig. 1.1 y Sección 2.2), a la vez que presenta singularidades de gran complejidad (Capítulo 2). La planificación de movimientos libres de singularidades en este tipo de robots es, de hecho, un problema abierto de la cinemática computacional, lo cual supone un reto sustancial para este proyecto.

La construcción del robot incluye el diseño y montaje de su esqueleto mecánico, la selección e instalación de los sensores y actuadores, y la implementación de toda la electrónica y los algoritmos asociados para la adquisición de señales y ejecución de instrucciones. La financiación disponible para esta construcción ha sido limitada, motivo por el cual se han centrado los esfuerzos en obtener un prototipo de bajo coste que sea suficiente para las necesidades del proyecto. Los gastos de construcción se han sufragado mediante fondos del proyecto intramural CSIC “CUIK-IT: Un sistema modular para el análisis cinemático y la construcción de manipuladores planares”, del Instituto de Robótica e Informática Industrial.

Figura 1.1: El robot 3-RRR construido.

Para la construcción del robot físico se ha contado con el apoyo del Servicio de Diseño Mecatrónico del IRI. El desarrollo del esqueleto mecánico ha ido a cargo de este servicio íntegramente, incluyendo su diseño, la selección de materiales y elementos mecánicos, la impresión estereolitográfica de varias piezas, y el montaje y ensamblado final. Los aspectos electrónicos o electromecánicos del robot, en cambio, se han resuelto íntegramente como parte de este

trabajo fin de máster. Éstos incluyen la selección, conexión y programación de los sensores, los actuadores, los elementos de transmisión de señales, y las placas de control utilizadas.

1.3.2. El planificador de trayectorias

El planificador debe generar trayectorias libres de singularidades. En un robot paralelo existen, en general, dos tipos de singularidades: las directas, que dan lugar a pérdidas de control y/o incapacidad para soportar fuerzas externas, y las inversas, que implican una pérdida de destreza en el efector final. El sistema desarrollado se ha centrado en evitar sólo las primeras porque son, en primera instancia, las perjudiciales para el robot. Como se explica en la Sección 2.3.2, el segundo tipo de singularidades también se podría evitar si se quisisera, implicando pocas modificaciones en el planificador desarrollado.

El planificador, además, debe evitar configuraciones en las que el robot entre en colisión consigo mismo. Aunque por simplicidad no se han tratado las colisiones del robot con su entorno, éstas se podrían incorporar siguiendo las indicaciones dadas en la Sección 2.4.3.

Utilizaremos la estrategia de planificación descrita en [1], con algunas mejoras tomadas de [2], ya que el propósito de este trabajo es verificar el funcionamiento de esta estrategia experimentalmente. La estrategia es aplicable a robots de geometría arbitraria, pero los programas sólo se prepararan para calcular trayectorias en el caso del robot 3-RRR construido. El tratamiento de robots con estructuras alternativas requeriría generalizar aspectos como la obtención del sistema de ecuaciones, la consideración de colisiones, o la visualización de resultados, quedando todo ello fuera del ámbito de este trabajo.

1.3.3. El sistema de control

Para simplificar el proyecto, se asumirá que los actuadores del robot se mueven con velocidades y aceleraciones bajas. Puesto que el robot se mantendrá alejado de singularidades directas, sus eslabones se moverán, normalmente, con velocidades y aceleraciones también bajas. Por tanto, las fuerzas de inercia del mecanismo se podrán considerar bajas o despreciables, y asimilables a perturbaciones que el sistema de control debe de poder compensar.

El robot se construirá mediante elementos ligeros, y en la mayoría de casos trabajará en un plano horizontal, haciendo que las fuerzas gravitatorias debidas al peso de los elementos sean también bajas o despreciables.

Estas hipótesis simplifican el diseño y sintonización del sistema de control, haciendo que un esquema de control articular independiente sea suficiente para seguir las trayectorias planificadas [3] con errores aceptables.

1.3.4. La interfaz de usuario

Para facilitar el desarrollo del sistema, y su mantenimiento futuro, la interfaz de usuario se implementará en Matlab, al ser éste un lenguaje de programación de alto nivel muy extendido en ingeniería, y con amplio soporte a largo plazo. Esta interfaz dispondrá de funcionalidad para:

1. Permitir una fácil especificación de las configuraciones inicial y final del robot, así como de eventuales puntos de paso intermedios requeridos.
2. Calcular una trayectoria factible que una las configuraciones especificadas.
3. Simular la ejecución de la trayectoria sobre un modelo gráfico del robot.
4. Ejecutar la trayectoria sobre el robot físico.
5. Interpretar el movimiento del robot, visualizando su evolución en pantalla, al tiempo que se grafican el espacio de trabajo y las singularidades presentes en él.
6. Guardar trayectorias o cargar otras previamente planificadas.
7. Graficar todos los datos necesarios para evaluar el funcionamiento del robot durante la ejecución de las trayectorias, incluyendo las consignas de posición y velocidad angular, las respuestas medidas por los sensores, y los errores asociados en función del tiempo.

1.4 Trabajos relacionados

La planificación de movimientos es un aspecto central en la robótica [4–6], y existen numerosas estrategias para llevarla a cabo, incluso en robots de geometría general [7–12]. Hasta la fecha, sin embargo, la mayoría de trabajos ha omitido el tratamiento de las configuraciones singulares, un aspecto que resulta clave cuando existen cadenas cinemáticas cerradas en el mecanismo [13, 14]. Si bien es cierto que hay estrategias para atravesar estas configuraciones [15, 16], éstas recurren a la inercia del mecanismo, y no garantizan la precisión de posicionado ni la equilibrabilidad de las fuerzas soportadas cerca de una singularidad. Lo aconsejable, por tanto, es mantener al robot alejado de estas configuraciones, evitando así pérdidas de rigidez, o demandas de par motor inasumibles. En los manipuladores comerciales, esto se consigue imponiendo límites articulares que excluyan la presencia de singularidades, y asumiendo, en contrapartida, reducciones drásticas del espacio de trabajo. El uso de un planificador como el presentado en este proyecto, en cambio, permite trabajar con el rango completo de movimientos posibles, circunvalando las singularidades como si de obstáculos físicos se tratara.

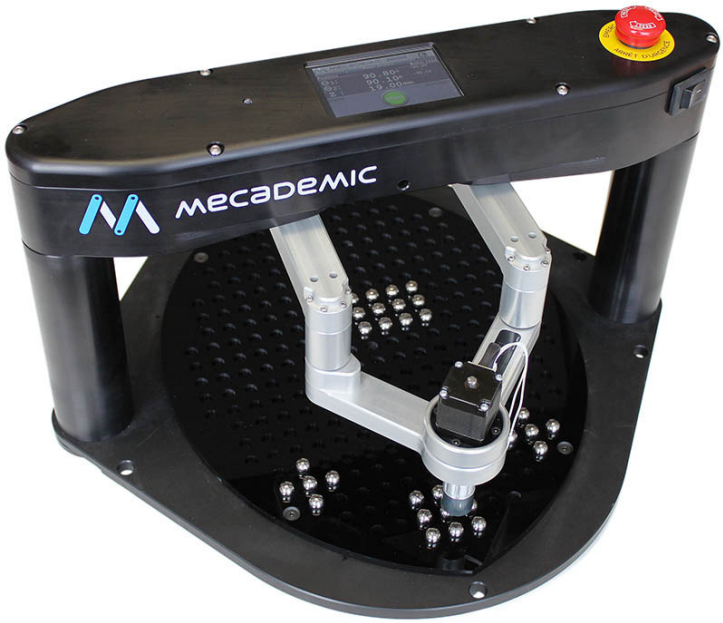


Figura 1.2: El robot 2-RRR *Dextar*.

Dejando de lado los estudios que se centran en la evasión de singularidades en tiempo real [17–19], que tienen un propósito distinto, los pocos algoritmos que permiten planificar trayectorias libres de singularidades se basan en la deformación de caminos paramétricos [20, 21], la resolución de ecuaciones diferenciales sujetas a condiciones de contorno [22], o el tratamiento del conjunto singular como un obstáculo explícitamente representado [23]. Estas estrategias funcionan bien en situaciones favorables, pero [20] y [22] mencionan limitaciones relativas al cálculo de las trayectorias en algunos casos, y el método en [23] es computacionalmente intensivo, porque requiere obtener aproximaciones politópicas del conjunto singular. De un modo u otro, además, en [20–23] se utilizan parametrizaciones explícitas del espacio de configuraciones, lo cual dificulta extender estos métodos para tratar manipuladores con una geometría distinta de la asumida. El método utilizado en este proyecto, en cambio, no recurre a tales parametrizaciones, y es aplicable a manipuladores no redundantes de geometría general. En contraste con [23], además, trata las singularidades implícitamente, no explícitamente como obstáculos, dando lugar a tiempos de cálculo muy reducidos.

De entre los anteriores trabajos, el más relacionado con este proyecto es el llevado a cabo por Bonev. y col. sobre el robot *Dextar* (Fig. 1.2 y todo su sistema de planificación [21]. Este robot presenta similitudes con respecto a nuestro robot 3-RRR. Ambos manipuladores son paralelos, utilizan patas de estructura RRR, y presentan singularidades directas e inversas, evitándose sólo las primeras en los movimientos planificados. *Dextar* posee un eficaz planificador capaz de calcu-

lar trayectorias de tiempo mínimo entre configuraciones dadas, y tanto su diseño mecatrónico, como su funcionamiento dinámico son excelentes, hasta el punto de haber dado lugar a una versión comercial de todo el sistema [24], orientado especialmente al entorno académico.

El robot *Dextar*, sin embargo, sólo posee dos actuadores, lo cual permite controlar la posición Cartesiana de su efector final, pero no su orientación. Esto simplifica notablemente su espacio de configuraciones y facilita una rápida planificación de sus movimientos explotando su estructura mecánica específica. Nuestro robot 3-RRR se encuentra aún en una fase de prototipo, pero supone un avance con respecto a *Dextar* en tanto que la orientación del efector sí se puede controlar, y su planificador, como ya se ha explicado, es aplicable a mecanismos de estructura general.

1.5 Estructura de la memoria

El resto de la memoria sigue la siguiente estructura.

- En el capítulo 2 explicamos el método de planificación de trayectorias empleado. Un aspecto interesante del enfoque utilizado es que se basa en la continuación numérica del espacio solución de un sistema de ecuaciones que incluye, solamente, las configuraciones no singulares y libres de colisión del robot. Al navegar por este espacio, las singularidades y colisiones se evitan de manera natural, sin tener que recurrir a los costosos tests de factibilidad comúnmente utilizados por los planificadores clásicos.
- En el capítulo 3 vemos los elementos necesarios para la ejecución de trayectorias en el robot 3-RRR desarrollado, haciendo énfasis en la función que desempeña cada componente, y el modo en el que interactúan entre sí. También exponemos la estrategia de control adoptada, así como los métodos seguidos para su ajuste.
- En el capítulo 4 mostramos un ejemplo de planificación y ejecución sobre el sistema desarrollado, viendo los resultados de un problema de planificación concreto. Analizamos el camino planificado, su posterior suavizado, la generación de la trayectoria factible a partir del camino, su ejecución, y todos los datos proporcionados por los sensores del robot a lo largo de la trayectoria para valorar la eficacia del sistema de control.
- En el capítulo 5 describimos las principales conclusiones de este proyecto, y enumeramos un conjunto de puntos que dan pie a trabajo futuro.

La memoria contiene también tres anexos: el presupuesto del proyecto, los ficheros de cálculo utilizados y la documentación técnica de los elementos montados en el robot. Finalmente, en un CD adjunto, se incluye el código íntegro de los programas implementados.

2

Planificación de las trayectorias

En este capítulo veremos cómo se pueden planificar trayectorias libres de singularidades en un robot 3-RRR. Comenzaremos caracterizando los dos grandes tipos de singularidades que afectan el comportamiento de un robot paralelo genérico (las singularidades directas e inversas) y viendo cuál es su relación con los fenómenos de pérdida de control y de destreza mencionados en el capítulo anterior. A continuación presentaremos la estructura particular del robot 3-RRR, y formularemos un sistema de ecuaciones que define su espacio de configuraciones "factibles", es decir, aquellas que no dan lugar a una singularidad, ni a colisiones del robot consigo mismo. Finalmente, explicaremos el método adoptado para planificar trayectorias dentro de este espacio, basado en técnicas numéricas de continuación multidimensional.

2.1 Configuraciones singulares

La configuración de un mecanismo robótico se puede describir mediante un vector $\mathbf{q}$ de n_q coordenadas generalizadas, el cual determina la orientación y posición de todos sus eslabones en un determinado instante de tiempo. Existe total libertad en la elección de la forma y dimensión de $\mathbf{q}$, pero debe describir una y sólo una configuración. Llamamos *espacio de configuraciones* de un mecanismo al conjunto $\mathcal{C}$ de todas las configuraciones que éste puede adoptar.

Los movimientos instantáneos de un mecanismo en una configuración $\mathbf{q}$ se pueden caracterizar mediante un sistema de ecuaciones lineales,

$$\mathbf{L}\,\mathbf{m} = \mathbf{0} \tag{2.1}$$

donde $\mathbf{L}$ es una matriz que depende de $\mathbf{q}$, y $\mathbf{m}$ es el *vector velocidad*, el cual contiene suficientes coordenadas como para calcular la velocidad de cualquier punto del mecanismo [25]. En general

el vector $\mathbf{m}$ toma la forma

$$\mathbf{m} = \begin{bmatrix} \mathbf{m}_v \\ \mathbf{m}_u \\ \mathbf{m}_p \end{bmatrix}$$

donde $\mathbf{m}_v$, $\mathbf{m}_u$, y $\mathbf{m}_p$ engloban las coordenadas de la velocidad de entrada (relativas a los grados de libertad actuados), las de la salida (relativas al efector final) y pasivas (las restantes coordenadas de velocidad) respectivamente. La Ec. (2.1) se puede obtener de diversas maneras, bien derivando las ecuaciones que definen $\mathcal{C}$ con respecto del tiempo, bien planteándolas directamente mediante la Teoría de Torsores.[1] En cualquier caso, por el hecho de que $\mathbf{m}$ caracteriza completamente el estado de velocidad, esta ecuación permite hacer un estudio exhaustivo de todas las singularidades del mecanismo [13, 14, 26, 27].

Las singularidades de un mecanismo robótico aparecen en situaciones donde el problema cinemático instantáneo directo, o bien el inverso, son irresolubles o de solución indeterminada.

Recordemos que:

- El problema cinemático instantáneo directo consiste en encontrar $\mathbf{m}$, cuando la velocidad de entrada $\mathbf{m}_v$ es conocida.

- El problema cinemático instantáneo inverso consiste en encontrar $\mathbf{m}$, cuando la velocidad de salida $\mathbf{m}_u$ es conocida.

Para definir los dos grandes grupos de singularidades con que se encuentra un robot, consideramos las particiones

$$\mathbf{m} = \begin{bmatrix} \mathbf{m}_v \\ \mathbf{m}_y \end{bmatrix}, \qquad \mathbf{m} = \begin{bmatrix} \mathbf{m}_u \\ \mathbf{m}_z \end{bmatrix},$$

donde $\mathbf{m}_y$ y $\mathbf{m}_z$ proporcionan las velocidades restantes en $\mathbf{m}$ tras la eliminación de $\mathbf{m}_v$ y $\mathbf{m}_u$, respectivamente. De la misma manera, consideramos las particiones de $\mathbf{L}$

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_v & \mathbf{L}_y \end{bmatrix}, \qquad \mathbf{L} = \begin{bmatrix} \mathbf{L}_u & \mathbf{L}_z \end{bmatrix}.$$

Podemos ahora escribir la Ec. (2.1) de las dos siguientes formas

$$\mathbf{L}_v\mathbf{m}_v + \mathbf{L}_y\mathbf{m}_y = \mathbf{0}, \tag{2.2}$$

$$\mathbf{L}_u\mathbf{m}_u + \mathbf{L}_z\mathbf{m}_z = \mathbf{0}. \tag{2.3}$$

[1] *Screw Theory* en inglés.

Como se asume que el mecanismo no es redundante, tanto $\mathbf{m}_v$ como $\mathbf{m}_u$ contienen n coordenadas de velocidad, donde n es la dimensión de $\mathcal{C}$. En particular, el número de columnas de $\mathbf{L}$ excede su número de filas en exactamente n, y, por tanto, las matrices $\mathbf{L}_y$ y $\mathbf{L}_z$ son de dimensión $n \times n$. Así pues, a partir de las ecuaciones anteriores, para configuraciones en las que $\mathbf{L}_y$ y $\mathbf{L}_z$ sean no singulares, podemos escribir

$$\mathbf{m}_y = -\mathbf{L}_y^{-1}\, \mathbf{L}_v\, \mathbf{m}_v,$$
$$\mathbf{m}_z = -\mathbf{L}_z^{-1}\, \mathbf{L}_u\, \mathbf{m}_u,$$

que proporcionan la solución a los problemas cinemáticos instantaneos directo e inverso del mecanismo.

Se dice que una configuración $\mathbf{q}$ es una *singularidad directa* cuando la matriz $\mathbf{L}_y$ evaluada en $\mathbf{q}$ es singular. De forma analoga, $\mathbf{q}$ es una *singularidad inversa* cuando $\mathbf{L}_z$ evaluada en $\mathbf{q}$ es singular.

Observemos los dos siguientes fenómenos que se producen en una singularidad directa. Por un lado, de la Ec. (2.2) vemos que si en una singularidad directa la velocidad de los actuadores $\mathbf{m}_v$ es nula, el mecanismo aún puede adoptar infinitas velocidades porque la ecuación queda reducida al sistema homogéneo $\mathbf{L}_y\mathbf{m}_y = \mathbf{0}$, y $\mathbf{L}_y$ es de rango deficiente. Por otro lado, la Ec. (2.2) se puede reescribir como $\mathbf{L}_y\mathbf{m}_y = \mathbf{b}$, donde $\mathbf{b} = -\mathbf{L}_v\mathbf{m}_v$. Fijémonos que cuando $\mathbf{L}_y$ es singular, si $\mathbf{m}_v$ es no nulo, $\mathbf{b} \neq 0$ en general, y el sistema sólo será resoluble si $\mathbf{b}$ pertenece al espacio imagen de $\mathbf{L}_y$. Al ser $\mathbf{L}_y$ singular, sin embargo, este espacio será de dimensión inferior a n, y por tanto, en una singularidad directa, un valor arbitrario del vector $\mathbf{m}_v$ difícilmente corresponderá a una velocidad factible del mecanismo.

Ambos fenómenos se pueden observar en la práctica. Al pasar por una singularidad directa, el mecanismo pierde rigidez (primer fenómeno), y puede incluso romperse porque su velocidad es raramente compatible con la disposición de los enlaces cinemáticos (segundo fenómeno).

Se puede hacer un análisis similar con la Ec. (2.3), y concluir que en una singularidad inversa, si bloqueamos el efector final (fijamos $\mathbf{m}_u = 0$), el mecanismo no es rígido, y no todas las velocidades de salida $\mathbf{m}_u$ son posibles, con lo cual el mecanismo experimenta una pérdida de destreza en su efector final. Es decir, el espacio de pequeños desplazamientos que este efector puede realizar pasa a ser de dimensión inferior.

A la hora de planificar los movimientos del mecanismo, es prioritario evitar las singularidades directas, porque son aquellas que pueden dañar más el mecanismo y conducen a una pérdida de control de su movimiento. En determinadas aplicaciones se querrá además evitar el paso por singularidades inversas, para mantener una destreza máxima a lo largo de toda la trayectoria, y poder así corregir errores de pose del efector en cualquier dirección.

2.2 Estructura del robot 3-RRR

El mecanismo 3-RRR consta de una plataforma triangular móvil unida mediante tres patas a un soporte o base. Cada pata es una cadena cinemática formada por tres articulaciones de revolución - una actuada y dos pasivas - y dos eslabones (Fig. 2.1). Los actuadores de este mecanismo estan montados en las articulaciones de la base, $J_{1,1}$, $J_{1,2}$, y $J_{1,3}$, de aquí el carácter subrayado en la denominación 3-RRR. Variando los ángulos girados en estas tres articulaciones, conseguimos controlar la posición y orientación de la plataforma triangular. Al actuar la plataforma mediante las tres patas en paralelo, el robot exhibe una mayor rigidez que la de un brazo serie triactuado estándar, lo cual lo hace especialmente efectivo en aplicaciones que requieran una mayor precisión de posicionado.

Nombraremos las diversas articulaciones y eslabones del mecanismo como se indica en la Fig. 2.1. Los eslabones 2, 3 y 4 reciben el nombre de eslabones *proximales*, y los 5, 6 y 7, eslabones *distales*. El eslabón 1 coincide con la base anclada al suelo y el 8 con el triángulo móvil, que actúa como *efector final*.

En el prototipo desarrollado para este proyecto, las coordenadas de las articulaciones de la base en el sistema de referencia absoluto OXY de la Fig. 2.1 son, en centímetros:

$$\begin{aligned} J_{1,1} &= (0, 0), \\ J_{1,2} &= (23.5, 0), \\ J_{1,3} &= (11.75, 20.35). \end{aligned}$$

Los eslabones proximales de cada pata tienen una longitud de $l_1 = 10$ cm, los distales de $l_2 = 13.5$ cm y los lados del triángulo equilátero móvil $l_3 = 12$ cm.

A cada eslabón k del robot se le asigna un sistema de referencia relativo $\mathcal{F}_k$. La Fig. 2.1 muestra en rojo el eje x de cada uno de estos sistemas de referencia. La orientación del eslabón k viene dada por el ángulo θ_k que forma el eje x de su sistema de referencia $\mathcal{F}_k$ con el eje x del sistema de referencia global OXY.

En todo el proyecto, para determinar la pose del efector final se utilizaran las coordenadas del baricentro P del triángulo móvil (Fig. 2.1), y el ángulo θ_8 de este triángulo. Así mismo, utilizaremos la notación

$$(x^k_{j,i}, y^k_{j,i})$$

para referirnos a las coordenadas de la articulación $J_{j,i}$ en el sistema de referencia $\mathcal{F}_k$. Cuando estas coordenadas estén referidas al sistema de referencia global OXY, no se especificará el superíndice k.

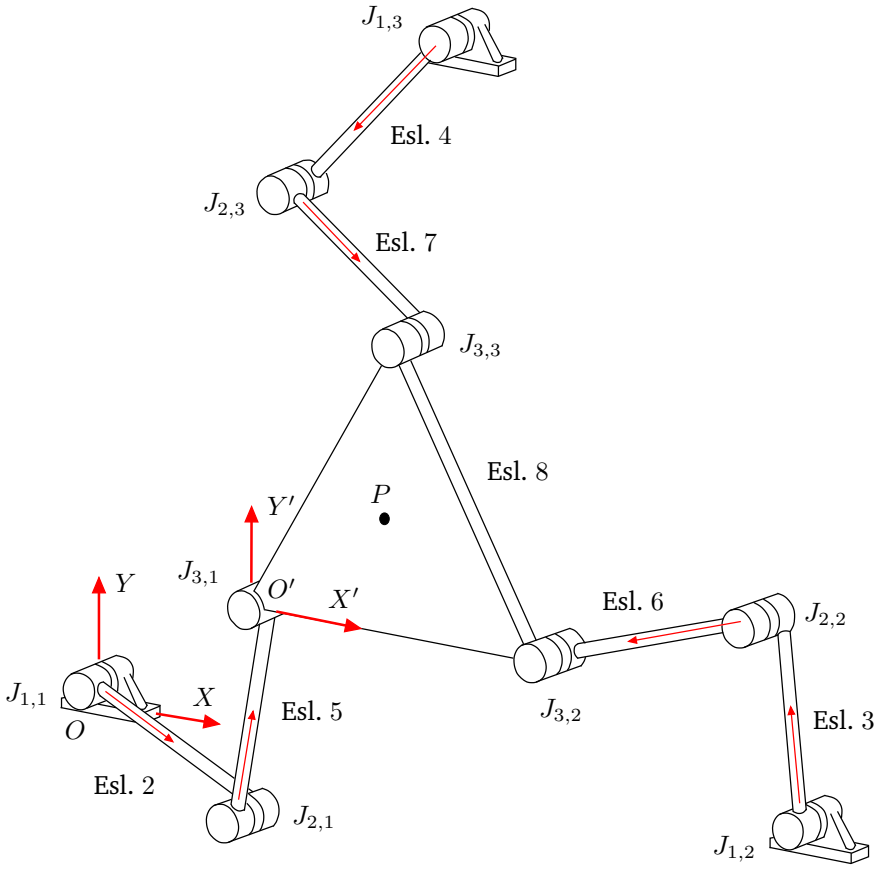


Figura 2.1: Estructura del robot paralelo 3-RRR. Todos los sistemas de referencia se indican en rojo. Para los sistemas de referencia solidarios a cada eslabón, sólo se indica la posición de su eje X. El eje Y se supone colocado a $90°$ en sentido antihorario.

2.3 El espacio de configuraciones

La configuración del robot en un determinado instante de tiempo se puede codificar mediante la tupla de ángulos

$$\mathbf{q} = (\theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, \theta_8).$$

Estos siete ángulos no pueden tomar valores arbitrarios. La configuración sólo será válida si los ángulos satisfacen las restricciones de ensamblado del robot, las restricciones de exclusión de singularidades, y las de no colisión. El espacio de configuraciones del robot será, por tanto, el conjunto de valores $\mathbf{q}$ que satisfacen estas restricciones. Veámoslas en detalle.

2.3.1. Las restricciones de ensamblado

Aunque hay otras opciones, las restricciones de ensamblado de un mecanismo planar se imponen habitualmente de dos maneras: o bien forzando la conexión de cada articulación individualmente, o bien imponiendo el cierre de cada lazo cinemático. En este proyecto adoptaremos la segunda opción, ya que es la que genera, normalmente, un menor número de ecuaciones y variables.

Siguiendo este planteamiento, el cierre de un lazo cinemático se impone estableciendo que la suma de los vectores que van de cada articulación a la siguiente sea cero (Fig. 2.2). Matemáticamente esta condición se puede expresar así

$$\sum_{k=1}^{n} \mathbf{R}_k \cdot \mathbf{v}_k = 0, \tag{2.4}$$

donde $\mathbf{v}_k$ es el vector que va del eslabón k al eslabón $k+1$, expresado en el sistema de referencia $\mathcal{F}_k$, y $\mathbf{R}_k$ es la matriz de rotación de ángulo θ_k, que proporciona la orientación de $\mathcal{F}_k$ con respecto al sistema de referencia absoluto OXY.

Un mecanismo puede exhibir un gran número de lazos cinemáticos, pero basta con imponer las restricciones de cierre relativas a un conjunto maximal de lazos independientes, porque las restricciones de otros lazos siempre se pueden expresar como combinación lineal de éstas. Este conjunto maximal se puede obtener calculando los ciclos fundamentales del grafo de eslabones i articulaciones del mecanismo [28].

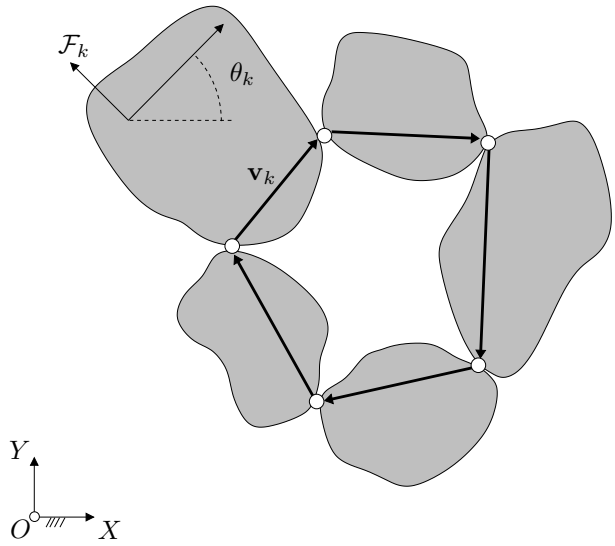


Figura 2.2: Condición de cierre de un lazo cinemático.

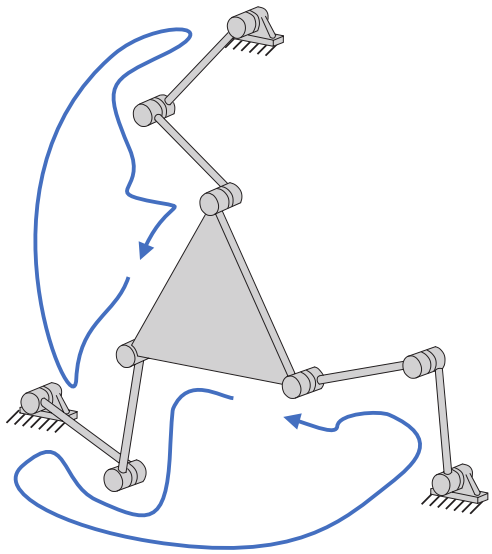


Figura 2.3: Dos lazos cinemáticos independientes del robot 3-RRR.

En particular, el mecanismo 3-RRR sólo posee dos ciclos fundamentales. Por ejemplo los de los lazos indicados en la Fig. 2.3. Utilizando estos dos lazos, las ecuaciones de ensamblado del robot 3-RRR son:

$$x_{1,2} + l_1 c_3 + l_2 c_6 - l_3 c_8 - l_2 c_5 - l_1 c_2 - x_{1,1} = 0, \tag{2.5}$$

$$y_{1,2} + l_1 s_3 + l_2 s_6 - l_3 s_8 - l_2 s_5 - l_1 s_2 - y_{1,1} = 0, \tag{2.6}$$

$$x_{1,3} + l_1 c_4 + l_2 c_7 - l_3 c_8 \cos(\phi) + l_3 s_8 \sin(\phi) - l_2 c_5 - l_1 c_2 - x_{1,1} = 0, \tag{2.7}$$

$$y_{1,3} + l_1 s_4 + l_2 s_7 - l_3 s_8 \cos(\phi) - l_3 c_8 \sin(\phi) - l_2 s_5 - l_1 s_2 - y_{1,1} = 0, \tag{2.8}$$

siendo $s_i = \sin(\theta_i)$, $c_i = \cos(\theta_i)$, y $\phi = \pi/3$.

Estas ecuaciones, por sí solas, ya garantizan que $\mathbf{q}$ define un configuración ensamblada del mecanismo. No obstante, para imponer fácilmente las restricciones de exclusión de singularidades y de no colisión, conviene introducir ecuaciones adicionales que expliciten las coordenadas $(x_{j,i}, y_{j,i})$ de todas las articulaciones móbiles. Estas restricciones son, simplemente:

$$x_{2,1} = x_{1,1} + l_1 c_2, \qquad x_{2,2} = x_{1,2} + l_1 c_3, \qquad x_{2,3} = x_{1,3} + l_1 c_4, \tag{2.9}$$

$$y_{2,1} = y_{1,1} + l_1 s_2, \qquad y_{2,2} = y_{1,2} + l_1 s_3, \qquad y_{2,3} = y_{1,3} + l_1 s_4, \tag{2.10}$$

$$x_{3,1} = x_{2,1} + l_2 c_5, \qquad x_{3,2} = x_{2,2} + l_2 c_6, \qquad x_{3,3} = x_{2,3} + l_2 c_7, \tag{2.11}$$

$$x_{3,1} = y_{2,1} + l_2 c_5, \qquad x_{3,2} = y_{2,2} + l_2 c_6, \qquad x_{3,3} = y_{2,3} + l_2 c_7. \tag{2.12}$$

2.3.2. Las restricciones de exclusión de singularidades

Utilizando la Teoría de Torsores [29] se puede ver que en el caso del robot 3-RRR, el estado de velocidad del robot queda unívocamente determinado por las velocidades angulares de las articulaciones activas $J_{1,1}$, $J_{1,2}$, $J_{1,3}$, y la velocidad de traslación y rotación del efector final. En este caso, por tanto, no hay que considerar coordenadas de velocidad pasivas, como podrían ser las de las articulaciones intermedias. La Ec. (2.1) se puede entonces formular en la forma simplificada

$$\mathbf{A} \cdot \hat{T} - \mathbf{B} \cdot \gamma = \mathbf{0} \tag{2.13}$$

donde los vectores $\hat{T}$ y γ corresponden a $\mathbf{m}_u$ y $\mathbf{m}_v$, y las matrices $\mathbf{A}$ y $\mathbf{B}$ a $\mathbf{L}_y$ y $\mathbf{L}_z$. Así pues, una configuración del robot 3-RRR será una singularidad directa si

$$\det(\mathbf{A}) = 0, \tag{2.14}$$

e inversa si

$$\det(\mathbf{B}) = 0. \tag{2.15}$$

Los vectores $\hat{T}$ y γ adoptan la forma

$$\hat{T} = \begin{bmatrix} v_{0,x} \\ v_{0,y} \\ \omega \end{bmatrix}, \qquad \gamma = \begin{bmatrix} \omega_{1,1} \\ \omega_{1,2} \\ \omega_{1,3} \end{bmatrix}$$

donde:

- ($v_{0,x}$,$v_{0,y}$) es el vector velocidad del punto del efector que coincide instantáneamente con el origen del sistema de referencia OXY.
- ω es la velocidad angular del efector final.
- $\omega_{1,i}$ es la velocidad angular de la articulación $J_{1,i}$.

Las matrices $\mathbf{A}$ y $\mathbf{B}$, por su lado, tienen la estructura

$$\mathbf{A} = \begin{bmatrix} \hat{s}_1^{\mathsf{T}} \\ \hat{s}_2^{\mathsf{T}} \\ \hat{s}_3^{\mathsf{T}} \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} \hat{s}_1^{\mathsf{T}} \hat{S}_{1,1} & 0 & 0 \\ 0 & \hat{s}_2^{\mathsf{T}} \hat{S}_{1,2} & 0 \\ 0 & 0 & \hat{s}_3^{\mathsf{T}} \hat{S}_{1,3} \end{bmatrix},$$

donde $\hat{s}_i$ es el vector de coordenadas de Plücker del eslabón distal de la pata i, y $\hat{S}_{j,i}$ es el

vector de coordenadas de Plücker de la recta ortogonal al plano del robot que pasa por $J_{j,i}$. Aunque en las matrices $\mathbf{A}$ y $\mathbf{B}$ los vectores $\hat{s}_i$ y $\hat{S}_{j,i}$ se suponen normalizados, a efectos de las ecuaciones (2.14) y (2.15) no es necesario que lo estén, porque la normalización simplemente multiplica estas ecuaciones por un factor constante. Así, podremos considerar que

$$\hat{s}_1 = \hat{S}_{1,2} \times \hat{S}_{1,3}, \tag{2.16}$$

$$\hat{s}_2 = \hat{S}_{2,2} \times \hat{S}_{2,3}, \tag{2.17}$$

$$\hat{s}_3 = \hat{S}_{3,2} \times \hat{S}_{3,3}. \tag{2.18}$$

siendo

$$\hat{S}_{j,i} = \begin{bmatrix} y_{j,i} \\ -x_{j,i} \\ 1 \end{bmatrix}, \tag{2.19}$$

para toda articulación $J_{j,i}$.

A partir de las Ecs. (2.14) y (2.15), y usando las expresiones anteriores, se puede ver que las singularidades directas aparecen cuando las rectas soporte de los tres eslabones distales intersectan en un punto común, mientras que las inversas se dan cuando se alinean las tres articulaciones de una pata. Las figuras 2.4 y 2.5 muestran, en secuencia, una configuración no singular, una singularidad directa, una singularidad inversa, y una configuración que es, a la vez, singularidad directa e inversa. En estas figuras vemos cómo, de acuerdo con lo explicado en la Sección 2.1, en una y otra singularidad el mecanismo pierde rigidez al bloquear la entrada o la salida, respectivamente. La secuencia completa de estas figuras se puede ver en vídeo en [30].

Para excluir las singularidades directas o inversas de nuestro espacio de configuraciones, simplemente necesitamos imponer

$$\det(\mathbf{A}) \neq 0 \tag{2.20}$$

y/o

$$\det(\mathbf{B}) \neq 0 \tag{2.21}$$

según convenga. Como se ha comentado en la Sección 1.3, en nuestro caso nos centraremos en excluir únicamente las singularidades directas, porque son las más perjudiciales para el mecanismo (comprometen el control y pueden producir la rotura del mecanismo), y se permitirá, en cambio, el paso por singularidades inversas. Con leves cambios, podríamos optar también por excluir el paso por singularidades inversas, pero por un lado esto no es deseable, porque

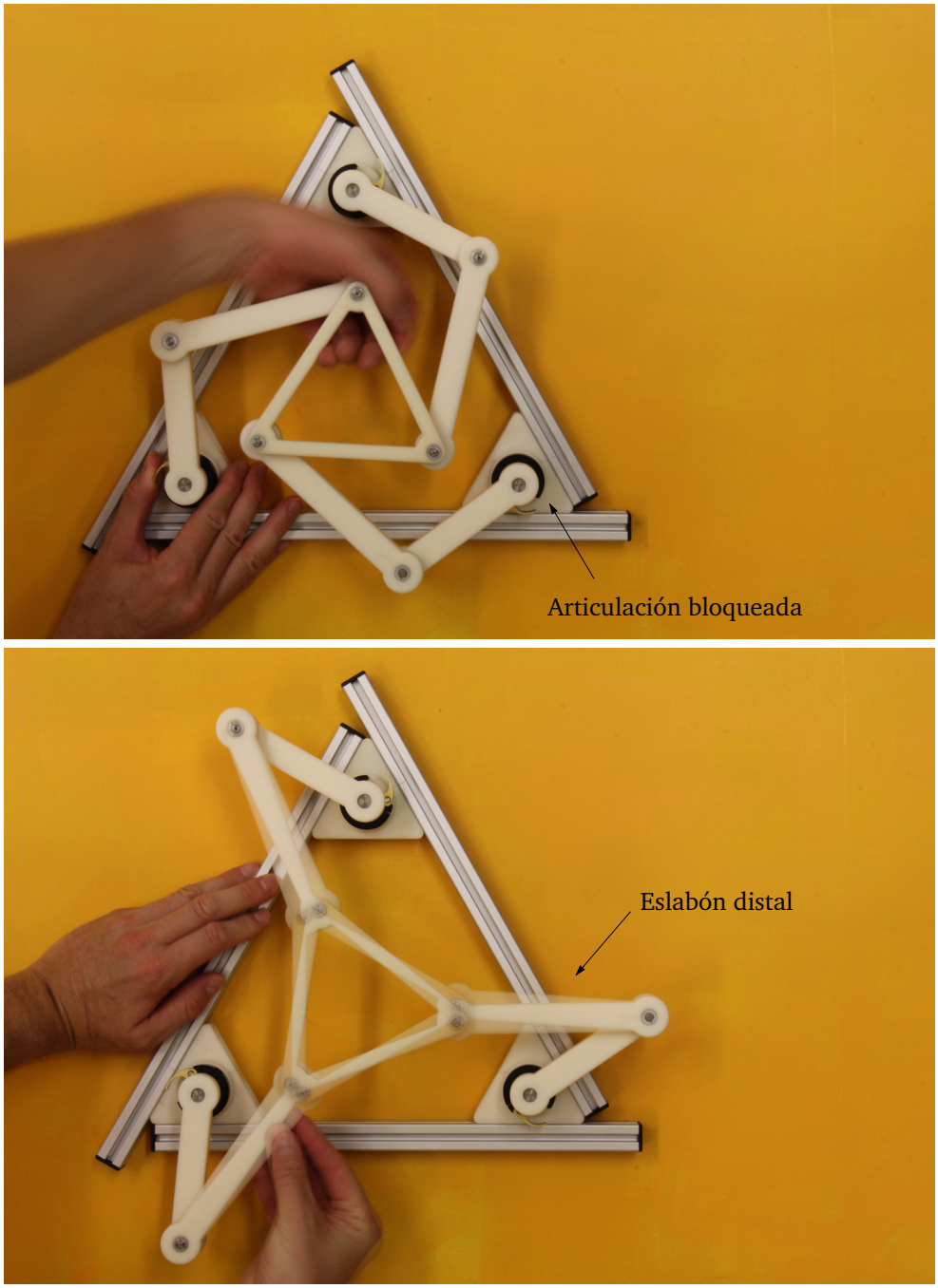


Figura 2.4: **Arriba:** Con las articulaciones de la base bloqueadas, el mecanismo se muestra rígido en configuraciones no singulares. **Abajo:** En una singularidad directa, los eslabones distales son concurrentes y tras bloquear las articulaciones de la base, el mecanismo pierde rigidez.

