ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

Diseño e Implementación de un Control Fraccionario Predictivo por Modelo para el Seguimiento de Rutas de un Robot Móvil

PROYECTO DE TITULACIÓN

Previo la obtención del Título de:

Magister en Automatización y Control Industrial

Presentado por:
Luis Alexander Solórzano Vera

GUAYAQUIL - ECUADOR
Año: 2023

DEDICATORIA

Dedico esta tesis a mis amados padres, motores e inspiración para seguir adelante en los momentos de adversidad y tribulación, entrego el resultado final de este trabajo y comparto con ellos la felicidad que me embarga. Mi intención es que esta dedicatoria sea un homenaje en vida a los seres más increíbles que Dios me ha dado, nada podrá compensar todo lo que ustedes han sembrado en mí: disciplina, perseverancia, constancia, carácter y honestidad.

Espero que el fruto de su arduo trabajo sea de su total regocijo y complacencia.

AGRADECIMIENTOS

Deseo agradecer con toda mi alma y corazón a mi Padre celestial, al creador de los cielos y la tierra quien ha estado conmigo siempre.

Quiero agradecer a mis amados padres Luis y Alexandra quienes me han brindado sus valiosos consejos y su apoyo incondicional para poder culminar mi trabajo de tesis. Nunca podre recompensarles todo lo que han hecho por mí.

También tengo que reconocer a Giuseppe y Sandrelly por ser unos hermanos ejemplares y en todo momento estar pendiente de mí.

De igual manera mi gratitud a todos mis profesores y compañeros de la maestría ya que compartimos momentos que siempre quedaran grabados en mi mente, un reconocimiento especial a mi tutor el profesor Ricardo Cajo gracias a su dirección y guía dedique mi tiempo a la realización de este trabajo.

Estaré eternamente agradecido con la Escuela Politécnica del Litoral por permitirme formar parte de la Cohorte VIII de la maestría en automatización y control.

Agradezco a mis colegas que aportaron con conceptualizaciones y modelos, en especial al profesor Marcelo que fue de gran ayuda para dar colores y matices a esta tesis.

Por último, quiero dejar inscrito en esta página a alguien muy especial en mi vida, destacar el apoyo de mi alma gemela Andrea gracias a ti, por tus oraciones, por estar pendiente de mí y por simplemente estar ahí conmigo cuando más lo necesitaba.

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, me corresponden conforme al reglamento de propiedad intelectual de la institución; Luis Alexander Solórzano Vera doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Firmado electrónicamente por:
LUIS ALEXANDER
SOLORZANO VERA

Luis Alexander Solórzano Vera

COMITÉ EVALUADOR

Firmado electrónicamente por:
RICARDO ALFREDO
CAJO DIAZ

Ricardo Cajo Díaz, Ph.D.

PROFESOR TUTOR

Firmado electrónicamente por:
DENNYS FABIAN
PAILLACHO CHILUIZA

Dennys Paillacho, Ph.D.

PROFESOR EVALUADOR

RESUMEN

Este trabajo analiza el uso de un modelo de orden fraccional para diseñar una estrategia de control predictivo de modelo (MPC) para un robot Khepera IV. El robot Khepera IV es un robot móvil comercial que se utiliza en diversas aplicaciones, como exploración, vigilancia y educación. El modelo de orden fraccional captura la dinámica del robot Khepera IV con mayor precisión que los modelos convencionales de orden entero, lo que permite un control más preciso del movimiento del robot.

El diseño del controlador MPC implica varios pasos, incluida la formulación de la función de coste, la selección del horizonte de predicción y el ajuste de los parámetros de control. La función de coste penaliza las desviaciones de la trayectoria deseada considerando las restricciones del movimiento del robot. El horizonte de predicción determina el número de pasos de tiempo futuros en el problema de optimización, equilibrando el equilibrio entre la eficiencia computacional y el rendimiento del control. Los parámetros de control se ajustan para optimizar el rendimiento del controlador en diferentes condiciones de funcionamiento.

El modelo de orden fraccional se construye utilizando la caja de herramientas FOMCON en MATLAB, la cual permite modelar y simular sistemas de orden fraccional utilizando una interfaz de fácil uso. El controlador MPC se implementa desarrollando una función de costo con un operador de orden fraccional para minimizar la diferencia entre la trayectoria deseada y la predicción.

El rendimiento del controlador MPC se evalúa a través de estudios de simulación, donde el robot sigue una trayectoria predefinida mientras evita obstáculos. Los resultados muestran que el controlador MPC que usa el modelo de orden fraccional supera a los controladores convencionales de orden entero, logrando un control de movimiento más suave y preciso. El enfoque propuesto proporciona un marco prometedor para el diseño de controladores MPC para robots móviles, con aplicaciones potenciales en robótica, automatización e ingeniería de control.

Palabras Clave: Modelo de Orden Fraccional, Modelo de Control Predictivo, Robots Móviles, Sistemas de Control.

ABSTRACT

This work discusses using a fractional order model to design a Model Predictive Control (MPC) strategy for a commercial robotic platform named Khepera IV. This robot is used in several applications: surveillance, exploration, and education. A fractional-order model captures the dynamics of the Khepera IV robot more accurately than conventional integer-order models, enabling more precise control of the robot's motion.

The MPC controller's design encompasses various phases, such as defining the cost function, choosing the prediction horizon, and fine-tuning the control parameters. Considering the robot's motion limits, the cost function imposes penalties for straying from the target trajectory. The prediction horizon sets the count of upcoming time steps for optimisation, weighing the balance between computational speed and control quality. The control parameters are modified to optimise the controller functions across diverse operational scenarios.

The fractional order model is built using the FOMCON toolbox in MATLAB, offering an intuitive platform for modelling and simulating fractional order systems. The MPC controller is realised by crafting a cost function incorporating a fractional order operator. This aims to reduce the discrepancy between the targeted and forecasted trajectory.

The efficacy of the MPC controller is assessed via simulation tests, where the robot's objective is to adhere to a set trajectory and dodge obstacles. The findings indicate that when utilising the fractional order model, the MPC controller surpasses the traditional integer-order controllers, delivering a more seamless and precise movement control. This suggested method offers an encouraging blueprint for crafting MPC controllers for mobile robots, holding potential use in fields like robotics, automation, and control engineering.

Keywords: Fractional Order Model, Model Predictive Control, Mobile Robots, Control Systems.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

1. Introducción

En años recientes, los modelos de orden fraccional han ganado popularidad debido a su habilidad para representar sistemas del mundo real con más exactitud que los modelos de orden entero. Los beneficios de este cálculo fraccionario han sido destacados en diversas áreas, tales como la reología, la mecánica, la química, la física, la bioingeniería y la robótica [1]. Naturalmente, los controladores de orden fraccional son adecuados para estos modelos de orden fraccional [2].

Dentro de estos controladores, el Control Predictivo por Modelo Fraccionario (FOMPC) representa una perspectiva alentadora para el control óptimo de sistemas con orden fraccional [3]. El FOMPC integra dinámicas de orden fraccional en el control, manifestándose como memoria a partir de las características del operador de orden fraccionario, lo que ofrece mayores grados de libertad [4]. El uso del cálculo de orden fraccionario en el método de estabilidad combinado del MIT (Massachussets Institute of Technology) y Lyapunov también ha sido propuesto para sistemas de control [5].

El uso de controladores de orden fraccional en el modelado y diseño de control ha motivado un interés renovado en varias aplicaciones de control de orden fraccional [6]. En general, los modelos y controladores de orden fraccional han mostrado un gran potencial en varias aplicaciones, y su uso en el control predictivo de modelos es un área de investigación prometedora. Una de las ventajas destacadas de los sistemas de control de orden fraccional es su robustez y un rendimiento de control más elevado en contraste con los sistemas de control clásicos de orden entero [7].

1.1 Planteamiento del problema

El rastreo de trayectorias es un desafío significativo en el ámbito de la robótica móvil, dado que es fundamental para asegurar un movimiento seguro y eficaz del robot en su ambiente. Una estrategia alentadora para enfrentar este reto es la aplicación del Control Fraccionario Predictivo por Modelo, que posibilita una representación más exacta de la dinámica compleja del sistema, optimizando el rendimiento del control en contraposición a los modelos convencionales.