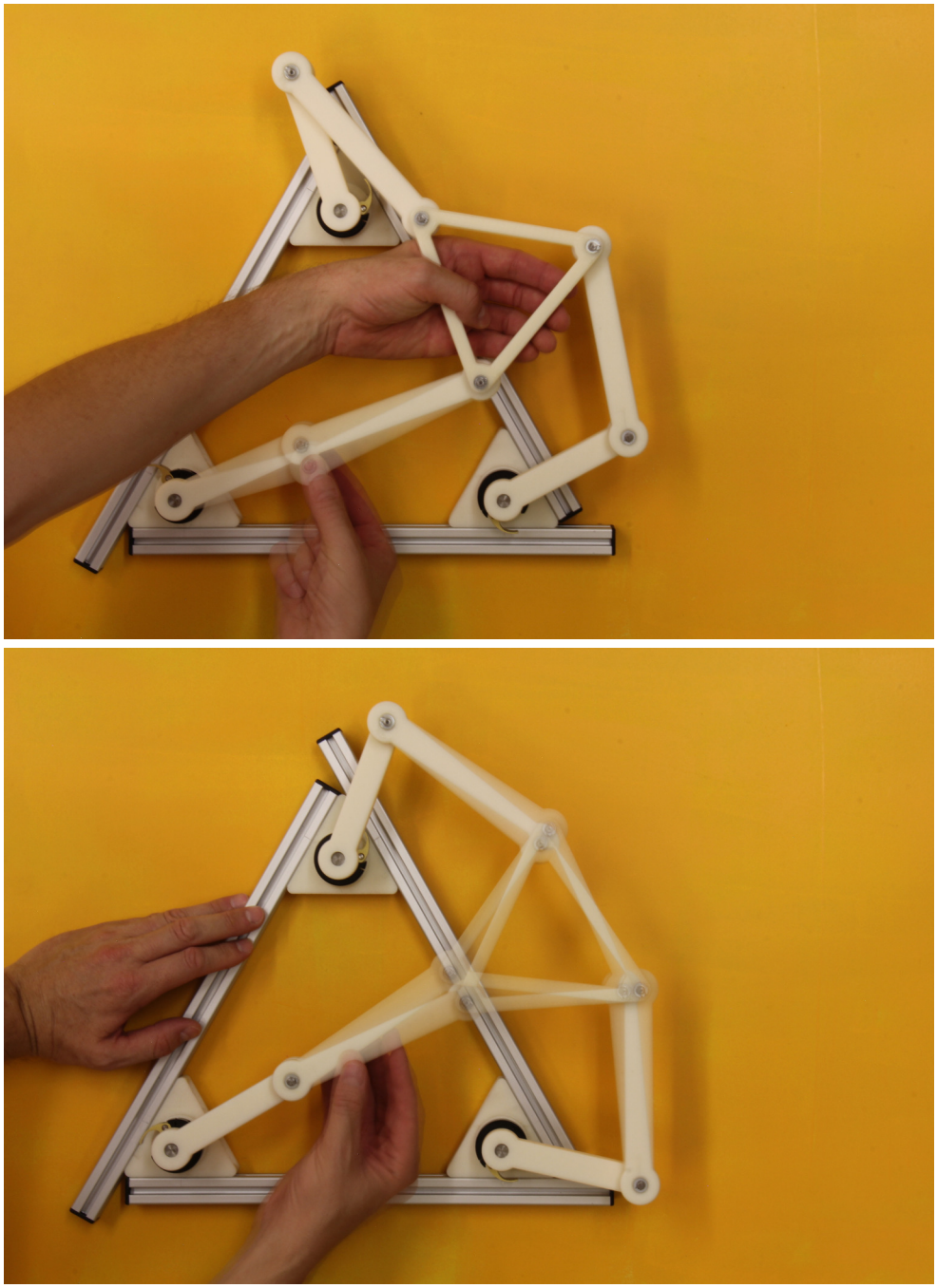


Figura 2.5: **Arriba:** En una singularidad inversa las articulaciones de una pata están alineadas y, bloqueando la plataforma móvil notamos que la pata no tiene rigidez. **Abajo:** Una configuración puede ser a la vez una singularidad directa e inversa.

reduciríamos el espacio de trabajo del robot, y tampoco es necesario, porque las pérdidas de destreza del efector no suponen, en nuestro contexto, ningún riesgo para el robot.

2.3.3. Las restricciones de no colisión

Tal y como se ha diseñado la geometría del robot (Fig. 2.1) sólo se pueden producir colisiones entre los eslabones distales de las patas. Los eslabones distales no pueden colisionar ni con los proximales ni con la plataforma móvil ya que se encuentran situados en planos diferentes, y los eslabones proximales no pueden colisionar entre ellos, porque las articulaciones $J_{1,1}$, $J_{1,2}$, $J_{1,3}$ están suficientemente alejadas entre sí. Nos centraremos pues, en imponer la no colisión entre parejas de eslabones distales.

Los eslabones distales constan de dos zonas circulares centradas en las articulaciones, unidas mediante un tronco rectilíneo, con las medidas que se indican en la Fig. 2.6(a). Gracias a esta geometría, los troncos de los dos eslabones nunca pueden entrar en contacto y, como se ve en la Fig. 2.6(b), la única colisión posible es la del círculo de un eslabón con el contorno de otro eslabón. Por tanto, podemos decir que se producirá una colisión cuando el centro C del círculo entre en contacto con el perímetro p formado por los puntos que se encuentran a distancia r del eslabón. Sin embargo, utilizando el prototipo físico del robot podemos ver que el círculos de los eslabones distales tampoco pueden entrar en contacto entre ellos. Por tanto, en la práctica sólo tendremos que comprobar la inclusión de C en el rectángulo R indicado en la Fig. 2.6(c), de vértices P_1, P_2, P_3, P_4.

El test de inclusión de un punto en un rectángulo es fácil de implementar algorítmicamente, pero para el método de planificación que propondremos necesitamos que la restricción de no colisión venga dada en la forma funcional

$$\psi(\mathbf{q}) > 0 \tag{2.22}$$

donde $\psi(\mathbf{q})$ es una función diferenciable de $\mathbf{q}$. Evaluada en $\mathbf{q}$, esta función debe determinar si el centro C está en el exterior ($\psi(\mathbf{q}) > 0$) o bien en el interior ($\psi(\mathbf{q}) < 0$) del rectángulo R. Para encontrar esta función, aproximaremos el rectángulo R mediante la superelipse circunscrita indicada en la Fig. 2.6. Esta superelipse tiene la expresión

$$\left(\frac{x' - a}{s_x}\right)^4 + \left(\frac{y' - b}{s_y}\right)^4 = 1$$

en el sistema de referencia $O'X'Y'$ de la figura, donde (a, b) son las coordenades de su centro, y s_x y s_y son las longitudes de sus semiejes. En nuestro caso, $(a, b) = (6.75, 0)$ cm en $O'X'Y'$, $s_x = 6.75$ cm, y $s_y = 2.70$ cm. Así pues, consideraremos que un círculo de un eslabón distal

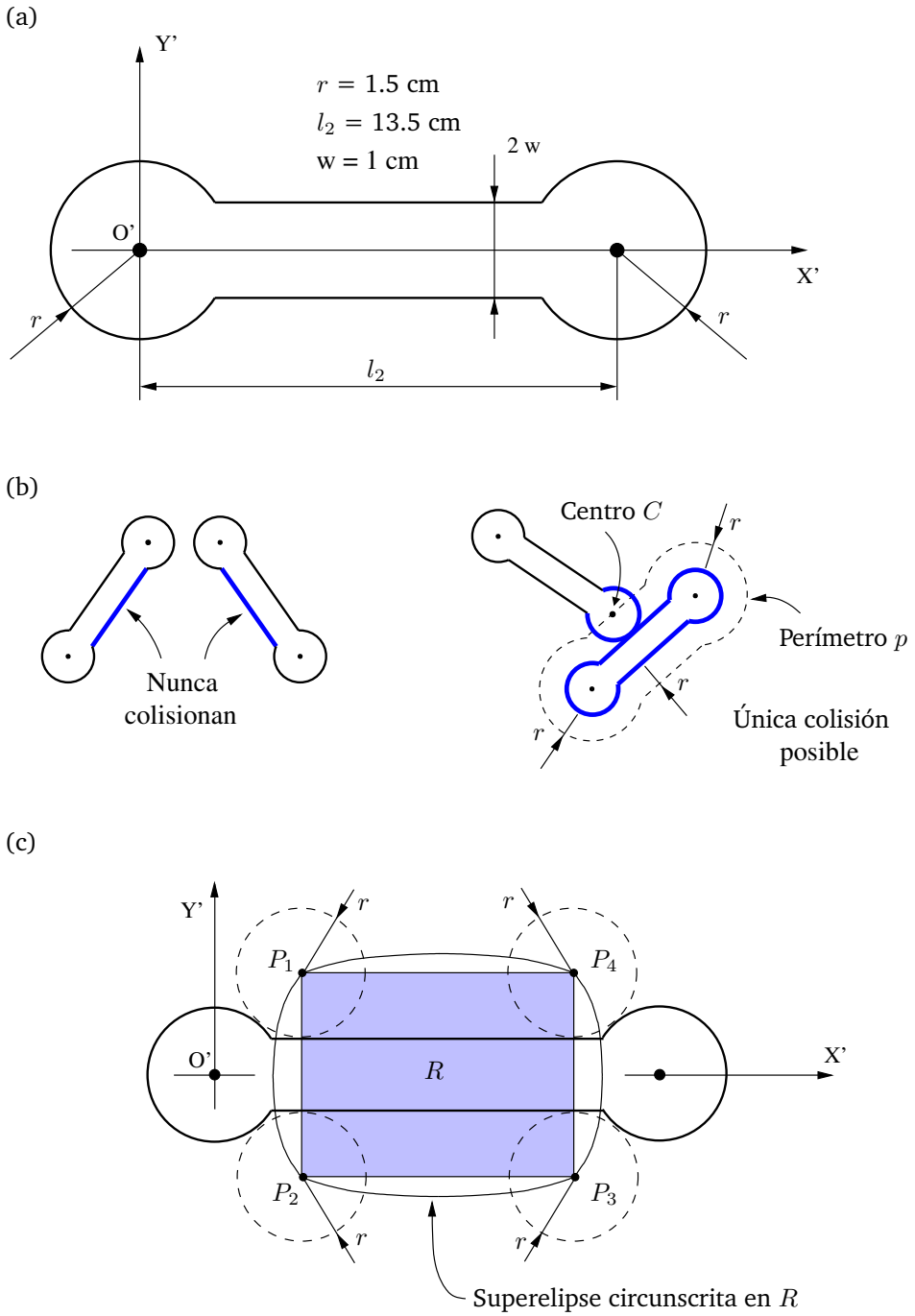


Figura 2.6: **(a)** Geometría de un eslabón distal. **(b)** Posibilidades de colisión entre eslabones distales. **(c)** Región R y su aproximación mediante una superelipse.

colisiona con un segundo eslabón, si el centro C de este círculo se encuentra dentro de la superelipse asociada al segundo eslabón.

Usando la expresión anterior y la notación definida en la Sección 2.2, podemos decir que el círculo de radio r centrado en $J_{j,i}$ no colisionará con el eslabón distal k siempre y cuando

$$\left(\frac{x_{j,i}^k - a}{s_x}\right)^4 + \left(\frac{y_{j,i}^k - b}{s_y}\right)^4 - 1 > 0, \tag{2.23}$$

expresión que ya adopta la forma deseada en la Ec. (2.22).

Fijémonos en que el círculo de la articulación $J_{j,i}$ de una pata puede entrar en colisión con los eslabones distales de las otras dos patas. Por tanto, como tenemos seis círculos en eslabones distales, deberemos considerar un total de doce desigualdades para garantizar que el robot no se encuentra en una autocolisión.

En la Ec. (2.23), sin embargo, nos faltan las expresiones de las coordenadas $x_{j,i}^k$ y $y_{j,i}^k$ en función de la configuración $\mathbf{q}$. Estas expresiones se obtienen utilizando el cambio de sistema de referencia

$$\begin{bmatrix} c_k & -s_k \\ s_k & c_k \end{bmatrix} \begin{bmatrix} x_{j,i}^k \\ y_{j,i}^k \end{bmatrix} + \begin{bmatrix} x_{2,l} \\ y_{2,l} \end{bmatrix} = \begin{bmatrix} x_{j,i} \\ y_{j,i} \end{bmatrix},$$

donde hemos supuesto que el índice l es el de la pata donde está montado el eslabón k, y que el sistema de referencia $\mathcal{F}_k$ está centrado en la articulación $J_{2,l}$, de acuerdo con el convenio establecido en la Fig. 2.1. Aislando $x_{j,i}^k$ y $y_{j,i}^k$ resulta, en forma matricial:

$$\begin{bmatrix} x_{j,i}^k \\ y_{j,i}^k \end{bmatrix} = \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix} \begin{bmatrix} x_{j,i} - x_{2,l} \\ y_{j,i} - y_{2,l} \end{bmatrix}. \tag{2.24}$$

2.4 La estrategia de planificación

Com acabemos de ver, el espacio de configuraciones $\mathcal{C}$ del robot está formado por el conjunto de valores del vector $\mathbf{q}$ que satisfacen, simultáneamente, las condiciones:

- De ensamblado: Ecs. (2.5)-(2.12).
- De exclusión de singularidades: Ecs. (2.16)-(2.20).
- De no colisión: Ecs. (2.23) y (2.24), usando todos los pares de eslabones implicados.

La búsqueda de un movimiento factible entre dos configuraciones $\mathbf{q}_s$ y $\mathbf{q}_g$ no singulares y libres de colisión equivale a encontrar un camino dentro del espacio $\mathcal{C}$ que cumple estas ecuaciones.

Para calcular este camino aplicaremos métodos de continuación mutlidimensional sobre las ecuaciones anteriores [31]. Usando estos métodos podemos partir de $\mathbf{q}_s$ para explorar $\mathcal{C}$ sistemáticamente, utilizando el espacio tangente a $\mathcal{C}$ para avanzar en cada momento, hasta eventualmente encontrar $\mathbf{q}_g$. Dependiendo del tipo de implementación escogida, el proceso de continuación acabará cuando se agote un tiempo máximo de búsqueda, o bien cuando se haya explorado el componente conexo de $\mathcal{C}$ alcanzable desde $\mathbf{q}_s$. Esta metodología, sin embargo, requiere que el sistema de ecuaciones sólo contenga igualdades, y en nuestro caso las condiciones (2.20) y (2.23) son desigualdades. Así pues, antes de detallar la metodología, veremos como podemos transformar estas condiciones en igualdades.

2.4.1. Conversión de las restricciones en igualdades

Observemos que la restricción (2.20) se puede substituir por la igualdad equivalente

$$\det(\mathbf{A}) \cdot b = 1, \tag{2.25}$$

donde b es una nueva variable auxiliar añadida al sistema. Claramente, esta ecuación impide que $\det(\mathbf{A})$ se anule, porque $\det(\mathbf{A})$ será cero sólo cuando $b = \pm\infty$. Si bien es cierto que $\det(\mathbf{A})$ se puede acercar a cero cuando b toma valores muy altos o bajos, esta posibilidad se elimina fijando un intervalo de validez para b

$$b \in [b_{min}, b_{max}].$$

Este intervalo, además, acota el valor de $\det(\mathbf{A})$, y así aseguramos que el robot se mantiene suficientemente rígido y controlable a lo largo del camino. Fijémonos también que, para que haya un camino solución, el signo de $\det(\mathbf{A})$ deberà ser el mismo tanto en $\mathbf{q}_s$ com en $\mathbf{q}_g$, y el valor de $b = 1/\det(A)$ en estas configuraciones deberá estar dentro del rango $[b_{min}, b_{max}]$ que hayamos fijado.

La restricción (2.23) también se puede convertir en igualdad substituyéndola por

$$\left[\left(\frac{x^k_{j,i} - a}{s_x}\right)^4 + \left(\frac{y^k_{j,i} - b}{s_y}\right)^4 - 1\right] \cdot g_{i,j,k} = 1, \tag{2.26}$$

donde $g_{i,j,k}$ es una nueva variable auxiliar. Dado que el término entre corchetes será positivo en $\mathbf{q}_s$, y la ecuación impide que se anule, el término se mantendrá positivo durante todo el proceso de continuación.

2.4.2. La variedad de navegación

Con estos cambios obtenemos un nuevo sistema de ecuaciones formado por las Ecs. (2.5)-(2.12), que fuerzan el ensamblado de los distintos eslabones, las Ecs. (2.16)-(2.19) y (2.25), que evitan las singularidades, y las Ecs (2.24) y (2.26), que impiden las autocolisiones. Este nuevo sistema se puede escribir de forma compacta así

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}, \tag{2.27}$$

donde $\mathbf{x}$ es una tupla que aglutina todas sus variables. Esta tupla contiene:

- Los ángulos del vector $\mathbf{q}$.
- Las coordenades absolutas, $x_{j,i}$ y $y_{j,i}$, y relativas, $x^k_{j,i}$ y $y^k_{j,i}$, de todas las articulaciones móviles.
- La variable b de la Ec. (2.25)
- Las variables $g_{j,i,k}$ de las Ecs. (2.26).

El conjunto solución de este nuevo sistema

$$\mathcal{M} = \{\mathbf{x} : \mathbf{F}(\mathbf{x}) = \mathbf{0}\},$$

se denominará la *variedad de navegación* del robot, porque es una variedad diferencial que permite buscar movimientos factibles entre $\mathbf{q}_s$ y $\mathbf{q}_g$ mediante continuación numérica. Ciertamente, fijémonos que cada configuración $\mathbf{q} \in \mathcal{C}$ corresponde a un punto $\mathbf{x} \in \mathcal{M}$, porque los valores de $x_{j,i}$, $y_{j,i}$, $x^k_{j,i}$ y $y^k_{j,i}$ correspondientes a $\mathbf{q}$ se obtienen de las Ecs. (2.9)-(2.12) y (2.24), el valor de b se obtiene de la Ec. (2.25), y los valores de $g_{j,i,k}$ se obtienen de la Ec. (2.26). Así, si $\mathbf{x}_s$ y $\mathbf{x}_g$ son los puntos de $\mathcal{M}$ que corresponden a $\mathbf{q}_s$ y $\mathbf{q}_g$, buscar un camino que conecte $\mathbf{q}_s$ con $\mathbf{q}_g$ sobre $\mathcal{C}$ equivale a encontrar un camino de conexión entre $\mathbf{x}_s$ y $\mathbf{x}_g$ sobre $\mathcal{M}$.

2.4.3. Búsqueda sobre la variedad

Para determinar un camino que conecte $\mathbf{x}_s$ y $\mathbf{x}_g$ sobre $\mathcal{M}$ utilizaremos el método de continuación multidimensional de Henderson [32], con las modificaciones introducidas en [1, 2] para guiar la búsqueda hacia $\mathbf{x}_g$. Este método proporciona una manera sistemática de recorrer $\mathcal{M}$ contínuamente desde $\mathbf{x}_s$, hasta que, o bien se encuentra $\mathbf{x}_g$, o bien se ha rastreado todo el componente conexo de $\mathcal{M}$ alcanzable desde $\mathbf{x}_s$. El método es una generalización de las técnicas de continuación clásicas [33], y no es hasta hace muy poco se ha reconocido su valor en el

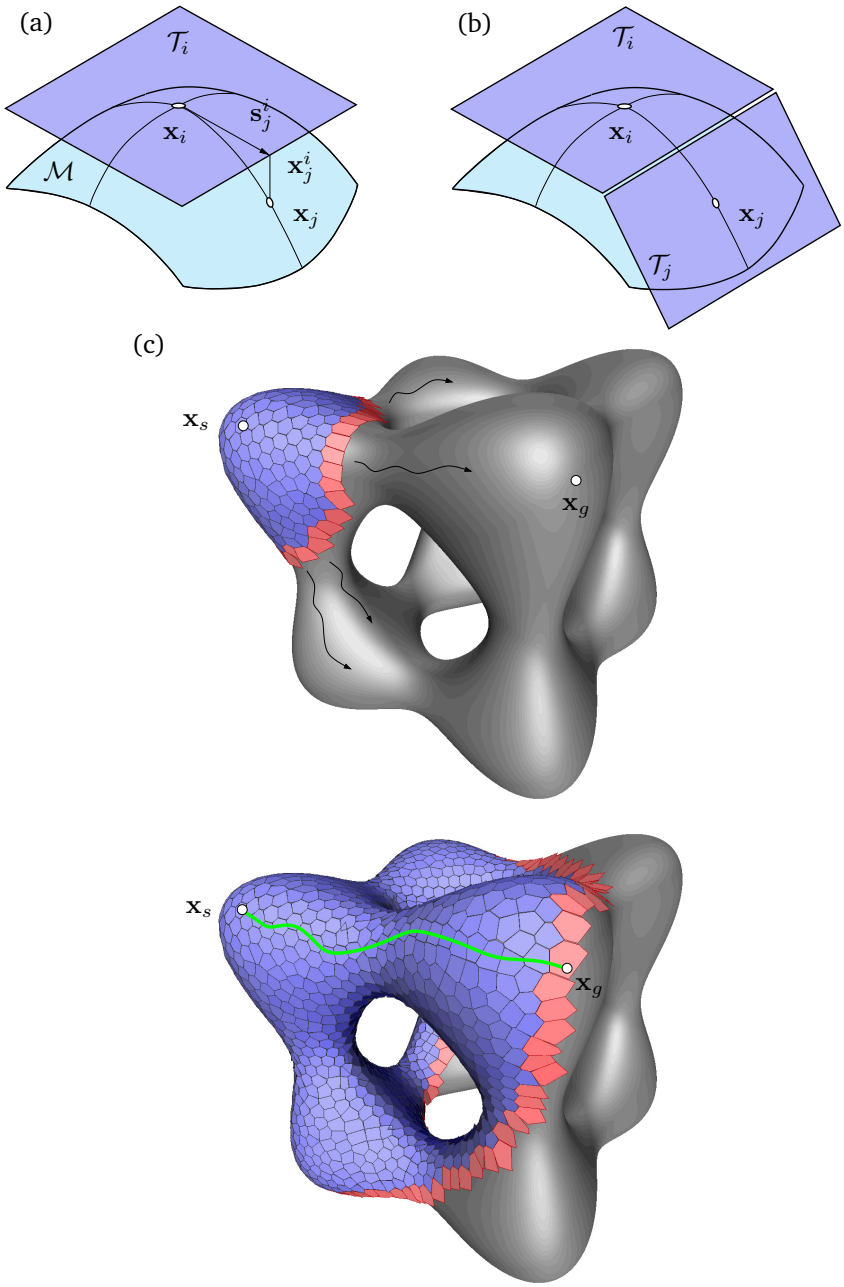


Figura 2.7: (a) Construcción de una carta local alrededor de $\mathbf{x}_i \in \mathcal{M}$, y expansión en la dirección $\mathbf{s}_j^i$ para producir un nuevo punto $\mathbf{x}_j \in \mathcal{M}$. (b) Construcción de una carta vecina en x_j, y recorte de su dominio teniendo en cuenta los dominios de cartas previamente existentes. (c) Evolución del método de continuación sobre la superficie de Chmutov, definida por $3 + 8(x^4 + y^4 + z^4) = 8(x^2 + y^2 + z^2)$, con el camino encontrado indicado en verde.

ámbito de la Robótica [34]. A continuación explicamos como funciona, y referimos al lector a [1, 2, 32] para más detalles.

El método explora $\mathcal{M}$ construyendo un *atlas* de $\mathcal{M}$ gradualmente, es decir, un conjunto de cartas locales que parametrizan, cada una, una pequeña región de $\mathcal{M}$. Para un punto dado $\mathbf{x}_i \in \mathcal{M}$, inicialmente fijado a $\mathbf{x}_s$, el método calcula el espacio tangente a $\mathcal{M}$ en $\mathbf{x}_i$, $\mathcal{T}_i$, y usa este espacio para parametrizar los puntos de $\mathcal{M}$ en el vecindario de $\mathbf{x}_i$ [Fig. 2.7 (a)]. Para continuar el rastreo de $\mathcal{M}$, el método escoge una dirección $\mathbf{s}_j^i \in \mathcal{T}_i$, proyecta el punto $\mathbf{x}_j^i = \mathbf{x}_i + \mathbf{s}_j^i$ sobre $\mathcal{M}$ para obtener $\mathbf{x}_j$, y genera una nueva carta sobre $\mathcal{T}_j$, el espacio tangente a $\mathcal{M}$ en $\mathbf{x}_j$. El método guarda traza de las regiones de $\mathcal{M}$ exploradas recortando mútuamente las cartas definidas sobre $\mathcal{T}_i$ y $\mathcal{T}_j$ [Fig. 2.7 (b)], e itera el proceso hasta que se localiza $\mathbf{x}_g$ [Fig. 2.7 (c)], o bien se agota todo el componente conexo alcanzable desde $\mathbf{x}_s$. La estrategia dispone de mecanismos para adaptar la extensión de cada carta a la curvatura de $\mathcal{M}$ [32], y se puede avanzar desde $\mathbf{x}_i$ hasta $\mathbf{x}_j$ mediante técnicas de multiprecisión que aseguren la continuidad [35].

El atlas se puede extender en amplitud, como en la Fig. 2.7 (c), o guiándolo heurísticamente hacia $\mathbf{x}_g$. Una posibilidad es utilizar una estrategia A* [36], mediante la cual sólo se generan las cartas estrictamente necesarias para obtener un camino de coste mínimo $\mathbf{x}_s \rightarrow \mathbf{x}_g$. En cada iteración, este método avanza desde la carta cuyo centro $\mathbf{x}_i$ da lugar al menor coste estimado en el movimiento $\mathbf{x}_s \rightarrow \mathbf{x}_g$, manteniendo una cola ordenada de caminos alternativos. El coste mencionado es la suma de un término $g(\mathbf{x}_i)$, que proporciona el menor coste conocido para la transición $\mathbf{x}_s \rightarrow \mathbf{x}_i$, y un término $h(\mathbf{x}_i)$, que proporciona una cota inferior del coste de la transición $\mathbf{x}_i \rightarrow \mathbf{x}_g$. El valor del primer término se actualiza durante la expansión del atlas utilizando una función $c(\mathbf{x}_k, \mathbf{x}_l)$ que determina el coste de transición entre los centros de dos cartas adyacentes centradas en $\mathbf{x}_k$ y $\mathbf{x}_l$.

Durante la construcción del atlas, se va construyendo un grafo G cuyos nodos representan los centros $\mathbf{x}_i$ de las cartas generadas, y cuyas aristas guardan las relaciones de vecindad entre cartas. Cuando se alcanza $\mathbf{x}_g$, por lo tanto, se puede usar este grafo para generar un camino de coste mínimo que una $\mathbf{x}_s$ con $\mathbf{x}_g$, de acuerdo con la función $c(\mathbf{x}_k, \mathbf{x}_l)$ asumida. La función puede ser cualquiera, pudiendo reflejar el consumo energético del robot, la distancia que recorre, o una penalización debida a colisiones no consideradas en $\mathcal{M}$. Para tener en cuenta estas colisiones, la función sólo tiene que asignar un coste infinito a las transiciones $\mathbf{x}_k \rightarrow \mathbf{x}_l$ que darían lugar a una colisión [1]. En nuestro caso, no hay colisiones adicionales a las ya consideradas, y simplemente hemos escogido $c(\mathbf{x}_j, \mathbf{x}_k) = \|\mathbf{x}_j - \mathbf{x}_k\|^2$, que tiende a proporcionar el camino de longitud mínima sobre $\mathcal{M}$.

La estrategia A* es rápida cuando la variedad rastreada es de dimensión uno o dos, pero los tiempos de cálculo incrementan notablemente en dimensión tres o superiores. Por ello, sólo es aconsejable aplicar la estrategia A* cuando planifiquemos movimientos en los que alguna

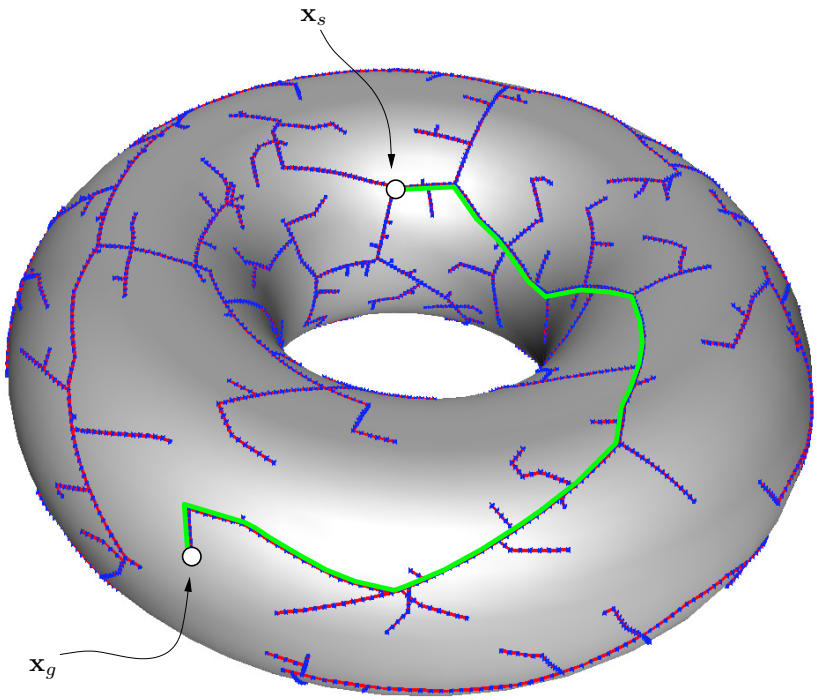


Figura 2.8: Resultado de la estrategia de continuación basada en árboles de expansión rápida al conectar dos puntos $\mathbf{x}_s$ y $\mathbf{x}_g$ sobre la superficie de un toro. El camino encontrado se indica en verde.

variable del robot quede fijada, como por ejemplo el ángulo θ_8 de la plataforma constante. Para movimientos generales, $\mathcal{M}$ tendrá dimensión tres, y será mejor utilizar la estrategia propuesta en [2], basada en el crecimiento de árboles de expansión rápida. Este método utiliza un atlas parcial de $\mathcal{M}$ para extender un árbol de expansión sobre $\mathcal{M}$, el cual, a su vez, se utiliza para decidir en qué direcciones hay que extender el atlas parcial. La Fig. 2.8 ilustra el resultado sobre la superficie de un toro.

En ambas estrategias, el camino obtenido viene dado en forma de una secuencia de n_c cartas

$$C_1, C_2, \ldots, C_{n_c}$$

donde cada carta C_i está centrada en un punto $\mathbf{x}_i \in \mathcal{M}$. La primera carta y la última están centradas en $\mathbf{x}_s$ y $\mathbf{x}_g$, y las cartas consecutivas son adyacentes entre sí. La representación del camino $\mathbf{x}_s \to \mathbf{x}_g$ es, por lo tanto, discreta, puesto que sólo se proporcionan los puntos de paso intermedios $\mathbf{x}_2, \ldots, \mathbf{x}_{n_c-1}$. Sin embargo, el camino se puede refinar tanto como se quiera, usando los planos tangentes a las cartas para añadir puntos intermedios en cada transición $\mathbf{x}_i \to \mathbf{x}_{i+1}$. Si bien es cierto que el camino obtenido puede no ser suave, utilizaremos técnicas de suavizado

que lo deforman para que sí lo sea [12, 34].

Al final, el camino $\mathbf{x}_s \to \mathbf{x}_g$ se obtiene en forma de una secuencia de n_p puntos sobre $\mathcal{M}$

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n_p},$$

donde $\mathbf{x}_1 = \mathbf{x}_s$, $\mathbf{x}_{n_p} = \mathbf{x}_g$, y el camino $\mathbf{q}_s \to \mathbf{q}_g$ sobre $\mathcal{C}$ se devuelve tomando las coordenadas $\mathbf{q}$ de cada uno de estos puntos, dando lugar a la secuencia

$$\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{n_p}, \tag{2.28}$$

donde $\mathbf{q}_i$ es la proyección de $\mathbf{x}_i$ sobre $\mathcal{C}$.

Todos estos métodos de búsqueda y suavizado estan implementados en el paquete de código abierto *Cuik*, desarrollado por el Grupo de Cinemática y Diseño de Robots del IRI [34]. El planificador de este proyecto, por lo tanto, se ha implementado realizando llamadas a bajo nivel sobre los comandos de este paquete.

2.4.4. Visualización del espacio de configuraciones

Para apreciar la estructura de $\mathcal{C}$, la Fig. 2.9 muestra dos secciones del espacio de trabajo del robot, suponiendo que el ángulo θ_8 se mantiene constante, a $\theta_8 = -\pi/4$ y $\theta_8 = 0$ respectivamente. La zona gris corresponde a los puntos del plano XY que puede alcanzar el baricentro P del triángulo móvil. Las curvas azules corresponden a singularidades inversas del robot, y son los puntos en los que el efector final tiene movilidad reducida. Las curvas rojas, en cambio, corresponden a singularidades directas, o puntos en los que se da una pérdida de rigidez. Como se ve, estas curvas son de una notable complejidad. Bonev demostró, de hecho, que quedan descritas por polinomios irreductibles de grado 42 en las coordenadas de P [37].

Ambas curvas se han calculado utilizando los métodos de poda y bisección del paquete *Cuik*, siguiendo la metodología explicada en [27], y utilizando los ficheros de cálculo del Apéndice B, Sección B.2. Si se cambia el valor de θ_8 en tales ficheros, se puede observar que la complejidad de la curva de singularidades es parecida en muchas otras secciones de $\mathcal{C}$, lo cual, como ya se ve, dificulta notablemente el problema de planificación.

A partir de la Fig. 2.9, no obstante, podría parecer que el rango de movimientos viables del robot queda severamente reducido por la presencia de singularidades, pero fijémonos que estamos observando una proyección del espacio de configuraciones sobre las coordenadas (x, y) de P, y que para cada pose (x, y, θ_8) del efector hay un total de ocho configuraciones compatibles del robot. Cada configuración corresponde a un *modo de trabajo distinto* del mecanismo, identificado mediante una tripleta de signos $(\sigma_1, \sigma_2, \sigma_3)$, donde σ_i nos da la orientación del triángulo

$J_{1,i}J_{2,i}J_{3,i}$ [37]. El espacio $\mathcal{C}$, por lo tanto, tiene una estructura muy rica, estando formado por distintas "capas" correspondientes a los distintos modos de trabajo. Si proyectamos cada una de estas capas por separado, podemos ver que las regiones libres de singularidades son mucho mayores (Fig. 2.10). Observemos que las dos configuraciones marcadas mediante puntos sólidos en la Fig. 2.10, por ejemplo, se pueden conectar mediante caminos libres de singularidades en las capas $(+,+,+)$ y $(-,+,-)$, mientras que esto parecía imposible en la Fig. 2.9.

Para verificar la estrategia de continuación, y apreciar el efecto de las restricciones de exclusión de singularidades y autocolisiones, hemos definido las ecuaciones de $\mathcal{M}$ en el fichero `scf.cuik` indicado en el Apéndice B, y hemos ejecutado el método de continuación desde una configuración en la que el triángulo del efector se encuentra concéntrico y paralelo al de la base en todos sus lados. Si fijamos $\theta_8 = 0$, $[b_{min}, b_{max} = [-4, 4]$, y mantenemos constante el modo de trabajo de las patas, entonces el método obtiene los rangos de movimiento indicados en azul en la Fig. 2.11. Para cada subfigura, la tupla de signos indica el modo de trabajo que se ha mantenido constante. Como se ve, el método explora una región que respeta los límites impuestos por las singularidades, y analizando las fronteras vemos que las que no son cercanas a las curvas rojas coinciden, como era de esperar, con configuraciones en autocolisión.

Si ahora partimos de la misma configuración inicial y mantenemos $\theta_8 = 0$, pero permitimos cambios de signo en el modo de trabajo, entonces el método encuentra regiones de movimiento

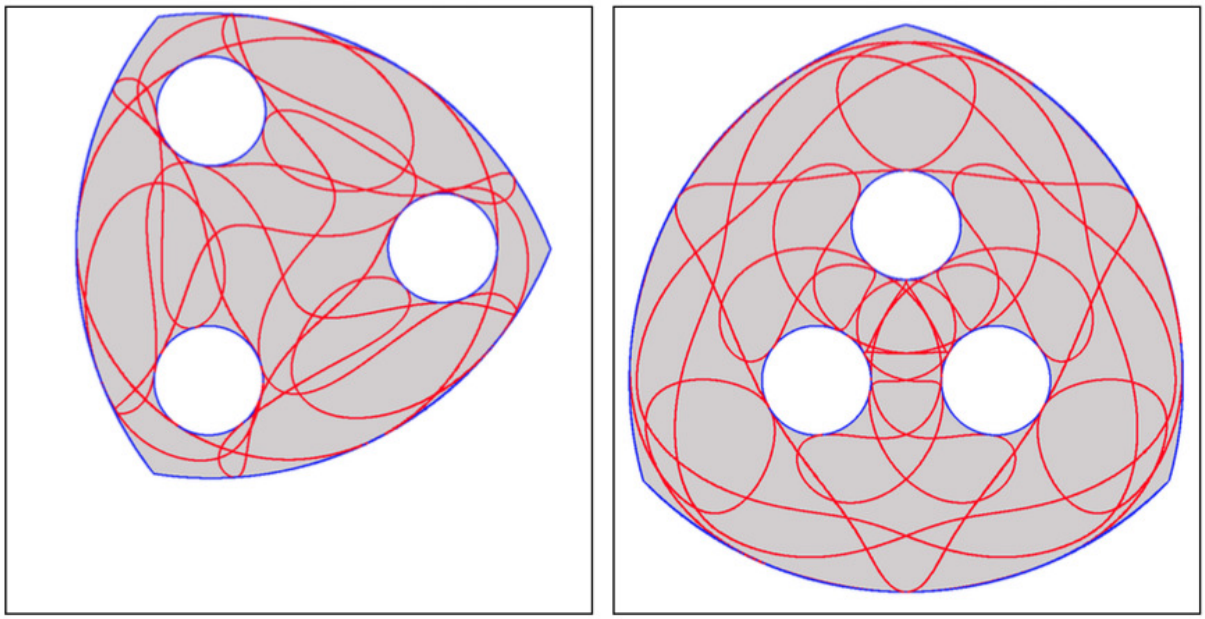


Figura 2.9: Secciones del espacio de trabajo del robot 3-RRR, fijando el ángulo del efector a $\theta_8 = -\pi/4$ y $\theta_8 = 0$ respectivamente.

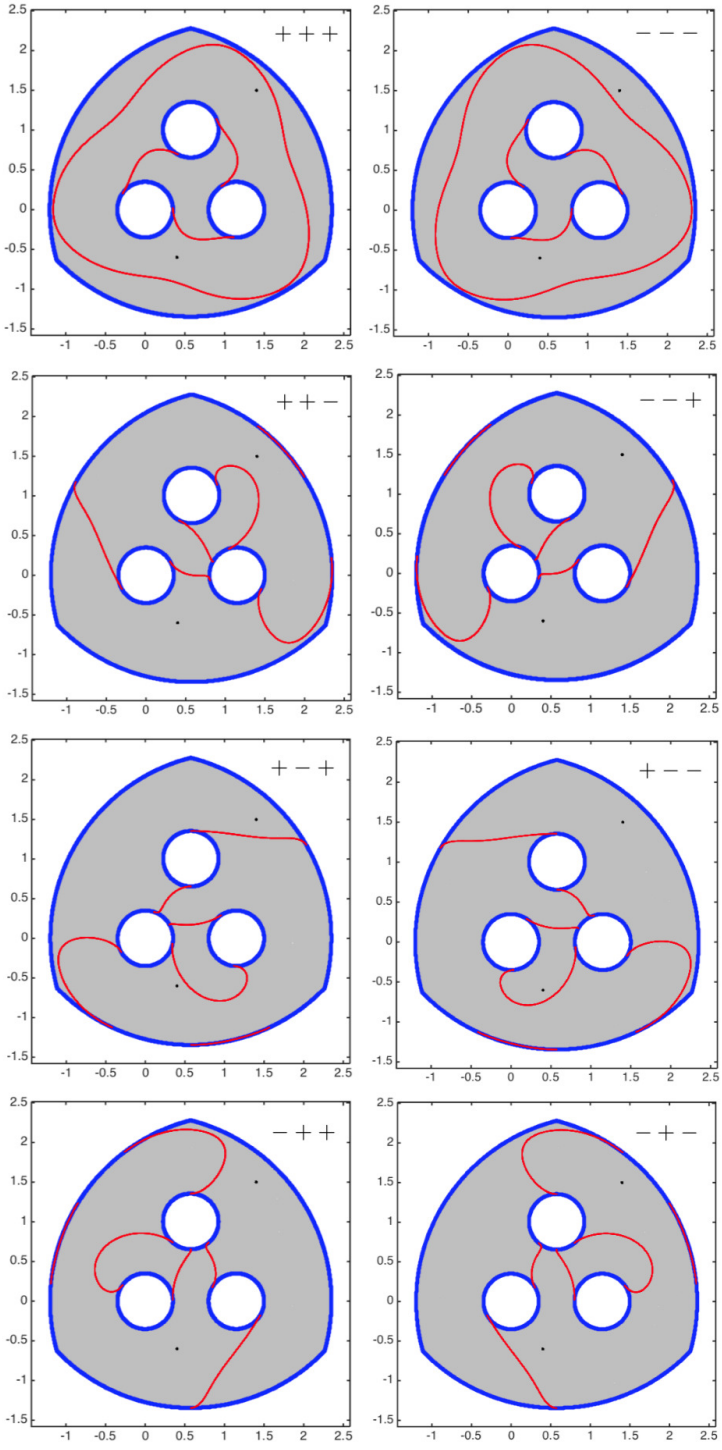


Figura 2.10: La capas de la sección $\theta_8 = 0$ que se obtienen al fijar cada modo de trabajo.