Sin embargo, el uso de modelos de orden fraccional en el control predictivo todavía presenta desafíos importantes, como la selección adecuada de parámetros y la implementación en tiempo real en un entorno dinámico. Así pues, es necesario realizar más investigaciones para diseñar y analizar un control predictivo eficaz fundamentado en modelos de orden fraccional destinado al rastreo de trayectorias de un robot móvil en un entorno dinámico.

El presente trabajo utiliza el robot móvil Khepera IV que posee dos entradas y dos salidas, clasificándose dentro de los sistemas del tipo MIMO cuyo campo de acción se enfoca en la velocidad lineal (v) y la velocidad angular (ω) tomando en cuenta el punto de destino y la posición presente del robot. Para lo cual se toma el modelo del robot en mención para controlar sus variables y lograr seguir la ruta trazada minimizando el error.

La flexibilidad, tamaño compacto, sensores avanzados y bajo coste del Khepera IV permite experimentar con diferentes técnicas de control y desarrollar prototipos de sistemas de control predictivo del modelo de orden fraccional de una manera accesible y asequible.

1.2 Delimitación del problema

En la presente investigación se desarrolla un controlador FOMPC Fractional Order Model Predictive Control para cumplir una ruta trazada comprobando su robustez lo cual generara una valiosa información para quienes estén relacionados en el área de control automático.

La investigación del FOMPC en el robot Khepera IV toma en cuenta aspectos como la elaboración del modelo matemático del robot, la puesta en marcha del controlador, la contraposición con otras técnicas de control y la mejora del control predictivo basado en el modelo de orden fraccional para el rastreo de trayectorias.

El desarrollo del modelo matemático del Khepera IV es utilizado en la implementación del Control Predictivo por Modelo Fraccionario, es necesario tener un modelo matemático preciso del robot. El trabajo se enfoca en el desarrollo de este modelo matemático y en la validación de este mediante experimentos en el robot utilizando software de simulación.

La implementación del FOMPC contempla seleccionar parámetros adecuados en el controlador y evaluar el rendimiento del controlador en diferentes condiciones. Se incluye la comparación del FOMPC con otros métodos de control utilizados en robótica, como el control proporcional-integral-derivativo (PID), para evaluar la eficacia del método en comparación con otras técnicas.

Una vez implementado el sistema se evalúa el rendimiento del controlador en términos de precisión en el seguimiento de la ruta y capacidad de recuperación ante obstáculos o perturbaciones. Finalmente se analiza el efecto del orden fraccional en el controlador, su rendimiento y como afecta al seguimiento de la ruta.

1.3 Objetivos

1.3.1 Objetivo General

Diseñar un controlador avanzado para el control de la velocidad lineal (v) y la velocidad angular (ω) del robot Khepera IV para el seguimiento de ruta trazada.

1.3.2 Objetivos Específicos

1. Modelar el robot Khepera IV en MATLAB/Simulink.
2. Extender el control predictivo por modelo utilizando la teoría del control fraccionario.
3. Establecer la ruta de evaluación pare el robot Khepera IV.
4. Implementar el controlador FOMPC en la plataforma CoppeliaSim para el Robot Khepera IV.
5. Evaluar el desempeño del controlador FOMPC en el robot Khepera IV.

1.4 Metodología

1.4.1 Métodos

La investigación realizada es teórica-deductiva-indirecta para lo cual utilizaremos el método científico. Se parte del algoritmo de control FOMPC que es conocido como un método de control robusto. A continuación, desarrollamos el modelo en teoría robusto y simulamos la trayectoria de recorrido del cual debe traducirse mínimas desviaciones. De esta forma por medio de pruebas utilizando software determinaremos la efectividad del sistema. También se utiliza el método teórico para demostrar a partir de una hipótesis,

por lo que se emplean principios, leyes y teorías para obtener un modelo matemático del sistema que se tiene como objetivo controlar. Se llevarán a cabo simulaciones en contextos dinámicos para medir el desempeño del controlador sugerido, y se efectuarán pruebas en un ambiente simulado para el robot móvil, lo cual facilitará la validación de los resultados alcanzados. Se utilizarán técnicas de análisis matemático y de sistemas de control para desarrollar y optimizar el modelo de orden fraccional y el controlador predictivo. Además, se compararán los resultados obtenidos con los enfoques de control tradicionales para analizar la eficacia y la eficiencia del enfoque propuesto.

CAPÍTULO 2

2. Análisis de herramientas y conocimientos disponibles

Mario Cimoli afirma que, “para construir una globalización más igualitaria, la cooperación en ciencia, tecnología e innovación debe ser efectiva e ir de la mano con los acuerdos de comercio e inversiones y la apertura de los tratados de propiedad intelectual” [8].

Para el desarrollo de la programación del robot Khepera IV, el presente trabajo se basará en estudios de implementaciones realizadas en programaciones anteriores; simulación y resultados experimentales de una nueva estrategia de control para la estabilización puntual de robots móviles no holonómicos y ajuste MPC práctico para aplicaciones industriales.

2.1 Estado del Arte

En los últimos años, los modelos de orden fraccional han capturado un interés notable debido a su habilidad para representar diversos materiales reales con mayor precisión que los modelos de orden entero [9]. Para estos modelos son adecuados los controladores de orden fraccional [9], [10]. Munoz-Vazquez y Parra-Vega diseñaron un controlador PID de orden fraccional para controlar la posición angular de la barra de nivel de un sistema aerodinámico de dos rotores. Los parámetros del controlador se ajustan mediante la optimización de Nelder-Mead, concluyendo que la simulación del modelo no lineal muestra una mejora significativa en el rendimiento del controlador PID de orden fraccional en comparación con un controlador PID clásico [11]. Petras propone un esquema de control absolutamente continuo de orden fraccionario para sistemas de Euler-Lagrange, que impone en tiempo finito un régimen de orden fraccionario racional conmensurable. Además, se lleva a cabo un análisis en el ámbito frecuencial, que resulta muy beneficioso para ciertas aplicaciones [12]. Zhou et al. presentan un enfoque novedoso para el control predictivo del modelo de orden fraccionario en el espacio de estados, que proporciona más grados de libertad e incorpora dinámicas de orden fraccionario en el control en forma de memoria debido a la propiedad del operador de orden fraccionario [13]. Chen et al. proponen un controlador de modo deslizante fraccional que combina el cálculo de orden fraccional y el método de control de modo

deslizante jerárquico para el control de un robot esférico bajo saturación de entrada. Las ventajas del controlador propuesto se ilustran comparando los resultados de simulación de los controladores de modo deslizante de orden fraccionario y el controlador de orden entero, para esto se propone un control lateral para un sistema de estacionamiento autónomo con controlador de orden fraccional. La idea de utilizar controladores de orden fraccionario para sistemas dinámicos proviene de Oustaloup, quien desarrolló el llamado controlador CRONE [14]. Gao se presenta un método analítico sobre la estabilización de plantas de orden fraccionario con un término de orden fraccionario e incertidumbres de intervalo utilizando controladores PID de orden fraccionario [15]. Finalmente, Zhao et al. se diseña un sistema de control de orden fraccional basado en el método de fase plana para un servosistema práctico, y los resultados de la simulación muestran que el controlador de orden fraccional tiene mejor robustez y mayor precisión de control en comparación con un PID de orden entero [16].

2.2 Control de procesos

El campo de control, particularmente en lo que se refiere a la supervisión de las acciones de un robot móvil autónomo, ha visto un aumento notable en la investigación. Se han creado algoritmos más fuertes y sofisticados. La capacidad de un robot móvil para operar de forma independiente se basa en gran parte en su sistema de navegación automática, que incluye responsabilidades en áreas como la planificación, percepción y control [17].

El reto de planificar trayectorias de forma global se centra en trazar una ruta óptima para alcanzar un fin, conduciendo el diseño de control que dirigirá las tareas del robot, aspecto que se investiga en este trabajo. La atención se dirige hacia robots móviles autónomos de dos ruedas con restricciones no holonómicas, provenientes de la premisa de evitar el deslizamiento. El diseño y estudio de sistemas de control desarrollados para robots móviles es un campo amplio, con un vasto conjunto de algoritmos de control que pueden ofrecer usos sorprendentes. Aunque el robot móvil puede mostrar movimientos que parezcan simples a simple vista, en términos matemáticos es intrincado, dado que el modelo dinámico no es lineal. Para tratar de reducir un índice de desempeño cuadrático y solucionar el problema, se adopta la solución convencional al reto del control preciso para un sistema lineal. Esto puede verificarse mediante la técnica de programación dinámica aplicada a la ecuación de Hamilton-Jacobi-Bellman, con el fin de establecer la

estrategia idónea para el sistema de control lineal en un marco temporal continuo. Dentro del control óptimo, se encuentra la resolución de la ecuación diferencial matricial de Riccati, que contiene detalles del sistema lineal y del índice de desempeño cuadrático, esta matriz resulta crucial para estabilizar el sistema lineal. Elegir adecuadamente las matrices de peso para el índice de rendimiento es esencial para la eficacia en la estabilización del sistema. El control óptimo diseñado para el sistema lineal alrededor de una ruta determinada se emplea después en el sistema de ecuaciones no lineales de movimiento del robot móvil con el objetivo de conseguir la trayectoria prevista.

2.3 Velocidad lineal y angular

La velocidad se caracteriza como la proporción en que varía la posición en función del tiempo. La posición (R) posee una naturaleza vectorial. La velocidad puede manifestarse de forma angular (ω) o lineal (V).

La velocidad lineal, v, denota la celeridad con la que un punto se desplaza a través de un camino circular. Por su parte, la velocidad angular indica cuán rápido ocurre una rotación. Esta se establece como el ángulo que se recorre en una unidad temporal y se representa con la letra griega ω. En el Sistema Internacional, su unidad es el radián por segundo.

2.4 Control en lazo cerrado

En estos sistemas, la entrada y la salida se contrastan para ajustar cualquier alteración presente. A este proceso se le denomina realimentación.

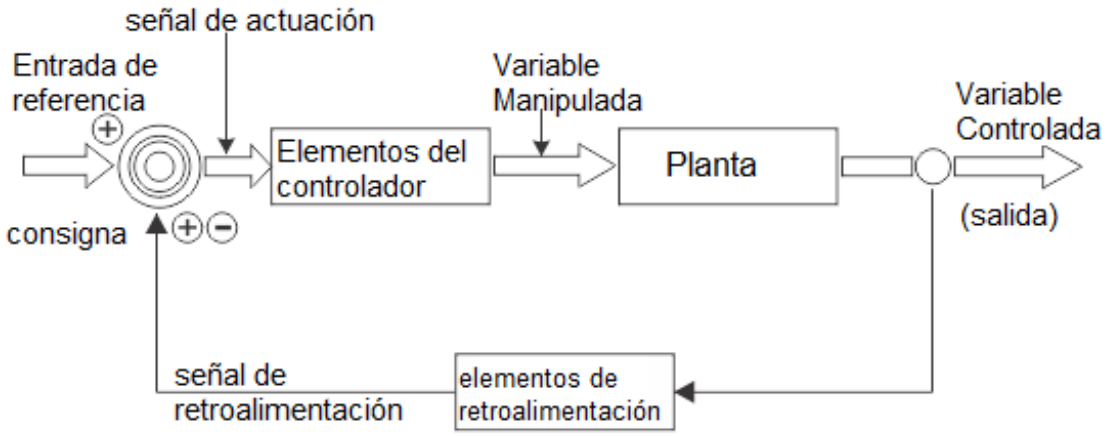


Figura 2.1. Control de lazo cerrado [18]

2.5 Elementos finales de control (Actuadores)

El robot Khepera IV está equipado con un amplio conjunto de dispositivos de detección (sensores y actuadores):

- Doce sensores ópticos reflectantes Vishay Telefunken TCRT5000. Ocho de estos sensores están igualmente espaciados en un anillo alrededor del cuerpo del robot, mientras que cuatro de ellos están orientados hacia abajo. Cuando están en modo de proximidad, los sensores emiten una longitud de onda de 950 mm y su rango publicado es de 2-250 mm. También pueden operar en modo pasivo y medir la luz ambiental. La frecuencia de muestreo para los sensores infrarrojos es 200 Hz, independientemente del modo de funcionamiento.

- Cinco transceptores ultrasónicos Prowave 400PT12B de 40 kHz. El rango publicado de los sensores es de 25-200 cm con un ángulo de haz de 85◦a -6 dBm, y se puede muestrear un sensor cada 20 ms. La tasa de muestreo efectiva depende de la cantidad de sensores habilitados, que van desde 50 Hz para un solo sensor hasta 10 Hz si se usa todo el conjunto.

- Una unidad de medición inercial (IMU) ST LSM330DLC iNEMO de paquete único montada en el centro, con un acelerómetro 3D y un giroscopio 3D. El acelerómetro está configurado para un rango de 2 g y una velocidad de datos de 100 Hz, y el giroscopio está configurado para un Alcance de 2000 dps y

tasa de datos de 95 Hz. Los datos son leídos por el microcontrolador en grupos de 10, por lo tanto, un nuevo conjunto de lecturas del acelerómetro está disponible cada 100 ms y un nuevo conjunto de lecturas de giroscopio está disponible cada 105 ms.

- Dos micrófonos MEMS amplificados Knowles SPU0414HR5H-SB, uno a cada lado. Los micrófonos omnidireccionales tienen una ganancia de 20 dB y un rango de frecuencia de 100-10000 Hz. La SNR nominal es de 59 y la sensibilidad es de -22 dBV a 1 kHz.

- Una cámara en color Aptina MT9V034C12ST montada en la parte frontal con un sensor CMOS WVGA de 1/3”, que produce una resolución de 752x480 px. El robot viene con una lente de 2,1 mm de foco fijo con filtro de corte IR, montada en una rosca M12x0.5.

Los campos de visión especificados son 150º diagonal, 131º horizontales y 101º vertical.

Las capacidades de movimiento son proporcionadas por dos motores de CC Faulhauber 1717, uno que impulsa cada rueda. La potencia nominal de cada motor es de 1.96 W, transferida a través de dos cajas de engranajes con una relación de transmisión general de 38:1 y una eficiencia general del 66.3 %, lo que genera una potencia útil de 1.3 W por rueda. Los motores están emparejados con codificadores de alta resolución Faulhaber IE2-128, con una revolución de rueda completa correspondiente a 19456 pulsos. Esto produce aproximadamente 147.4 pulsos por milímetro de desplazamiento de la rueda. La velocidad del motor se ajusta mediante la modulación de ancho de pulso (PWM), y es posible establecer los motores en varios modos: control en bucle cerrado de velocidad, gestión de perfil de velocidad, control de posición y también en modo de bucle abierto.

Los robots están equipados con tres LED RGB, montados en la parte superior del robot en un triángulo isósceles, con guías de luz en la cubierta superior. El color del LED se puede controlar con una resolución de 6 bits en cada canal, lo que los hace útiles para el seguimiento y la identificación. Finalmente, un altavoz PUI Audio SMS-1308MS-2-R, con potencia nominal 0.7 W, SPL 88 dBA y el rango de frecuencia 400-20000 Hz se pueden usar para comunicación o interacción.

2.6 Descripción de la planta Khepera IV

El robot Khepera IV es un modelo compacto, que requiere un pequeño espacio para funcionar, incluso en para las aplicaciones de robótica de enjambre. Con su alta flexibilidad, el robot Khepera IV se puede utilizar en casi cualquier aplicación, como navegación, enjambre, inteligencia artificial, computación, demostración, etc. El robot incluye una matriz de 8 sensores infrarrojos para la detección de obstáculos, cuatro más para evitar caídas. o seguimiento de línea, y 5 sensores ultrasónicos para la detección de objetos de largo alcance. Los bloques de motor del robot utilizan motores de CC de alta calidad para aumentar la eficiencia y la precisión. La batería interna proporciona un tiempo de funcionamiento de aproximadamente 7 horas.

2.7 Hardware

En las Figura 2.2 se puede apreciar la ubicación de los diferentes elementos del robot Khepera IV.