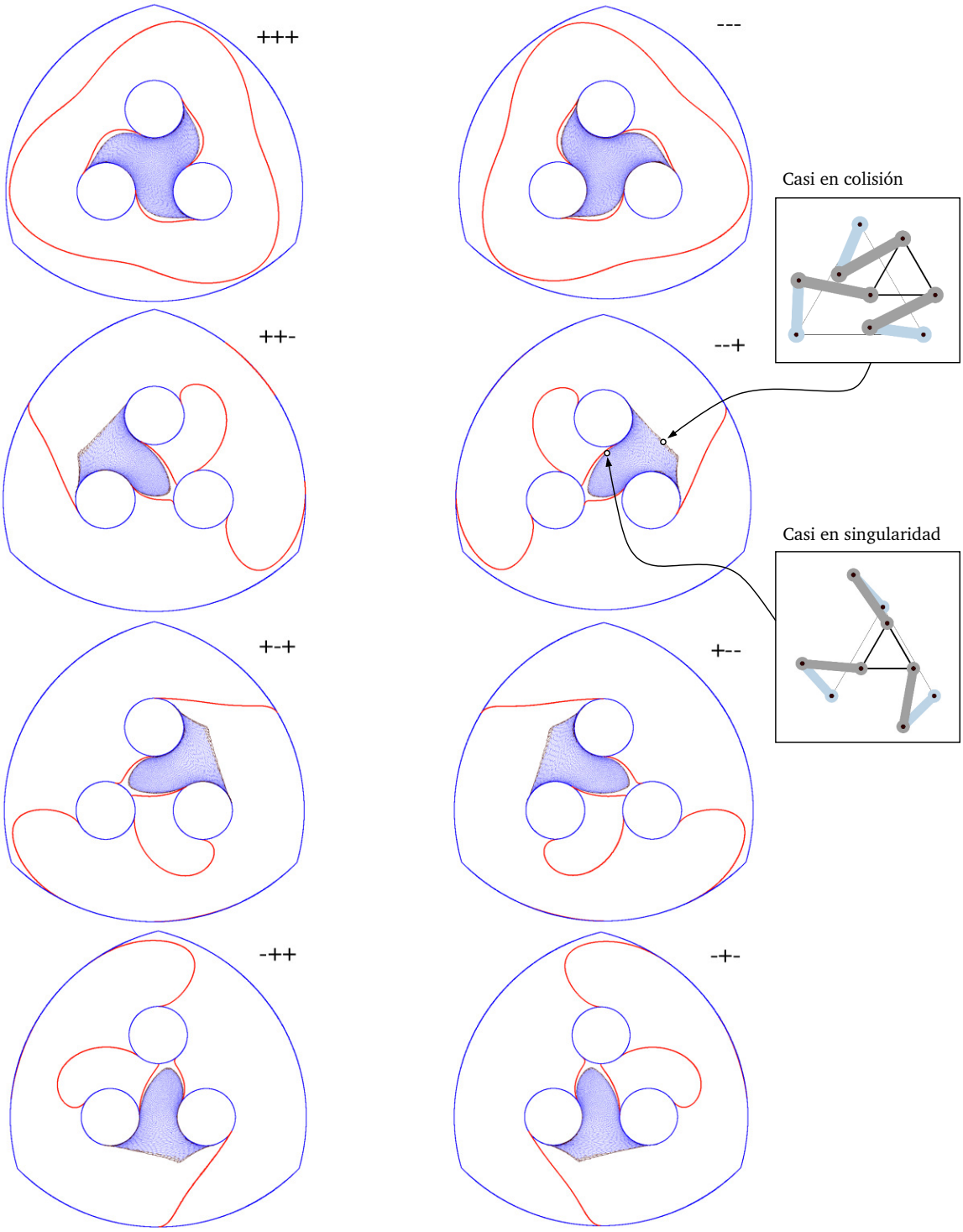


Figura 2.11: Exploración de $\mathcal{C}$ manteniendo $\theta_8 = 0$ y fijando el modo de trabajo al valor indicado.

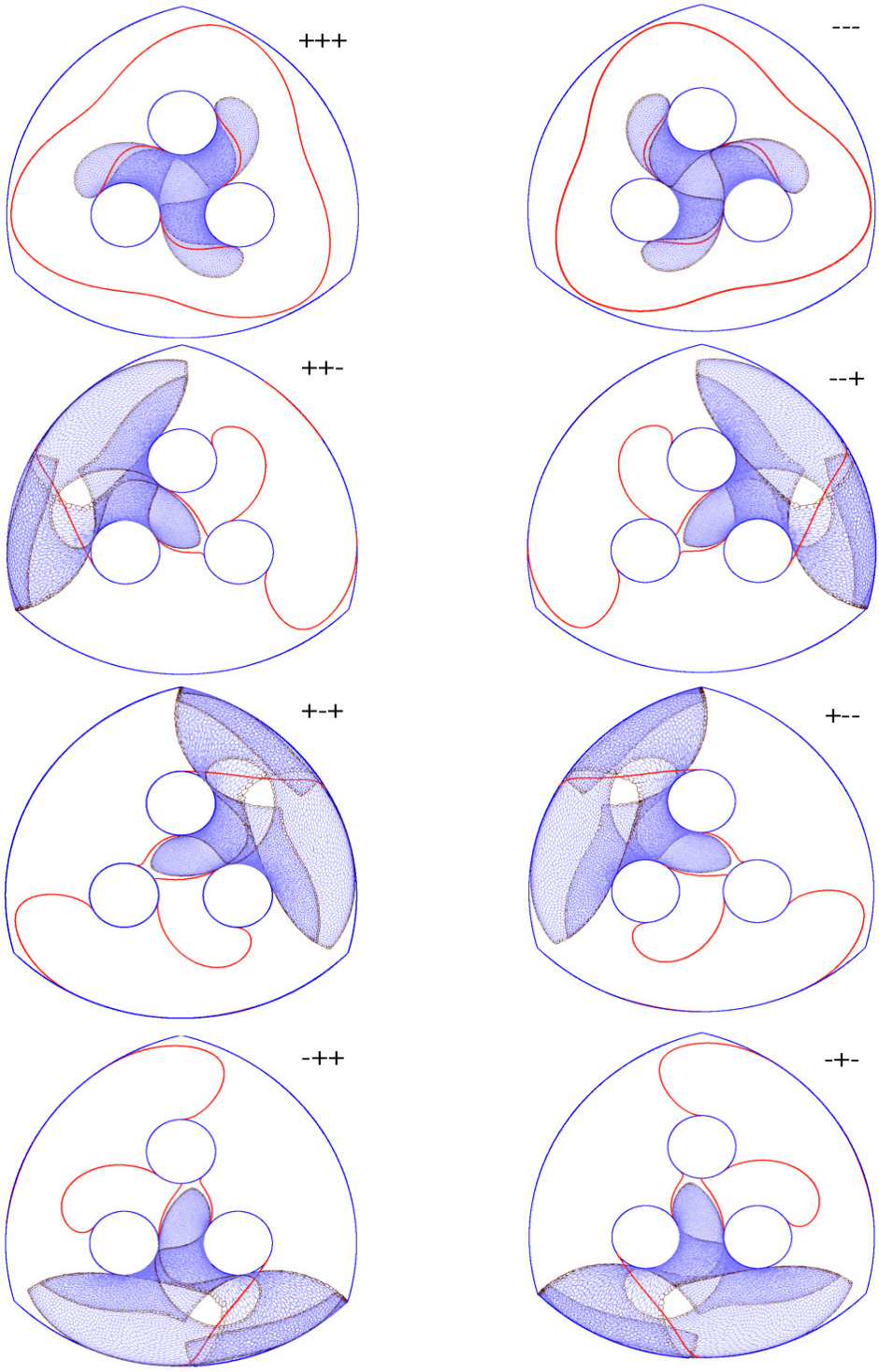


Figura 2.12: Exploración con $\theta_8 = 0$ pero permitiendo cambios de modo de trabajo en las patas.

factible mucho mayores (Fig. 2.12). Vemos que, de esta manera, el robot puede cambiar entre las distintas zonas indicadas en la Fig. 2.10.

El mayor rango de movimiento del robot, sin embargo, se obtiene permitiendo que θ_8 sea variable, motivo por el cual hemos preferido que los movimientos que planifiquemos se hagan siempre bajo esta asunción.

2.5 Conversión de caminos en trayectorias

En la Sección 2.4 hemos proporcionado un método que permite obtener un camino contínuo $\mathbf{q}_s \to \mathbf{q}_g$ sobre $\mathcal{C}$. Este camino se obtiene muestreado, en la forma descrita en la expresión (2.28), y nuestro objetivo es ahora generar una *trayectoria* en el espacio articular de los tres ángulos actuados

$$\boldsymbol{\theta}(t) = [\theta_2(t), \theta_3(t), \theta_4(t)] \tag{2.29}$$

de manera que el robot pueda seguir el camino sobre $\mathcal{C}$ al ejecutarla. Esta trayectoria deberá tener en cuenta las limitaciones de velocidad de los motores montados, y las necesidades del sistema de control utilizado.

Como se verá en la Sección 3.6, implementaremos un sistema de control digital, y necesitaremos tanto de $\boldsymbol{\theta}(t)$ como $\dot{\boldsymbol{\theta}}(t)$ en sus formas muestreadas

$$\begin{aligned}\boldsymbol{\theta}(kT_s) &= [\theta_2(kT_s), \theta_3(kT_s), \theta_4(kT_s)],\\ \dot{\boldsymbol{\theta}}(kT_s) &= [\dot{\theta}_2(kT_s), \dot{\theta}_3(kT_s), \dot{\theta}_4(kT_s)],\end{aligned}$$

donde k es el índice de muestra, y T_s es un período de tiempo fijo prestablecido.

Obtendremos $\boldsymbol{\theta}(kT_s)$ y $\dot{\boldsymbol{\theta}}(kT_s)$ en cuatro pasos (Fig. 2.13):

1. Proyectaremos los puntos $\mathbf{q}_1, \ldots, \mathbf{q}_{n_p}$ de (2.28) sobre el espacio articular $\theta_2\theta_3\theta_4$, obteniendo los puntos

   $$\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_{n_p}, \tag{2.30}$$

   donde $\boldsymbol{\theta}_i$ es el subvector de $\mathbf{q}_i$ formado por θ_2, θ_3, y θ_4.

2. Ajustaremos un *spline* paramétrico $\boldsymbol{\theta}(s) = [\theta_2(s), \theta_3(s), \theta_4(s)]$ a la curva poligonal inducida por $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n_p}$, donde s es el parámetro arco del *spline* [Fig. 2.13(a)]. Este ajuste también nos proporcionará el vector velocidad del spline $\dot{\boldsymbol{\theta}}(s) = [\dot{\boldsymbol{\theta}}_2(s), \dot{\boldsymbol{\theta}}_3(s), \dot{\boldsymbol{\theta}}_4(s)]$.

3. Muestrearemos $\boldsymbol{\theta}(s)$ y $\dot{\boldsymbol{\theta}}(s)$ para distintos valores de s, obteniendo una nueva secuencia de n_t puntos del espacio articular, $\boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \ldots, \boldsymbol{\psi}_{n_t}$, consecutivamente equiespaciados una

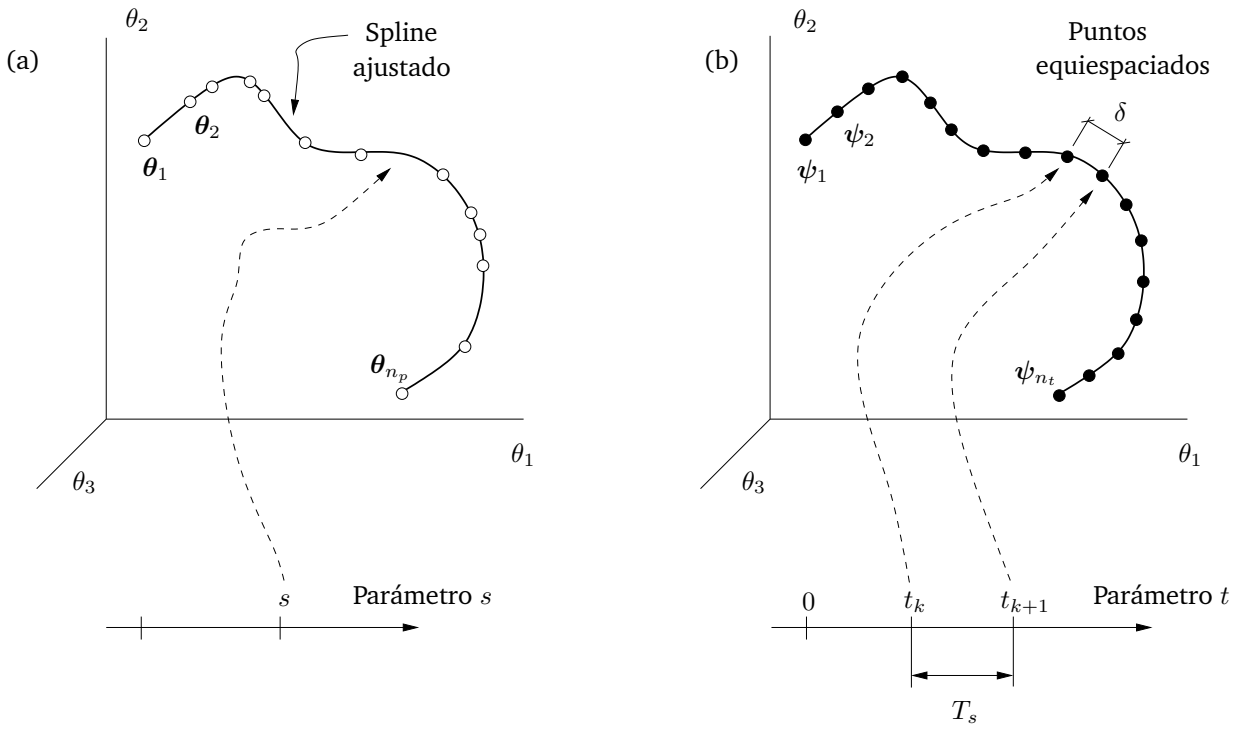


Figura 2.13: Obtención de la trayectoria angular muestreada. (a) Ajuste de un *spline* parametrizado mediante el parámetro arco a la secuencia inicial de puntos del camino. (b) Muestreado del *spline* mediante puntos equiespaciados, y asignación de incrementos de tiempo constante entre puntos.

distancia δ sobre el *spline* [(Fig. 2.13(b)], junto con sus respectivos vectores velocidad $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \ldots, \boldsymbol{\omega}_{n_t}$.

4. Asignaremos el instante de tiempo $t_k = kT_s$ a cada muestra $\boldsymbol{\psi}_i$, y a su respectivo vector velocidad $\boldsymbol{\omega}_i$, de manera que las funciones deseadas seran, finalmente

$$\boldsymbol{\theta}(kT_s) = \boldsymbol{\psi}_{k+1},$$
$$\dot{\boldsymbol{\theta}}(kT_s) = \boldsymbol{\omega}_{k+1},$$

para $k = 0, \ldots, n_t - 1$.

En relación a este procedimiento, hay que tener en cuenta los siguientes aspectos. En primer lugar, al muestrear el *spline* mediante puntos equiespaciados en δ se asegura que el incremento angular máximo entre muestras consecutivas es inferior a δ en cualquiera de los ángulos θ_2, θ_3, y θ_4. Al ser constante T_s, esto asegura que la velocidad angular máxima que deberá suministrar un motor a lo largo de la trayectoria es $\omega_{max} = \frac{\delta}{T_s}$. Deberemos escoger δ y T_s de manera que

ω_{max} esté dentro del rango admitido por los actuadores del robot, procurando, también, que las aceleraciones sean bajas, para simplificar posteriormente el sistema de control (Sección 3.6). Además, el incremento δ se debe fijar lo suficientemente pequeño como para que el camino articular quede bien discretizado.

En segundo lugar, el equiespaciado de puntos sobre el *spline*, y la constancia de T_s, aseguran que recorremos el camino articular a velocidad constante. Esto nos protege de cambios de velocidad inesperados al pasar cerca de las singularidades inversas. Si hubiéramos escogido movernos a velocidad constante sobre el espacio de tareas x-y-θ_8, por ejemplo, en los cambios de modo de trabajo notaríamos el efecto de la sobreaceleración articular [27], que daría lugar a cambios bruscos de velocidad en los actuadores, quizá inadmisibles, y a efectos inerciales indeseados.

En tercer lugar, para ajustar el *spline* $\boldsymbol{\theta}(s)$, y obtener sus derivadas $\dot{\boldsymbol{\theta}}(s)$, hemos utilizado la estrategia propuesta en [38], que formula el problema en términos de ecuaciones diferenciales que describen la parametrización buscada. Esta estrategia nos permite encontrar un *spline* que pasa por los puntos $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n_p}$ subministrados, parametrizado mediante el parámetro arco s. Este parámetro toma valores en el rango $[0, 1]$, de manera que $\boldsymbol{\theta}(0) = \boldsymbol{\theta}_1$ y $\boldsymbol{\theta}(1) = \boldsymbol{\theta}_{n_p}$, y s es proporcional a la longitud del camino articular recorrido desde $\boldsymbol{\theta}(0)$ hasta $\boldsymbol{\theta}(s)$. Por lo tanto, para encontrar puntos equiespaciados sobre el camino, basta con evaluar $\boldsymbol{\theta}(s)$ para $s = k\frac{\delta}{L}$, $k = 1, \ldots, n_t - 1$, donde L es la longitud total del camino articular.

Cabe decir, finalmente, que un mejor procedimiento para la obtención de $\boldsymbol{\theta}(kT_s)$ hubiera consistido en reparametrizar el *spline* $\boldsymbol{\theta}(s)$ en función del tiempo de acuerdo con la rapidez de ejecución deseada, y analizar el espectro frecuencial de sus señales $\theta_2(t)$, $\theta_3(t)$, y $\theta_4(t)$ resultantes. Un muestreo correcto de estas señales requiere respetar el Teorema de Shannon, con lo cual deberíamos encontrar la frecuencia máxima del espectro, f_{max}, y muestrear las señales con un período $T_s \leq \frac{\pi}{f_{max}}$. Para simplificar la implementación, sin embargo, se ha preferido el método anteriormente expuesto, escogiendo un valor δ que proporcione suficientes puntos representativos sobre el camino a recorrer.

3

Ejecución y control de las trayectorias

En este capítulo se detalla cómo se llega a la ejecución de una trayectoria previamente calculada. Para tal fin, necesitaremos un robot 3-RRR, un dispositivo que lo controle, y una interfaz de usuario que nos facilite actuar sobre el robot y analizar su funcionamiento. Empezaremos viendo qué componentes físicos se han utilizado, sus características y su función dentro del sistema. A continuación, se definirá el modo en el que se comunican los tres elementos principales del sistema (el servomotor, la placa de control y la interfaz de usuario). Se seguirá con una descripción general de las funciones de la interfaz gráfica y, para finalizar, se expondrá la estrategia de control adoptada, así como los métodos seguidos para el ajuste de sus controladores.

3.1 Arquitectura del sistema

Para poder probar de forma práctica el planificador de trayectorias libres de singularidades y colisiones, se ha diseñado y construido un robot 3-RRR (Fig. 3.1). El sistema desarrollado consta de tres bloques diferenciados (Fig. 3.2):

- *Bloque mecánico*. Incluye la base, la plataforma móvil, eslabones, soportes, y para cada una de las tres patas, un servomotor como actuador, un codificador angular situado en la articulación intermedia, y un conector rotativo que nos permite llevar la señal del codificador angular a la base del robot.
- *Bloque de control*. Tiene como función la comunicación directa con los tres servomotores, actuando de puente entre el usuario y el robot físico. Con este fin se utiliza una placa Arduino, más una circuitería asociada diseñada para adaptarse al protocolo del servomotor.
- *Interfaz de usuario*. Se ha implementado una interfaz en Matlab desde la que el usuario puede, entre otras opciones, ver la configuración actual del robot, planificar, simular o ejecutar trayectorias entre configuraciones dadas.

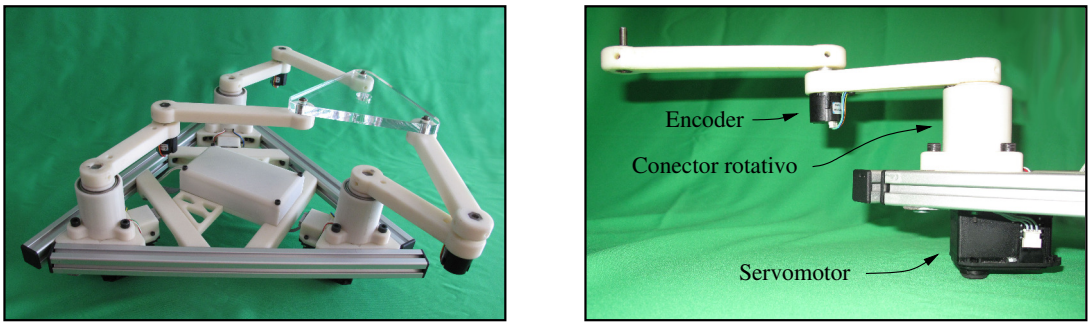


Figura 3.1: Imagen del bloque mecánico y detalle de una pata.

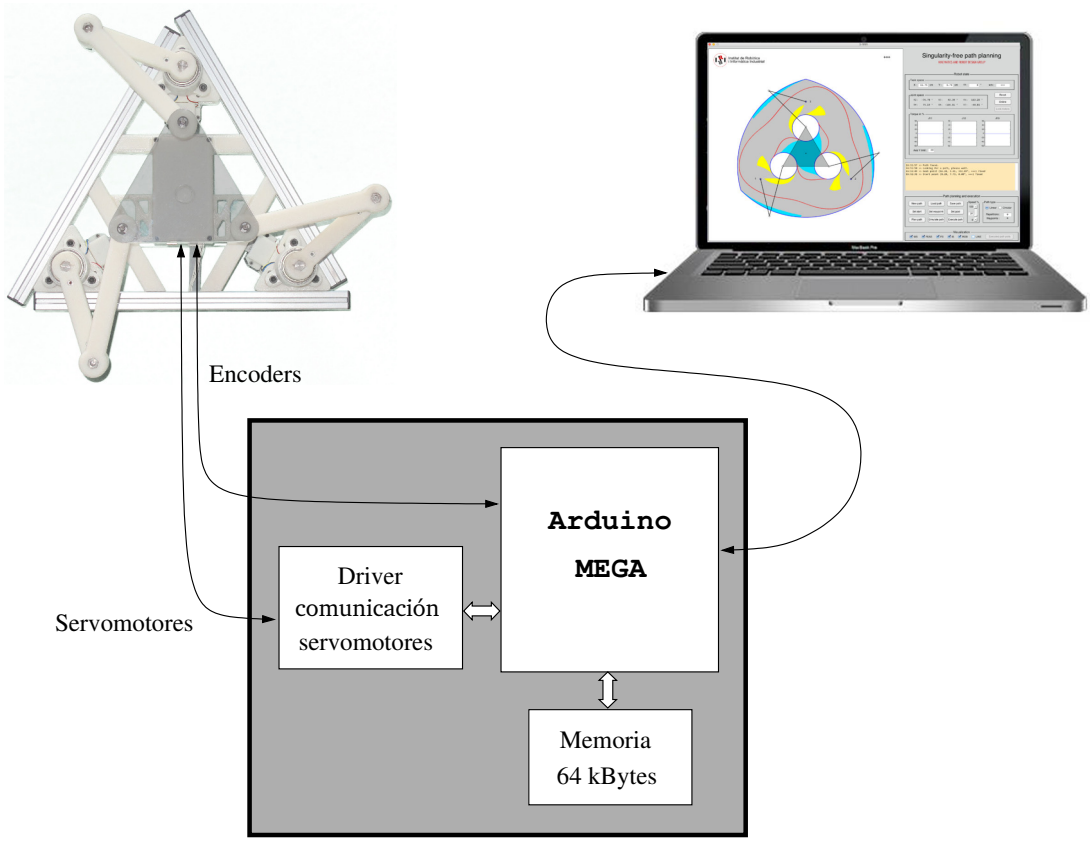


Figura 3.2: Diagrama de bloques de la solución adoptada.

3.2 Elementos integrantes

3.2.1. Servomotor

Debido a que se pretende que el robot sea fácilmente reproducible y de coste moderado, pero a la vez permita una ejecución suficientemente precisa de trayectorias, se ha seleccionado como actuador el servomotor MX-64T de la serie Dynamixel de la empresa Robotis (Fig. 3.3). Este servomotor, de prestaciones elevadas, ha sido diseñado para aplicaciones robóticas no industriales. Puede trabajar en tres modos distintos: en "modo articulación" (pensado para seguir consignas de posición de 0 a 360º), en "modo rueda" (giro continuo a una velocidad especificada), y en "modo multivuelta" (pensado para seguir consignas de posición en un rango de ± 7 vueltas). Algunas de sus características principales son:

- Microcontrolador ST CORTEX-M3 (STM32F103C8 @ 72MHZ, 32BIT)
- Codificador angular magnético de 12 bits, rango 360º. (resolución: 0.088º).
- Motor Maxon sin escobillas, serie RE-Max.
- Velocidad máxima de 63 rpm (alimentado a 12V).
- Comunicación serie semi-dúplex (8 bits, 1 stop, sin paridad), velocidad hasta 4.5 Mbps
- Par máximo de 6 Nm a 12V (4.1A).
- PID integrado.
- Realimentación de posición, velocidad, carga, voltaje de entrada, temperatura y corriente.
- Modos de funcionamiento: articulación, rueda y multivuelta.

Para comunicarnos con este servomotor, hay que enviarle instrucciones y en su caso recibir su respuesta, todo ello siguiendo el protocolo de comunicación *Dynamixel communication 1.0* [39].

Figura 3.3: Servomotor Dynamixel MX-64T.

Un ejemplo de instrucción podría ser la lectura de la posición o velocidad actual, o la escritura de consignas de posición, velocidad, parámetros del PID interno, cambios de modo de trabajo, etc. Toda la información relativa al estado actual y el modo de operación del servomotor la tenemos disponible en su tabla de control [40]. El modo de interactuar con el MX-64 será leyendo y/o escribiendo en dicha tabla.

3.2.2. Codificador angular

El codificador angular absoluto utilizado es el modelo AEAT-6012 de la marca AVAGO (Fig. 3.4). Es un codificador angular magnético de bajo coste, y de características muy similares al implementado en el servomotor MX-64T. Su función será permitir la medida de los ángulos de las articulaciones intermedias, para así poder conocer la configuración del robot. Algunas de sus características principales son:

- Rango de medición de 0 a 360º.
- Resolución de 12 bits. (0.088º)
- Velocidad de rotación máxima: 12.000 rpm.
- Salida serie SPI, frecuencia máxima: 1 MHz.

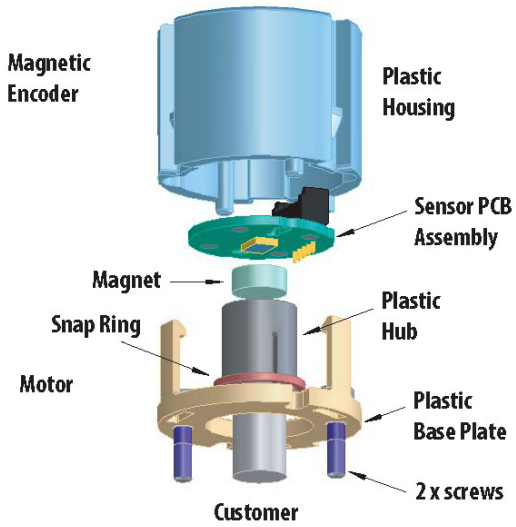


Figura 3.4: Codificador angular AVAGO AEAT-6012.

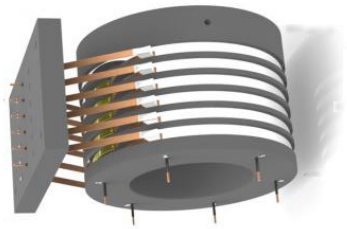


Figura 3.5: Esquema de funcionamiento de un conector rotativo.

3.2.3. Conector rotativo

Un conector rotativo[1] es un sistema electromecánico que posibilita la transmisión de señales desde un elemento fijo a otro rotativo. Este componente nos permitirá acoplar la señal del codificador angular de la articulación intermedia a la base del manipulador, evitando así tener que utilizar un cable directo entre codificador y base, que estaría expuesto a la fatiga de la rotación de la articulación de la base y a enrollamientos indeseados. El modelo utilizado es el BTH1256 de la marca ByTune Electronics (Fig. 3.5). Algunas de sus características principales son:

- Número de circuitos: 6.
- Corriente máxima por circuito: 5A.
- Velocidad de rotación máxima: 1.000 rpm.

3.2.4. Placa Arduino

Arduino es una plataforma de hardware y código abierto, de bajo coste, muy extendida para la realización de prototipos electrónicos de complejidad media-baja. Su hardware consta de una placa electrónica con un microcontrolador y diversos periféricos como puertos de entrada/salida analógicos y digitales, generadores de PWM, puertos de comunicación, etc. Existen muchas versiones diferentes de hardware Arduino, cada una con sus características[2]. Se programa mediante un lenguaje de programación propio (basado en Wiring), y el software de Arduino (IDE), basado en Processing [41].

Para este proyecto se ha seleccionado la placa Arduino Mega Rev. 3, básicamente por tener más de un puerto serie hardware y por destacar dentro de estos dispositivos por su tamaño de

[1] También llamado de anillos rozantes, colector de anillos, o *slip ring* en inglés.

[2] En la página https://www.arduino.cc/en/Main/Products podremos encontrar una comparación de las características principales de las diferentes placas Arduino.

memoria RAM. Sus funciones en el sistema serán:

- Hacer posible la comunicación entre los servomotores y la interfaz gráfica de Matlab.
- Permitir la lectura de los codificadores angulares.
- Controlar la lectura-escritura en la memoria externa.
- Control a bajo nivel de la trayectoria planificada.

3.2.5. Driver de comunicación con el servomotor

Para controlar los actuadores Dynamixel, la placa Arduino requiere de un driver para convertir su señal del puerto serie, duplex, (dos lineas de comunicación, RX y TX) a una señal semi-dúplex (una línea de comunicación en la que se alternan señales de transmisión y recepción). Con este fin se ha utilizado el integrado TTL 74F244, compuesto por ocho buffers con salida tri-estado. (Fig. 3.6)

Los pines 2 y 3 de Arduino nos servirán para marcar los intervalos en los que la señal serie semi-dúplex del bus de los servomotores estará en modo de transmisión o de recepción.

3.2.6. Memoria externa

Utilizaremos una memoria serie SPI SRAM de 512 kbits, modelo 23LCV512, para almacenar dos tipos de datos:

- La tabla de consignas que Matlab enviará a Arduino cada vez que se tenga que ejecutar una trayectoria en el robot. Por cada punto de la trayectoria, se enviarán 14 bytes almacenados en una fila de la tabla de consignas.

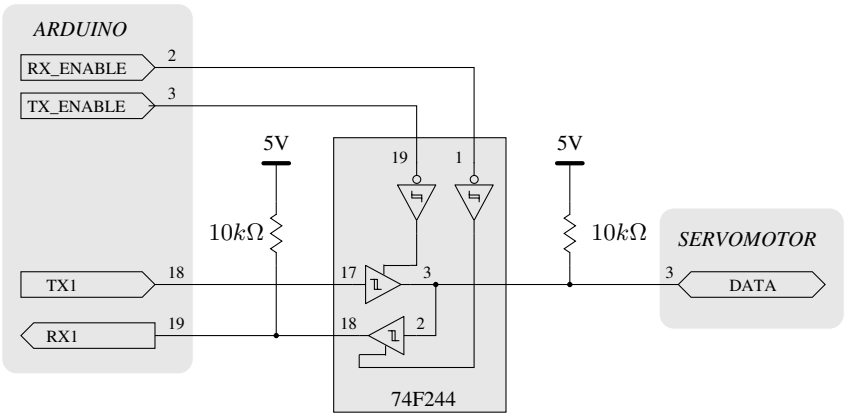


Figura 3.6: Esquema del driver de comunicación implementado.

- La información leída desde los servomotores mientras se está ejecutando una trayectoria en el robot. Por cada punto de la trayectoria se generará una muestra de 42 bytes.

Dividiendo la capacidad total de la memoria (64 kBytes) por la cantidad de bytes que deberemos almacenar en la memoria externa por cada punto de la trayectoria (56 bytes), obtenemos que el número máximo de puntos que podrá tener la trayectoria a ejecutar es de 1142. Teniendo en cuenta este valor, se ha dividido la memoria externa en dos bloques (Tabla 3.1). En el primero se almacenará la tabla de consignas y en el segundo las muestras adquiridas durante la trayectoria.

Área	Tamaño(bytes)	Dirección
Tabla de consignas	15.988	0 .. 3E74h
Muestras	47.964	3E75h .. F9D1h

Tabla 3.1: Uso de la memoria externa SRAM.

3.2.7. Conversor serie-USB

Para facilitar la depuración del código de la placa Arduino, se ha incorporado un conversor serie-USB. Este nos permite utilizar el puerto serie 2 de Arduino Mega para enviar información, vía USB, a una aplicación de terminal. En el programa de Arduino se ha creado una variable booleana llamada *DEBUG*, que habilita el envío de dicha información, que incluye el estado actual del programa, la ocurrencia de errores al enviar o recibir tramas de comunicación o la ejecución de determinadas funciones, entre otros aspectos.

3.3 Conexionado de los elementos integrantes

En la Fig. 3.7 se representa de forma esquemática la conexión física de todos los elementos de la parte hardware del sistema. Cabe destacar tres aspectos de la implementación efectuada:

- El módulo USB-Serie es prescindible durante la ejecución, ya que se ha usado en la etapa de depuración del programa realizado en Arduino.
- Los dos pulsadores indicados en el esquema no tienen ninguna función asignada en la versión final, por lo que también son prescindibles.
- La placa Arduino Mega viene habilitada por defecto con la opción de efectuar un reset automático por software. Esto permite que la escritura en el dispositivo la pueda iniciar el software de programación sin necesidad de pulsar el botón de reset. Sin embargo, en la aplicación final se ha desactivado dicha opción (cortando una pista del circuito impreso

diseñada con este propósito) ya que este reset provocaba un retardo de unos dos segundos cada vez que se abría el puerto de comunicación con el robot. Esto implica que para programar la placa Arduino, tras pulsar el botón *Upload* de la aplicación Arduino, habrá que pulsar el botón físico de Reset para que la placa Arduino entre en modo programación.

3.4 Comunicación entre dispositivos

Para que dos elementos capaces de enviar y recibir información se comuniquen con éxito se requiere que "hablen el mismo idioma". En esta sección nos centraremos en ver de qué modo se comunican los tres elementos más relevantes del sistema. En un primer apartado se tratará la comunicación servomotor – Arduino, que como veremos se rige por el protocolo establecido por Robotis para el control de sus servomotores Dynamixel [39]. El segundo apartado lo dedicaremos a ver la comunicación Arduino – Matlab. En este caso se ha desarrollado un protocolo específico con el objeto de establecer una comunicación fiable entre ambos dispositivos.

3.4.1. Comunicación servomotor - Arduino

El servomotor MX-64T y la placa Arduino se comunican mediante el envío y recepción de datos encapsulados en paquetes, siguiendo el protocolo definido por Robotis, *Dynamixel communication 1.0*. En este protocolo se establecen dos tipos de paquetes:

- ***Instruction packet***. Paquete enviado por el dispositivo maestro para controlar el servomotor. La estructura de un *Instruction Packet* es:

0xFF	0xFF	ID	LENGTH	INSTRUCTION	PARAMETER 1	...	PARAMETER N	CHECK SUM

  El controlador puede enviar siete ordenes distintas, en función del valor del campo INSTRUCTION del paquete. La Tabla 3.2 enumera los siete posibles valores de este campo.

Valor	Nombre	Función
0x01	PING	Envío de Status Packet desde el servomotor.
0x02	READ_DATA	Lectura de datos desde el servomotor.
0x03	WRITE_DATA	Escritura de datos hacia el servomotor.
0x04	REG WRITE	Escribe en el servomotor sin ejecutar la acción.
0x05	ACTION	Ejecuta la escritura de valores registrados.
0x06	RESET	Reinicia valores de fabrica de los parámetros del servomotor.
0x83	SYNC WRITE	Sirve para controlar diferentes servomotores simultáneamente.

Tabla 3.2: Comandos aceptados por el servomotor.

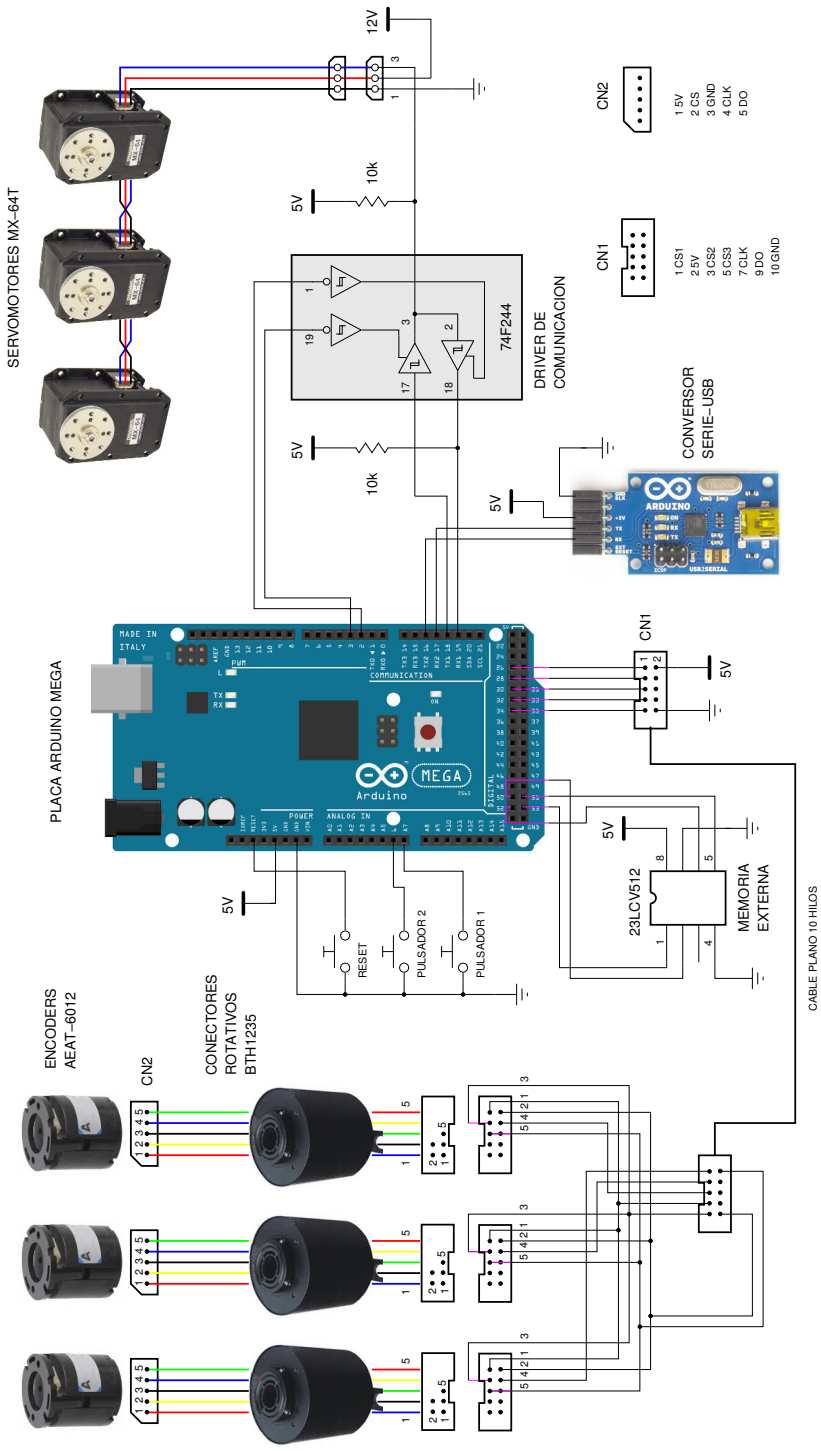


Figura 3.7: Esquema de conexionado de los elementos integrantes.