1. Ruedas giratorias
2. Interruptor encendido / apagado
3. LED de estado
4. Conector mini-USB B (modo dispositivo, sin carga)
5. Conector USB A (modo anfitrión, 500mA)
6. Conector de fuente de alimentación (9 V, 1.5 A, centro positivo de 0.65 mm)
7. LED de estado de carga
8. Botón de reinicio
9. Sensores infrarrojos (8x)
10. Sensores ultrasónicos (5x)
11. Cámara
12. Sensores IR inferiores (4x, para evitar caídas y seguimiento de línea)
13. Ruedas
14. Pegatina
15. Tuercas inferiores M3 (4x)
16. Conectores de extensión KB-250
17. Tuercas superiores M3 (4x)
18. Imanes (3x)
19. LED RGB (3x)

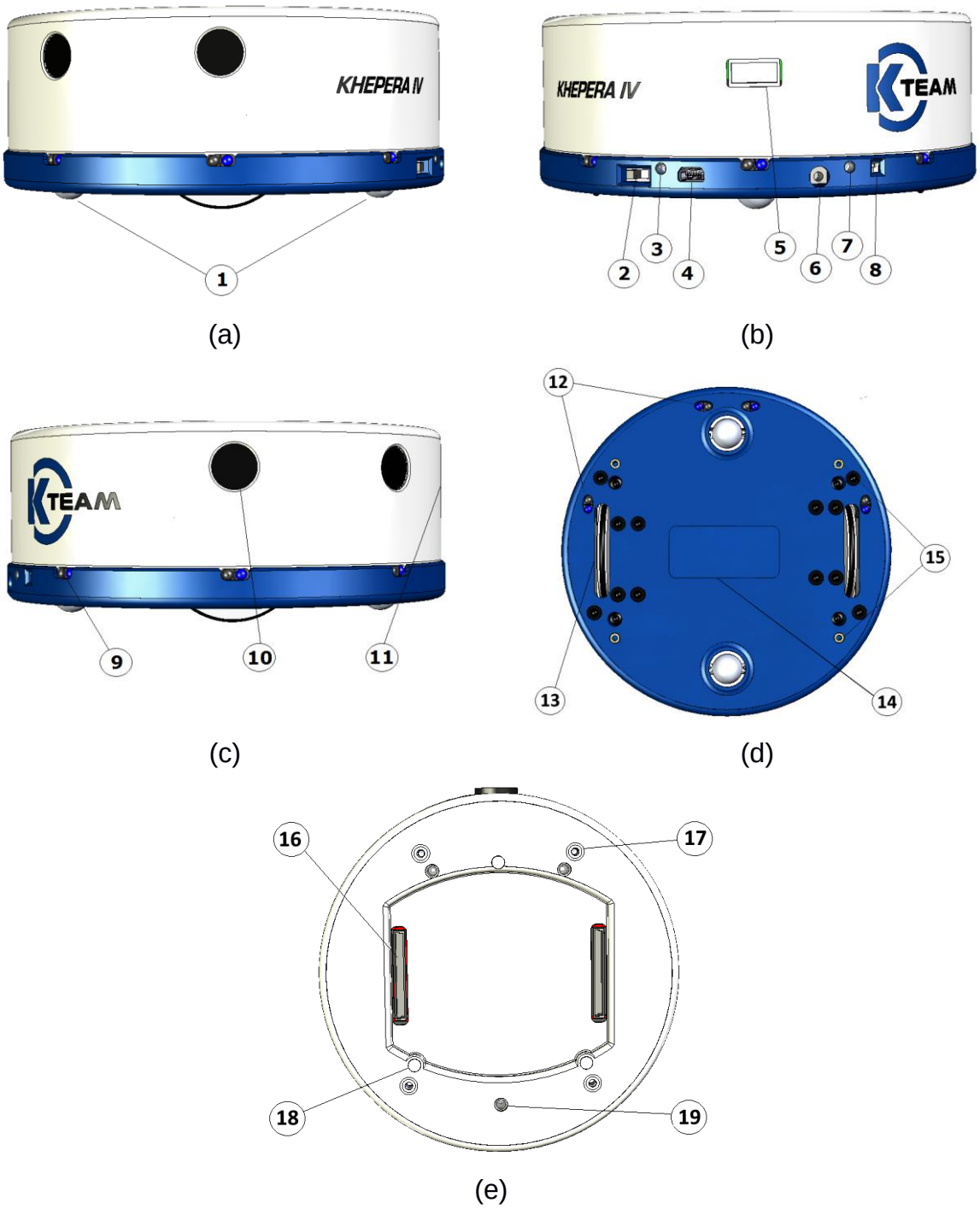


Figura 2.2. Vista global del robot Khepera IV a) Vista lateral izquierda, b) Vista posterior, c) Vista lateral derecha, d) Vista inferior, e) Vista superior [19]**.**

2.7.1 Sensores Infrarojos

El Khepera IV tiene 8 sensores infrarrojos colocados alrededor del robot y 4 colocados en la parte inferior. Estos últimos permiten experimentos como el seguimiento de líneas o evitar caídas. Están posicionados y numerados como se muestra en la Figura 2.2c.

Estos sensores incorporan un emisor de luz infrarroja y un receptor. Los doce sensores son ópticos reflectantes TCRT5000 de Vishay Telefunken. El rango de medición es de 2 a 250 mm. Cada sensor está separado de su vecino desde un ángulo de 45°.

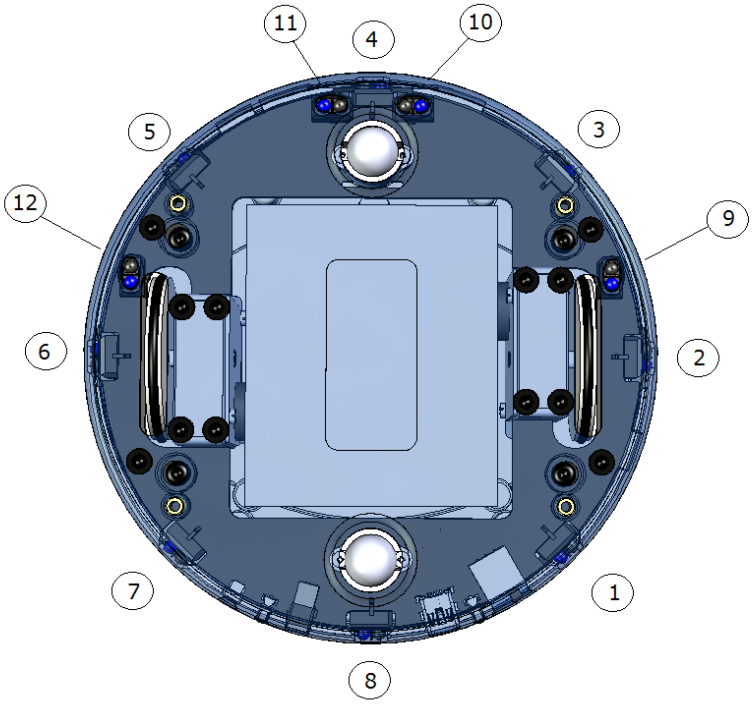


Figura 2.3. Vista inferior de los sensores infrarrojos [19]**.**

Este tipo de sensores permite dos medidas:

- La luz ambiental normal. Esta medida se realiza utilizando únicamente la parte receptora del dispositivo, sin emitir luz con el emisor. Se realiza una nueva medición cada 5 ms. El valor devuelto en un momento dado es el resultado de la última medición realizada.
- La luz reflejada por los obstáculos (=proximidad). Esta medida se realiza emitiendo luz utilizando la parte emisora del dispositivo. El valor devuelto es la diferencia entre la medición realizada mientras se emite luz y la luz medida sin emisión de luz (luz ambiental). Se realiza una nueva medición cada 5ms. El valor devuelto en un momento dado es el resultado de la última medición realizada.

2.7.2 Sensores Ultrasónicos

El robot Khepera IV dispone de cinco sensores Prowave 400PT12B alrededor del robot, los cuales se ubican y numeran como se muestra en la Figura 2.4. Estos sensores son transceptores, lo que significa que pueden emitir y recibir los pulsos.

Los sensores ultrasónicos están alimentados por una fuente de 85 Vpp. La frecuencia nominal de estos transductores es de 40kHz +/- 1kHz.

El valor devuelto es la distancia al objeto en centímetros, con una tolerancia de +/-2 cm. El rango de medición es de 25 a 200 cm. Cada transductor está separado de su vecino por un ángulo de 45°.

Cada sensor se puede deshabilitar para obtener una frecuencia de actualización más alta para uno (o un grupo) en particular. Una medida del sensor tarda 20 ms. Los 5 sensores necesitan 100ms para ser leídos.

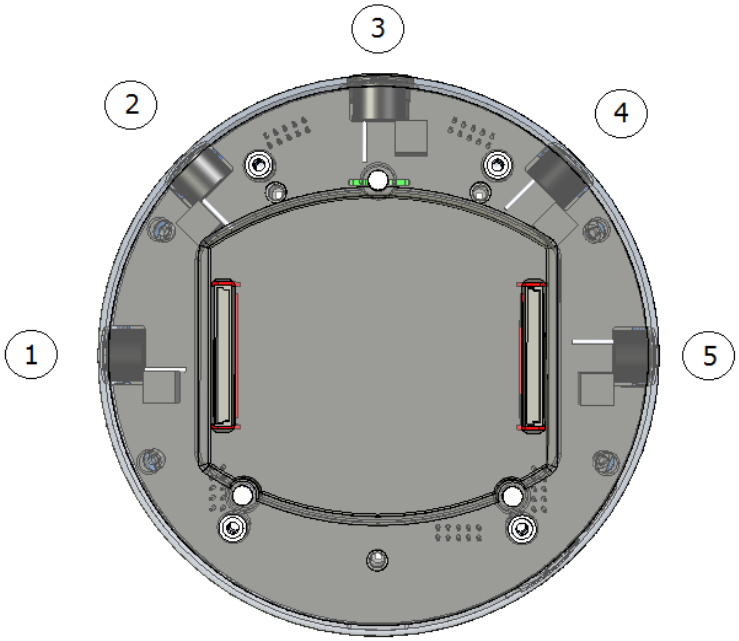


Figura 2.4. Vista superior de los sensores ultrasónicos [19]**.**

2.7.3 Sensores Ultrasónicos

El Khepera IV está equipado con una batería interna de polímero de litio no extraíble cuyas características se muestran en la Tabla 2.1.

Tabla 2.1. Propiedades de la batería del robot Khepera IV

Voltaje Nominal	: 7.4 V
Tensión de corte	: 6.0 V
Voltaje de carga	: 8.4 V
Capacidad nominal	: 3400 mAh
Corriente de descarga máxima	: 3400 mA (1C)
Corriente de carga	: 1100 mA
Corriente de carga	: 4 - 5 horas

Utilizando su potencia integrada, el robot puede funcionar de forma totalmente autónoma por un período superior a 5 horas con los motores al 100 % y 7 horas con los motores apagados, funcionando con una configuración básica. Cuando se utiliza equipo adicional, la autonomía se reduce ya que las extensiones de Khepera, como la pinza, dependen de las baterías como fuente de energía.

El Khepera no cuenta con un sistema de gestión de energía determinado. Si el voltaje de la batería desciende por debajo de 6V, esta interrumpe el circuito para prevenir una descarga intensa de sus celdas.

La batería se puede cargar desde dos lugares diferentes:

- Desde el conector
- Desde los conectores de las extensiones KB-250.

La batería se carga a través de un circuito integrado de carga de batería que necesita 9 V de voltaje de entrada. Durante su fase de corriente constante, la batería se carga con una corriente de 1.1 A.

2.7.4 Sensores Ultrasónicos

El Khepera IV está equipado con una cámara a color frontal, dispuesta debajo del sensor ultrasónico frontal. El sensor es un MT9V034C12ST de Aptina. Es un sensor CMOS WVGA de 1/3”, el tamaño del reproductor de imágenes activo es de 4,51x2,88 mm y los píxeles activos son 752x480.

La lente por defecto tiene una distancia focal de 2,1 mm, con filtro de corte IR y foco fijo. La rosca de montaje es M12x0,5. El campo de visión diagonal es de 150°, el horizontal es de 131° y el vertical es de 101°.

2.7.5 Micrófonos

El Khepera IV está equipado con un micrófono amplificado PU0414HR5H-SB de Knowles. Está conectado directamente a la entrada SUB MIC derecha de Overo Analog. Las características del micrófono son las que se indican en la Tabla 2.2.

Tabla 2.2. Propiedades del micrófono Knowles PU0414HR5H-SB.

Ganancia	: 20 dB
Sensibilidad (típica)	: -22 dbv/Pa
Directividad	: Omnidireccional
Tensión de alimentación	: 2,5 V

2.7.6 Altoparlante

Un altavoz SMS-1308MS-2-R de PUI Audio está montado en el Khepera IV. Este altavoz es impulsado por un amplificador de potencia de baja distorsión de 1W. El altavoz se

conecta en la salida de audio HSOLF del OVERO. El OVERO también puede silenciar el amplificador con GPIO64 (0 = MUTE, 1 = amplificación).

Tabla 2.3. Propiedades del altavoz SMS-1308MS-2-R.

Potencia del altavoz	: 0,7 W (máx. 1 W)
Impedancia	: 8 ohmios
Salida SPL	: 88 dBA
Distorsión (máx.)	: 5 %
Frecuencia resonante	: 850 Hz ± 20%
Rango de frecuencia	: 400 ~ 20 000 Hz

2.7.7 Gumstix Overo FireSTORM-Y COM

El Khepera IV incorpora una placa de procesador Gumstix Overo. El módulo de computadora montado por defecto es el Gumstix Overo FireSTORM-Y COM. Esta computadora tiene un DSP adicional para realizar tareas especiales, capacidades de Bluetooth y WiFi (antena SMD montada en el Khepera), adicionalmente se proporciona un sistema Linux ya instalado (distribución Yocto).

Tabla 2.4. Propiedades del Gumstix Overo FireSTORM-Y COM.

Arquitectura	: ARM Cortex-A8
Flash NAND	: 512 MB
Procesador	: Texas Instruments DaVinci DM3730 @ 800 MHz
DSP	: C64x Punto Fijo DSP 660,800 Mhz
Wi-Fi	: 802.11 b/g/n incluido
Bluetooth	: versión 4.1+BLE incluida

2.7.8 Acelerómetro

El acelerómetro montado en el Khepera IV es un LSM330DLC de ST. Este dispositivo incluye en un paquete un acelerómetro 3D y un giroscopio 3D. El dispositivo está exactamente en el centro del robot (colocado en el centro de rotación) y ubicado en la parte superior de la PCB principal. El acelerómetro está orientado con el pin 1 al frente a la derecha; esto devuelve un valor positivo para el eje X al avanzar. El eje Y es positivo a la izquierda, y finalmente el eje Z es negativo con la gravedad.

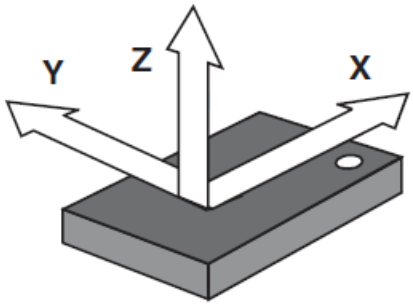


Figura 2.5. Vista superior de las direcciones de las aceleraciones detectables

El acelerómetro entrega datos de 12 bits con un rango de +/-2g, por lo que un valor de 1g devolverá un valor de 16384. La tasa de datos está configurada a 100 Hz, ya que el dsPIC de Khepera actualiza 10 valores a la vez. Por lo tanto, es necesario leer cada 100 ms (10 Hz) para obtener datos nuevos.

2.7.9 Giroscopio

El giroscopio del Khepera IV está incluido en el mismo paquete que el acelerómetro. Las direcciones de las velocidades angulares detectables se definen alrededor de los ejes del acelerómetro.

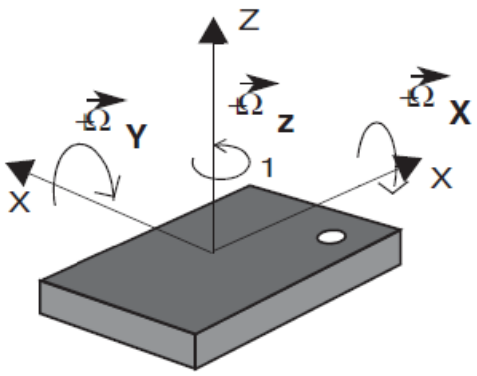


Figura 2.6. Vista superior de las direcciones de velocidades angulares detectable

El formato de datos es también es de 12 bits, el rango completo está configurado en +/- 2000dps (360dps = 5898) y la velocidad de datos está configurada en 95Hz. Los datos del giroscopio se leen en paquetes de 10 valores a la vez, lo que significa que el usuario puede leer datos nuevos cada 105 ms para obtener un valor nuevo. La salida debe multiplicarse por 0.066 para tener unidades [grados/s].

2.7.10 Motores

El Khepera IV tiene 2 motores de corriente continua para impulsar sus dos ruedas con una potencia nominal de 1.96W. La caja reductora integrada tiene una relación de 19:1 y una eficiencia del 78%. Existe otra caja de cambios dentro del cárter del bloque motor, con una relación de 2:1 y una eficiencia del 85%. La relación total es entonces de 38:1 y la eficiencia es del 66.3 %, lo que significa que hay 1,3 W de potencia mecánica utilizable por rueda.