- ***Status packet***. Paquete de respuesta del servomotor al dispositivo maestro. La estructura de un *Status Packet* es:

0xFF	0xFF	ID	LENGTH	ERROR	PARAMETER 1	...	PARAMETER N	CHECK SUM

  En función del tipo de instrucción enviada por el controlador, variará la información de respuesta escrita en los campos de parámetros del *Status Packet*[3].

Para que la placa Arduino pueda controlar los servomotores, se han programado una serie de funciones que permiten el intercambio de datos, haciendo uso de los dos formatos de trama definidos anteriormente. Estas funciones se han escrito en lenguaje Arduino y se ejecutarán en esta placa. A continuación se explican estas funciones de forma resumida.

En primer lugar se han implementado estas funciones de bajo nivel:

- *instructionPacket*: Genera y envía un paquete *Instruction packet* al servomotor.
- *statusPacket*: Recibe un paquete *Status packet* devuelto por el servomotor como respuesta a un paquete *Status packet*.
- *instructionStatus*: Envía un *Instruction packet* y recibe su correspondiente *Status packet*.

Haciendo uso de las funciones de bajo nivel, se han implementado otras funciones para facilitar el envío de comandos aceptados por el servomotor.

- *mxPing*: Envía una instrucción PING al servomotor especificado y recibe su respuesta.
- *mxReadDataByte*: Lee un byte de la memoria (Tabla de control) del servomotor.
- *mxReadDataWord*: Lee una palabra (dos bytes) de la memoria del servomotor.
- *mxWriteDataByte*: Escribe un byte de la memoria del servomotor.
- *mxWriteDataWord*: Escribe una palabra de la memoria del servomotor.
- *mxRegWriteSpeed*: Registra una consigna de velocidad en la memoria del servomotor sin ejecutarla hasta que se reciba la instrucción ACTION. Esta función la utilizaremos al ejecutar trayectorias con el fin de poder escribir las consignas de velocidad en los tres servomotores a la vez.
- *mxAction*: Envía una instrucción ACTION a los servomotores.
- *initializeControlTable*: Inicializa la Tabla de control del servomotor a un estado definido.

[3]En la página http://support.robotis.com/en/product/dynamixel/communication/dxl_packet.htm se encuentra una descripción detallada de cada uno de los campos de ambos paquetes.

3.4.2. Comunicación Arduino - Matlab

La interfaz gráfica de usuario, programada en Matlab, necesita interactuar con los servomotores para poder ejecutar las trayectorias planificadas o para leer la configuración actual del robot. Estas tareas implican la adquisición o envío de información de forma cíclica con unos tiempos de ciclo, en algunos casos, del orden de decenas de milisegundos, y dependiendo de la exactitud con la que se mida este tiempo, el resultado será mejor o peor. Por esta razón se decidió utilizar la placa Arduino para llevar la comunicación más directa con los servomotores, pudiendo su microcontrolador alcanzar resoluciones del orden de microsegundos en vez de milisegundos, que es la que obtendríamos si el control de los servomotores se hiciera directamente desde Matlab. Resuelta la comunicación servomotores - Arduino, en la que Arduino actuaba de dispositivo maestro y el servomotor de esclavo, ahora Matlab será el que comande a Arduino, y este responderá realizando la acción que le ha sido solicitada. Para que ambos se entiendan, se ha definido un protocolo muy parecido al utilizado por el servomotor. En él se definen dos tipos de paquetes:

- ***Matlab Command Packet***. Paquete enviado por Matlab para solicitar una acción a Arduino. Su estructura es:

0xFF	LENGTH	INSTRUCTION	PARAMETER 1	...	PARAMETER N	CHECK SUM

  El campo INSTRUCTION puede contener los valores enumerados en la Tabla 3.3.

Valor	Función
0x02	Envía la posición de los tres servomotores y encoders AVAGO.
0x03	Recibe una Tabla de trayectoria y ejecuta dicha trayectoria.
0x04	Envía un paquete *Instruction packet* directamente desde Matlab a un servomotor.
0x05	Envía un paquete *Instruction packet* directamente desde Matlab a un servomotor y devuelve el paquete de respuesta *Status packet* a Matlab.
0x06	Escribe una palabra (dos bytes) en la memoria externa SRAM.
0x07	Lee una palabra desde la memoria externa SRAM.
0x08	Envía la posición de los tres servomotores, de los encoders AVAGO y de la variable de la Tabla de control CURRENT LOAD.

Tabla 3.3: Comandos aceptados por Arduino desde Matlab.

- ***Matlab Answer Packet***. Paquete de respuesta de Arduino a la petición desde Matlab. Su estructura es:

0xFF	LENGTH	ERROR	PARAMETER 1	...	PARAMETER N	CHECK SUM

Esta vez, el protocolo no ha venido fijado por uno de los dos dispositivos que se han de comunicar. Por ello, para hacer que tanto Arduino como Matlab lo utilicen, habrá que programar ambos dispositivos. Veamos las funciones implementadas en cada uno de ellos.

Las funciones implementadas en Arduino son:

- *rcvMatlabCommand*: Recibe un paquete *Matlab Command Packet* y lo guarda en la matriz MatlabPacket con la estructura:

  MatlabPacket[] = {length, instruction, parameter 1, ..., parameter N, checkSumReceived}

- *sendMatlabAnswer*: Envía un paquete *Matlab Answer Packet* desde la matriz MatlabAnswer, con la estructura:

  MatlabAnswer[] = {head, length, error, parameter 1, ..., parameter N, checkSum}

- *matlabInstrPacket*: Permite llamar directamente a la función de bajo nivel *instructionPacket* de la comunicación servomotor - Arduino. Con esta función Matlab puede enviar una trama *Instruction packet* hacia el servomotor.

- *matlabInstructionStatus*: Permite llamar directamente a la función de bajo nivel *instructionStatus* de la comunicación servomotor - Arduino. Con esta función Matlab puede enviar una trama *Instruction packet* hacia el servomotor y recibir la trama de respuesta generada.

Las funciones implementadas en Matlab son:

- *commandtoarduino*: Envía un paquete *Matlab Command Packet* hacia Arduino, para solicitar una de las acciones descritas en la Tabla 3.3.

- *answerfromarduino*: Recibe un paquete *Matlab Answer Packet* desde Arduino, como respuesta a una acción previa solicitada por Matlab.

3.5 La interfaz de usuario

La inferfaz gráfica de usuario es el entorno gráfico que facilita el acceso a las capacidades principales del sistema diseñado: la definición de las configuraciones inicial y final, la planificación, ejecución y visualización de trayectorias, etc. En su concepción, además de servir como base de experimentación de las distintas partes del proyecto, se ha intentado que fuera una herramienta didáctica que facilitara la comprensión del concepto de singularidad en manipuladores paralelos. Para este uso no es necesario contar con el robot ya que en modo simulación se puede interactuar con el manipulador así como planificar y simular la ejecución de trayectorias.

La interfaz desarrollada (Fig. 3.8) tiene dos modos de funcionamiento diferenciados según si hay comunicación con el robot físico o no. En modo de funcionamiento “sin conexión” podemos:

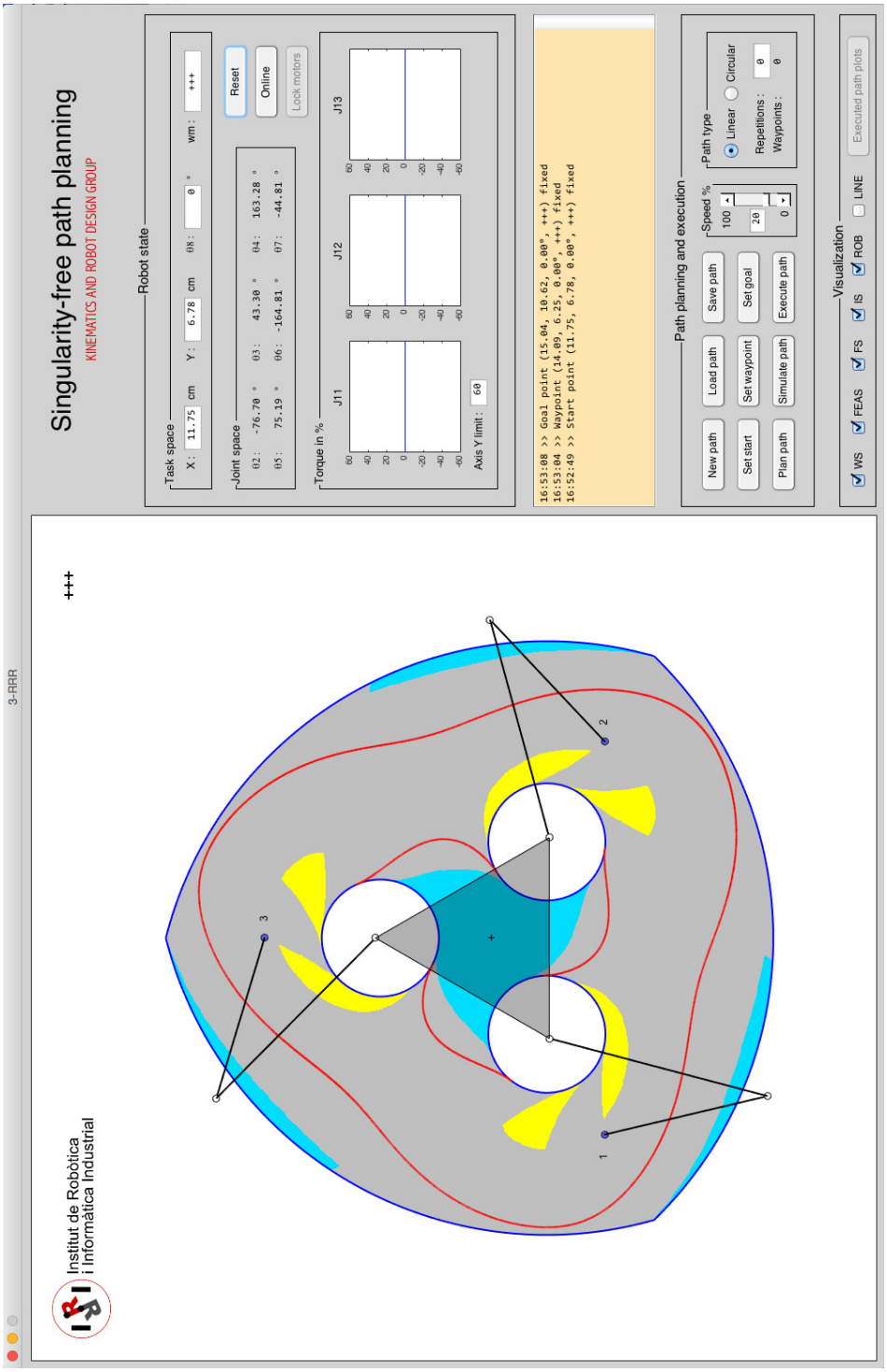


Figura 3.8: Interfaz de usuario implementada.

- Visualizar, habilitando las siguientes capas como deseemos,
  1. La configuración del robot.
  2. El espacio de trabajo sin considerar colisiones.
  3. El espacio de trabajo útil, considerando colisiones y zonas próximas a singularidades. Este espacio lo representamos en la interfaz con zonas de color azul y amarillo, que se corresponden a las configuraciones en las que $\det(\mathbf{A}) = 0$ (Secc. 2.3.2) tiene signo negativo o positivo, respectivamente.
  4. Las singularidades directas.
  5. Las singularidades inversas.
  6. Las rectas soporte de los tres eslabones distales.
- Cambiar el modo de trabajo de cada una de las patas, mediante las teclas “1”, “2” y “3”.
- Arrastrar el efector final, manteniendo pulsado el botón izquierdo del ratón, a cualquier punto del espacio de trabajo.
- Modificar la posición del triángulo móvil con las teclas de dirección, y su orientación mediante las teclas “+” y “-”.
- Modificar la configuración del robot editando las coordenadas del efector en el espacio de trabajo.
- Planificar caminos libres de singularidades directas entre una configuración de inicio y otra final, pasando por los puntos intermedios que definamos, siempre que ese camino exista.
- Guardar o cargar una trayectoria desde fichero.
- Simular la ejecución de una trayectoria previamente cargada o calculada.

En el modo de funcionamiento “con conexión” podemos:

- Visualizar en tiempo real, combinando las capas que deseemos:
  1. La configuración del robot.
  2. El espacio de trabajo sin considerar colisiones.
  3. El espacio de trabajo útil, considerando colisiones y zonas próximas a singularidades.
  4. Las singularidades directas.
  5. Las singularidades inversas.

6. Las rectas soporte de los tres eslabones distales.

- Mover cada una de las articulaciones actuadas del robot en ambos sentidos mediante las teclas:

Articulación	Giro derecha	Giro izquierda
1	`q`	`a`
2	`w`	`s`
3	`e`	`d`

- Mover el efector final. Con las flechas de dirección del teclado podremos cambiar su posición, mientras que pulsando las teclas "+" y "-", cambiaremos su orientación.
- Bloquear las articulaciones actuadas evitando que se muevan.
- Ver el par ejercido por cada actuador, expresado en porcentaje sobre el par máximo que puede efectuar.
- Planificar caminos libres de singularidades directas entre una configuración de inicio y otra final pasando por los puntos intermedios que definamos, siempre que ese camino exista.
- Guardar o cargar una trayectoria.
- Ejecutar o simular una trayectoria previamente cargada o calculada.
- Modificar la velocidad a la que se ejecuta la trayectoria, el número de veces que se repite la ejecución de la trayectoria y si la trayectoria es circular o no.
- Visualizar la configuración del robot y los puntos por los que va pasando mientras se ejecuta una trayectoria.
- Visualizar los gráficos en los que se muestran los datos adquiridos de los servomotores durante la ejecución de la trayectoria. (Fig. 3.9)

3.6 Control de movimientos

3.6.1. Estrategia de control

Para conseguir que el efector final siga una trayectoria dentro de su espacio de trabajo, cada articulación actuada del robot debe controlarse de modo que siga con el mínimo error posible

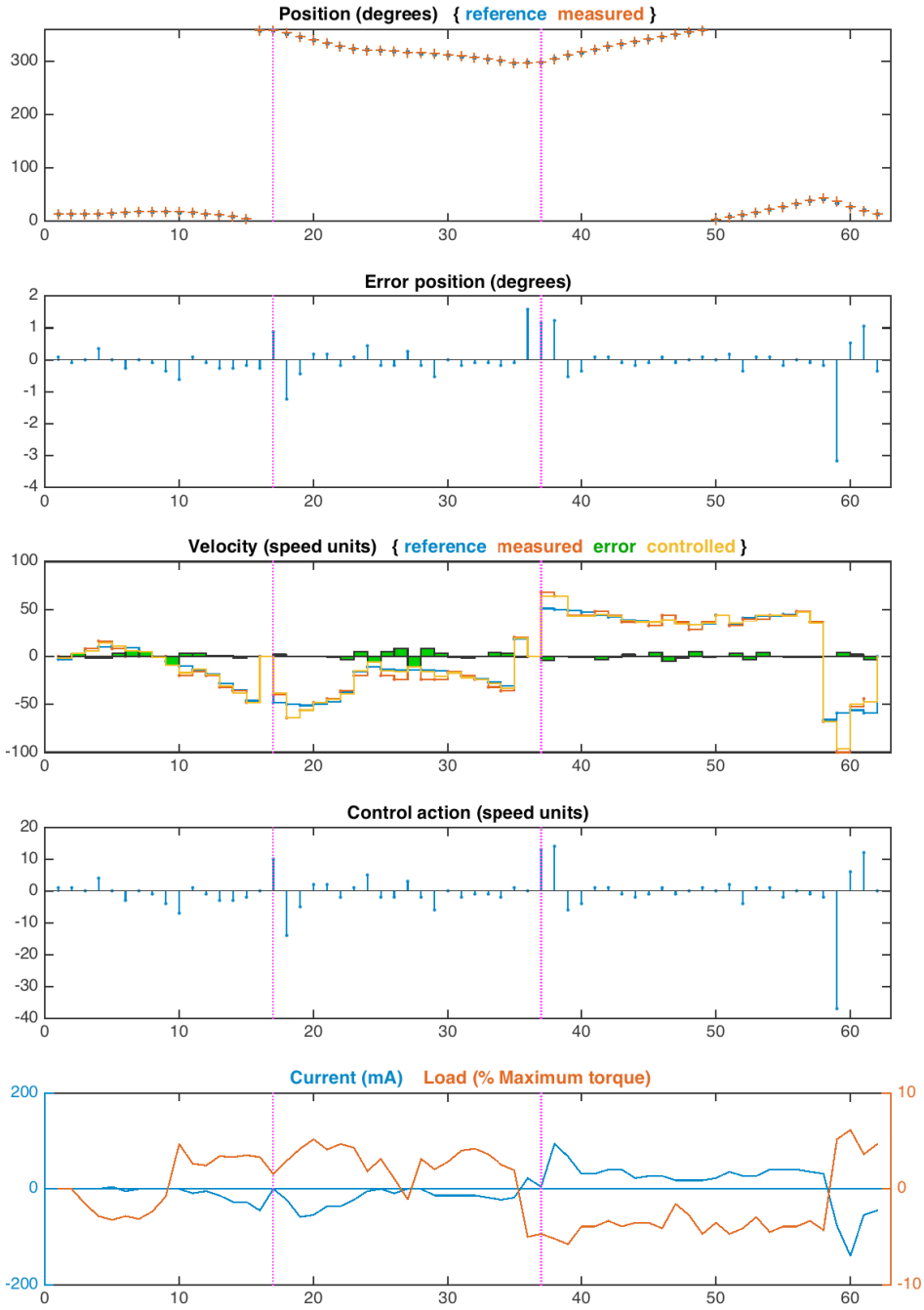


Figura 3.9: Ejemplo de gráficas con datos adquiridos, de un servomotor, durante la ejecución de una trayectoria.

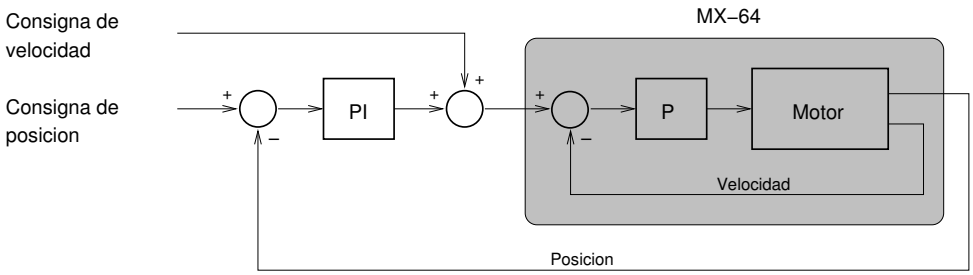


Figura 3.10: Esquema de bloques del controlador de un actuador.

su correspondiente trayectoria en el espacio articular. Se ha utilizado una estrategia de control articular independiente, en la que la trayectoria de cada servomotor se controla de manera individual, sin considerar la trayectoria seguida por las restantes articulaciones [3, Sección 9.4.2]. De este modo tendremos tres esquemas de control iguales, uno por cada articulación actuada (Fig. 3.10). Cada controlador estará formado por dos lazos de control anidados. El lazo externo seguirá la consigna de posición, y actuará ajustando el valor de la consigna de velocidad previamente calculada. La consigna de velocidad resultante será la entrada del lazo interno de control de velocidad.

Como el robot se moverá lentamente, las aceleraciones a que estará sometido serán pequeñas, y por tanto, con respecto al par resistente de inercia, se podrá suponer que es despreciable. Además, como el robot es ligero, el par resistente debido al efecto de la gravedad será también pequeño, y para nuestro propósito, no lo tendremos en consideración. En [3] se explica cómo tratar estas perturbaciones de par en caso de que deban tenerse en cuenta.

3.6.2. Ajuste del controlador interno

Cada servomotor lleva integrado un controlador PID que podemos ajustar mediante los registros “P Gain”, “I Gain” y “D Gain” de su tabla de control. La relación entre estos parámetros y las constantes K_p, K_i y K_d del PID es la siguiente:

$$K_p = P_{GAIN}/8,$$
$$K_i = (I_{GAIN} \cdot 1000)/2048,$$
$$K_d = (D_{GAIN} \cdot 4)/1000.$$

Su ajuste se ha realizado de forma heurística, entrando una señal de consignas de velocidad al servomotor y graficando su salida para diferentes valores de P Gain, I Gain y D Gain. (Fig. 3.11). En base a los resultados de estos experimentos y teniendo en cuenta que el robot se moverá a

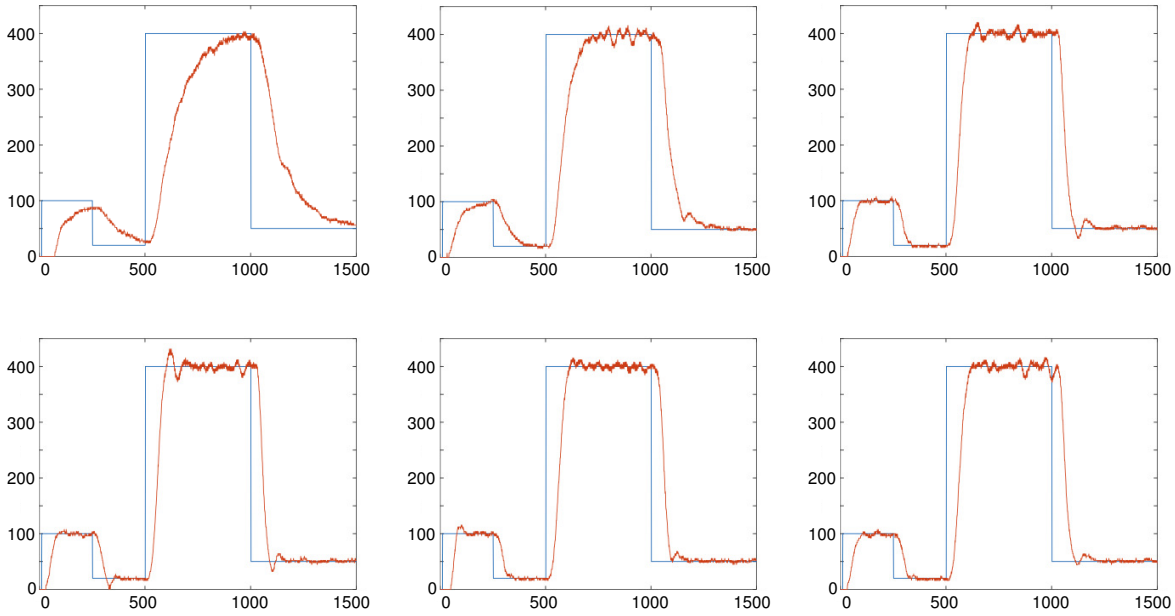


Figura 3.11: Respuesta del servomotor a diferentes parámetros en su PID interno. En color azul se representa la consigna de velocidad y en rojo la lectura de la velocidad del motor. En el eje de abscisas tenemos el número de muestra (el periodo de muestreo es de 1 ms). El eje de ordenadas está medido en unidades de velocidad del MX-64 (1 unidad de velocidad son 1.1245 rpm). De arriba a abajo, izquierda a derecha, representados con el vector (P Gain, I Gain, D Gain), (8, 0, 0), (16, 0, 0), (32, 0, 0), (40, 0, 0), (32, 1, 0), (32, 0, 1). El experimento se ha realizado sin el efector final acoplado y con el eslabón distal colocado sobre el proximal.

baja velocidad, se ha configurado su PID interno para trabajar como controlador proporcional, con $K_p = 4$ ($P_{GAIN} = 32$). Con este valor se obtiene un tiempo de subida reducido manteniendo el sistema alejado de la inestabilidad y evitando en lo posible los sobrepicos. Por otro lado, este es el valor configurado por defecto en el servomotor.

3.6.3. Diseño del controlador externo

Con vistas al diseño del controlador externo de posición, es deseable disponer de un modelo dinámico del controlador interno. El método utilizado para su obtención ha sido la identificación del sistema mediante datos experimentales. El elemento principal del sistema a identificar será el servomotor, con su PID configurado para trabajar en modo control proporcional, con $K_p = 4$, tal como se determinó en el punto anterior. De esta manera, el sistema de control de velocidad en lazo cerrado es esencialmente de primer orden, simplificándose así el modelo que se obtendrá mediante la identificación. El primer paso es obtener datos experimentales del sistema. Para

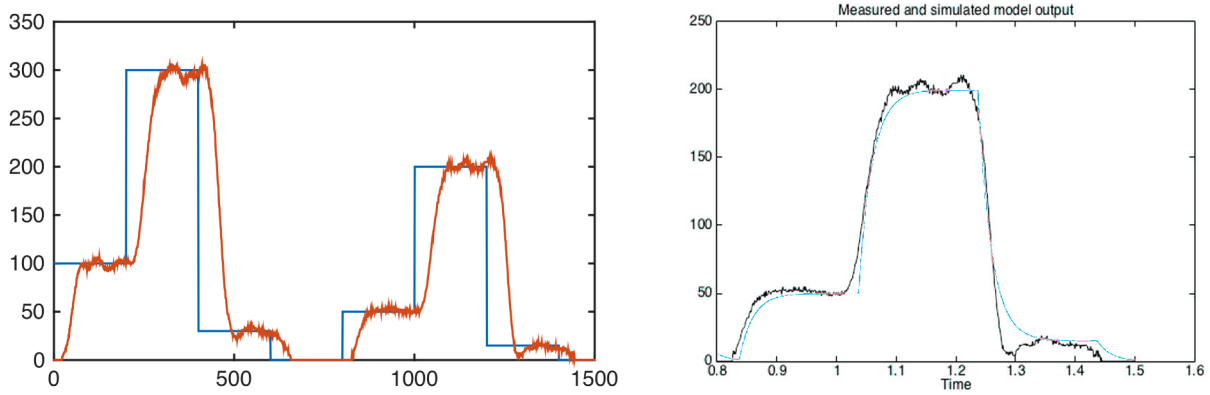


Figura 3.12: A la izquierda se grafican los datos experimentales: la primera mitad son utilizados para la identicación y la segunda para la validación del modelo. A la derecha se representan los datos de salida del modelo superpuestos a los experimentales.

ello lo excitamos con una señal de consignas de velocidad y registramos su respuesta. La señal de entrada se ha seleccionado de forma que la salida sea suficientemente informativa sobre la dinámica del sistema (Fig. 3.12). El servomotor actúa sobre el motor mediante una señal PWM (*Pulse-width modulation*) con una frecuencia medida de 23,4 kHz, muy superior a la frecuencia del lazo externo de posición considerada, de unos 6.3 Hz, por lo que identificaremos un modelo en tiempo continuo del controlador interno. Con estos datos experimentales, utilizamos la aplicación de identificación de sistemas de la *System Identification Toolbox* de Matlab, obteniendo la siguiente función de transferencia de primer orden con retardo como modelo del sistema:

$$TF_{sistema} = \frac{1.0009}{0.0271 \cdot s + 1} \cdot e^{-0.037 \cdot s}. \tag{3.1}$$

En la fase de validación del modelo, se utilizan parte de los datos experimentales no considerados en la identificación para ver la concordancia entre la salida del modelo y la salida real del sistema. En nuestro caso, la aplicación de identificación nos devuelve una concordancia del 87.77 % (Fig. 3.12).

Para determinar los parámetros del PID de posición se ha implementado una simulación en Simulink (Fig. 3.13). Como se aprecia en la figura, el controlador de posición es un PID con prealimentación de la consigna de velocidad. El control anticipativo permite obtener errores de posición bajos con valores también bajos de la ganancia proporcional del PID. Haciendo pruebas con el simulador, se ha visto que fijando $K_p = 1$ y $K_i = 0.4$, se obtienen errores de posición angular de máximos de ± 15 grados en cada articulación (Fig. 3.14), mientras que estos errores se reducen a ± 2 grados cuando se activa la prealimentación (Fig. 3.15). Con los

valores de K_p y K_i mencionados, y la prealimentación de velocidad activada, la trayectoria se sigue notablemente bien, tanto en simulación como experimentalmente. Por tanto, se han fijado estos parámetros en la implementación real.

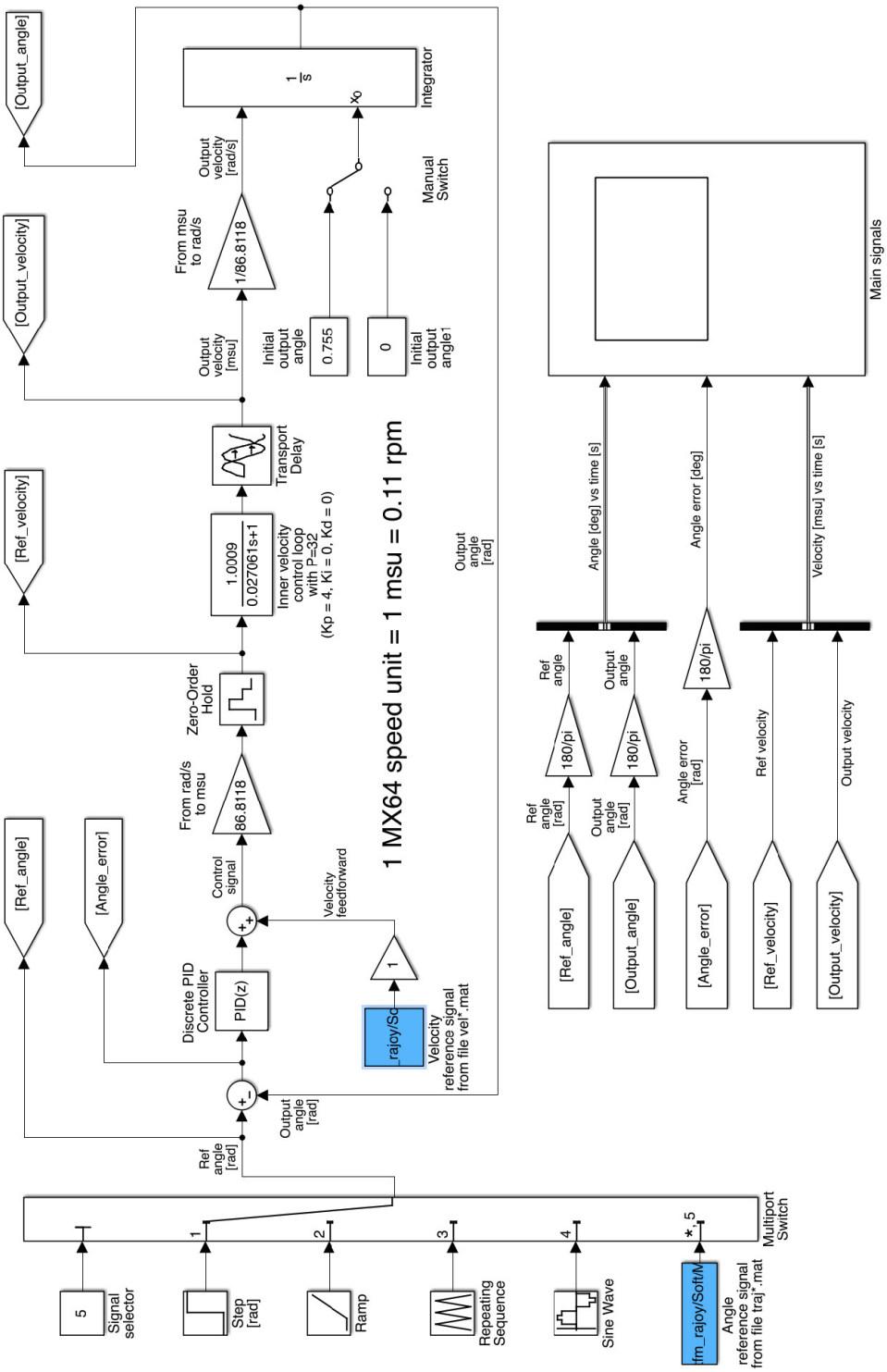


Figura 3.13: Modelo Simulink del sistema de control de una articulación.

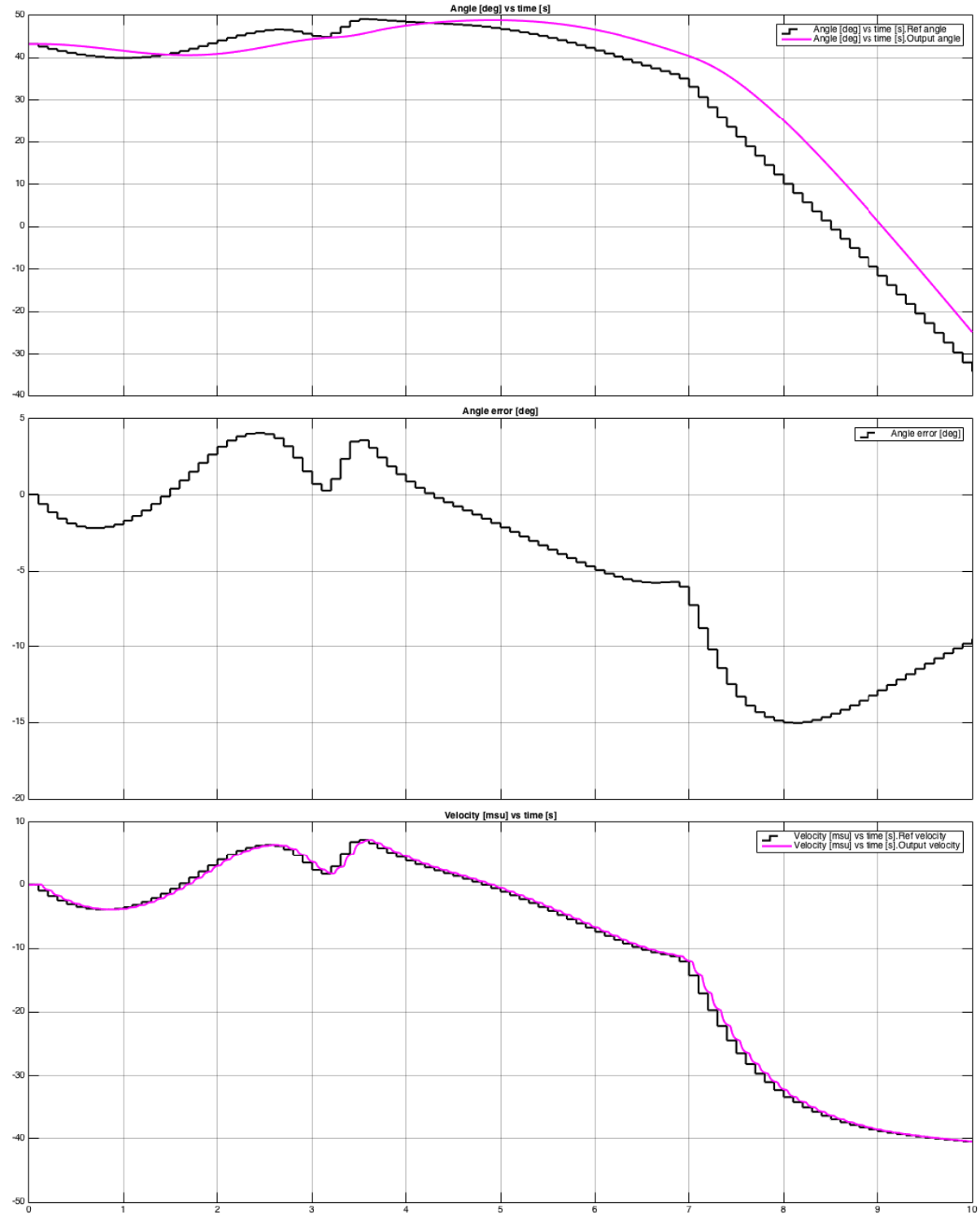


Figura 3.14: Resultados de simulación sin prealimentación de velocidad.

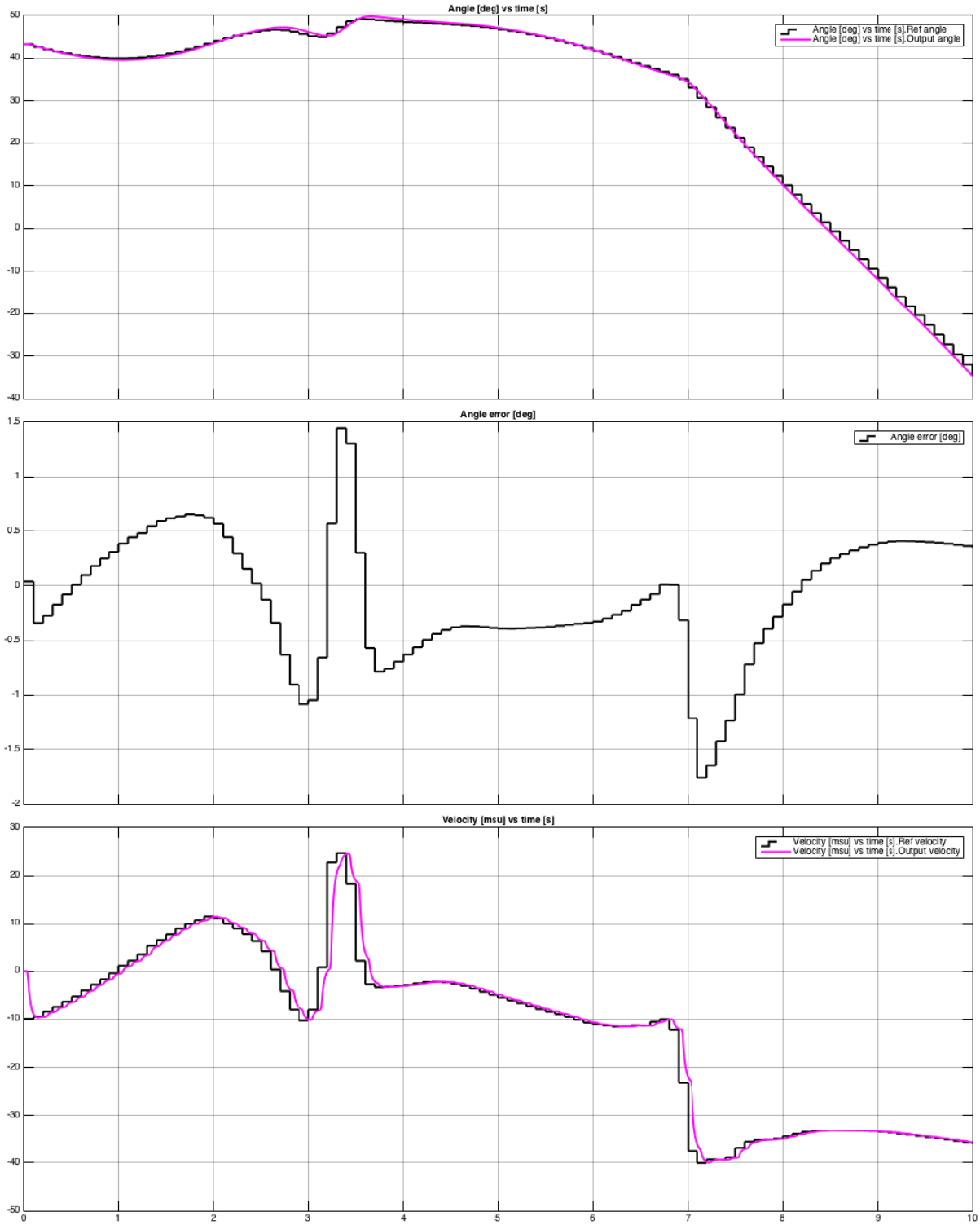


Figura 3.15: Resultados de simulación con prealimentación de velocidad.

4

Experimentos

En este capítulo examinaremos experimentalmente el sistema implementado para la planificación y ejecución de trayectorias libres de singularidades. Para ello partiremos de una configuración de inicio, una intermedia y una final, e iremos viendo el resultado de cada uno de los pasos seguidos hasta conseguir que el robot siga una trayectoria libre de singularidades y colisiones pasando por los puntos definidos. Finalizaremos viendo la influencia del lazo externo de control de posición en el resultado de la ejecución.