El encoder tiene una resolución de 128 pulsos por vuelta. Con la relación de reducción de 38:1 y un multiplicador 4x de hardware interno, tenemos 19 456 pulsos por giro de rueda. Como el diámetro de la rueda es de 42 mm (el perímetro es entonces de 131.94 mm), esto da 147.4 pulsos por milímetro o 1 pulso es 0.006782mm (6.7818 um).

Ambos motores se controlan mediante modulación de ancho de pulso (PWM) a 20 kHz. Esta técnica enciende y apaga el motor a una frecuencia determinada y durante un tiempo determinado. De esta forma, el motor reacciona al promedio de la fuente de alimentación, que puede modificarse cambiando el período en que el motor está encendido.

El dsPIC calcula el PWM para aplicar a cada motor en control de velocidad y control de posición. El usuario puede anular el PID y aplicar directamente un PWM deseado al motor mediante el comando de bucle abierto.

Los ajustes predeterminados del PID aplicados al controlador de velocidad del robot Khepera IV son, Kp=10, Ki=5 y Kd=1 [19]. En el control de posición, el PID es similar al de velocidad. El sistema calcula una velocidad que luego el controlador debe alcanzar. El usuario puede modificar estos valores para mejorar el comportamiento en cada uso particular. Finalmente, se debe considerar que al seleccionar un tipo de control, este modo se aplicará a ambos motores. No es posible configurar el motor izquierdo en control de velocidad y el motor derecho en otro modo.

2.7.10.1 Control de Velocidad

Los motores DC son controlados por un PID ejecutado cada 10ms en una rutina de interrupción del dsPIC. Cada término de este controlador (Proporcional, Integral, Derivativo) está asociado a una constante, fijando el peso del término correspondiente: Kp para el proporcional, Ki para el integral, Kd para el derivativo.

El controlador tiene como entrada el valor de velocidad de las ruedas y controla el motor para mantener esta velocidad de rueda. La modificación de la velocidad se realiza lo más rápido posible, de forma brusca. En este modo no se considera limitación en la aceleración.

La unidad de velocidad corresponde a la diferencia de posición medida entre la rutina de los dos controladores (10ms) que se convierte a unidad métrica utilizando la ecuación (2.1), donde el tiempo de actualización es de 10ms, el diámetro de la rueda: 42 mm y la resolución de revolución: 19456 [pulsos].

$$Velocidad\left[\frac{mm}{s}\right] = \frac{v_{pulsos}}{t_{Actualización}}\frac{\phi_{Rueda}\pi}{Nb_{pulsos}} = \frac{v_{pulsos}}{0.01}\frac{42\pi}{19456} = 0.678181v_{pulsos} \qquad (2.1)$$

La velocidad mínima para asegurar un control correcto es de 3mm/s. Por debajo de este valor, el control no es muy estable con los parámetros PID predeterminados. Para mejorar el control a velocidades muy bajas es necesario modificar los valores del PID.

La velocidad máxima es de aproximadamente 813 mm/s. Sin embargo, es posible mover el robot más rápido si el modo de control está configurado en bucle abierto. En este caso, la velocidad máxima variará con el voltaje de la batería y la carga útil.

2.7.10.2 Control de Perfil de Velocidad

Este tipo de control utiliza el mismo PID que el controlador de velocidad estándar, pero agrega una rampa de aceleración para pasar de la velocidad real a la nueva velocidad.

La rampa utilizada en este modo se puede configurar con los parámetros del perfil de velocidad. Tres parámetros definen la rampa:

Acc_Inc: valor de incremento a sumar o restar en cada lazo de control Acc_Div +1 (valor de 1 a 255). Predeterminado = 3

Acc_Div: define el número de lazos de control donde no se agrega ningún incremento al orden de velocidad. Por ejemplo, un valor de 0 significa que, en cada lazo de control, la velocidad aumentará en Acc_Inc. Un valor de 4 significa que cada 5 lazos de control (50ms) se modificará la velocidad. (valor de 0 a 255) Predeterminado = 0.

Min_Speed_Acc: este parámetro define la velocidad mínima utilizada por el controlador. Este valor evita establecer una velocidad demasiado baja donde el controlador no es eficiente. Si el valor de la orden es menor que este parámetro, el controlador limitará automáticamente la velocidad a Min_Speed_Acc. No establezca valores inferiores a 1. Predeterminado = 20.

2.7.10.3 Control de Posición

En este modo, el robot calculará una velocidad (que será procesada por el PID) para mover el robot utilizando una rampa de aceleración, una velocidad constante y finalmente una rampa de desaceleración.

El modo de control de posición utiliza el mismo parámetro que el control de perfil de velocidad para calcular la rampa de aceleración. El parámetro Min_Speed_Acc se usa

solo al principio. Al llegar a la posición de destino, la velocidad está limitada por el parámetro Min_Speed_Dec (predeterminado = 1).

Además de estos tres parámetros, la velocidad de desplazamiento se puede configurar a través del parámetro “Speed_Order” (predeterminado = 400).

Finalmente, el parámetro “Pos_Margin” define el umbral cuando el controlador de posición detiene completamente el motor (establece 0 en el controlador de velocidad). Un margen bajo aumentará la precisión, pero agregará inestabilidad al control.

Para calcular la distancia real recorrida por el motor se utiliza la siguiente fórmula:

$$Posición[mm] = P_{pulsos}\frac{\phi_{Rueda}\pi}{Nb_{pulsos}} = P_{pulsos}\frac{42\pi}{19456} = \frac{P_{pulsos}}{147453} \qquad (2.2)$$

La posición se almacena en datos de 32 bits con signo, lo que significa que el orden de posición máximo es +/- 231 pulsos (= 14563m). Al realizar un desplazamiento recto (misma velocidad en cada rueda), la mejor solución es restablecer el encoder de posición antes de enviar el comando de posición de destino.

2.7.10.4 Control de lazo abierto

Este modo de control desactiva el controlador PID y establece directamente el PWM a los dos motores. Esto puede ser útil si la aplicación quiere calcular su propio PID.

El rango de este comando es +/- 2940 donde 2940 corresponde al 100% de PWM en dirección hacia adelante y -2940 en dirección hacia atrás.

Si la aplicación quiere deshabilitar el motor (para disminuir el consumo de corriente), la mejor manera es usar este modo y configurar el PWM a 0. El motor estará en modo de rueda libre.

2.8 Computadoras software y operación

La computadora va a utilizar es una portátil con un procesador Intel i7 de octava generación con 12 de RAM con GPU integrado y disco de estado sólido de 512 GB. Los softwares por utilizar para la simulación son los siguientes:

- *CoppeliaSim:* Es un simulador de robótica con un ambiente de desarrollo unificado, opera sobre una estructura de control distribuido. Cada objeto o modelo puede ser gestionado de manera individual mediante un script incorporado, un plugin, un nodo ROS, un cliente API a distancia o una estrategia diseñada a medida. Esta característica confiere a CoppeliaSim una gran adaptabilidad, siendo perfecto para usos con múltiples robots. Los sistemas de control pueden ser redactados en lenguajes como C/C++, Python, Java, Lua, MATLAB o Octave.

  Para el control del robot creado en CoppeliaSim se usará MATLAB/Simulink que nos permite una programación con una interfaz gráfica muy eficiente por lo que es la herramienta óptima a usar en este proyecto.

  CoppeliaSim es empleado para la creación ágil de algoritmos, simulaciones de sistemas automatizados, desarrollo acelerado de prototipos, comprobación, formación en robótica, supervisión a distancia, validación de seguridad redundante, funcionando como gemelo digital, entre otras aplicaciones.

- *Simulink:* Es un ambiente de programación gráfica que opera dentro del marco de programación de MATLAB. Proporciona un nivel de abstracción superior al lenguaje interpretativo de MATLAB. Los archivos creados en Simulink poseen la extensión mdl.

- *MATLAB:* Es un sistema de cálculo numérico con un ambiente de desarrollo unificado y un lenguaje de programación específico. Se encuentra disponible para plataformas como Unix, Windows, macOS y GNU/Linux.

- *FOMCON MATLAB:* FOMCON Toolbox for MATLAB está dedicado al modelo de orden fraccionario y al control de sistemas dinámicos y es compatible con la versión de MATLAB que se está usando con CoppeliaSim. Este plugin nos permite comunicar de manera bidireccional.