4.1 Generación de trayectorias

Empezaremos el experimento definiendo en la interfaz gráfica los puntos por los que queremos que pase el robot. A modo de ejemplo, utilizaremos los indicados en la siguiente tabla:

Configuración inicial	Configuración de paso	Configuración final	Archivo de trayectoria
(11.75, 6.78, 0, "+++")	(15.93, -0.74, 320.86, "+−+")	(2.15, 7.51, 319.14, "− −+")	`\Experimentos\path.mat`

En la 4-tupla que define cada configuración, los valores primero y segundo son las coordenadas (x, y) del baricentro del triángulo móvil (en centímetros), el tercer valor es su orientación, θ_8 (en grados), y el cuarto valor es el modo de trabajo del robot.

Seleccionamos en la interfaz la opción *Path type: circular*, de modo que en el último tramo de la trayectoria el robot regrese desde la configuración final a la inicial.

Al pulsar el botón *Plan path*, por cada una de las configuraciones especificadas se crea su correspondiente punto en la variedad de navegación $\mathcal{M}$ (Secc. 2.4.2). Estos puntos se almacenan mediante tuplas de 67 elementos, que se corresponden con las 67 variables del fichero *scf.cuik* (Apéndice B).

Cada tramo del camino se calcula de manera independiente, enviando al planificador los vectores de inicio y fin de tramo. En nuestro experimento tendremos tres tramos, que van de

la configuración de inicio a la de paso, de la de paso a la final y, al haber especificado una trayectoria circular, de la final a la de inicio. En la Fig. 4.1 se representa, en color azul, el camino total devuelto por el planificador tras las tres llamadas.

Como podemos ver, el camino obtenido presenta zonas de zigzagueo que evitaremos aplicándole un filtro de suavizado a cada tramo. Para ello nos serviremos de la aplicación *cuiksmoothpath*, incluida en el paquete *Cuik*, que trabaja directamente sobre la variedad de navegación $\mathcal{M}$, asegurándonos así que este filtrado no hará que el robot atraviese singularidades directas o zonas de auto-colisión. En la Fig. 4.1 vemos representado en rojo el camino filtrado.

El siguiente paso es equiespaciar los puntos del camino en el espacio articular, tal y como se

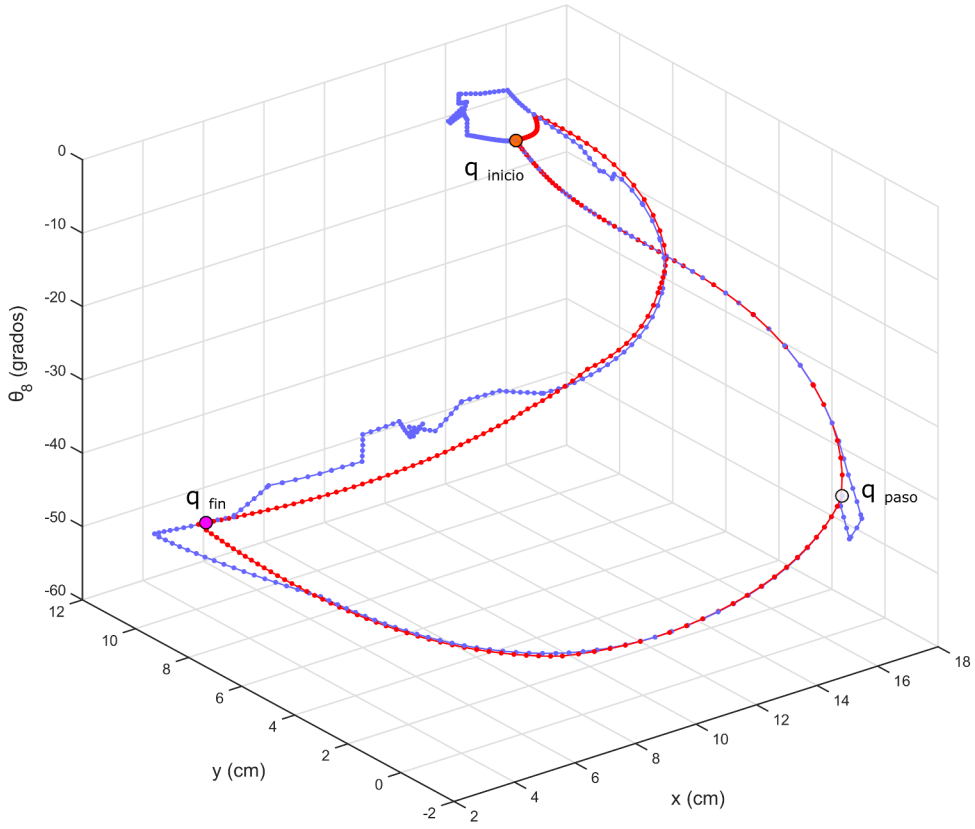


Figura 4.1: Camino utilizado en el experimento, representado en el espacio de trabajo. En color azul se grafica el camino circular encontrado por el planificador pasando por los tres puntos definidos. En color rojo se muestra el mismo camino tras aplicarle el filtro de suavizado.

ha explicado en la Sección 2.5. Para ello se ajusta una *spline* al camino y se muestrea, obteniendo ya los puntos que se enviarán como consigna de posición a cada servomotor (Fig. 4.2).

A partir de estos puntos, se genera la tabla de consignas, que incluye consignas de posición, velocidad, y tiempo entre muestras, para cada uno de los puntos. (Fig. 4.3). Esta tabla, una vez transformada a las unidades de velocidad y posición aceptadas por el servomotor, será la que se envíe a la placa Arduino.

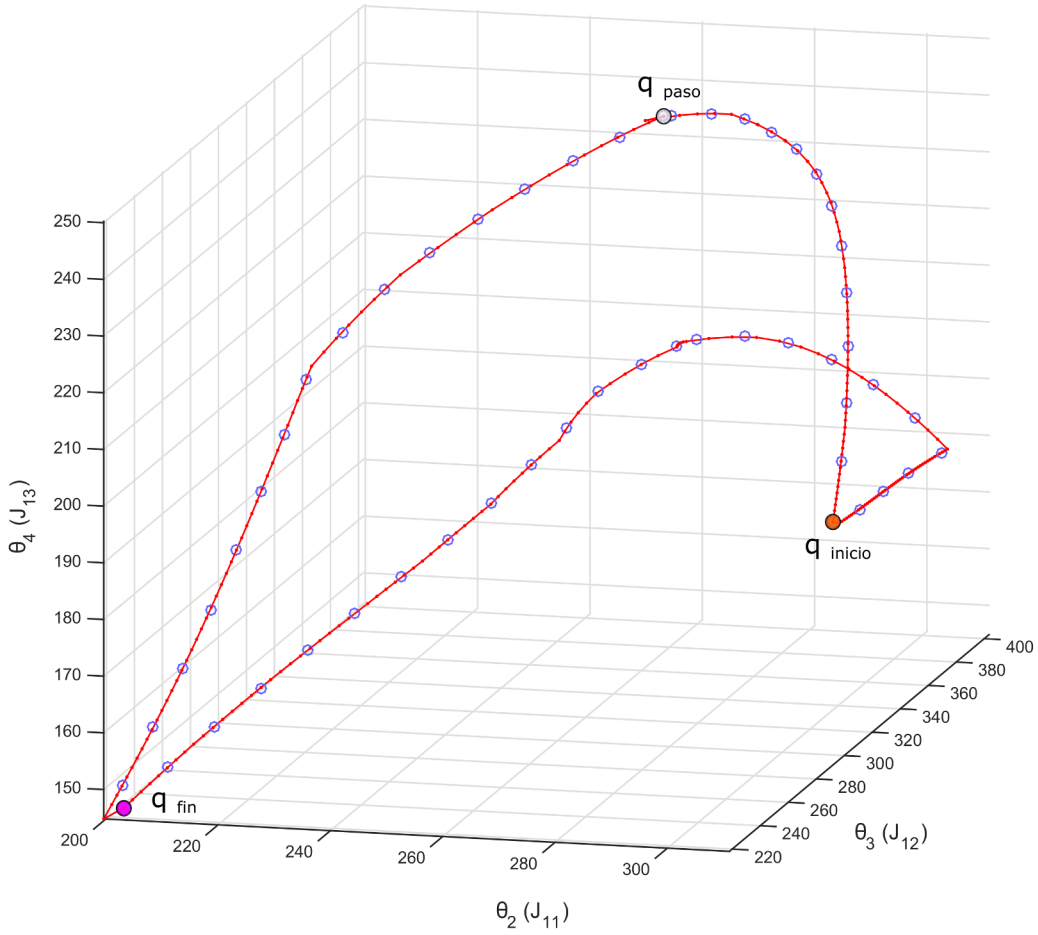


Figura 4.2: Camino utilizado en el experimento, representado en el espacio articular. En rojo se muestra la *spline* ajustada al camino devuelto por el filtro de suavizado y en azul las muestras equiespaciadas. Los ángulos θ_2, θ_3 y θ_4 se especifican en grados.

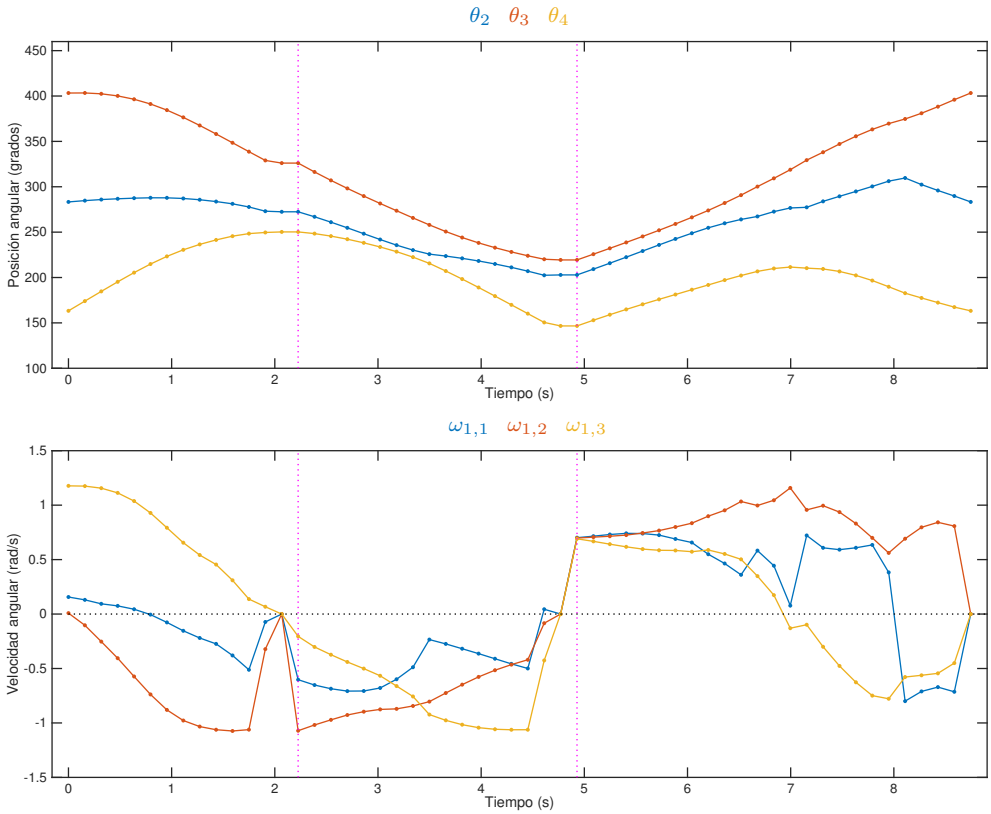


Figura 4.3: Posición y velocidad angular de cada punto de la trayectoria experimentada. Las señales representadas son discretas pero se han graficado mediante lineas poligonales para facilitar su interpretación.

4.2 Ejecución de trayectorias

Para ejecutar cada tramo de la trayectoria, le enviamos a Arduino un paquete de datos *Matlab Command Packet* con la instrucción 3 y el número de puntos de la trayectoria (Secc. 3.4.2). Arduino reacciona esperando la recepción de la tabla de consignas. Una vez recibida, Arduino empieza a enviar, a los servomotores, las consignas de velocidad, modificadas por el lazo externo de control de posición. Antes de enviar cada punto, lee el estado de los servomotores y representa, en la interfaz gráfica, la posición del baricentro del efector final. En la Fig. 4.4 tenemos diversas capturas de la secuencia seguida por el robot en la ejecución de toda la trayectoria. Como podemos apreciar, el baricentro del triángulo móvil siempre permanece dentro de la zona azul, que es una de las dos zonas libres de singularidades directas y colisiones. En verde se representa el camino planificado, y los puntos negros marcan la posición por la que va pasando el baricentro del efector final justo antes de que los servomotores reciban una nueva consigna.

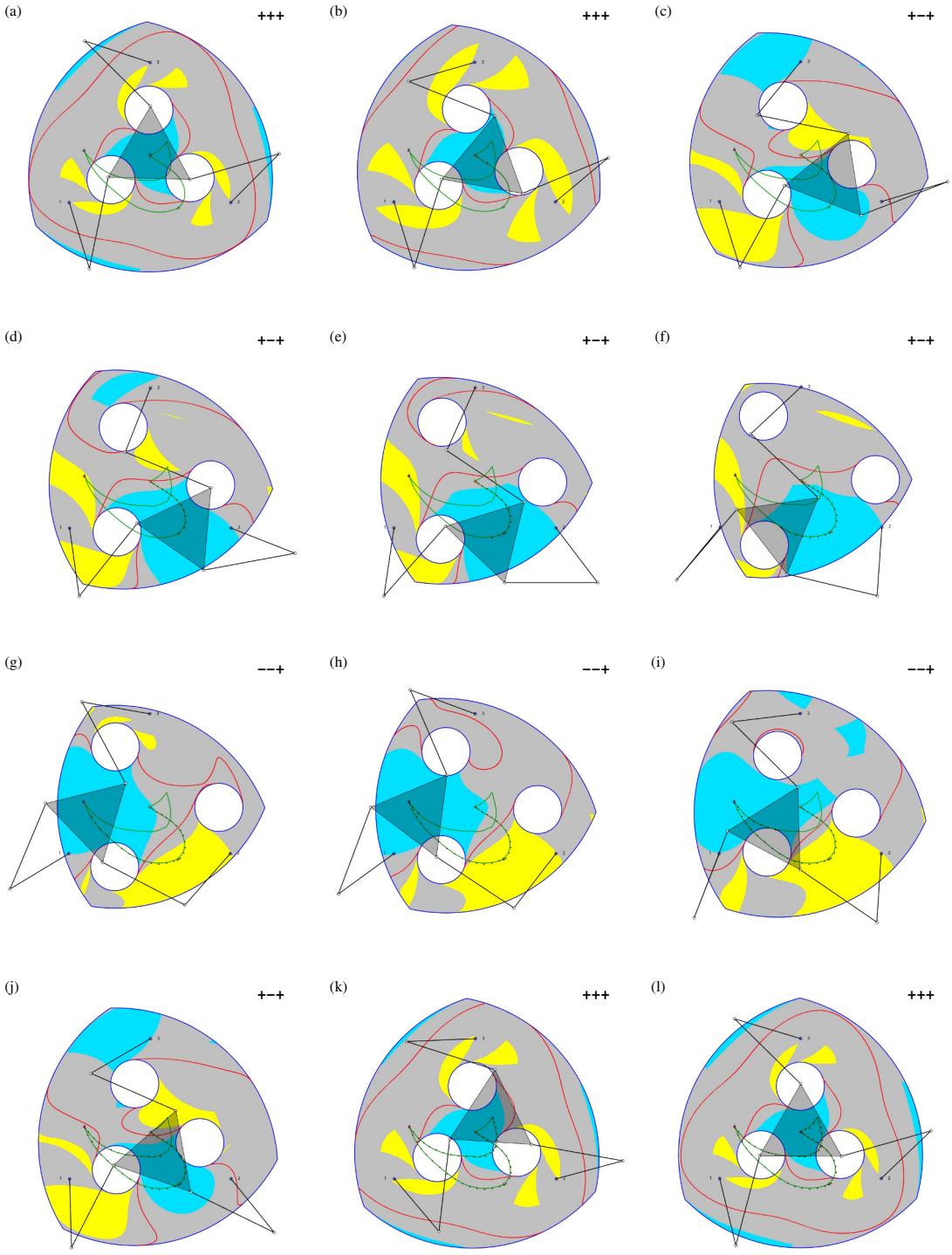


Figura 4.4: Capturas de la secuencia seguida en la ejecución de la trayectoria.

Como podemos ver, el robot sigue notablemente bien la trayectoria planificada. Encontramos cambios del modo de trabajo (indicado en la parte superior derecha de cada imagen), entre la imagen (b) y (c) de la pata 2, entre la (f) y (g), y (i) y (j) de la pata 1, y entre la (j) y (k) de la pata 2.

Para poder apreciar el efecto del lazo externo de control de posición, se ha realizado el experimento dos veces, una sólo con el lazo de control de velocidad activado y otra con los dos lazos de control activados, el interno de velocidad y el externo de posición. En la Fig. 4.5 podemos ver los puntos de paso del robot (en negro) sobrepuestos a la trayectoria planificada (en verde).

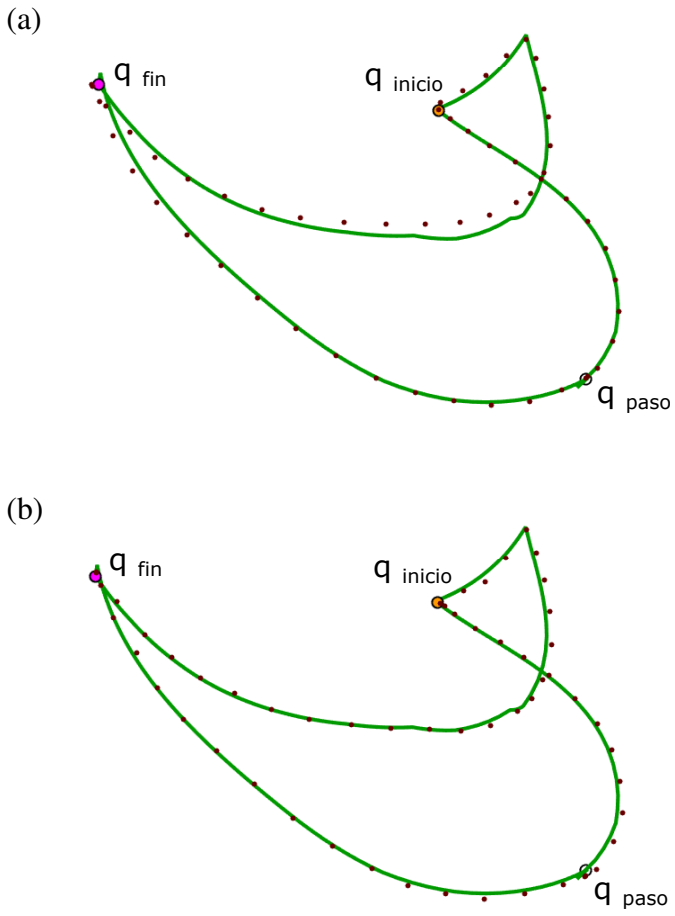


Figura 4.5: Camino seguido por el baricentro P del efector final durante la ejecución de la trayectoria. En (a) sólo está habilitado el lazo interno de control de posición. En (b) ambos lazos de control están habilitados. En (a) se observa como el robot incurre en un mayor error de posición con respecto a (b).

En (a), solo se esta controlando velocidad. Se aprecia que sigue la trayectoria bastante bien pero no corrige errores de posición, por lo que, en presencia de perturbaciones, el robot puede llegar a salir de la zona libre de singularidades y colisiones. En (b), con los dos lazos de control habilitados, vemos como el robot corrige de forma adecuada los errores de posición, desviándose menos del camino a seguir.

Como se ha comentado, antes de enviar cada punto de consigna, se hace una lectura del estado de los servomotores, lo que permite la realimentación de estos datos al controlador, saber la configuración actual del robot, y analizar la ejecución de la trayectoria. Estos datos leídos se almacenan primero en la memoria externa para, más tarde, cuando ha finalizado la ejecución, enviarlos a Matlab para su posterior tratamiento. En la Fig. 4.6 se muestran los datos de mayor interés capturados en la ejecución sin control de posición. Cada inicio de un nuevo tramo de la trayectoria está marcado por una línea vertical punteada de color magenta. En (a) tenemos los datos recogidos del servomotor de la pata 1 en función del número de muestra, en (b) de la 2, y en (c) de la tres. Notemos como no hay acción de control y solo actúa la consigna de velocidad. El efecto de ello es que la señal de error de posición no es corregida. En la Fig. 4.7 tenemos la realización en la que están activos los dos lazos de control. En este caso, vemos como Arduino aplica una acción de control a la consigna de velocidad en función del error de posición, consiguiendo así reducir sensiblemente estas desviaciones. La señal titulada “speed controlled” es la consigna de velocidad que se le manda al servomotor tras ser modificada por la acción de control (salida del controlador PI de posición), por lo que la velocidad controlada será igual a la referencia de velocidad más la acción de control.

Cabe notar que la última muestra de cada tramo tiene consigna de velocidad cero, para que el servomotor se pare. Esto hace que en esta última muestra no se aplique acción de control, por lo que el error en posición en la última muestra de tramo y en la primera del siguiente no esta controlado. Lo apreciamos en las gráficas de cada pata en forma de un incremento de error de posición en estos puntos.

Todos los datos relativos a este experimento se encuentran disponibles en el CD adjunto, directorio `\Experimentos`.

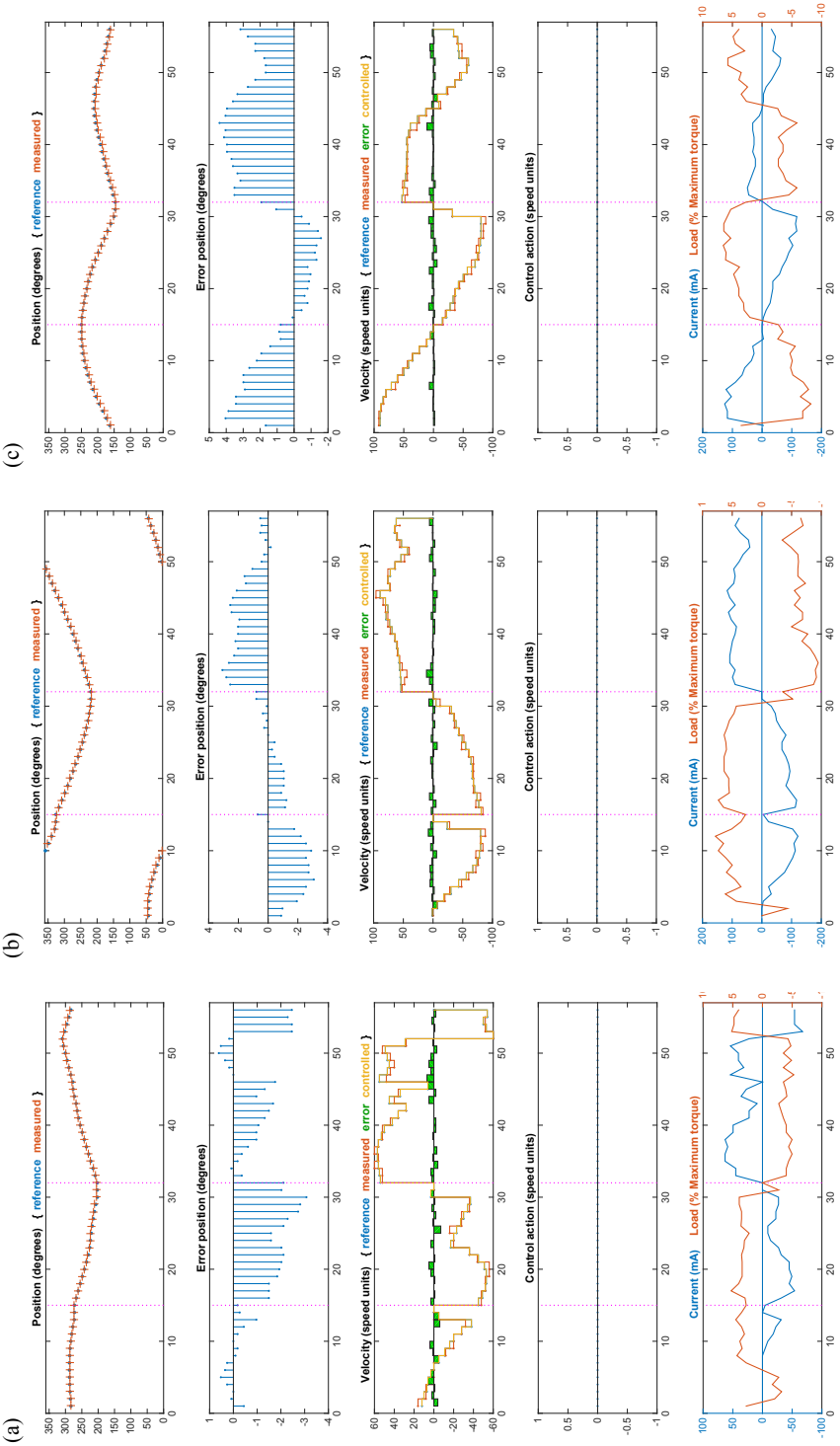


Figura 4.6: Gráfica con datos adquiridos durante la ejecución de la trayectoria sin el lazo externo de control de posición habilitado.

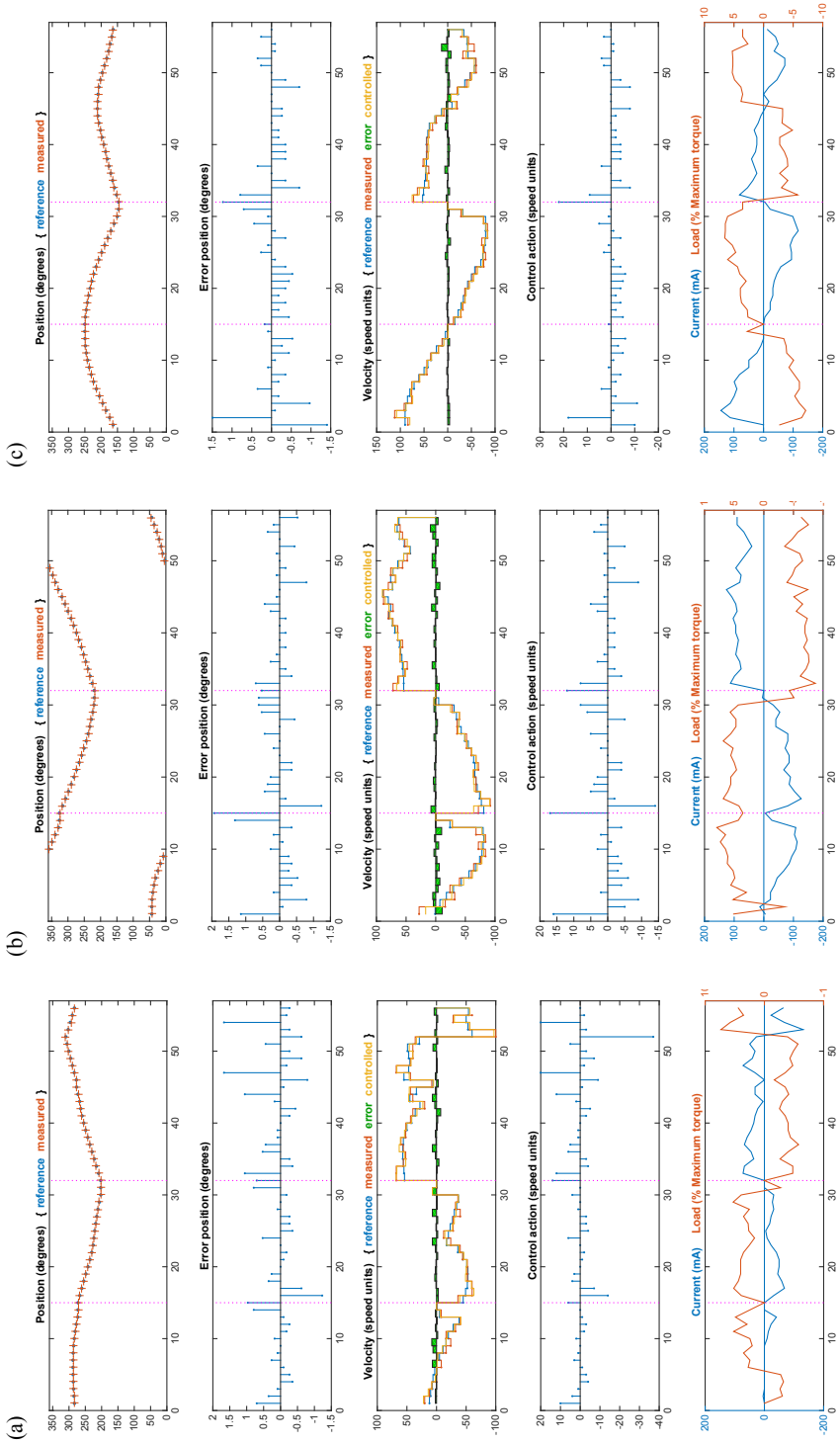


Figura 4.7: Gráfica con datos adquiridos durante la ejecución de la trayectoria con el lazo externo de control de posición habilitado.

5 Conclusiones

5.1 Contribuciones

En este proyecto se ha construido un robot 3-RRR que permite evaluar experimentalmente el método de planificación de movimientos libres de singularidades propuesto en [1]. Para ello se ha diseñado, montado y programado un robot 3-RRR, se ha definido un sistema de ecuaciones cuya solución es directamente el espacio de configuraciones libre de singularidades y colisiones, y se ha desarrollado una interfaz que permite la planificación, simulación y ejecución de trayectorias dentro de dicho espacio. En su conjunto, el sistema funciona correctamente, cumpliendo las funciones para las que ha sido diseñado.

5.2 Posibles mejoras y trabajo futuro

Durante el proyecto se han detectado posibles mejoras y caminos de investigación que se podrían llevar a cabo en futuras extensiones de este trabajo. Se enumeran a continuación:

1. El material plástico utilizado para la impresión en 3D de las piezas mecánicas del robot presenta flexiones que puede llegar a hacer que haya colisión entre eslabones proximales y distales. Sería recomendable seleccionar otro material menos flexible y más robusto para estas piezas.

2. El servomotor utilizado tiene una etapa de reducción basada en engranajes circulares. Este tipo de transmisión presenta juego o *backslash* (ángulo que gira el eje de salida cuando cambia su sentido de giro sin que llegue a girar el eje de entrada). Este juego hace que el espacio de trabajo disponible para planificar trayectorias se reduzca ya que hay que alejarse más de las singularidades para evitar su influencia. Una solución posible sería utilizar un reductor de engranajes planetarios con muy bajo juego.

3. La plataforma Arduino limita las capacidades del robot. Se deberían estudiar otras alternativas de hardware electrónico que permitan una mayor frecuencia de trabajo y potencia de cálculo. De esta manera, gran parte de la funciones que ahora realizan Matlab y el paquete *Cuik* podrían ser ejecutadas, de manera más eficiente, en la placa electrónica de control. Por ejemplo, se podría integrar el planificador de trayectorias íntegramente en esta placa electrónica, dando así autonomía al robot.

4. En el supuesto de la se lleve a termino la mejora anterior, se podría hacer que este nuevo robot, con capacidad de planificación integrada, utilizara ROS (*Robot Operating System*) como medio de interacción con sistemas externos. De esta manera se podría generar un nodo, correspondiente al robot, el cual podría publicar información sobre su estado actual, y suscribirse a otros tópicos que le informen de hacia donde ha de ir o la situación de obstáculos en el área de trabajo.

5. En las ecuaciones que definen el espacio en el que planificar caminos, se ha utilizado el parámetro b_{max} para establecer una distancia mínima con las configuraciones singulares. Este parámetro no tiene una interpretación geométrica que nos permita saber exactamente cuanto nos estamos alejando de la configuración singular más próxima. Sería interesante disponer de un índice que cuantifique para cada configuración del robot, el error de posición del efector final debido a un error de posición en sus actuadores.

6. Siguiendo con las ecuaciones de definición del espacio libre de singularidades directas y auto-colisiones, se podrían definir nuevas ecuaciones que permitan restringuir el área de trabajo, por ejemplo para definir la posición de obstáculos fijos o áreas de seguridad por las que el robot no pueda pasar. Esta definición se podría hacer dibujando los límites de dicha zona desde la interfaz gráfica.

7. En el prototipo actual la única manera de saber si hay continuidad entre dos configuraciones diferentes es, o bien navegando manualmente por las gráficas que facilita la interfaz gráfica, o lanzando el planificador, el cual, en casos extremos, puede tardar varios minutos en dar una respuesta. Habría que implementar un método que permita conocer, en un tiempo razonable, a que área del espacio de trabajo podemos llegar partiendo de una configuración conocida, y definir algún tipo de métrica que nos permita intuir la complejidad del camino, como por ejemplo, el número de cambios de modos del trabajo.

8. El resultado del filtro de alisado del camino encontrado por el planificador tiene una repercusión directa en la suavidad con la que se desplaza el robot al ejecutar una trayectoria. La estrategia actual utiliza el método *shortcut* de la aplicación *cuiksmoothpath*, incluida en el paquete *Cuik*. Este método es rápido, pero el resultado puede presentar cambios de

dirección bruscos en el camino. Una alternativa sería utilizar el método *gradient*, de la misma aplicación, pero su ejecución es demasiado lenta para obtener el resultado deseado en un tiempo razonable. Habría que buscar un método que consiga resultados comparables a los del método *gradient* con una velocidad comparable a la del método *shortcut*.

9. Con las asunciones hechas en la Sección 1.3, no se han tenido en cuenta en la estrategia de control, ni la dinámica del sistema, ni las posibles perturbaciones que puedan afectarle. Podrían incluirse estos supuestos, para así eliminar las restricciones de trabajo a baja velocidad y peso reducido. En este mismo aspecto, deberían considerarse las perturbaciones producidas por la interacción entre las diferentes piezas mecánicas del robot.

10. Actualmente no hay implementado en el robot ningún método que le permita saber si se está produciendo alguna anomalía en su movimiento, como en el caso de que entre en colisión con un objeto externo o que vaya a parar a una configuración singular. Se podría monitorizar el consumo de los motores de tal manera que se les corte la alimentación en tales supuestos.

Bibliografía

[1] O. Bohigas, M. E. Henderson, L. Ros, M. Manubens, and J. M. Porta, "Planning singularity-free paths on closed-chain manipulators," *IEEE Transactions on Robotics*, vol. 29, pp. 888–898, 2013.

[2] L. Jaillet and J. M. Porta, "Path planning under kinematic constraints by rapidly exploring manifolds," *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 105–117, 2013.

[3] P. I. Corke, *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*, ch. Dynamics and Control, pp. 204–211. Springer, 2011. ISBN 978-3-642-20143-1.

[4] J. Canny, *The Complexity of Robot Motion Planning*. Cambridge, Massachusetts: The MIT Press, 1988.

[5] J.-C. Latombe, *Robot motion planning*. Kluwer Academic Pub., 1991.

[6] L. E. Kavraki and S. M. LaValle, *Springer Handbook of Robotics*, ch. Motion Planning, pp. 109–131. Berlin, Heidelberg: Springer, 2008.

[7] L. Han and N. M. Amato, "A kinematics-based probabilistic roadmap method for closed chain systems," *Proc. of the International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pp. 233–246, 2000.

[8] J. H. Yakey, S. M. LaValle, and L. E. Kavraki, "Randomized path planning for linkages with closed kinematic chains," *IEEE Transactions on Robotics*, vol. 17, no. 6, pp. 951–958, 2001.

[9] J. Cortés and T. Siméon, "Sampling-based motion planning under kinematic loop-closure constraints," *Algorithmic Foundations of Robotics VI*, pp. 75–90, 2005.

[10] J. M. Porta, J. Cortés, L. Ros, and F. Thomas, "A space decomposition method for path planning of loop linkages," *Proc. of IEEE International Conference on Intelligent Robots and Systems*, pp. 1882–1888, 2007.

[11] J. M. Porta, L. Jaillet, and O. Bohigas, "Randomized path planning on manifolds based on higher-dimensional continuation," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 201–215, 2012.

[12] D. Berenson, S. S. Srinivasa, and J. J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[13] O. Bohigas, M. Manubens, and L. Ros, "Singularities of non-redundant manipulators: A short account and a method for their computation in the planar case," *Mechanism and Machine Theory*, vol. 68, pp. 1–17, October 2013.

[14] O. Bohigas, D. Zlatanov, L. Ros, M. Manubens, and J. M. Porta, "A general method for the numerical computation of manipulator singularity sets," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 340–351, 2014.

[15] C. Jui and Q. Sun, "Path tracking of parallel manipulators in the presence of force singularity," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 127, pp. 550–563, 2005.

[16] S. Briot and V. Arakelian, "On the dynamic properties and optimum control of parallel manipulators in the presence of singularity," in *IEEE International Conference on Robotics and Automation, ICRA*, pp. 1549–1555, 2008.

[17] S. Bhattacharya, H. Hatwal, and A. Ghosh, "Comparison of an exact and an approximate method of singularity avoidance in platform type parallel manipulators," *Mechanism and Machine Theory*, vol. 33, no. 7, pp. 965–974, 1998.

[18] H. Schaub and J. L. Junkins, "Singularity avoidance using null motion and variable-speed control moment gyros," vol. 23, no. 1, pp. 11–16.

[19] G. Marani, J. Kim, J. Yuh, and W. K. Chung, "A real-time approach for singularity avoidance in resolved motion rate control of robotic manipulators," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 2, pp. 1973–1978 vol.2, 2002.

[20] B. Dasgupta and T. S. Mruthyunjaya, "Singularity-free path planning for the Stewart platform manipulator," *Mechanism and Machine Theory*, vol. 33, no. 6, pp. 711–725, 1998.

[21] F. Bourbonnais, P. Bigras, and I. Bonev, "Minimum-time trajectory planning and control of a pick-and-place five-bar parallel robot," *Mechatronics, IEEE/ASME Transactions on*, vol. 20, pp. 740–749, April 2015.

[22] S. Sen, B. Dasgupta, and A. K. Mallik, "Variational approach for singularity-free path-planning of parallel manipulators," *Mechanism and Machine Theory*, vol. 38, no. 11, pp. 1165–1183, 2003.

[23] A. K. Dash, I. Chen, S. H. Yeo, and G. Yang, "Workspace generation and planning singularity-free path for parallel manipulators," *Mechanism and Machine Theory*, vol. 40, no. 7, pp. 776–805, 2005.

[24] "MECADEMIC: Affordable desktop high-accuracy robot arms." http://www.mecademic.com.

[25] J. G. De Jalón and E. Bayo, *Kinematic and Dynamic Simulation of Multibody Systems*. Springer Verlag, 1993.

[26] D. Zlatanov, *Generalized Singularity Analysis of Mechanisms*. PhD thesis, University of Toronto, 1998.

[27] O. Bohigas, *Numerical Computation and Avoidance of Manipulator Singularities*. PhD thesis, Universitat Politècnica de Catalunya, 2013. Available through the web address http://goo.gl/wlR0i.

[28] J. M. Porta, L. Ros, and F. Thomas, "A linear relaxation technique for the position analysis of multi-loop linkages," *IEEE Transactions on Robotics*, vol. 25, no. 2, pp. 225–239, 2009.

[29] L. Ros, "Geometric Fundamentals for Robot Design: On-line Lecture Notes," *Universitat Politècnica de Catalunya - BarcelonaTech*, 2008-2015. http://www.iri.upc.edu/gfrd.

[30] L. Ros, "Singularities of a 3-RRR mechanism." https://youtu.be/xV3m6ioilnc.

[31] M. E. Henderson, *Numerical Continuation Methods for Dynamical Systems: path following and boundary value problems*, ch. Higher-Dimensional Continuation, pp. 77–115. Springer, 2007.

[32] M. E. Henderson, "Multiple parameter continuation: Computing implicitly defined k-manifolds," *International Journal of Bifurcation and Chaos*, vol. 12, no. 3, pp. 451–476, 2002.

[33] E. L. Allgower and K. Georg, "Numerical path following," *Handbook of Numerical Analysis*, vol. 5, no. 3, pp. 3–207, 1997.

[34] J. Porta, L. Ros, O. Bohigas, M. Manubens, C. Rosales, and L. Jaillet, "The CUIK Suite: Motion analysis of closed-chain multibody systems," *IEEE Robotics and Automation Magazine*, vol. 21, pp. 105–114, September 2014.

[35] D. J. Bates, J. D. Hauenstein, A. J. Sommese, and C. W. Wampler, "Adaptive multiprecision path tracking," *SIAM Journal on Numerical Analysis*, vol. 46, no. 2, pp. 722–746, 2008.

[36] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[37] I. A. Bonev, *Geometric Analysis of Parallel Mechanisms*. PhD thesis, Faculté des Sciences et de Génie, Université de Laval, 2002.

[38] J. D'Errico, "INTERPARC: Distance based interpolation along a general curve in space," *Mathworks*. Matlab file in File Exchange, Matlab Central, Version 1.3. http://goo.gl/RdhUw7.

[39] "Protocolo dynamixel communication 1.0." http://goo.gl/AtVxgY.

[40] "Manual del servomotor MX-64 de Robotis." http://goo.gl/aiwCKP.

[41] "Página web de Arduino." https://www.arduino.cc/.

[42] "Documentation of the CUIK suite." http://www.iri.upc.edu/cuik.

A
Presupuesto

Este apéndice pretende dar una estimación del coste de construcción de un robot 3-RRR como el desarrollado. Se relacionan todos los materiales y componentes utilizados, así como su precio, pero no se incluye la mano de obra dedicada al diseño e implementación de la partes electrónica, mecánica, y de programación, debido a su subjetividad y poca relevancia en el contexto de este proyecto.

Los precios de los materiales son de venta al público, en Euros, a fecha de la presentación de esta memoria, y sin considerar descuentos. Cuando en el proveedor indicado se ha de realizar un pedido mínimo de piezas, se indica el precio unitario.

En la Tabla A.1 se muestra el listado de material mecánico. En los productos en los que no se indica proveedor se ha hecho una estimación de su coste. En la Tabla A.2 se muestra el listado de material eléctrico-electrónico. El precio total apróximado de la construcción del robot asciende a **1,939.24 €**.

Ref.	Descripción	Proveedor	Precio unitario	Núm. unid.	Precio total
747-800	Rodamiento de bolas NMB, radial	Amidata S.L.U.	3.00	3	9.00
619-0812	Rodamiento de bolas, ranura profunda	Amidata S.L.U.	5.45	3	16.35
521-7758	Casquillo eje 5mm, Salida 8mm, long. 12mm,	Amidata S.L.U.	1.59	3	4.77
278-562	Acoplamiento M5 chapado en níquel	Amidata S.L.U.	0.21	4	0.84
424-3469	Barra hexagonal de acero dulce 1x10mm	Amidata S.L.U.	3.05	1	3.05
466-7219	Perfil aluminio 20x20	Amidata S.L.U.	25.31	1	25.31
466-7247	Tapa negra para perfil 20x20	Amidata S.L.U.	0.85	6	5.01
	Plancha metacrilato + corte		100.00	1	100.00
	Impresión cada Arduino		40.00	1	40.00
	Impresión soporte motor-patas		66.00	3	198.00
	Impresión eslabón corto + largo		50.00	3	150.00
	Impresión soporte peana		60.00	1	60.00
	Total material mecánico				612.33

Tabla A.1: Material mecánico utilizado en la construcción del robot.

Ref.	Descripción	Proveedor	Precio unitario	Núm. unid.	Precio total
DXL0008	Servomotor Dynamixel MX-64T	Robótica Global,S.L.	380.50	3	1,141.50
714-2693	Codificador Avago AEAT-6012	Amidata S.L.U.	22.52	3	67.56
2126	Fuente alimentación 12V 5A EU-220V	Robótica Global,S.L.	48.35	1	48.35
A000067	Placa Arduino Mega rev 3	Diotronic,S.A.	39.00	1	39.00
A000059	Arduino USB2Serial	Diotronic,S.A.	10.00	1	10.00
803-2184	Memoria SRAM 23LCV512	Amidata S.L.U.	2.00	1	2.00
008	Zocalo DIL 8 pins soldar	Diotronic,S.A.	0.06	1	0.06
630-437	IC 74F244	Amidata S.L.U.	0.91	1	0.91
020	Zocalo DIL 20 pins soldar	Diotronic,S.A.	0.12	1	0.12
1035025	Resistencia 10k, 1/4W	Diotronic,S.A.	0.02	2	0.04
DTS62N	Pulsador membrana	Diotronic,S.A.	0.09	3	0.27
CT17	Placa fibra de vidrio 100x160	Diotronic,S.A.	6.54	1	6.54
2X25PY	Tira doble de pines 2X25CTS macho	Diotronic,S.A.	0.39	1	0.39
40PY	Tira pines 40 CTS macho	Diotronic,S.A.	0.39	1	0.39
625-7369	Conector IDC 10 vías hembra	Amidata S.L.U.	1.38	4	5.52
CO1710	Conector ICD 10 vías acodado macho	Onda Radio,S.A.	0.66	1	0.66
CP10	Cable plano 10,AWG28-10, 1 metro	Diotronic,S.A.	0.41	1	0.41
687-8095	Conector Molex acodado 3 pines	Amidata S.L.U.	0.05	2	0.10
797-9105	Conector SPOX, serie 5264 3 vías	Amidata S.L.U.	0.09	7	0.63
687-7152	Contacto de conector SPOX 5263	Amidata S.L.U.	0.03	21	0.63
Q1140B	Cable 1x0.5 blanco, 1 metro	Diotronic,S.A.	0.31	3	0.93
447-6580	Carcasa con.hembra 5polos paso 1,25mm	Amidata S.L.U.	0.20	3	0.60
CP10	Contacto conector Molex 50079-8000	Amidata S.L.U.	0.02	15	0.30
	Total material eléctrico/electrónico				1,326.91

Tabla A.2: Material eléctrico/electrónico utilizado en la construcción del robot.

B
Los ficheros de cálculo

B.1 Ficheros para efectuar la planificación

Para resolver un problema de planificación, el entorno MATLAB lanza el siguiente comando *Cuik* sobre el sistema operativo [32].

```
cuikatlas_rrt scf
```

Este comando busca un camino que una dos puntos $\mathbf{x}_s$ y $\mathbf{x_g}$ sobre la variedad $\mathcal{M}$ (Secc. 2.4.2), utilizando el método probabilístico de planificación basado en árboles de expansión rápida (Secc. 2.4.3). Los puntos $\mathbf{x_g}$ de $\mathcal{M}$ corresponden a las configuraciones inicial y final $\mathbf{q}_s$ y $\mathbf{q_g}$ deseadas, fijadas por el usuario desde la interfaz gráfica.

El comando `cuikatlas_rrt` lee dos ficheros de ínput y produce un fichero de output. Los ficheros de ínput son `scf.links`, que almacena los puntos $\mathbf{x}_s$ y $\mathbf{x_g}$, y `scf.cuik`, que especifica las ecuaciones de $\mathcal{M}$. El fichero de output es `scf.path`, y contiene una descripción del camino encontrado entre $\mathbf{x}_s$ y x_g. Los ficheros `scf.links` y `scf.path` see escriben y leen, respectivamente, desde la librería MATLAB implementada, utilizando el formato específico del paquete *Cuik* (Véase [42] para más información). A continuación se incluye el contenido del fichero `scf.cuik`, añadiendo comentarios que clarifican su estructura.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File scf.cuik
% Defines the equations of the navigation manifold M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[CONSTANTS]

%-----------------------------------------------
% Edit this section to define a new 3-Rrr robot
%-----------------------------------------------

% Coordinates of the base joints expressed in the absolute frame (in dm)

J11x := 0.000000
J11y := 0.000000
J12x := 2.350000
J12y := 0.000000
J13x := 1.175000
J13y := 2.035000
```

```
% Sides (in dm) and angle (in rad) of the moving triangle

phi := 1.047198
l3  := 1.200000
l4  := 1.200000

% Length of the proximal (l1) and distal (l2) limbs of each leg (in dm)

l1  := 1.000000
l2  := 1.350000

% Parameters of the superellipse that bounds the region R of a link (in dm)

ctrx := 0.675000
ctry := 0.000000
sx := 0.675000
sy := 0.270000

% Coordinates of the platform joints expressed in the moving frame (in dm)

J31rx := 0
J31ry := 0
J32rx := l3
J32ry := 0
J33rx := l4 * cos(phi)
J33ry := l4 * sin(phi)

% Conservative bound used for the position of all joints

maxpos := l1 + l2 + l3 + l4 + l2 + l1

% Distances from the base anchor points to the origin, used to bound s1, s2, and s3

d1 := sqrt(J11x^2 + J11y^2)
d2 := sqrt(J12x^2 + J12y^2)
d3 := sqrt(J13x^2 + J13y^2)

% Bounds of the auxiliary variables tJ...
% (where mu is computed as mu = 2*l2 + max[l3,l4])

mu    := 3.900000
txmin := -mu/sx - ctrx/sx
txmax :=  mu/sx - ctrx/sx
tymin := -mu/sy - ctry/sy
tymax :=  mu/sy - ctry/sy

% Clearance w.r.t. the singularity locus

bmax := 0.500000



% Clearance with respect to the collision locus, computed as 1 / (k*offset),
```

```
% where offset = 1 (superellipse offset) and k = 10 (hyperbola shape factor)

gmax := 0.100000

[SYSTEM VARS]

% Orientation angle of the moving triangle

th8~[-pi,pi]

% Barycenter coordinates

bx:[-maxpos,maxpos]
by:[-maxpos,maxpos]

% Absolute coords of the moving revolute joints

J31x:[-maxpos,maxpos]
J31y:[-maxpos,maxpos]

J21x:[-maxpos,maxpos]
J21y:[-maxpos,maxpos]

J32x:[-maxpos,maxpos]
J32y:[-maxpos,maxpos]

J22x:[-maxpos,maxpos]
J22y:[-maxpos,maxpos]

J33x:[-maxpos,maxpos]
J33y:[-maxpos,maxpos]

J23x:[-maxpos,maxpos]
J23y:[-maxpos,maxpos]

% Orientation angles of links 2, 3, and 4

th2~[-pi,pi]
th3~[-pi,pi]
th4~[-pi,pi]

% Orientation angles of links 5, 6, and 7

th5~[-pi,pi]
th6~[-pi,pi]
th7~[-pi,pi]

% Vectors s1, s2, s3 (note that they are not normalized)

s1x:[-l2,l2]
s1y:[-l2,l2]
s1z:[-(d1+l1)*l2,(d1+l1)*l2]

s2x:[-l2,l2]
s2y:[-l2,l2]
```

```
s2z:[-(d2+l1)*l2,(d2+l1)*l2]

s3x:[-l2,l2]
s3y:[-l2,l2]
s3z:[-(d3+l1)*l2,(d3+l1)*l2]

% Auxiliary variable to send singular configs. to infinity
% The bound is symmetric [-value,value], to account for all
% possible working modes

b:[-bmax,bmax]

% Auxiliary variables to send collision configs. to infinity

g1:[0,gmax]
g2:[0,gmax]
g3:[0,gmax]
g4:[0,gmax]
g5:[0,gmax]
g6:[0,gmax]
g7:[0,gmax]
g8:[0,gmax]
g9:[0,gmax]
g10:[0,gmax]
g11:[0,gmax]
g12:[0,gmax]

% Auxiliary variables to have a simpler system

tJ21x2:[txmin,txmax]
tJ21y2:[tymin,tymax]
tJ31x2:[txmin,txmax]
tJ31y2:[tymin,tymax]
tJ22x1:[txmin,txmax]
tJ22y1:[tymin,tymax]
tJ32x1:[txmin,txmax]
tJ32y1:[tymin,tymax]
tJ22x3:[txmin,txmax]
tJ22y3:[tymin,tymax]
tJ32x3:[txmin,txmax]
tJ32y3:[tymin,tymax]
tJ23x2:[txmin,txmax]
tJ23y2:[tymin,tymax]
tJ33x2:[txmin,txmax]
tJ33y2:[tymin,tymax]
tJ21x3:[txmin,txmax]
tJ21y3:[tymin,tymax]
tJ31x3:[txmin,txmax]
tJ31y3:[tymin,tymax]
tJ23x1:[txmin,txmax]
tJ23y1:[tymin,tymax]
tJ33x1:[txmin,txmax]
tJ33y1:[tymin,tymax]

[SYSTEM EQS]
```

```
% Loop closure equations

J12x + l1*cos(th3) + l2*cos(th6) - l3*cos(th8)
- l2*cos(th5) - l1*cos(th2) - J11x = 0;
J12y + l1*sin(th3) + l2*sin(th6) - l3*sin(th8)
- l2*sin(th5) - l1*sin(th2) - J11y = 0;

J13x + l1*cos(th4) + l2*cos(th7) - l4*cos(th8)*cos(phi) +
l4*sin(th8)*sin(phi) - l2*cos(th5) - l1*cos(th2) - J11x = 0;
J13y + l1*sin(th4) + l2*sin(th7) - l4*sin(th8)*cos(phi) -
l4*cos(th8)*sin(phi) - l2*sin(th5) - l1*sin(th2) - J11y = 0;

% Equations defining the positions of all moving joints

J21x = J11x + l1 * cos(th2);
J21y = J11y + l1 * sin(th2);

J22x = J12x + l1 * cos(th3);
J22y = J12y + l1 * sin(th3);

J23x = J13x + l1 * cos(th4);
J23y = J13y + l1 * sin(th4);

J31x = J21x + l2 * cos(th5);
J31y = J21y + l2 * sin(th5);

J32x = J22x + l2 * cos(th6);
J32y = J22y + l2 * sin(th6);

J33x = J23x + l2 * cos(th7);
J33y = J23y + l2 * sin(th7);

% Equations defining the barycenter

3 * bx = J31x + J32x + J33x;
3 * by = J31y + J32y + J33y;

% Equations defining the s1, s2, and s3 vectors
% (note that they are not normalized)

s1x = J31x - J21x;
s1y = J31y - J21y;
s1z = J21x * J31y - J21y * J31x;

s2x = J32x - J22x;
s2y = J32y - J22y;
s2z = J22x * J32y - J22y * J32x;

s3x = J33x - J23x;
s3y = J33y - J23y;
s3z = J23x * J33y - J23y * J33x;

% Equation to ensure a full-rank Jacobian
```

```
b * s1x*s2y*s3z + b * s1z*s2x*s3y + b * s1y*s2z*s3x -
b * s1z*s2y*s3x - b * s2z*s3y*s1x - b * s1y*s2x*s3z = 1;

% Collision avoidance equations

g1 * tJ21x2^4 + g1 * tJ21y2^4 - g1 - 0.100000 = 0;
- sx * tJ21x2 + cos(th6)*J21x - cos(th6)*J22x + sin(th6)*J21y - sin(th6)*J22y - ctrx = 0;
- sy * tJ21y2 -sin(th6)*J21x + sin(th6)*J22x + cos(th6)*J21y - cos(th6)*J22y - ctry = 0;

g2 * tJ31x2^4 + g2 * tJ31y2^4 - g2 - 0.100000 = 0;
- sx * tJ31x2 + cos(th6)*J31x - cos(th6)*J22x + sin(th6)*J31y - sin(th6)*J22y - ctrx = 0;
- sy * tJ31y2 -sin(th6)*J31x + sin(th6)*J22x + cos(th6)*J31y - cos(th6)*J22y - ctry = 0;

g3 * tJ22x1^4 + g3 * tJ22y1^4 - g3 - 0.100000 = 0;
- sx * tJ22x1 + cos(th5)*J22x - cos(th5)*J21x + sin(th5)*J22y - sin(th5)*J21y - ctrx = 0;
- sy * tJ22y1 -sin(th5)*J22x + sin(th5)*J21x + cos(th5)*J22y - cos(th5)*J21y - ctry = 0;

g4 * tJ32x1^4 + g4 * tJ32y1^4 - g4 - 0.100000 = 0;
- sx * tJ32x1 + cos(th5)*J32x - cos(th5)*J21x + sin(th5)*J32y - sin(th5)*J21y - ctrx = 0;
- sy * tJ32y1 -sin(th5)*J32x + sin(th5)*J21x + cos(th5)*J32y - cos(th5)*J21y - ctry = 0;

g5 * tJ22x3^4 + g5 * tJ22y3^4 - g5 - 0.100000 = 0;
- sx * tJ22x3 + cos(th7)*J22x - cos(th7)*J23x + sin(th7)*J22y - sin(th7)*J23y - ctrx = 0;
- sy * tJ22y3 -sin(th7)*J22x + sin(th7)*J23x + cos(th7)*J22y - cos(th7)*J23y - ctry = 0;

g6 * tJ32x3^4 + g6 * tJ32y3^4 - g6 - 0.100000 = 0;
- sx * tJ32x3 + cos(th7)*J32x - cos(th7)*J23x + sin(th7)*J32y - sin(th7)*J23y - ctrx = 0;
- sy * tJ32y3 -sin(th7)*J32x + sin(th7)*J23x + cos(th7)*J32y - cos(th7)*J23y - ctry = 0;

g7 * tJ23x2^4 + g7 * tJ23y2^4 - g7 - 0.100000 = 0;
- sx * tJ23x2 + cos(th6)*J23x - cos(th6)*J22x + sin(th6)*J23y - sin(th6)*J22y - ctrx = 0;
- sy * tJ23y2 -sin(th6)*J23x + sin(th6)*J22x + cos(th6)*J23y - cos(th6)*J22y - ctry = 0;

g8 * tJ33x2^4 + g8 * tJ33y2^4 - g8 - 0.100000 = 0;
- sx * tJ33x2 + cos(th6)*J33x - cos(th6)*J22x + sin(th6)*J33y - sin(th6)*J22y - ctrx = 0;
- sy * tJ33y2 -sin(th6)*J33x + sin(th6)*J22x + cos(th6)*J33y - cos(th6)*J22y - ctry = 0;

g9 * tJ21x3^4 + g9 * tJ21y3^4 - g9 - 0.100000 = 0;
- sx * tJ21x3 + cos(th7)*J21x - cos(th7)*J23x + sin(th7)*J21y - sin(th7)*J23y - ctrx = 0;
- sy * tJ21y3 -sin(th7)*J21x + sin(th7)*J23x + cos(th7)*J21y - cos(th7)*J23y - ctry = 0;

g10 * tJ31x3^4 + g10 * tJ31y3^4 - g10 - 0.100000 = 0;
- sx * tJ31x3 + cos(th7)*J31x - cos(th7)*J23x + sin(th7)*J31y - sin(th7)*J23y - ctrx = 0;
- sy * tJ31y3 -sin(th7)*J31x + sin(th7)*J23x + cos(th7)*J31y - cos(th7)*J23y - ctry = 0;

g11 * tJ23x1^4 + g11 * tJ23y1^4 - g11 - 0.100000 = 0;
- sx * tJ23x1 + cos(th5)*J23x - cos(th5)*J21x + sin(th5)*J23y - sin(th5)*J21y - ctrx = 0;
- sy * tJ23y1 -sin(th5)*J23x + sin(th5)*J21x + cos(th5)*J23y - cos(th5)*J21y - ctry = 0;

g12 * tJ33x1^4 + g12 * tJ33y1^4 - g12 - 0.100000 = 0;
- sx * tJ33x1 + cos(th5)*J33x - cos(th5)*J21x + sin(th5)*J33y - sin(th5)*J21y - ctrx = 0;
- sy * tJ33y1 -sin(th5)*J33x + sin(th5)*J21x + cos(th5)*J33y - cos(th5)*J21y - ctry = 0;
```

B.2 Ficheros para el cálculo de las singularidades

Para poder visualizar las singularidades directas e inversas del robot en la interfaz gráfica, éstas se han tenido que calcular primero mediante el paquete *Cuik*. Las singularidades directas delimitan la frontera del espacio de trabajo del robot, y las inversas indican las configuraciones donde aparece una pérdida de rigidez. En la interfaz gráfica, los lugares geométricos de estas singularidades se indican en azul y rojo, respectivamente. A continuación se incluyen los ficheros *Cuik* necesarios para calcular tales singularidades.

B.2.1. Fichero para calcular las singularidades inversas

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File inverse.cuik
%
% Inverse kinematic singularities of the 3-Rrr manipulator
% for a fixed platform orientation (given by theta_8)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[CONSTANTS]

% Coordinates of the base joints expressed in the absolute frame

J11x := 0
J11y := 0
J12x := 2.35
J12y := 0
J13x := 1.175
J13y := 2.035

% Coordinates of the platform joints expressed in the moving frame

J31rx := 0
J31ry := 0
J32rx := 1.2
J32ry := 0
J33rx := 0.6
J33ry := 1.03

% Distances l3, from J31 to J32, and l4, from J31 to J33

l3 := sqrt((J31rx-J32rx)^2 + (J31ry-J32ry)^2)
l4 := sqrt((J31rx-J33rx)^2 + (J31ry-J33ry)^2)

% Cosine and sine of the angle between segments l3 and l4, assumed in the 1st quadrant

cphi := ( (J32rx-J31rx)*(J33rx-J31rx)+(J32ry-J31ry)*(J33ry-J31ry) ) / (l3*l4)
sphi := sqrt(1-cphi^2)

% Length of the lower (l1) and upper (l2) limbs of each leg

l1 := 1
l2 := 1.350
```

```
% Value of the orientation angle theta_8 of the platform and its sine and cosine

theta_8 := 0
ct8 := cos(theta_8)
st8 := sin(theta_8)

% Conservative bound used for the position of all joints

maxpos := l1 + l2 + l3 + l2 + l1

% Distances from the base anchor points to the origin, used to bound s1, s2, and s3

d1 := sqrt(J11x^2 + J11y^2)
d2 := sqrt(J12x^2 + J12y^2)
d3 := sqrt(J13x^2 + J13y^2)

[SYSTEM VARS]

% Absolute coords of the moving revolute joints

J31x:[-maxpos,maxpos]
J31y:[-maxpos,maxpos]

J21x:[-maxpos,maxpos]
J21y:[-maxpos,maxpos]

J32x:[-maxpos,maxpos]
J32y:[-maxpos,maxpos]
J22x:[-maxpos,maxpos]
J22y:[-maxpos,maxpos]

J33x:[-maxpos,maxpos]
J33y:[-maxpos,maxpos]
J23x:[-maxpos,maxpos]
J23y:[-maxpos,maxpos]

% Cosines and sines of the orientation angles of links 2, 3, and 4

ct2:[-1,1]
st2:[-1,1]
ct3:[-1,1]
st3:[-1,1]
ct4:[-1,1]
st4:[-1,1]

% Cosines and sines of the orientation angles of links 5, 6, and 7

ct5:[-1,1]
st5:[-1,1]
ct6:[-1,1]
st6:[-1,1]
ct7:[-1,1]
st7:[-1,1]

% Vectors s1, s2, s3 (note that they are not normalized)
```

```
s1x:[-l2,l2]
s1y:[-l2,l2]
s1z:[-(d1+l1)*l2,(d1+l1)*l2]

s2x:[-l2,l2]
s2y:[-l2,l2]
s2z:[-(d2+l1)*l2,(d2+l1)*l2]

s3x:[-l2,l2]
s3y:[-l2,l2]
s3z:[-(d3+l1)*l2,(d3+l1)*l2]

% Components of the omega vector

om11:[-1,1]
om12:[-1,1]
om13:[-1,1]

% Diagonal elements of the B matrix

B11:[-l1*l1*l2,l1*l1*l2]
B22:[-l1*l1*l2,l1*l1*l2]
B33:[-l1*l1*l2,l1*l1*l2]

[SYSTEM EQS]

% Loop closure equations

J12x + l1*ct3 + l2*ct6 - l3*ct8 - l2*ct5 - l1*ct2 - J11x = 0;
J12y + l1*st3 + l2*st6 - l3*st8 - l2*st5 - l1*st2 - J11y = 0;

J13x + l1*ct4 + l2*ct7 - l4*ct8*cphi + l4*st8*sphi - l2*ct5 - l1*ct2 - J11x = 0;
J13y + l1*st4 + l2*st7 - l4*st8*cphi - l4*ct8*sphi - l2*st5 - l1*st2 - J11y = 0;

% Circle equations due to angle algebraizations

ct2^2 + st2^2 = 1;
ct5^2 + st5^2 = 1;
ct3^2 + st3^2 = 1;
ct6^2 + st6^2 = 1;
ct4^2 + st4^2 = 1;
ct7^2 + st7^2 = 1;

% Equations defining the positions of all moving joints

J21x = J11x + l1 * ct2;
J21y = J11y + l1 * st2;

J22x = J12x + l1 * ct3;
J22y = J12y + l1 * st3;

J23x = J13x + l1 * ct4;
J23y = J13y + l1 * st4;
```

```
J31x = J21x + l2 * ct5;
J31y = J21y + l2 * st5;

J32x = J22x + l2 * ct6;
J32y = J22y + l2 * st6;

J33x = J23x + l2 * ct7;
J33y = J23y + l2 * st7;

% Equations defining the s1, s2, and s3 vectors (note that they are not normalized)

s1x = J31x - J21x;
s1y = J31y - J21y;
s1z = J21x*J31y - J21y*J31x;

s2x = J32x - J22x;
s2y = J32y - J22y;
s2z = J22x*J32y - J22y*J32x;

s3x = J33x - J23x;
s3y = J33y - J23y;
s3z = J23x*J33y - J23y*J33x;

% Equations defining the B matrix

B11 = s1x*J11y - s1y*J11x + s1z;
B22 = s2x*J12y - s2y*J12x + s2z;
B33 = s3x*J13y - s3y*J13x + s3z;

% Equations forcing the rank deficiency of matrix B

B11 * om11 = 0;
B22 * om12 = 0;
B33 * om13 = 0;

om11^2 + om12^2 + om13^2 = 1;

% Inequality to select only one solution of the gamma = [om11,om12,om13] vector

0.1394*om11 + 0.4143*om12 + 0.8994*om13 >= 0;
```

B.2.2. Fichero para calcular las singularidades directas

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File forward.cuik
%
% Forward kinematic singularities of the 3-Rrr manipulator
% for a fixed platform orientation (given by theta_8)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[CONSTANTS]

% Coordinates of the base joints expressed in the absolute frame

J11x := 0
J11y := 0
J12x := 2.35
J12y := 0
J13x := 1.175
J13y := 2.035

% Coordinates of the platform joints expressed in the moving frame

J31rx := 0
J31ry := 0
J32rx := 1.2
J32ry := 0
J33rx := 0.6
J33ry := 1.03

% Distances l3, from J31 to J32, and l4, from J31 to J33

l3 := sqrt((J31rx-J32rx)^2 + (J31ry-J32ry)^2)
l4 := sqrt((J31rx-J33rx)^2 + (J31ry-J33ry)^2)

% Cosine and sine of the angle between segments l3 and l4, assumed in the 1st quadrant

cphi := ( (J32rx-J31rx)*(J33rx-J31rx)+(J32ry-J31ry)*(J33ry-J31ry) ) / (l3*l4)
sphi := sqrt(1-cphi^2)

% Length of the lower (l1) and upper (l2) limbs of each leg

l1 := 1
l2 := 1.350

% Value of the orientation angle theta_8 of the platform and its sine and cosine

theta_8 := 0
ct8 := cos(theta_8)
st8 := sin(theta_8)

% Conservative bound used for the position of all joints

maxpos := l1 + l2 + l3 + l2 + l1

% Distances from the base anchor points to the origin, used to bound s1, s2, and s3
```

```
d1 := sqrt(J11x^2 + J11y^2)
d2 := sqrt(J12x^2 + J12y^2)
d3 := sqrt(J13x^2 + J13y^2)

[SYSTEM VARS]

% Absolute coords of the moving revolute joints

J31x:[-maxpos,maxpos]
J31y:[-maxpos,maxpos]

J21x:[-maxpos,maxpos]
J21y:[-maxpos,maxpos]

J32x:[-maxpos,maxpos]
J32y:[-maxpos,maxpos]
J22x:[-maxpos,maxpos]
J22y:[-maxpos,maxpos]

J33x:[-maxpos,maxpos]
J33y:[-maxpos,maxpos]
J23x:[-maxpos,maxpos]
J23y:[-maxpos,maxpos]

% Cosines and sines of the orientation angles of links 2, 3, and 4

ct2:[-1,1]
st2:[-1,1]
ct3:[-1,1]
st3:[-1,1]
ct4:[-1,1]
st4:[-1,1]

% Cosines and sines of the orientation angles of links 5, 6, and 7

ct5:[-1,1]
st5:[-1,1]
ct6:[-1,1]
st6:[-1,1]
ct7:[-1,1]
st7:[-1,1]

% Vectors s1, s2, s3 (note that they are not normalized)

s1x:[-l2,l2]
s1y:[-l2,l2]
s1z:[-(d1+l1)*l2,(d1+l1)*l2]

s2x:[-l2,l2]
s2y:[-l2,l2]
s2z:[-(d2+l1)*l2,(d2+l1)*l2]

s3x:[-l2,l2]
s3y:[-l2,l2]
```

```
s3z:[-(d3+l1)*l2,(d3+l1)*l2]

% Components of the Tw vector

Twx:[-1,1]
Twy:[-1,1]
Twz:[-1,1]

[SYSTEM EQS]

% Loop closure equations

J12x + l1*ct3 + l2*ct6 - l3*ct8 - l2*ct5 - l1*ct2 - J11x = 0;
J12y + l1*st3 + l2*st6 - l3*st8 - l2*st5 - l1*st2 - J11y = 0;

J13x + l1*ct4 + l2*ct7 - l4*ct8*cphi + l4*st8*sphi - l2*ct5 - l1*ct2 - J11x = 0;
J13y + l1*st4 + l2*st7 - l4*st8*cphi - l4*ct8*sphi - l2*st5 - l1*st2 - J11y = 0;

% Circle equations due to angle algebraizations

ct2^2 + st2^2 = 1;
ct5^2 + st5^2 = 1;
ct3^2 + st3^2 = 1;
ct6^2 + st6^2 = 1;
ct4^2 + st4^2 = 1;
ct7^2 + st7^2 = 1;

% Equations defining the positions of all moving joints

J21x = J11x + l1 * ct2;
J21y = J11y + l1 * st2;

J22x = J12x + l1 * ct3;
J22y = J12y + l1 * st3;

J23x = J13x + l1 * ct4;
J23y = J13y + l1 * st4;

J31x = J21x + l2 * ct5;
J31y = J21y + l2 * st5;

J32x = J22x + l2 * ct6;
J32y = J22y + l2 * st6;

J33x = J23x + l2 * ct7;
J33y = J23y + l2 * st7;

% Equations defining the s1, s2, and s3 vectors (note that they are not normalized)

s1x = J31x - J21x;
s1y = J31y - J21y;
s1z = J21x*J31y - J21y*J31x;

s2x = J32x - J22x;
s2y = J32y - J22y;
```

```
s2z = J22x*J32y - J22y*J32x;

s3x = J33x - J23x;
s3y = J33y - J23y;
s3z = J23x*J33y - J23y*J33x;

% Equations to force the rank deficiency of matrix A

s1x*Twx + s1y*Twy + s1z*Twz = 0;
s2x*Twx + s2y*Twy + s2z*Twz = 0;
s3x*Twx + s3y*Twy + s3z*Twz = 0;

% Equation to force a twist vector Tw of unit norm

Twx^2 + Twy^2 + Twz^2 = 1;

% Inequality to select only one solution of the Tw vector

0.1394*Twx + 0.4143*Twy + 0.8994*Twz >= 0;