2.9 Programación de sistemas de control

La programación del sistema de control está basada en cálculo fraccionario, el cual es más eficiente.

Los modelos matemáticos actúan como representaciones aproximadas del comportamiento de sistemas reales. Entre las técnicas matemáticas disponibles, el Cálculo de Orden Fraccional (COF) destaca por sus características únicas en comparación con el cálculo tradicional de orden entero. Cuando se trata de modelar dinámicamente sistemas, el COF ofrece una representación más precisa que el cálculo de orden entero y es aplicable a una variedad de sistemas.

Los robots manipuladores se pueden representar utilizando distintas estrategias. Una ecuación de modelado frecuentemente utilizada para robots es la de Euler-Lagrange, que resulta en un sistema de ecuaciones diferenciales de orden entero. Antes de proponer un modelo basado en orden fraccional, es esencial asegurarse de que cumple con las condiciones de estabilidad. Si el modelo fraccional satisface estos criterios, entonces es viable para simulación y uso práctico.

2.10 Sistemas de control en tiempo continuo y discreto

Sistema de control en tiempo continuo: Los controladores en tiempo continuo funcionan de manera constante en el circuito de control de bucle cerrado, enviando una señal persistente al actuador. Los componentes P, I y D son elementos reconocidos en el ámbito del control continuo.

Sistema de control en tiempo discreto: Son sistemas dinámicos que involucran una o más variables que cambian en momentos específicos de muestreo. Estos momentos se denotan como KT, que se interpreta como la consulta a un espacio de memoria. Los sistemas de control digital gestionan señales digitales o discretas. Una señal digital surge del muestreo de una señal continua y se estudia en el ámbito temporal mediante la transformada z. Una señal discreta es aquella conocida solo en momentos determinados, denominados KT, con K=0,1,2...n, y donde T representa el intervalo de muestreo.

2.11 Control predictivo por modelo (MPC)

Es un controlador ampliamente aplicado en procesos industriales por su capacidad para lidiar con restricciones tales como la saturación de entrada y los límites de velocidad. Para anticipar y optimizar el error futuro y el esfuerzo de control, es necesario contar con el modelo dinámico del sistema, minimizando así una función de costo. El MPC ha

emergido como la técnica de control predominante en diversos campos, incluyendo las industrias de petróleo, refinación y química.

MPC es una técnica de control óptimo en la que las acciones de control calculadas minimizan una función de coste para un sistema dinámico restringido en un horizonte finito y en retroceso.

En cada intervalo temporal, un controlador MPC determina o estima el estado presente del sistema. Posteriormente, determina la serie de acciones de control que reducen el coste en el horizonte, resolviendo un problema de optimización con restricciones basado en un modelo interno del sistema y considerando su estado actual. A continuación, solo se implementa la primera acción de control determinada en el sistema, descartando las subsiguientes. Este proceso se repite en cada nuevo intervalo temporal.

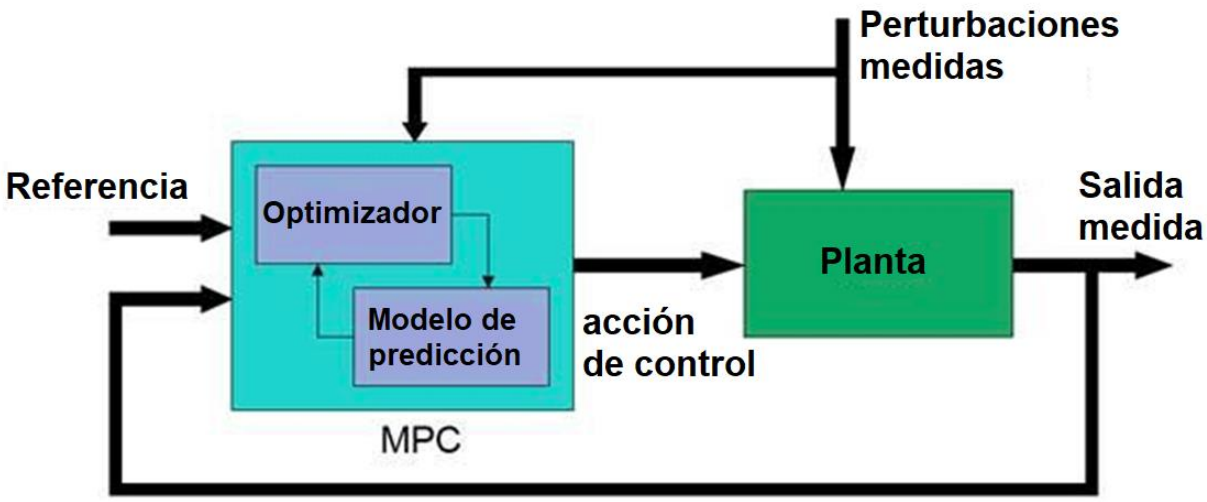


Figura 2.7. Esquema del controlador MPC [20]

Cuando la función de coste es cuadrática, la planta es lineal y sin restricciones, y el horizonte tiende al infinito, MPC es equivalente al control del regulador lineal cuadrático (LQR), o al control lineal cuadrático gaussiano (LQG) si un filtro de Kalman estima el estado de la planta a partir de sus entradas y salidas.

En la práctica, a pesar del horizonte finito, MPC a menudo hereda muchas características útiles del control óptimo tradicional, como la capacidad de manejar de forma natural plantas de múltiples entradas y múltiples salidas (MIMO), la capacidad de manejar

retrasos de tiempo (posiblemente de diferentes duraciones en diferentes canales) y propiedades de robustez incorporadas contra errores de modelado. La estabilidad nominal también se puede garantizar mediante el uso de restricciones terminales específicas. Otras características importantes adicionales de MPC son su capacidad para manejar explícitamente las restricciones y la posibilidad de hacer uso de información sobre futuras referencias y señales de perturbación, cuando esté disponible.

Resolver un control óptimo restringido en línea en cada paso de tiempo puede requerir grandes recursos computacionales. Sin embargo, en algunos casos, como en plantas con restricciones lineales, puede precalcular y almacenar la ley de control en todo el espacio de estado en lugar de resolver la optimización en tiempo real. Este enfoque se conoce como MPC explícito.

En los casos en que la planta a controlar se puede aproximar con precisión mediante una planta lineal solo localmente, alrededor de un punto de operación dado. Es posible que esta aproximación ya no sea precisa a medida que pasa el tiempo y cambia el punto de operación de la planta.

2.12 Modelo de Espacio de Estados de Orden Fraccional

Un modelo lineal invariante en el tiempo (LTI) de orden fraccionario habitual en el espacio de estado se puede escribir como [21]:

$$D_t^{\alpha} x(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t) \tag{2.3}$$

Donde $x \in R^n$, $u \in R^r$ and $y \in R^p$ son los vectores de estados, entradas y salidas del sistema, y $A \in R^{n\times n}$, $B \in R^{n\times r}$, $C \in R^{p\times n}$, $D \in R^{p\times r}$, y $\alpha = [\alpha_1, \alpha_2, \dots, \alpha_n]^T$ son las ordenes fraccionales, $\alpha \in R$. Si $\alpha_1 = \alpha_2 = \cdots \alpha_n \equiv \alpha$, el Sistema se denomina Sistema de orden conmensurado, por el contrario, es un sistema de orden inconmensurado [12].