% Inequalities to select a specific working mode
% (uncomment if necessary)
%
% s1x*J11y - s1y*J11x + s1z >= 0;
% s2x*J12y - s2y*J12x + s2z >= 0;
% s3x*J13y - s3y*J13x + s3z <= 0;
```

C

Documentación técnica de los elementos

A continuación se incluye la siguiente documentación técnica de los principales elementos que integran el robot:

- Documentación del servomotor Dynamixel MX-64 de Robotis, incluyendo:
  1. Manual del servomotor MX-64T.
  2. Formato detallado de los *instruction* y *status packets*.
  3. Listado de instrucciones posibles en un *instruction packet*.
- Características y parámetros físicos del motor Maxon RE-max 21, modelo 250002, montado en el servomotor.
- Características del codificador angular absoluto AVAGO, modelo AEAT-6012.
- Características del conector rotativo BTH1256 de la marca ByTune Electronics.
- Características de la memoria externa SPI SRAM de 512 kbits, modelo 23LCV512, de Microchip Technology Inc.

Documentación del servomotor Dynamixel MX-64 de Robotis

Manual del servomotor MX-64T
Formato detallado de los *instruction* y *status packets*
Listado de instrucciones posibles en un *instruction packet*

MX-64T / MX-64R

Parts Photo

[MX-64T] [MX-64R]

※ Control Table's Compliance replaced by PID.

※ The control table's order for PID has changed to DIP from this version onwards. Please make reference of this change.

※ Although the MX-64T (TTL) and MX-64R (RS-485) differ in communications protocols both have the same features and perform equally. (TTL uses 3-pin connectors while RS-485 uses 4)

H/W Specification

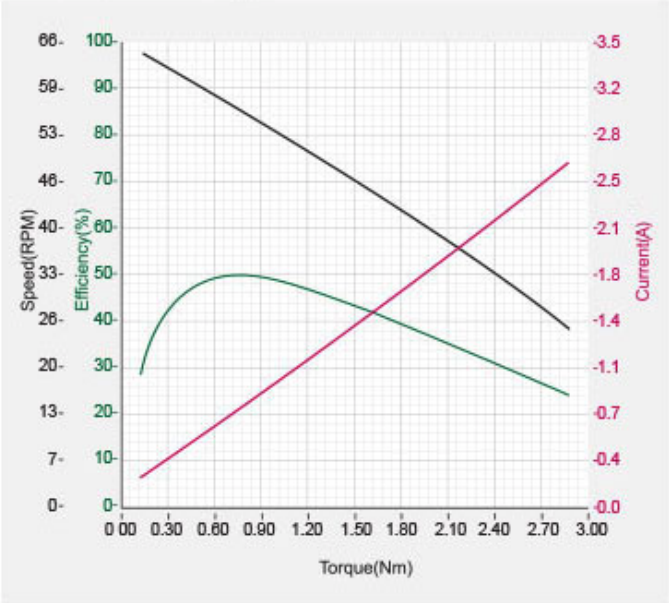
- MCU : ST CORTEX-M3 (STM32F103C8 @ 72MHZ,32BIT)
- POSITION SENSOR : Contactless absolute encoder (12BIT,360 DEGREE)
- MOTOR : Maxon
- BAUD RATE : 8000 bps ~ 4.5 Mbps
- CONTROL ALGORITHM : PID CONTROL
- Resolution : 0.088°
- Running Degree
  - 0° ~ 360°
  - Endless Turn
- Weight : 126g
- Dimension : 40.2mm x 61.1mm x 41mm
- Gear Reduction Ratio : 200 : 1
- Stall Torque
  - 5.5N.m (at 11.1V, 3.9A),
  - 6.0N.m (at 12V, 4.1A)
  - 7.3N.m (at 14.8V, 5.2A)
- No load speed
  - 58rpm (at 11.1V)
  - 63rpm (at 12V)
  - 78rpm (at 14.8V)
- Running Temperature : -5℃ ~ +80℃
- Voltage : 10 ~ 14.8V (Recommended Voltage 12V)

- Command Signal : Digital Packet
- Protocol Type
  - MX-64T (Half duplex Asynchronous Serial Communication (8bit,1stop, No Parity))
  - MX-64R (RS485 Asynchronous Serial Communication (8bit,1stop, No Parity))
- Link (Physical)
  - MX-64T (TTL Level Multi Drop Bus)
  - MX-64R (RS485 Multi Drop Bus)
- ID : 254 ID (0~253)
- Feedback : Position, Temperature, Load, Input Voltage, etc.
- Material : Full Metal Gear, Engineering Plastic Body
- Standby current : 100 mA

Data

	Unit	Data
Weight	g	135
Dimension	mm	40.2 x 61.1 x 41
Gear Ratio	type/material	200:1 (Spur/Metal)
Network	-	TTL / RS-485
Position Sensor (resolution)	-	Contactless Absolute Encoder (360°/ 4096)
Motor	-	Maxon Motor
Operation Voltage	V	10~14.8
Stall Torque	N.m	6.0 at 12V
Stall Current	A	4.1 at 12V
No Load Speed	RPM	63 at 12V

Performance Graph



Precautions when connecting to power supply!

- For the stable power supply, we recommend using ROBOTIS controller or SMPS2Dynamixel.
- Connect your DYNAMIXEL to power supply while it's off and turn on/off with the power switch.

Control Table

Control Table consists of data regarding the current status and operation, which exists inside of Dynamixel. The user can control Dynamixel by changing data of Control Table via Instruction Packet.

EEPROM and RAM

Data in RAM area is reset to the initial value whenever the power is turned on while data in EEPROM area is kept once the value is set even if the power is turned off.

Address

It represents the location of data. To read from or write data to Control Table, the user should assign the correct address in the Instruction Packet.

Access

Dynamixel has two kinds of data: Read-only data, which is mainly used for sensing, and Read-and-Write data, which is used for driving.

Initial Value

In case of data in the EEPROM Area, the initial values on the right side of the below Control Table are the factory default settings. In case of data in the RAM Area, the initial values on the right side of the above Control Tables are the ones when the power is turned on.

Highest/Lowest Byte

In the Control table, some data share the same name, but they are attached with (L) or (H) at the end of each name to distinguish the address. This data requires 16bit, but it is divided into 8bit each for the addresses (low) and (high). These two addresses should be written with one Instruction Packet at the same time.

Area	Address (Hexadecimal)	Name	Description	Access	Initial Value (Hexadecimal)
EEPROM	0 (0X00)	Model Number(L)	Lowest byte of model number	R	54 (0X36)
	1 (0X01)	Model Number(H)	Highest byte of model number	R	1 (0X01)
	2 (0X02)	Version of Firmware	Information on the version of firmware	R	-
	3 (0X03)	ID	ID of Dynamixel	RW	1 (0X01)
	4 (0X04)	Baud Rate	Baud Rate of Dynamixel	RW	34 (0X22)
	5 (0X05)	Return Delay Time	Return Delay Time	RW	250 (0XFA)
	6 (0X06)	CW Angle Limit(L)	Lowest byte of clockwise Angle Limit	RW	0 (0X00)
	7 (0X07)	CW Angle Limit(H)	Highest byte of clockwise Angle Limit	RW	0 (0X00)
	8 (0X08)	CCW Angle Limit(L)	Lowest byte of counterclockwise Angle Limit	RW	255 (0XFF)
	9 (0X09)	CCW Angle Limit(H)	Highest byte of counterclockwise Angle Limit	RW	15 (0X0F)
	11 (0X0B)	the Highest Limit Temperature	Internal Limit Temperature	RW	80 (0X50)
	12 (0X0C)	the Lowest Limit Voltage	Lowest Limit Voltage	RW	60 (0X3C)
	13 (0X0D)	the Highest Limit Voltage	Highest Limit Voltage	RW	160 (0XA0)
	14 (0X0E)	Max Torque(L)	Lowest byte of Max. Torque	RW	255 (0XFF)
	15 (0X0F)	Max Torque(H)	Highest byte of Max. Torque	RW	3 (0X03)
	16 (0X10)	Status Return Level	Status Return Level	RW	2 (0X02)
	17 (0X11)	Alarm LED	LED for Alarm	RW	36 (0X24)
	18 (0X12)	Alarm Shutdown	Shutdown for Alarm	RW	36 (0X24)
	20 (0X14)	Multi Turn Offset(L)	multi-turn offset least significant byte (LSB)	RW	0 (0X00)
	21 (0X15)	Multi Turn Offset(H)	multi-turn offset most significant byte (MSB)	RW	0 (0X00)
	22 (0X16)	Resolution Divider	Resolution divider	RW	1 (0X01)
	24 (0X18)	Torque Enable	Torque On/Off	RW	0 (0X00)
	25 (0X19)	LED	LED On/Off	RW	0 (0X00)
	26 (0X1A)	D Gain	Derivative Gain	RW	0 (0X00)
	27 (0X1B)	I Gain	Integral Gain	RW	0 (0X00)
	28 (0X1C)	P Gain	Proportional Gain	RW	32 (0X20)
	30 (0X1E)	Goal Position(L)	Lowest byte of Goal Position	RW	-
	31 (0X1F)	Goal Position(H)	Highest byte of Goal Position	RW	-
	32 (0X20)	Moving Speed(L)	Lowest byte of Moving Speed (Moving Velocity)	RW	-
	33 (0X21)	Moving Speed(H)	Highest byte of Moving Speed (Moving Velocity)	RW	-
	34 (0X22)	Torque Limit(L)	Lowest byte of Torque Limit (Goal Torque)	RW	ADD14
	35 (0X23)	Torque Limit(H)	Highest byte of Torque Limit (Goal Torque)	RW	ADD15
	36 (0X24)	Present Position(L)	Lowest byte of Current Position (Present Velocity)	R	-
	37 (0X25)	Present Position(H)	Highest byte of Current Position (Present	R	-

R			Velocity)		
A	38 (0X26)	Present Speed(L)	Lowest byte of Current Speed	R	-
M	39 (0X27)	Present Speed(H)	Highest byte of Current Speed	R	-
	40 (0X28)	Present Load(L)	Lowest byte of Current Load	R	-
	41 (0X29)	Present Load(H)	Highest byte of Current Load	R	-
	42 (0X2A)	Present Voltage	Current Voltage	R	-
	43 (0X2B)	Present Temperature	Current Temperature	R	-
	44 (0X2C)	Registered	Means if Instruction is registered	R	0 (0X00)
	46 (0X2E)	Moving	Means if there is any movement	R	0 (0X00)
	47 (0X2F)	Lock	Locking EEPROM	RW	0 (0X00)
	48 (0X30)	Punch(L)	Lowest byte of Punch	RW	0 (0X00)
	49 (0X31)	Punch(H)	Highest byte of Punch	RW	0 (0X00)
	68 (0X44)	Current(L)	Lowest byte of Consuming Current	RW	0 (0X00)
	69 (0X45)	Current(H)	Highest byte of Consuming Current	RW	0 (0X00)
	70 (0X46)	Torque Control Mode Enable	Torque control mode on/off	RW	0 (0X00)
	71 (0X47)	Goal Torque(L)	Lowest byte of goal torque value	RW	0 (0X00)
	72 (0X48)	Goal Torque(H)	Highest byte of goal torque value	RW	0 (0X00)
	73 (0X49)	Goal Acceleration	Goal Acceleration	RW	0 (0X00)

Address Function Help

EEPROM Area

Model Number

It represents the Model Number.

Firmware Version

It represents the firmware version.

ID

It is a unique number to identify Dynamixel.

The range from 0 to 252 (0xFC) can be used, and, especially, 254(0xFE) is used as the Broadcast ID.

If the Broadcast ID is used to transmit Instruction Packet, we can command to all Dynamixels.

> Please be careful not to duplicate the ID of connected Dynamixel.

Baud Rate

It is the baud rate to communicate with controller. It is available in between 0~254(0XFE).

If the data value is in between 0~249 :

Baudrate(BPS) = 2000000 / (Data + 1)

Data	Set BPS	Target BPS	Tolerance
1	1000000.0	1000000.0	0.000 %
3	500000.0	500000.0	0.000 %
4	400000.0	400000.0	0.000 %
7	250000.0	250000.0	0.000 %
9	200000.0	200000.0	0.000 %

16	117647.1	115200.0	-2.124 %
34	57142.9	57600.0	0.794 %
103	19230.8	19200.0	-0.160 %
207	9615.4	9600.0	-0.160 %

If the date value is over the 250 :

Data	Set BPS	Target BPS	Tolerance
250	2250000.0	2250000.0	0.000 %
251	2500000.0	2500000.0	0.000 %
252	3000000.0	3000000.0	0.000 %

> Note : Maximum Baud Rate error of 3% is within the tolerance of UART communication.

Return Delay Time

It is the delay time per data value that takes from the transmission of Instruction Packet until the return of Status Packet.
0 to 254 (0xFE) can be used, and the delay time per data value is 2 usec.
That is to say, if the data value is 10, 20 usec is delayed. The initial value is 250 (0xFA) (i.e., 0.5 msec).

CW/CCW Angle Limit

The angle limit allows the motion to be restrained.
The range and the unit of the value is the same as Goal Position(Address 30, 31).

- CW Angle Limit: the minimum value of Goal Position(Address 30, 31)
- CCW Angle Limit: the maximum value of Goal Position(Address 30, 31)

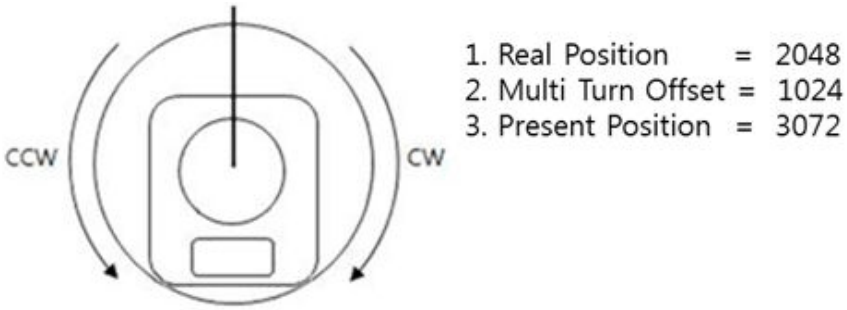
The following two modes can be set pursuant to the value of CW and CCW.

Operation Type	CW / CCW
Wheel Mode	both are 0
Joint Mode	neither at 0
Multi-turn Mode	both are 4095

The wheel mode can be used to wheel-type operation robots since motors of the robots spin infinitely.
The joint mode can be used to multi-joints robot since the robots can be controlled with specific angles.
Multi-turn mode allows joints have range of controllable position values from -28672 to 28672.

Multi Turn Offset

Adjusts position (zeroing). This value gets included in Present Position (36).
Present position + multi-turn offset.
Initial value is 0 and range is from -24576 to 24576
A Dynamixel with a position of 2048 with an applied offset of 1024 outputs a Present position of 3072.



> Note: This feature is only applied in multi-turn mode and ignored in other modes.

Resolution Divider

It allows the user to change Dynamixel's resolution.

The default Resolution Divider Value is set as 1. (1 ~ 4 available)

When resolution is lowered, revolutions (in both directions) can be increased (up to 28 turns in each direction).

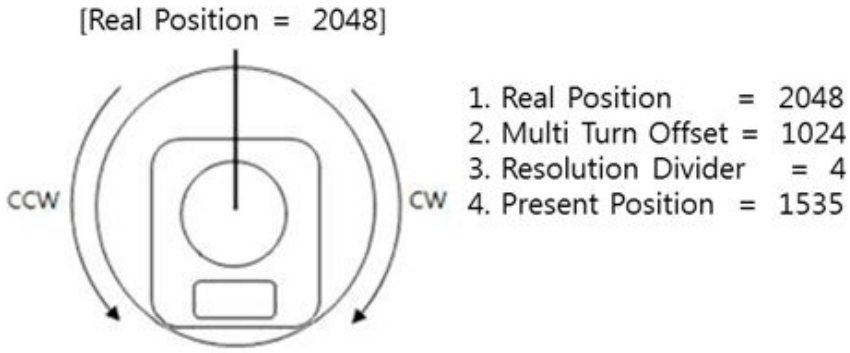
Present Position = Real Position / Resolution Divider

For example, a Real Position of 2048 with a Resolution Divider set as 2 will yield a Present Position value of 1024 (2048/2 = 1024). A Dynamixel with Resolution Divider set as 2 will have a resolution 2048 for a single revolution.

The Present Position can be obtained while Multi-turn Offset and Resolution Divider are taken into account.

Present position = (Real Position / Resolution Divider) + Multi-turn Offset

For example, a Dynamixel with a Real Position of 2048 with a Resolution Divider set as 4 and Multi-turn Offset as 1024 will yield a Present Position of 1535 ((2048/4) + 1024 = 1535).



> Note: This feature is only applied in multi-turn mode and ignored in other modes.

The Highest Limit Temperature

> Caution : Do not set the temperature lower/higher than the default value.
> When the temperature alarm shutdown occurs, wait 20 minutes to cool the temperature before re-use.
> Using the product when the temperature is high may and can cause damage.

The Lowest (Highest) Limit Voltage

It is the operation range of voltage.

50 to 250 (0x32 ~ 0x96) can be used. The unit is 0.1V.

For example, if the value is 80, it is 8V.

If Present Voltage (Address42) is out of the range, Voltage Range Error Bit (Bit0) of Status Packet is returned as '1' and Alarm is triggered as set in the addresses 17 and 18.

Max Torque

It is the torque value of maximum output. 0 to 1023 (0x3FF) can be used, and the unit is about 0.1%.

For example, Data 1023 (0x3FF) means that Dynamixel will use 100% of the maximum torque it can produce while Data 512 (0x200) means that Dynamixel will use 50% of the maximum torque. When the power is turned on, Torque Limit (Addresses 34 and 35) uses the value as the initial value.

Status Return Level

It decides how to return Status Packet. There are three ways like the below table.

Value	Return of Status Packet
0	No return against all commands (Except PING Command)
1	Return only for the READ command
2	Return for all commands

> When Instruction Packet is Broadcast ID, Status Packet is not returned regardless of Status Return Level.

Alarm LED

Alarm Shutdown

Dynamixel can protect itself by detecting errors occur during the operation.
The errors can be set are as the table below.

Bit	Name	Contents
Bit 7	0	-
Bit 6	Instruction Error	When undefined Instruction is transmitted or the Action command is delivered without the reg_write command
Bit 5	Overload Error	When the current load cannot be controlled with the set maximum torque
Bit 4	CheckSum Error	When the Checksum of the transmitted Instruction Packet is invalid
Bit 3	Range Error	When the command is given beyond the range of usage
Bit 2	OverHeating Error	When the internal temperature is out of the range of operating temperature set in the Control Table
Bit 1	Angle Limit Error	When Goal Position is written with the value that is not between CW Angle Limit and CCW Angle Limit
Bit 0	Input Voltage Error	When the applied voltage is out of the range of operating voltage set in the Control Table

It is possible to make duplicate set since the function of each bit is run by the logic of 'OR'. That is, if 0X05 (binary 00000101) is set, both Input Voltage Error and Overheating Error can be detected.
If errors occur, in case of Alarm LED, the LED blinks; in case of Alarm Shutdown, the motor output becomes 0 % by making the value of Torque Limit(Address 34, 35) as 0.

RAM Area

Torque Enable

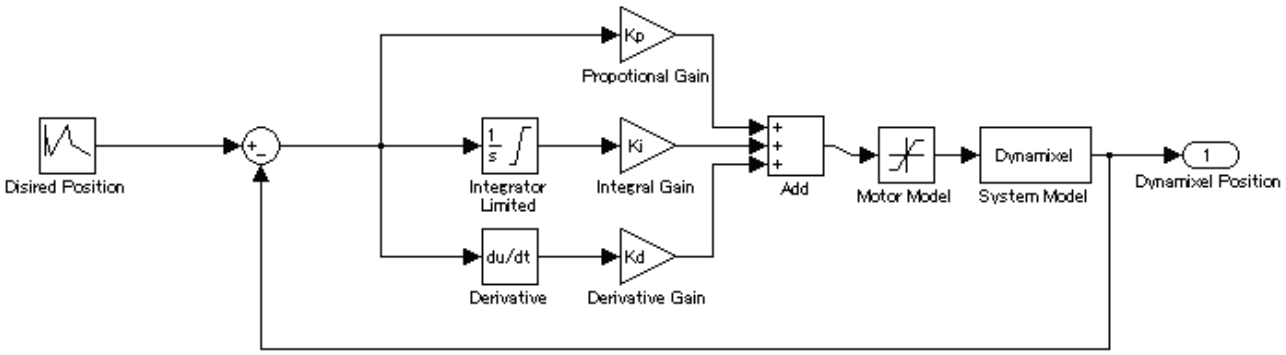
Value	Meaning
0	Keeps Torque from generating by interrupting the power of motor.
1	Generates Torque by impressing the power to the motor.

LED

Bit	Meaning
0	Turn OFF the LED
1	Turn ON the LED

PID Gain

MX series will use the PID controller as a main control method.
P gain refers to the value of proportional band.
I gain refers to the value of integral action.
D Gain refers to the value of derivative action.
Gains values are in between 0~254.



K_p = P Gain / 8
K_i = I Gain * 1000 / 2048
K_d = D Gain * 4 / 1000

※ The relationship between Compliance Slop and PID

Slope	P Gain
8	128
16	64
32	32
64	16
128	8

The less the P gain, The larger the back lash, and the weaker the amount of output near goal position.

At some extent, it is like a combined concept of margine and slope.

It does not exactly match the previous concept of compliance. So it is obvious if you see the difference in terms of motion.

※ Explanation for PID required.

For the brief explanation about general PID, please refer to the website(link) below.

http://en.wikipedia.org/wiki/PID_controller

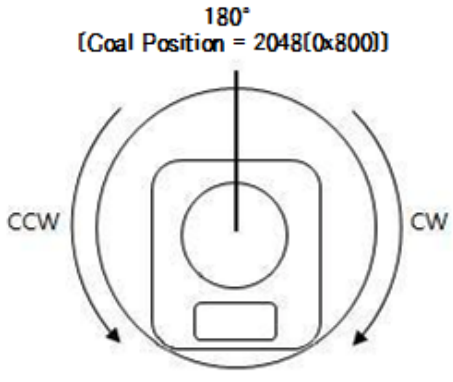
FYI, PID control theory is not only limited to the control of motor(actuator) but is a generic theory that can be applied to all kinds of control.

Goal Position

It is a position value of destination.

0 to 4095 (0xFFF) is available. The unit is 0.088 degree.

If Goal Position is out of the range, Angle Limit Error Bit (Bit1) of Status Packet is returned as '1' and Alarm is triggered as set in Alarm LED/Shutdown.



Moving Speed

- Joint Mode, Multi-Turn mode

It is a moving speed to Goal Position.
0~1023 (0X3FF) can be used, and the unit is about 0.114rpm.
If it is set to 0, it means the maximum rpm of the motor is used without controlling the speed.
If it is 1023, it is about 117.07rpm.
For example, if it is set to 300, it is about 34.33 rpm.

- Wheel Mode
  It is a moving speed to Goal direction.
  0~2047 (0X7FF) can be used, and the unit is about 0.114rpm.
  If a value in the range of 0~1023 is used, it is stopped by setting to 0 while rotating to CCW direction.
  If a value in the range of 1024~2047 is used, it is stopped by setting to 1024 while rotating to CW direction.
  That is, the 10th bit becomes the direction bit to control the direction.

> Note: This mode allows to check max rpm. Any values set higher than max rpm will not take effect.

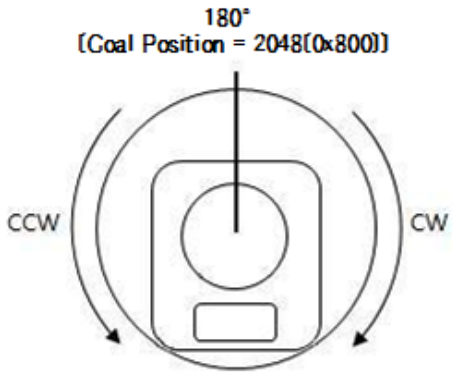
Torque Limit

It is the value of the maximum torque limit.
0 to 1023 (0x3FF) is available, and the unit is about 0.1%.
For example, if the value is 512, it is about 50%; that means only 50% of the maximum torque will be used.
If the power is turned on, the value of Max Torque (Address 14, 15) is used as the initial value.

> Notes: If the function of Alarm Shutdown is triggered, the motor loses its torque because the value becomes 0. At this moment, if the value is changed to the value other than 0, the motor can be used again.

Present Position

It is the current position value of Dynamixel.
The range of the value is 0~4095 (0xFFF), and the unit is 0.088 degree.



In multi-turn mode range is from -28672 to 28672 with unit values dependent on Resolution Divider (0.088 * Resolution Divider)

> Note: in multi-turn mode Present position depends on resolution divider and multi-turn offset For more information turn to the section on Multi Turn offset and Resolution Divider

Present Speed

Is the current moving speed.
0~2047 (0x000~0X7FF) can be used.
If a value is in the rage of 0~1023 then the motor rotates to the CCW direction.
If a value is in the rage of 1024~2047 then the motor rotates to the CW direction.
The 10th bit becomes the direction bit to control the direction; 0 and 1024 are equal.
The value unit is about 0.11rpm.
For example, if it is set to 300 then the motor is moving to the CCW direction at a rate of about 34.33rpm.

Present Load

It means currently applied load.
The range of the value is 0~2047, and the unit is about 0.1%.
If the value is 0~1023, it means the load works to the CCW direction.
If the value is 1024~2047, it means the load works to the CW direction.
That is, the 10th bit becomes the direction bit to control the direction, and 1024 is equal to 0.
For example, the value is 512, it means the load is detected in the direction of CCW about 50% of the maximum torque.

BIT	15~11	10	9	8	7	6	5	4	3	2	1	0
Value	0	Load Direction	Data (Load Ratio)									

Load Direction = 0 : CCW Load, Load Direction = 1: CW Load

> Notes: Current load is inferred from the internal torque value, not from Torque sensor etc.
> For that reason, it cannot be used to measure weight or torque; however, it must be used only to detect which direction the force works.

Present Voltage

It is the size of the current voltage supplied.
This value is 10 times larger than the actual voltage. For example, when 10V is supplied, the data value is 100 (0x64)

Present Temperature

It is the internal temperature of Dynamixel in Celsius.
Data value is identical to the actual temperature in Celsius. For example, if the data value is 85 (0x55), the current internal temperature is 85℃.

Registered Instruction

Value	Meaning
0	There are no commands transmitted by REG_WRITE
1	There are commands transmitted by REG_WRITE.

> Notes: If ACTION command is executed, the value is changed into 0.

Moving

Value	Meaning
0	Goal position command execution is completed.
1	Goal position command execution is in progress.

Lock

Value	Meaning
0	EEPROM area can be modified.
1	EEPROM area cannot be modified.

> Caution: If Lock is set to 1, the power must be turned off and then turned on again to change into 0.

Punch

Current to drive motor is at minimum.

Can choose vales from 0x00 to 0x3FF.

Current

Value at 2048(0x800) when current is consumption is idle.

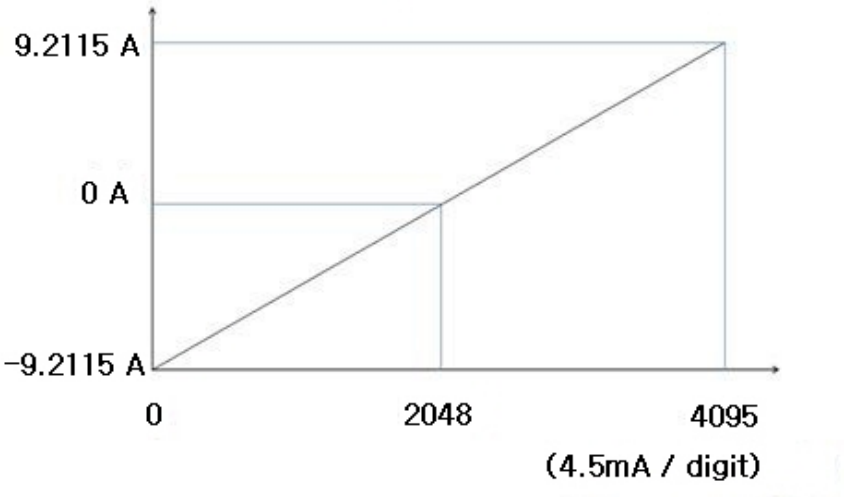
Values higher than 2048(0x800) during positive current flow

Values lower than 2048(0x800) during negative current flow

The following is a method to calculate current flow

I = (4.5mA) * (CURRENT – 2048) in amps unit (A).

For example, 68 gives a value of 2148, which corresponds to 450mA of current flow.



Torque Control Mode Enable

Value	Meaning
0	Turn off the torque mode. Executes Joint mode or Wheel mode.
1	Turn on the torque mode. Cannot control the position or moving speed but only Torque.

When Torque Control Mode Enable is at 1, DYNAMIXEL behaves like the followings

1. DYNAMIXEL does not control the position or the moving speed.
2. DYNAMIXEL controls with goal torque value.
3. DYNAMIXEL does not react to whatever value in Goal position and Goal speed.
4. Because position/moving speed is not controller, DYNAMIXEL behaves as if it is in the wheel mode.

Goal Torque

These are the goal torque value

You can use 0 ~ 2047 (0x7FF), and the unit is 4.5mA.

(torque is directly proportional to the current value.)

If you use from 0~1023, torque is on toward CCW, and when you set it to 0, it stops.

If you use from 1024~2047, torque is on toward CW, and when you set it to 1024, it stops.

That means, 10^{th} bit becomes the direction bit, which controls direction.

Goal Torque cannot be bigger than Torque Limit(34,35)

Goal Acceleration

This is Goal Acceleration value.

It can be used from 0~254(0XFE), and the unit is approximately 8.583 Degree / sec^2.

When it is set to 0, there is no control over acceleration and moves with the maximum acceleration of the motor.

When the goal speead is set to 0, there is no control over acceleration and moves with the maximum acceleration of the motor.

When it is set to 254, it becomes 2180 Degree / sec^2

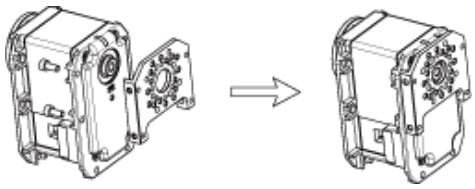
For example, the current speed of Dynamixel is 0, and Goal acceleration is 10,

The speed of Dynamixel after 1 second will be 14.3 RPM.

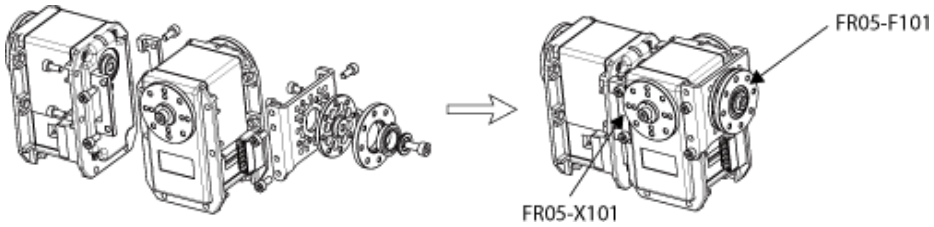
Option Frame

The types of MX-64 option frame are as follows.

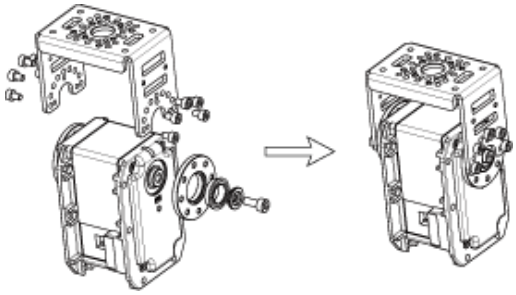
FR05-B101



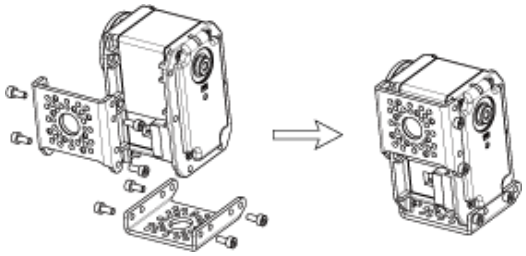
FR05-F101_FR05-X101



FR05-H101



FR05-S101

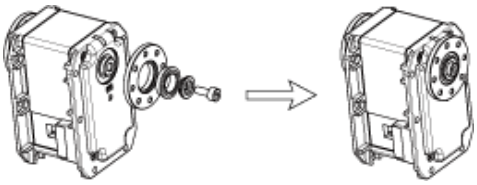


Horn

The types of MX-64 Horn are as follows.

HN05-N102

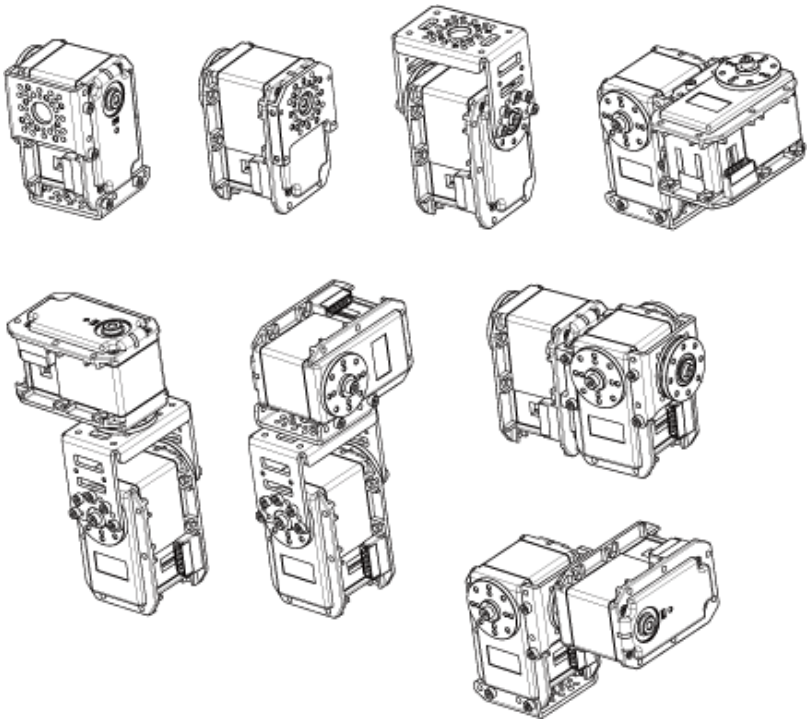
HN05-I101



Ref: place careful attention when aligning the horn of the RX-64 to prevent misalignment.

Combination

The following example shows the combination structure of option frames and horns.



Dimension

Drawing Information : DOWNLOAD RX64Dimension.pdf

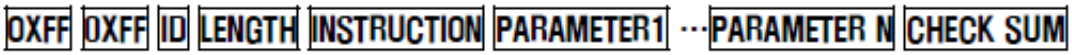
Videos

HOW TO REPLACE GEARS

Instruction Packet

Instruction Packet is command data that Main Controller sends to Dynamixel.

The structure of Instruction Packet is as follows:



The meaning of each byte composing packet is as follows:

0xFF 0xFF

This signal notifies the beginning of the packet

ID

It is the ID of Dynamixel which will receive Instruction Packet. It can use 254 IDs from 0 to 253 (0X00~0XFD).

Broadcasting ID

ID = 254 (0XFE)

If Broadcast ID is used, all linked Dynamixels execute command of Instruction Packet, and Status Packet is not returned.

LENGTH

It is the length of the packet. The length is calculated as “the number of Parameters (N) + 2”.

INSTRUCTION

This command gives an instruction to Dynamixel and has the following types.

Value	Name	Function	No.of Parameters
0x01	PING	No execution. It is used when controller is ready to receive Status Packet	0
0x02	READ_DATA	This command reads data from Dynamixel	2
0x03	WRITE_DATA	This command writes data to Dynamixel	2 or more
0x04	REG WRITE	It is similar to WRITE_DATA, but it remains in the standby state without being executed until the ACTION command arrives.	2 or more
0x05	ACTION	This command initiates motions registered with REG WRITE	0
0x06	RESET	This command restores the state of Dynamixel to the factory default setting.	0
0x83	SYNC WRITE	This command is used to control several Dynamixels simultaneously at a time.	4 or more

PARAMETER 0…N

Parameter is used when Instruction requires ancillary data.

CHECK SUM

It is used to check if packet is damaged during communication. Check Sum is calculated according to the following formula.

Check Sum = ~ (ID + Length + Instruction + Parameter1 + … Parameter N)

Where, "~" is the Not Bit operator.

When the calculation result of the parenthesis in the above formula is larger than 255 (0xFF), use only lower bytes.

For example, when you want to use Instruction Packet like the below

ID=1 (0x01), Length= 5 (0x05), Instruction= 3 (0x03),

Parameter1= 12 (0x0C), Parameter2= 100 (0x64), Parameter3= 170 (0xAA)

Check Sum = ~ (ID + Length + Instruction + Parameter1 + … Parameter 3)

= ~ [0x01 + 0x05 + 0x03 + 0x0C + 0x64 + 0xAA]

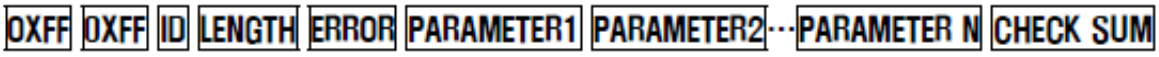
= ~ [0x123] // Only the lower byte 0x23 executes the Not operation.

= 0xDC

Thus, Instruction Packet should be 0x01, 0x05, 0x03, 0x0C, 0x64, 0xAA, 0xDC.

Status Packet (Return Packet)

Dynamixel executes command received from the Main controller and returns the result to the Main Controller. The returned data is called Status Packet. The structure of Status Packet is as follows:



Each byte composing the packet means as below.

0xFF 0xFF

This signal notifies the beginning of the packet.

ID

It is the ID of RX-64 which transfers Status Packet.

LENGTH

is the length of Status Packet, the value of which is"the number of Parameters (N) + 2".

ERROR

It displays the error status occurred during the operation of Dynamixel.

The meaning of each bit is described in the below table.

Bit	Name	Contents
Bit 7	0	-
Bit 6	Instruction Error	In case of sending an undefined instruction or delivering the action command without the reg_write command, it is set as 1.
Bit 5	Overload Error	When the current load cannot be controlled by the set Torque, it is set as 1.
Bit 4	Checksum Error	When the Checksum of the transmitted Instruction Packet is incorrect, it is set as 1.
Bit 3	Range Error	When a command is out of the range for use, it is set as 1.
Bit 2	Overheating Error	When internal temperature of Dynamixel is out of the range of operating temperature set in the Control table, it is set as 1.
Bit 1	Angle Limit Error	When Goal Position is written out of the range from CW Angle Limit to CCW Angle Limit , it is set as 1.
Bit 0	Input Voltage Error	When the applied voltage is out of the range of operating voltage set in the Control table, it is as 1.

For example, when Status Packet is returned as below

0xFF 0xFF 0x01 0x02 0x24 0xD8
It means that the error of 0x24 occurs from Dynamixel whose ID is 01. Since 0x24 is 00100100 as binary, Bit5 and Bit2 become 1. In order words, Overload and Overheating Errors have occurred.

> The error types on the table above are related to actuators, and the contents may vary depending on the type of Dynamixel.

PARAMETER 0…N

It returns data except ERROR. For the usage of parameters, refer to “3-5 How to Use Packet".

CHECK SUM

It is used to check if packet is damaged during communication. The below formula defines Check Sum. This formula is constructed in the same way as the Check Sum of Instruction Packet.
Check Sum = ~ (ID + Length + Error + Parameter1 + … Parameter N)

Kind of Instruction

To operate Dynamixel, Instruction Packet, which is binary type data, should be sent to Dynamixel from Main Controller. Instruction Packet has seven kinds of commands. (Refer to "Instruction Packet")
In addition, Dynamixel receives Instruction Packet to perform a command and returns the result as Status Packet to Main Controller. This section describes examples of the usage of each command of Instruction Packet.

> The following example is written based on Dynamixel Actuator RX-64. Since other Dynamixels such as AX-12/12+, DX etc. consist of equal command, the same Packet type can be used.

READ DATA

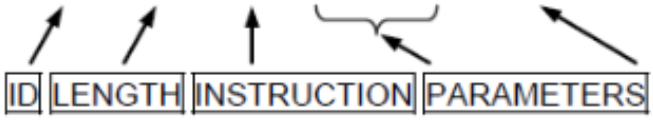
Function This command is to read data in the Control Table inside of RX-64.

Length 0X04
Instruction 0X02
Parameter1 Start Address of data to be read
Parameter2 Length of Data to be read

Example 1 Reads the current internal temperature of RX-64 whose ID is 1.

Reads 1 byte from the value of Address 43 (0x2B) in the Control Table.

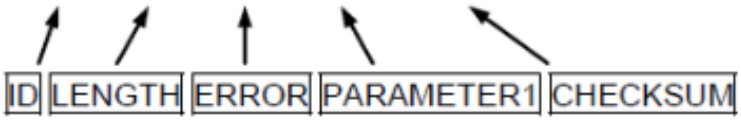
Instruction Packet : 0XFF 0XFF 0X01 0X04 0X02 0X2B 0X01 0XCC



CHECKSUM

Status Packet returned is as follows:

Status Packet : 0XFF 0XFF 0X01 0X03 0X00 0X20 0XDB



Data value read is 0x20 (i.e., 32 in decimal). Thus, the current internal temperature

WRITE DATA

Function This command is to write data to the Control Table inside of RX-64.

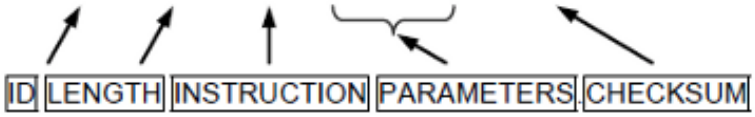
Length N+3 (if the number of writing data is N)
Instruction 0X03
Parameter1 Start address to write data
Parameter2 First data to write
Parameter3 Second data to write
Parameter N+1 Nth Data to write

Example 2	Sets the ID of RX-64 as "1".

Writes 1 to the Address 3 in the Control Table.

Sends ID as Broadcasting ID(0xFE).

Instruction Packet : 0XFF 0XFF 0XFE 0X04 0X03 0X03 0X01 0XF6`



Status Packet is not returned since Broadcast ID (0XFE) is transmitted.

REG WRITE

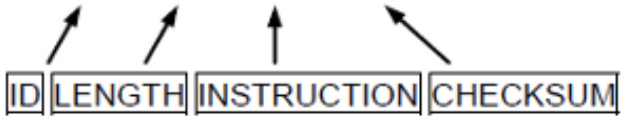
Function The REG_WRITE command is similar to the WRITE_DATA command in terms of function, but differs in terms of the timing that a command is executed. When Instruction Packet arrives, it is saved in Buffer and the Write operation remains in the standby state. At this moment, Registered Instruction (Address 44 (0x2C)) is set as "1". Then, when Action Instruction Packet arrives, Registered Instruction changes into "'0" and the registered Write command is finally executed.

Length N+3 (if the number of Writing Data is N)
Instruction 0X04
Parameter1 Start Address to write Data
Parameter2 First data to write
Parameter N+1 Nth data to write

Example 3 Receives Status Packet of RX-64 whose ID is 1.

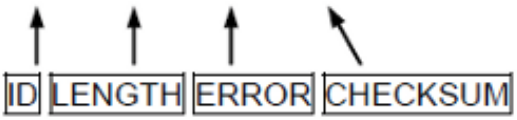
Reads 1 byte from the value of Address 43 (0x2B) in the Control Table.

Instruction Packet : 0XFF 0XFF 0X01 0X02 0X01 0XFB`



Status Packet returned is as follows:

Status Packet : 0XFF 0XFF 0X01 0X02 0X00 0XFC



ACTION

Function This command is to execute the Write action registered by REG_WRITE

Length 0X02
Instruction 0X05
Parameter NONE

The Action command is useful when several RX-64s are moved with accuracy at the same time. When several running gears are controlled via communication, there is a little time difference in terms of enabling time between the first and the last running gear getting commands. RX-64 has resolved this problem by using Action Instruction.

PING

Function This command does not instruct anything. It is only used when receiving Status Packet or confirming the existence of RX-64 with a specific ID. .

Length 0X02
Instruction 0X01
Parameter NONE

Although Status Return Level (Address 16 (0X10)) is 0, it returns Status Packet all the time for Ping Instruction. But, it does not return Status Packet when Check Sum Error occurs in spite of using PING Instruction.

RESET

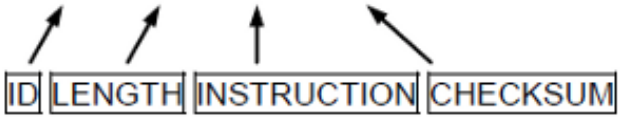
Function This command is to reset the Control Table of RX-64 to the factory default setting

> Please be careful since the value set by users can be erased if RESET command is used.

Length 0X02
Instruction 0X06
Parameter NONE

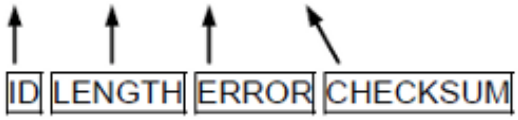
Example 4 Resets the Control Table of RX-64 whose ID is 0.

Instruction Packet : 0XFF 0XFF 0X00 0X02 0X06 0XF7`



Status Packet returned is as follows:

Status Packet : 0XFF 0XFF 0X00 0X02 0X00 0XFD



Please note that ID is changed into"‘1" after the execution of the RESET command.

SYNC WRITE

Function This command is used to control several RX-64s simultaneously with one Instruction Packet transmission. When this command is used, several commands are transmitted at once, so that the communication time is reduced when multiple RX-64s are controlled. However, the SYNC WRITE command can be used only if both of the address and length of the Control Table to write is identical. Besides, ID should be transmitted as Broadcasting ID.

> Generally, in the event 1 command packet is 4 byte, 26 Dynamixel can be controlled simultaneously. Make sure that the length of packet does not to exceed 143 bytes since the volume of receiving buffer of RX-64 is 143 bytes.

ID 0XFE
Length (L+1) X N + 4 (L: Data Length per RX-64, N: the number of RX-64s)
Instruction 0X83
Parameter1 Start address to write Data
Parameter2 Length of Data to write
Parameter3 First ID of RX-64
Parameter4 First data of the first RX-64
Parameter5 Second data of the first RX-64
...
Parameter L+3 Lth Data of the first RX-64
Parameter L+4 ID of the second RX-64
Parameter L+5 First data of the second RX-64
Parameter L+6 Second data of the second RX-64
...
Parameter 2L+4 Lth data of the second RX-64

Example 5 Moves to the following position and speed for each RX-64.

RX-64 with ID 0 : Moves to the position of 0x010 at the speed of 0x150
RX-64 with ID 1 : Moves to the position of 0x220 at the speed of 0x360
RX-64 with ID 2: Moves to the position of 0x030 at the speed of 0x170
RX-64 with ID 3: Moves to the position of 0x220 at the speed of 0x380

Instruction Packet : 0XFF 0XFF 0XFE 0X18 0X83 0X1E 0X04 0X00 0X10 0X00
0X50 0X01 0X01 0X20 0X02 0X60 0X03 0X02 0X30 0X00
0X70 0X01 0X03 0X20 0X02 0X80 0X03 0X12`

Status Packet is not returned since ID is transmitted as Broadcasting ID.

BULK READ (This command only works for MX series)

Function This command is used for reading the values of several DYNAMIXELs simultaneously, by sending a single Instruction Packet. The packet length is lessened compared to sending many READ commands, and the idle time between the status packets being returned is also lessened to save communication time. But, this cannot be used to read many times on a single module, and if several of the same module ID is designated, only the firstly designated parameter will be processed.

ID 0XFE
Length 3N3+3
Instruction 0X92
Parameter1 0X00
Parameter2 Length of the data to be read from the first module [L]
Parameter3 ID of the first module
Parameter4 Starting address of the data to be read from the first module

...

Parameter 3N+2 : Length of the data to be read from the Nth module [L]

Parameter 3N+3 : ID of the Nth module

Parameter 3N+4 : Starting address of the data to be read from the Nth module

example)

DYANMIXEL with ID 1 : Brings the goal position value (2 bytes from 0x1E).

DYNAMIXEL with ID 2 : Brings the current position value (2 bytes from 0x24).

The command packet to order this movement is as follows;

0XFF 0XFF 0XFE 0X09 0X92 0X00 0X02 0X01 0X1E 0X02 0X02 0X24 0X1D

During this time, module with ID 2 monitors the status packet being sent from ID 1 of the data bus (the very previous parameter ID), then right when module ID 1's status packet completes transmission, sends its own status packet. The returned status packet is as follows;

0XFF 0XFF 0X01 0X04 0X00 0X00 0X80 0X7A 0XFF 0XFF 0X02 0X04 0X00 0X00 0X80 0X79

Status packets from each of module ID 1 and ID 2 will come in one after another.

Other Example

The following examples are assumed that ID = 1, and Baud rate = 57142 BPS.

Example 6	Reads the Model Number and Firmware Version.
Hint	Instruction = READ_DATA, Address = 0x00, Length = 0x04
Communication	Instruction Packet : FF FF 01 04 02 00 03 F5 Status Packet : FF FF 01 05 00 40 00 08 B1
Status Packet Result	Model Number = 64 (0x40) Firmware Version = 0x08

Example 7	Changes the ID of RX-64 from 1 to 0.
Hint	Instruction = WRITE_DATA, Address = 0x03, DATA = 0x00
Communication	Instruction Packet : FF FF 01 04 03 03 00 F4 Status Packet : FF FF 01 02 00 FC
Status Packet Result	NO ERROR

Example 8	Changes the Baud Rate to 1M bps.
Hint	Instruction = WRITE_DATA, Address = 0x04, DATA = 0x01
Communication	Instruction Packet : FF FF 01 04 03 04 01 F2 Status Packet : FF FF 01 02 00 FC
Status Packet Result	NO ERROR

Example 9	Resets Return Delay Time as 4usec.
Hint	Instruction = WRITE_DATA, Address = 0x05, DATA = 0x02
Communication	Instruction Packet : FF FF 01 04 03 05 02 F0 Status Packet : FF FF 01 02 00 FC
Status Packet Result	NO ERROR

Example 10	Restricts the movement angle from 0 to 150°.
Hint	Since CCW Angle Limit 0x3FF means 300°, 150°corresponds to 0x200. Instruction = WRITE_DATA, Address = 0x08, DATA = 0x00, 0x02
Communication	Instruction Packet : FF FF 01 05 03 08 00 02 EC Status Packet : FF FF 01 02 00 FC
Status Packet Result	NO ERROR

Example 11	Resets the highest limit of operating temperature as 80°.
Hint	Instruction = WRITE_DATA, Address = 0x0B, DATA = 0x50
Communication	Instruction Packet : FF FF 01 04 03 0B 50 9C Status Packet : FF FF 01 02 00 FC
Status Packet Result	NO ERROR

Example 12	Sets the operating voltage as 10 to 17V.
Hint	Data of 10V is 100 (0x64) while 17V is 170 (0xAA). Instruction = WRITE_DATA, Address = 0x0C, DATA = 0x64, 0xAA
Communication	Instruction Packet : FF FF 01 05 03 0C 64 AA DC Status Packet : FF FF 01 02 00 FC
Status Packet Result	NO ERROR

Example 13	Only generates 50% of the maximum torque.
Hint	Sets the value of MAX Torque located in the EEPROM area as 0x1FF, which is 50% of the maximum value 0x3FF. Instruction = WRITE_DATA, Address = 0x0E, DATA = 0xff, 0x01
Communication	Instruction Packet: FF FF 01 05 03 0E FF 01 E8 Status Packet : FF FF 01 02 00 FC
Status Packet Result	NO ERROR
The change of Max Torque can be checked by turning the power off and then on.	

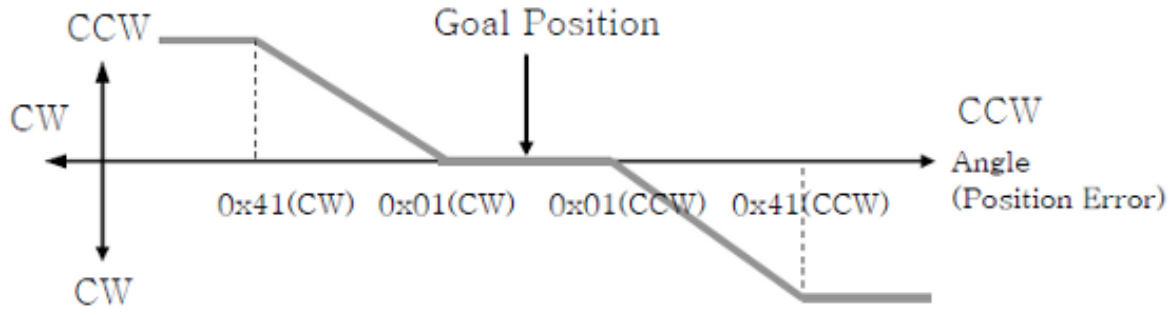
Example 15	Sets the Alarm as such that LED flickers and shutdown (torque off) when the operating temperature is higher than the limit temperature.
Hint	Since Overheating Error is Bit 2, set up Alarm value as 0x04. (0x04=00000100) Instruction = WRITE_DATA, Address = 0x11, DATA = 0x04, 0x04
Communication	Instruction Packet: FF FF 01 05 03 11 04 04 DD Status Packet : FF FF 01 02 00 FC
Status Packet Result	NO ERROR

Example 16	Turns on the LED and enables Torque.
Hint	Instruction = WRITE_DATA, Address = 0x18, DATA = 0x01, 0x01
Communication	Instruction Packet: FF FF 01 05 03 18 01 01 DC Status Packet : FF FF 01 02 00 FC
Status Packet Result	NO ERROR
You can check the Torque Enable state by touching the axis of Dynamixel you're your hand.	

Example 17	Locates at the Position 180° with the speed of 57RPM.
Hint	Sets Goal Position (Address 30 (0x1E))= 512 (0x200) and Moving Speed (Address 0x20))= 512 (0x200). Instruction = WRITE_DATA, Address = 0x1E, DATA = 0x00, 0x02, 0x00, 0x02
Communication	Instruction Packet: FF FF 01 07 03 1E 00 02 00 02 D2 Status Packet : FF FF 01 02 00 FC
Status Packet Result	NO ERROR

Example 18	Sets Compliance Margin=1 and Compliance Slope=0x40.

Hint The suggested condition can be depicted in a graph as below.



A: CCW Compliance Slope (Address 29 (0x1D)) = 0x40 (about 18.8°)
B: CCW Compliance Margin (Address 27 (0x1B)) = 0x01 (about 0.3°)
C: CW Compliance Margin (Address 26 (0x1A)) = 0x01 (about 0.3°)
D: CW Compliance Slope (Address 28 (0x1C)) = 0x40 (about 18.8°)

Instruction = WRITE_DATA, Address = 0x1A,
DATA = 0x01, 0x01, 0x40, 0x40

Communication Instruction Packet: FF FF 01 07 03 1A 01 01 40 40 58
Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR

Example 19	Sets the minimum output Torque (Punch) as 0x40.

Hint Instruction = WRITE_DATA, Address = 0x30,
DATA = 0x40, 0x00

Communication Instruction Packet : FF FF 01 05 03 30 40 00 86
Status Packet : FF FF 01 02 00 FC

Status Packet Result NO ERROR

Example 20	Locates RX-64 with ID 0 at Position 0° and RX-64 with ID 1 at Position 300°. Start only two RX-64s at the same point.
Hint	When the WRITE_DATA command is used, two RX-64s cannot be started at the same point. Thus, REG_WRITE and ACTION are used. ID=0, Instruction = REG_WRITE, Address = 0x1E, DATA = 0x00, 0x00 ID=1, Instruction = REG_WRITE, Address = 0x1E, DATA = 0xff, 0x03 ID=0xfe(Broadcasting ID), Instruction = ACTION,
Communication	Instruction Packet: FF FF 00 05 04 1E 00 00 D8 Status Packet : FF FF 00 02 00 FD Instruction Packet: FF FF 01 05 04 1E FF 03 D5 Status Packet : FF FF 01 02 00 FC Instruction Packet: FF FF FE 02 05 FA (LEN:006) Status Packet //No return packet
Status Packet Result	NO ERROR

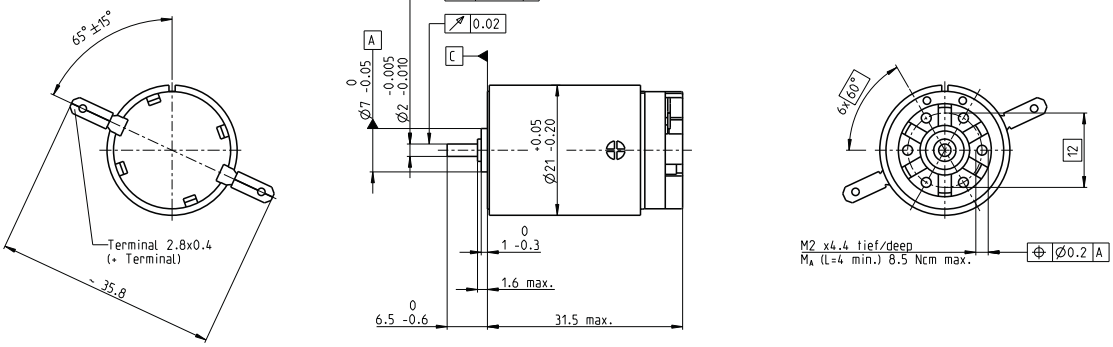
Example 21	Unable to change values except Address 24 to Address 35.
Hint	Sest Lock (Address 47 (0x2F)) as 1. Instruction = WRITE_DATA, Address = 0x2F, DATA = 0x01
Communication	Instruction Packet : FF FF 01 04 03 2F 01 C7 Status Packet : FF FF 01 02 00 FC
Status Packet Result	Status Packet Result NO ERROR

Once locked, It is impossible to unlock unless the power is off.

When other data is accessed while locked, an error is returned.

Características y parámetros físicos del motor
Maxon RE-max 21, modelo 250002

RE-max 21 ∅21 mm, Escobillas de grafito, 6 W



M 1:1

Programa Stock
Programa Estándar
Programa Especial (previo encargo)

Referencia

			250000	250001	250002	250003	250004	250005	250006	250007	250008
Datos del motor											
	Valores a tensión nominal										
1	Tensión nominal	V	4	6	9	15	18	21	24	36	48
2	Velocidad en vacío	rpm	11600	9730	10200	10600	9990	9770	8930	10100	9620
3	Corriente en vacío	mA	155	84.2	59.1	37.1	28.8	24.1	18.9	14.6	10.3
4	Velocidad nominal	rpm	11000	8500	8500	8410	7830	7560	6690	7830	7320
5	Par nominal (máx. par en continuo)	mNm	1.84	3.68	5.49	6.87	6.97	6.84	6.89	6.65	6.62
6	Corriente nominal (máx. corriente en continuo)	A	0.72	0.72	0.72	0.551	0.439	0.362	0.291	0.212	0.151
7	Par de arranque	mNm	43.9	29.9	33.7	34.1	32.7	30.6	27.9	30.2	28.2
8	Corriente de arranque	A	13.6	5.19	4.07	2.56	1.93	1.52	1.11	0.9	0.602
9	Máx. rendimiento	%	79	76	77	77	77	77	76	76	76
	Características										
10	Resistencia en bornes	Ω	0.295	1.16	2.21	5.86	9.32	13.8	21.7	40	79.7
11	Inductancia en bornes	mH	0.013	0.0411	0.0848	0.22	0.354	0.503	0.786	1.39	2.71
12	Constante de par	mNm/A	3.24	5.76	8.29	13.3	16.9	20.2	25.2	33.5	46.8
13	Constante de velocidad	rpm/V	2950	1660	1150	716	564	473	379	285	204
14	Relación velocidad/par	rpm/mNm	268	332	307	315	311	324	325	340	347
15	Constante de tiempo mecánica	ms	7.22	7.32	7.12	7.16	7.14	7.2	7.22	7.44	7.33
16	Inercia del rotor	gcm²	2.57	2.1	2.21	2.17	2.2	2.12	2.12	2.09	2.02

Especificaciones

Datos térmicos

17	Resistencia térmica carcasa/ambiente	28 K/W
18	Resistencia térmica bobinado/carcasa	8.0 K/W
19	Constante de tiempo térmica del bobinado	10.5 s
20	Constante de tiempo térmica del motor	501 s
21	Temperatura ambiente	-30…+85°C
22	Máx. temperatura del bobinado	+125°C

Datos mecánicos (cojinete sinterizado)

23	Máx. velocidad permitida	12000 rpm
24	Juego axial	0.05 - 0.15 mm
25	Juego radial	0.012 mm
26	Carga axial máx. (dinámica)	1 N
27	Máx. fuerza de empuje a presión (estática)	80 N
28	Carga radial máx. a 5 mm de la brida	2.7 N

Datos mecánicos (rodamiento a bolas)

23	Máx. velocidad permitida	12000 rpm
24	Juego axial	0.05 - 0.15 mm
25	Juego radial	0.025 mm
26	Carga axial máx. (dinámica)	3.3 N
27	Máx. fuerza de empuje a presión (estática)	45 N
28	Carga radial máx. a 5 mm de la brida	11.9 N

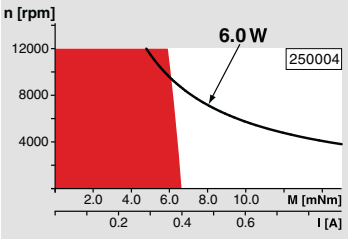
Otras especificaciones

29	Número de pares de polos	1
30	Número de delgas del colector	9
31	Peso del motor	42 g

Los datos de la tabla son valores nominales.
Explicación del diagrama en página 79.

Opción
Rodamientos a bolas en lugar de cojinetes sinterizados
Cablecillos en lugar de terminales para soldar

Rango de funcionamiento



Leyenda

Funcionamiento continuo
Teniendo en cuenta los valores de resistencia térmica antes mencionados (líneas 17 y 18). El rotor alcanzará la máxima temperatura durante funcionamiento continuo a 25°C de temperatura ambiente = límite térmico.

Funcionamiento intermitente
El motor puede ser sobrecargado durante cortos períodos (cíclicamente).

Potencia nominal asignada

Sistema Modular maxon

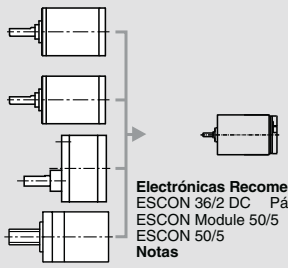
Esquema general en página 20–25

Reductor planetario
∅22 mm
0.5 - 1.0 Nm
Página 262

Reductor planetario
∅22 mm
0.5 - 2.0 Nm
Página 264

Reductor engranaje recto
∅38 mm
0.1 - 0.6 Nm
Página 282

Husillo
∅22 mm
Página 299/300



Electrónicas Recomendadas:

ESCON 36/2 DC	Página 342
ESCON Module 50/5	343
ESCON 50/5	344
Notas	**22**

Características del codificador angular AVAGO, modelo AEAT-6012

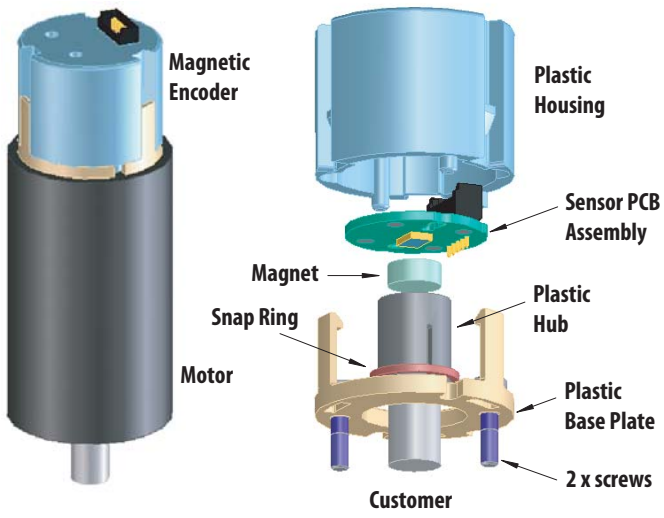
AEAT-6010/6012 Magnetic Encoder
10 or 12 bit Angular Detection Device

Data Sheet

Description

Avago Technologies' AEAT-60xx series of magnetic encoders provides an integrated solution for angular detection. With ease of use in mind, these magnetic encoders are ideal for angular detection within 360°. Based on magnetic technologies, the device is non-contact and ensures reliable operations. It is able to provide absolute angle detection upon power-up, with a resolution of 0.0879°(12 bits version) or 0.35°(10bits version), which is equivalent to 4096 and 1024 positions per revolution respectively. The positional data is provided in serial bit stream. There is no upper speed limit; the only restriction is that there will be fewer samples per revolution as the speed increases.

Exploded View



Features

- 10 or 12 bits resolution
- Contactless sensing technologies
- Wide temperature range from -40° to 125°C
- Absolute angular position detection
- Synchronous serial interface (SSI) output for absolute position data (binary format)
- Code monotony error = ± 1 LSB
- 5V supply
- Easy Assembly, No Signal Adjustment required
- RoHS compliant

Applications

- Flow meter
- Angular detection
- Knob control
- Rotary encoder

Device Selection Guide [1]

Part Number	Resolution (bit)	Operating Temperature (°C)	Output Communication	DC Supply Voltage (V), V_{DD}
AEAT-6012-A06	12	-40 to +125	Serial	+5.0
AEAT-6010-A06	10	-40 to +125	Serial	+5.0

Notes:
1. For other options of Magnetic Encoder, please refer to factory.

Table 1. Absolute Maximum Ratings [2, 3]

Parameter	Symbol	Limits	Units	Notes
DC Supply Voltage at pin V_{DD} = 5V	V_{DD}	-0.3 to + 7	V	
Input Voltage	V_i	-0.3 to VDD+0.3	V	
Storage Temperature	T_{STG}	-40 to 125	°C	

Notes:
2. Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.
3. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

Table 2. Recommended Operating Condition

Parameter	Symbol	Values	Units	Notes
DC Supply Voltage at pin V_{DD} = 5V	V_{DD}	+4.5 / +5.5	V	
Ambient Temperature	T_{amb}	–40 to +125	°C	
Maximum Read-out Frequency	f_{CLK}	≤1	MHz	>0 MHz

Table 3. DC Characteristics

DC Characteristics over Recommended Operating Range, typical at 25 °C

			Values				
Parameter	Symbol	Condition	Min	Typ.	Max	Units	Notes
VDD Supply Current	I_{DD}			16	20	mA	
Output High Voltage D0	V_{OH}		V_{DD}-0.5			V	
Output Low Voltage D0	V_{OL}				V_{SS}+0.4	V	
Output Current D0	I_O				4	mA	V_{DD} pin = 4.5V
Input High Voltage CLK, CSn	V_{IH}		0.7*V_{DD}				4
Input Low Voltage CLK, CSn	V_{IL}				0.3*V_{DD}		

Note:
4. CSn is internal pull-up.

Package Dimensions

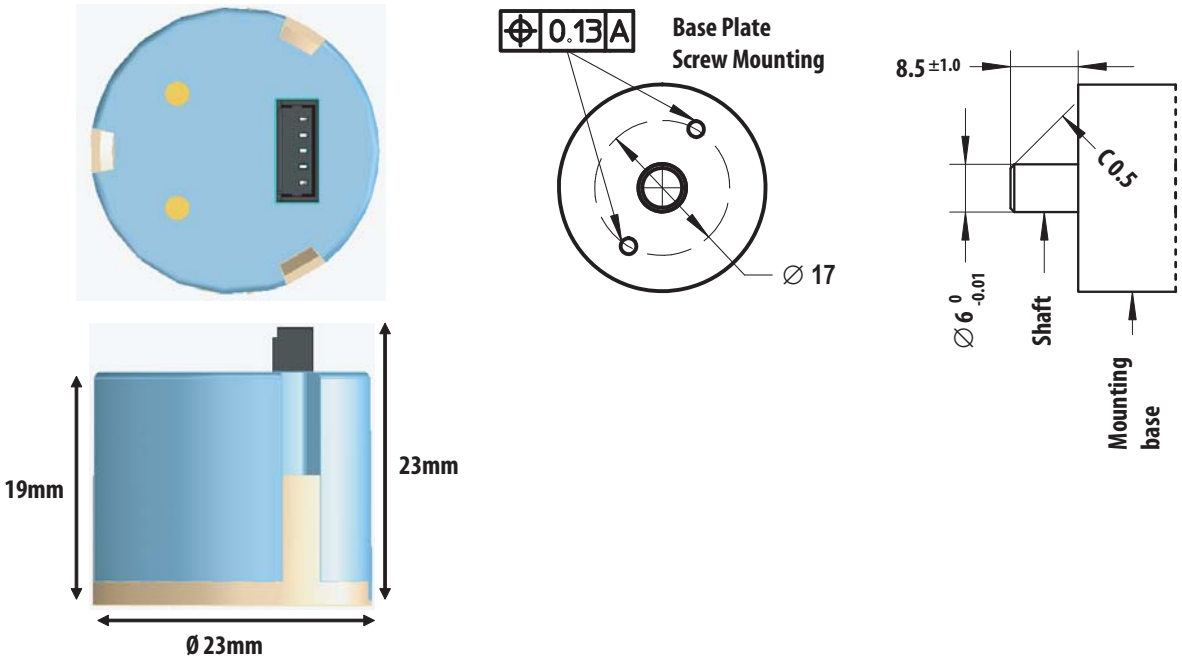


Figure 1. Package and recommended mounting dimension

Parameters

No.	Parameter	Value
1	Operating Temp(˚C)	- 40 to +125
2	Shaft axial play (mm)	± 0.08
3	Shaft TIR (mm)	0.05
4	Mechanical speed (rpm)	12,000
5	Shaft diameter (mm)	6 + 0 / -0.01
6	Moment inertia (g-cm^2)	0.104
7	Shaft length – (mm)	8.5 ±1.0
8	Mounting screw size (mm)	M2 x 0.4 x 8 (socket head cap screw, head Ø3.8 ± 0.18 mm)
9	Recommended screw torque	0.6 lb.inch
10	Encoder base plate thickness (mm)	2
11	Bolt circle	± 0.13

* Note:- For high temperature application, it is highly recommended that adhesive be applied at least to the screw and the base plate interface. Refer Application Note for further details.

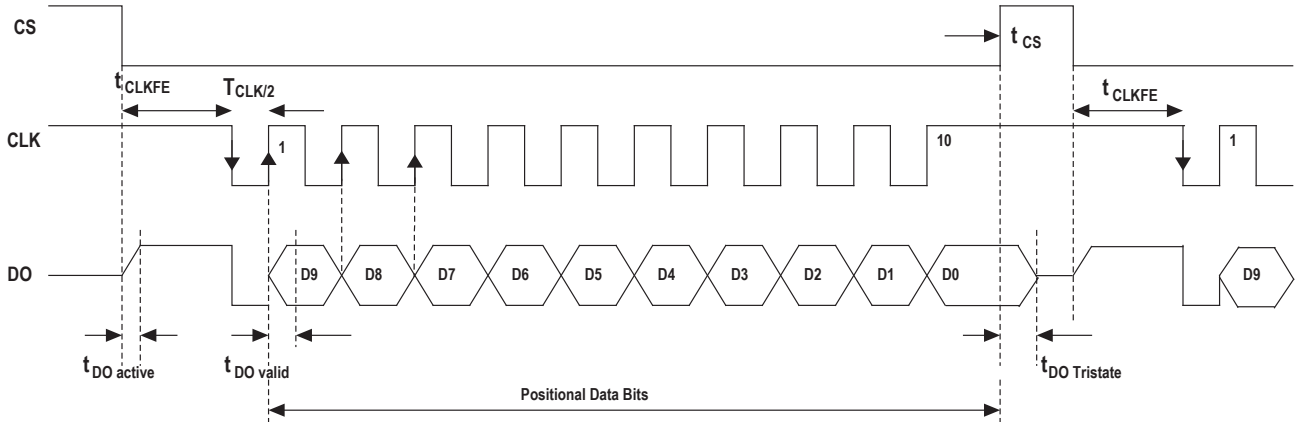
Table 4. Timing Characteristics

Timing Characteristics over Recommended Operating Range, typical at 25 °C

			Values				
Parameter	**Symbol**	**Condition**	**Min**	**Typ.**	**Max**	**Units**	**Notes**
Data output activated (logic high)	$T_{DO\ active}$				100	ns	1
First data shifted to output register	$t_{CLK\ FE}$		500			ns	2
Start of data output	$T_{CLK/2}$		500			ns	3
Data output valid	$T_{DO\ valid}$				375	ns	4
Data output tristate	$T_{DO\ tristate}$				100	ns	5
Pulse width of CSn	T_{CSn}		500			ns	6
Sampling rate for absolute output	f_{abs}		9.9	10.42	10.94	kHz	7
Power-up time	t_{CF}						8
10-bit version			-	-	50	ms	
12-bit version			-	-	20	ms	

Notes:
1. Time between falling edge of CSn and data output activated
2. Time between falling edge of CSn and first falling edge of CLK
3. Rising edge of CLK shifts out one bit a time
4. Time between rising edge of CLK and data output valid
5. After the last bit DO changes back to "tristate"
6. CSn=high; To initiate read-out of next angular position
7. Internal sampling rate.
8. Until internal compensation finished

Timing Characteristics



Notes:
1. Please refer to Table 4 for Timing Characteristics.
2. For 12 bits version; the Positional Data Bits will start with D11 instead and end at D0.

Figure 2. Timing Diagram for 10 bit Magnetic Encoder

Table 5. Linearity

Parameter	Symbol	Min.	Typ.	Max	Units	Notes
Integral Non-Linearity	INL	-	± 0.8 [1]	± 2.4 [2]	Deg.	
Differential Non-Linearity						
10-bit version	DNL	-	-	± 0.176	Deg.	No missing codes
12-bit version		-	-	± 0.044	Deg.	No missing codes

Notes:
1. Average value at typical operating and mounting conditions.
2. Maximum value over recommended operating range and over radial & axial mounting tolerances.

Linearity Definitions

Integral non-linearity

Integral non-linearity (INL) is the maximum deviation between actual angular position and the position indicated by the encoder's output count, over one revolution. It is defined as the most positive linearity error +INL or the most negative linearity error –INL from the best fit line, whichever is larger.

Differential non-linearity

Differential non-linearity (DNL) is the maximum deviation of the step length from one position to the next.

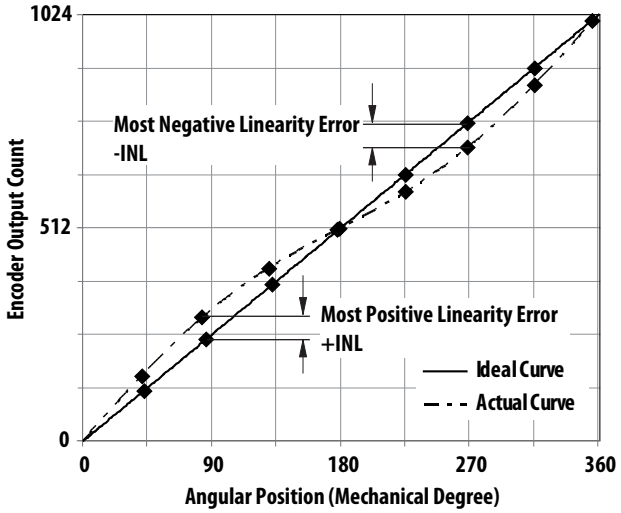


Figure 3. Integral non-linearity

Table 6. Environmental Specifications

Parameter	Reference Standard	Test Conditions	Level
Electromagnetic Compatibility (EMC) [1]			
Electrostatic discharge (ESD) immunity	IEC/EN 61000-4-2	8kV	
Power frequency magnetic field immunity	IEC/EN 61000-4-8	30 A/m (continuous field)	
		300 A/m (short duration field)	Level 4
Pulse magnetic field immunity	IEC/EN 61000-4-97	1000 A/m	Level 5
Damped oscillatory magnetic field immunity	IEC/EN 61000-4-10	100 A/m	Level 5
Mechanical Durability			
Vibration (Operating)	IEC/EN 60068-2-6	10-500Hz at 5G	
Shock	IEC/EN 60068-2-27	6ms at 200G	

Notes:
1. Suitable for applications in Industrial Environment Class 4.

Electrical Connections

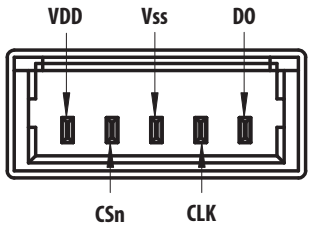


Figure 4. Electrical Connections

Pin	Symbol	Description
1	VDD	5V Supply Voltage
2	CSn	Chip Select – Input (See Figure 2)
3	VSS	Supply Ground
4	CLK	Serial Clock - Input (See Figure 2)
5	DO	Serial Data - Output. (See Figure 2)

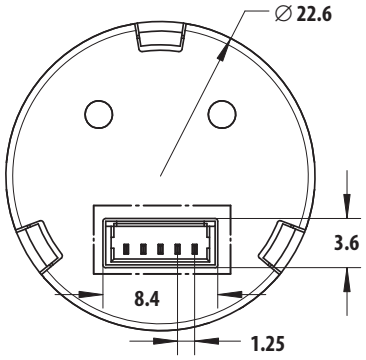
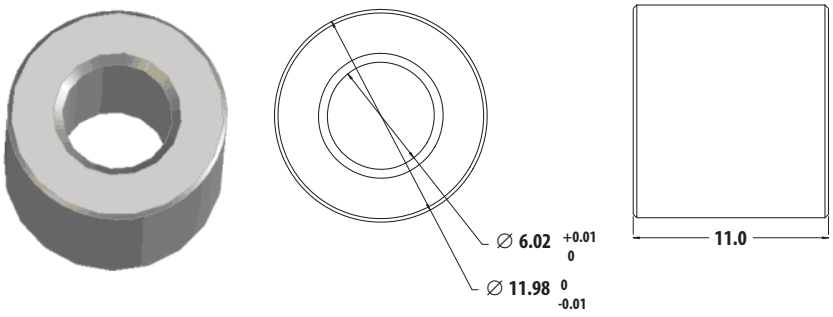


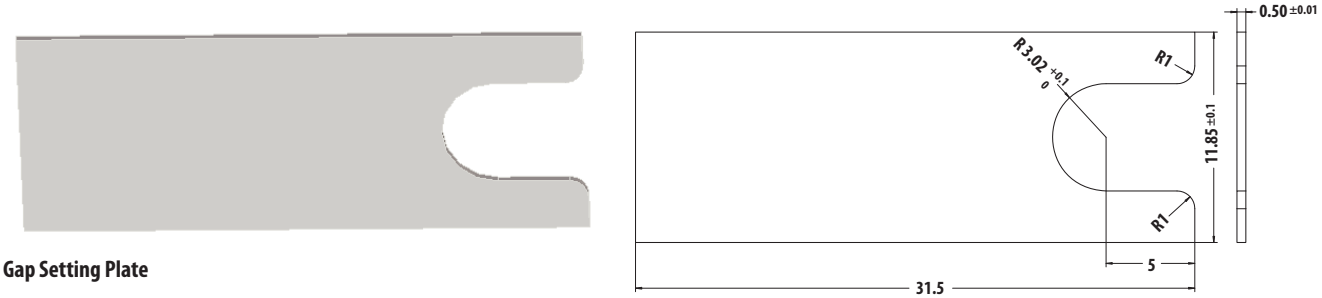
Figure 5. Basic connector dimensions

Alignment Tool Set - Part number HEDS-8934

This optional alignment tool set consists of a gap setting plate and a centering jig. Refer to Application Note 5317 for the assembly guide.



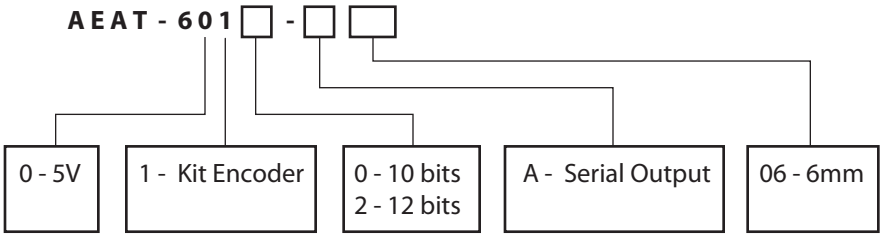
Centering Jig



Gap Setting Plate

Figure 6. Alignment tool set and recommended dimensions

Ordering Information



For product information and a complete list of distributors, please go to our web site: **www.avagotech.com**

AV02-0188EN - August 12, 2011

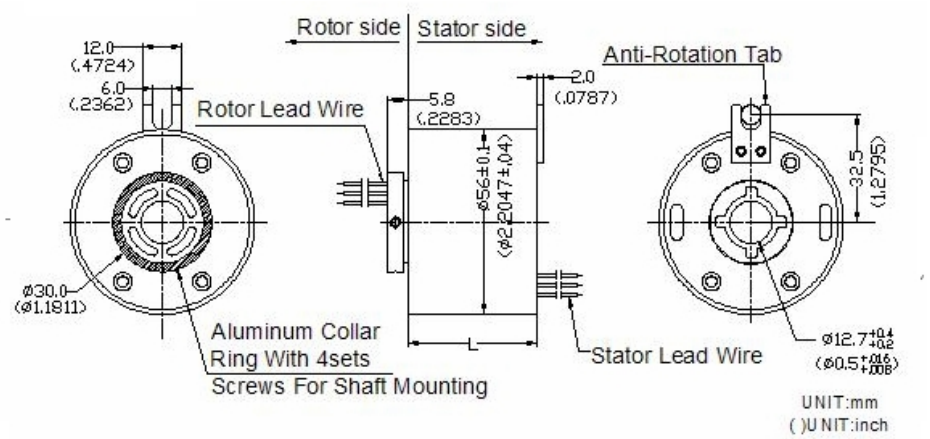
Características del conector rotativo BTH1256
de la marca ByTune Electronics

ByTune Electronics Co.,Ltd

Through hole slip ring---BTH1256

Picture

Drawing



Parameters:

Mechanical :

Working speed	1000rpm
Operating temperature	-40℃~ +100℃
Contact material	Precious metal
IP Grade	IP 54

Electrical:

Circuit	6 ,12,18,24
Current	5A,10A ,15A,20A or combined rings to get higher current
Voltage	≤380V,600V optional
Dielectric strength	≥1000V@50Hz between each circuit
Insulation resistance	1000MΩ@ 500VDC
Electric noise	≤7mΩ
Lead wire size	22AWG, 17AWG,UL, color Teflon, silver or tin plated
Lead wire length	200mm

Critical Dimension:

Model	Length "L"				
	Current	Formula (A means No. of ring)	6 circuits	12 circuits	18circuits
BTH1256	2A/5A	20+2.3×A	33.8mm	47.6mm	61.4mm
	10A	20+2.8×A	37.8mm	53.6mm	70.4mm
	15A/20A	20+3.8×A	42.8mm	65.6mm	88.4mm

Customization is available, OEM&ODM is welcome!

Further information，please feel free to contact our sales.

Características de la memoria externa SPI SRAM de 512 kbits, modelo 23LCV512, de Microchip Technology Inc.

Sólo se incluyen las características principales.

El manual completo se puede descargar de http://goo.gl/0azS2F.

23LCV512

512 Kbit SPI Serial SRAM with Battery Backup and SDI Interface

Device Selection Table

Part Number	Vcc Range	Dual I/O (SDI)	Battery Backup	Max. Clock Frequency	Packages
23LCV512	2.5-5.5V	Yes	Yes	20 MHz	SN, ST, P

Features:

- SPI-Compatible Bus Interface:
  - 20 MHz Clock rate
  - SPI/SDI mode
- Low-Power CMOS Technology:
  - Read Current: 3 mA at 5.5V, 20 MHz
  - Standby Current: 4 μA at +85°C
- Unlimited Read and Write Cycles
- External Battery Backup support
- Zero Write Time
- 64K x 8-bit Organization:
  - 32-byte page
- Byte, Page and Sequential mode for Reads and Writes
- High Reliability
- Temperature Range Supported:
  - Industrial (I): -40°C to +85°C
- Pb-Free and RoHS Compliant, Halogen Free.
- 8-Lead SOIC, TSSOP and PDIP Packages

Pin Function Table

Name	Function
$\overline{CS}$	Chip Select Input
SO/SIO1	Serial Output/SDI pin
Vss	Ground
SI/SIO0	Serial Input/SDI pin
SCK	Serial Clock
VBAT	External Backup Supply Input
Vcc	Power Supply

Description:

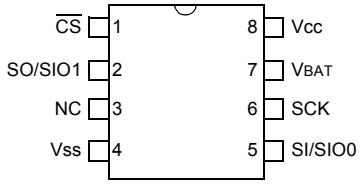
The Microchip Technology Inc. 23LCV512 is a 512 Kbit Serial SRAM device. The memory is accessed via a simple Serial Peripheral Interface (SPI) compatible serial bus. The bus signals required are a clock input (SCK) plus separate data in (SI) and data out (SO) lines. Access to the device is controlled through a Chip Select ($\overline{CS}$) input. Additionally, SDI (Serial Dual Interface) is supported if your application needs faster data rates.

This device also supports unlimited reads and writes to the memory array, and supports data backup via external battery/coin cell connected to VBAT (pin 7).

The 23LCV512 is available in standard packages including 8-lead SOIC, PDIP and advanced 8-lead TSSOP.

Package Types (not to scale)

SOIC/TSSOP/PDIP



2.0 FUNCTIONAL DESCRIPTION

2.1 Principles of Operation

The 23LCV512 is an 512 Kbit Serial SRAM designed to interface directly with the Serial Peripheral Interface (SPI) port of many of today's popular microcontroller families, including Microchip's PIC® microcontrollers. It may also interface with microcontrollers that do not have a built-in SPI port by using discrete I/O lines programmed properly in firmware to match the SPI protocol. In addition, the 23LCV512 is also capable of operating in SDI (or dual SPI) mode.

The 23LCV512 contains an 8-bit instruction register. The device is accessed via the SI pin, with data being clocked in on the rising edge of SCK. The $\overline{CS}$ pin must be low for the entire operation.

Table 2-1 contains a list of the possible instruction bytes and format for device operation. All instructions, addresses and data are transferred MSB first, LSB last.

2.2 Modes of Operation

The 23LCV512 has three modes of operation that are selected by setting bits 7 and 6 in the MODE register. The modes of operation are Byte, Page and Burst.

Byte Operation – is selected when bits 7 and 6 in the MODE register are set to `00`. In this mode, the read/write operations are limited to only one byte. The command followed by the 16-bit address is clocked into the device and the data to/from the device is transferred on the next eight clocks (Figure 2-1, Figure 2-2).

Page Operation – is selected when bits 7 and 6 in the MODE register are set to `10`. The 23LCV512 has 2048 pages of 32 bytes. In this mode, the read and write operations are limited to within the addressed page (the address is automatically incremented internally). If the data being read or written reaches the page boundary, then the internal address counter will increment to the start of the page (Figure 2-3, Figure 2-4).

Sequential Operation – is selected when bits 7 and 6 in the MODE register are set to `01`. Sequential operation allows the entire array to be written to and read from. The internal address counter is automatically incremented and page boundaries are ignored. When the internal address counter reaches the end of the array, the address counter will roll over to `0x0000` (Figure 2-5, Figure 2-6).

2.3 Read Sequence

The device is selected by pulling $\overline{CS}$ low. The 8-bit `READ` instruction is transmitted to the 23LCV512 followed by the 16-bit address. After the correct `READ` instruction and address are sent, the data stored in the memory at the selected address is shifted out on the SO pin.

If operating in Sequential mode, the data stored in the memory at the next address can be read sequentially by continuing to provide clock pulses. The internal Address Pointer is automatically incremented to the next higher address after each byte of data is shifted out. When the highest address is reached (FFFFh), the address counter rolls over to address 0000h, allowing the read cycle to be continued indefinitely. The read operation is terminated by raising the $\overline{CS}$ pin.

2.4 Write Sequence

Prior to any attempt to write data to the 23LCV512, the device must be selected by bringing $\overline{CS}$ low.

Once the device is selected, the Write command can be started by issuing a `WRITE` instruction, followed by the 16-bit address, and then the data to be written. A write is terminated by the $\overline{CS}$ being brought high.

If operating in Page mode, after the initial data byte is shifted in, additional bytes can be shifted into the device. The Address Pointer is automatically incremented. This operation can continue for the entire page (32 bytes) before data will start to be overwritten.

If operating in Sequential mode, after the initial data byte is shifted in, additional bytes can be clocked into the device. The internal Address Pointer is automatically incremented. When the Address Pointer reaches the highest address (FFFFh), the address counter rolls over to (0000h). This allows the operation to continue indefinitely, however, previous data will be overwritten.

3.0 PIN DESCRIPTIONS

The descriptions of the pins are listed in Table 3-1.

TABLE 3-1: PIN FUNCTION TABLE

Name	SOIC/ PDIP TSSOP	Function
$\overline{CS}$	1	Chip Select Input
SO/SIO1	2	Serial Data Output/SDI Pin
NC	3	No Connect
VSS	4	Ground
SI/SIO0	5	Serial Data Input/SDI Pin
SCK	6	Serial Clock Input
VBAT	7	External Backup Supply
VCC	8	Power Supply

3.1 Chip Select ($\overline{CS}$)

A low level on this pin selects the device. A high level deselects the device and forces it into Standby mode. When the device is deselected, SO goes to the high-impedance state, allowing multiple parts to share the same SPI bus. After power-up, a low level on $\overline{CS}$ is required, prior to any sequence being initiated.

3.2 Serial Output (SO)

The SO pin is used to transfer data out of the 23LCV512. During a read cycle, data is shifted out on this pin after the falling edge of the serial clock.

3.3 Serial Input (SI)

The SI pin is used to transfer data into the device. It receives instructions, addresses, and data. Data is latched on the rising edge of the serial clock.

3.4 Serial Dual Interface Pins(SIO0, SIO1)

The SIO0 and SIO1 pins are used for SDI mode of operation. Functionality of these I/O pins is shared with SO and SI.

3.5 Serial Clock (SCK)

The SCK is used to synchronize the communication between a master and the 23LCV512. Instructions, addresses or data present on the SI pin are latched on the rising edge of the clock input, while data on the SO pin is updated after the falling edge of the clock input.

3.6 VBAT supply Input

The VBAT pin is used as an input for external backup supply to maintain SRAM data when VCC is below the VTRIP point. If the VBAT function is not being used, it is recommended to connect this pin to VSS.

3.7 SPI and SDI Pin Designations