Para proponer la estructura del modelo para el MPC, se utiliza la derivada del control como su señal de entrada, manteniendo la misma salida. Como resultado, se añade un integrador en el modelo diseñado. Además, debido al principio de control del horizonte en retroceso, donde se requiere la salida de la planta para la predicción del control,

asumimos que la entrada no puede afectar la salida al mismo tiempo y, por lo tanto, la matriz $D = 0$ en el modelo de planta. Dicho modelo LTI de orden fraccional aumentado del sistema se puede representar mediante el siguiente nuevo modelo de espacio de estados [22]:

$$\begin{aligned} D_t^{\alpha} x(t) &= Ax(t) + BD_t^{\lambda} u(t) \\ y(t) &= Cx(t) \end{aligned} \tag{2.4}$$

Las definiciones más comunes utilizadas para calcular el orden fraccional son las de Grünwald-Letnikov (GL) y Riemann-Liouville (RL). La definición de GL se utiliza en cálculos discretos, y la RL es utilizada en el dominio continuo. El orden fraccional de acuerdo con la definición de GL para una función $f(t)$ está dada por la ecuación (2.5) [23].

$${}_{t_0}D_t^{\beta} f(t) = \lim_{h \to 0} \frac{1}{h^{\beta}} \sum_{j=0}^{[(t-t_0)/h]} \omega_j^{(\beta)} f(t - jh) \tag{2.5}$$

Donde ${}_{t_0}D_t^{\beta}$ es el operador general en el cálculo de orden fraccional, t_0 y t son los límites superior e inferior, h es el paso de cálculo, $[\cdot]$ redondea hacia abajo al entero más cercano, β es el orden, $\omega_j^{(\beta)}$ es el coeficiente cuadrático y los valores numéricos pueden ser aproximados como $\omega_0^{(\beta)} = 1$, $\omega_j^{(\beta)} = (1 - (\beta + 1)/j)\omega_{j-1}^{(\beta)}$, $j = 1,2, \ldots$

La definición de RL se expresa por la ecuación (2.6).

$${}_{t_0}D_t^{\beta} f(t) = \frac{1}{\Gamma(n - \beta)} \frac{d^n}{dt^n} \int_{t_0}^{t} \frac{f(\xi)}{(t - \xi)^{\beta - 1}} d\xi \tag{2.6}$$

Donde, $n - 1 < \beta < n$, $n \epsilon \mathrm{N}$, y $\Gamma(\cdot)$ es la función gamma de Euler.

Para describir los sistemas de orden fraccional se utiliza la transformada de Laplace de la definición RL. Bajo condiciones iniciales nulas, la transformada de Laplace del cálculo de orden fraccional de RL es la que se indica en (2.7).

$$\mathcal{L}\{D^{\beta}f(t)\} = s^{\beta}\mathcal{L}\{f(t)\} \tag{2.7}$$

2.13 Fractional Order Model Predictive Control

Un sistema de orden fraccionario es aquel sistema descrito por la siguiente ecuación diferencial de orden fraccionario:

$$\begin{aligned} a_n D^{\propto_n} f(x) + a_n - 1^{D^{\propto_{n-1}}} f(x) + a_n - 2^{D^{\propto_{n-2}}} f(x) + \cdots \\ = b_n D^{\beta_n} f(x) + b_n + 1^{D^{\beta_{n-i}}} f(x) + b_n + 2^{D^{\beta_{n-2}}} f(x) + \cdots, \end{aligned} \tag{2.8}$$

Para llevar a cabo simulaciones e implementaciones, es esencial aproximarnos a las funciones de transferencia de orden fraccionario con potencias de ∈ R utilizando funciones de transferencia de orden entero convencional n ∈ Z que exhiban comportamientos parecidos. La función de transferencia entera podría requerir un número ilimitado de polos y ceros. Sin embargo, siempre es factible conseguir aproximaciones precisas.

Para poner en práctica la táctica de control FOMPC, se toma en cuenta la estructura fundamental ilustrada en la Figura 2.8. Se emplea un modelo para anticipar las respuestas futuras del sistema basándose en los valores anteriores y presentes, así como en las futuras acciones de control óptimas sugeridas. El optimizador calcula estas acciones considerando tanto la función de coste como las limitaciones pertinentes.

El concepto habitual de esta estrategia de control utiliza el modelo de orden fraccionario en varios dominios, como, por ejemplo, el dominio del tiempo, donde el modelo tiene la forma de la ecuación diferencial fraccionaria (ecuaciones de términos m o forma canónica del espacio de estado), dominio de frecuencia (y de Laplace), donde el modelo tiene la forma de función de transferencia, así como dominio de tiempo discreto y función de transferencia discreta.

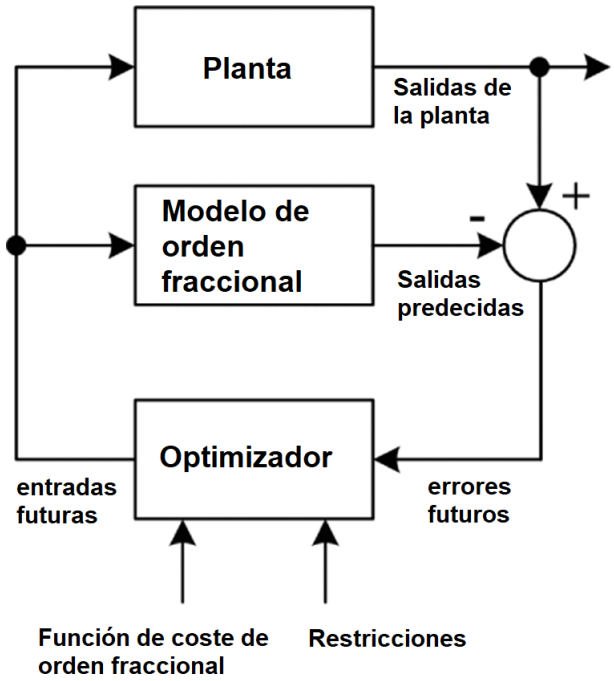


Figura 2.8. Estructura Básica del FOMPC [24]**.**

CAPÍTULO 3

3. Diseño de la solución

Model Predictive Control (MPC) es una técnica de control popular que ha ganado atención en los últimos años debido a su capacidad para manejar problemas de control complejos. MPC es un tipo de control de retroalimentación que optimiza una secuencia de control en un horizonte de tiempo finito mediante el uso de un modelo del sistema. El problema de optimización se resuelve en cada paso de tiempo y se aplica al sistema la primera acción de control de la secuencia óptima.

El enfoque MPC convencional asume que el modelo del sistema es un sistema lineal invariante en el tiempo (LTI), lo que limita su aplicabilidad a los sistemas que se pueden modelar con precisión utilizando modelos LTI. Sin embargo, muchos sistemas del mundo real exhiben dinámicas de orden fraccional, que los modelos LTI no pueden describir con precisión. En tales casos, se puede utilizar un enfoque MPC de orden fraccional (FOMPC) para manejar la dinámica de orden fraccional del sistema.

FOMPC utiliza modelos de orden fraccional (FOM) para representar la dinámica del sistema. Los FOM amplían el concepto de derivadas de orden entero a órdenes no enteros y pueden representar sistemas con respuestas decrecientes no exponenciales. El enfoque FOMPC utiliza estos modelos para diseñar el controlador MPC.

En este capítulo presentamos el desarrollo de una solución para el control FOMPC de un robot diferencial. El robot Khepera IV se utiliza como caso de estudio. El capítulo está organizado de la siguiente manera: Primero, presentamos los conceptos de cálculo fraccionario y los modelos de orden fraccionario. Luego, describimos el robot Khepera IV y derivamos su FOM (Fractional Orden Model). Luego presentamos el diseño FOMPC y los resultados de la simulación.

3.1 Khepera IV

El Khepera IV es un robot móvil de accionamiento diferencial desarrollado por K-Team Corporation. El robot es compacto, liviano y está equipado con varios sensores, incluidos sensores de proximidad, infrarrojos y de color. La cinemática del robot se describe mediante las siguientes ecuaciones:

$$\dot{x} = v\cos\theta$$

$$\dot{y} = v\sin\theta \tag{3.1}$$

$$\dot{\theta} = \omega$$

Donde x, y y θ son la posición y orientación del robot. Podemos derivar la FOM del robot Khepera IV usando cálculo fraccionario. Las ecuaciones de espacio de estado del robot son:

$$D^{\alpha}\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} 0 & 0 & -v\sin\theta \\ 0 & 0 & v\cos\theta \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.2}$$

La matriz A en el FOM entonces viene dada por:

$$A = \begin{bmatrix} 0 & 0 & -v\sin\theta \\ 0 & 0 & v\cos\theta \\ 0 & 0 & 0 \end{bmatrix}$$

La matriz de entrada B y la matriz de salida C están dadas por:

$$B = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & \dfrac{v}{L} \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

3.1.1 Modelo cinemático

El modelo cinemático de un robot diferencial establece una relación entre las velocidades del punto focal o de control y las velocidades de los actuadores. Esto ayuda a identificar la posición del robot dentro de un plano de referencia (Modelo Cinemático).

Para realizar el modelado se toma en cuenta sólo las fuerzas con las cuales se desplaza el robot y no así a las que se interponen a su paso. El robot es considerado como una masa puntual, el cual integra una herramienta en el centro de este.

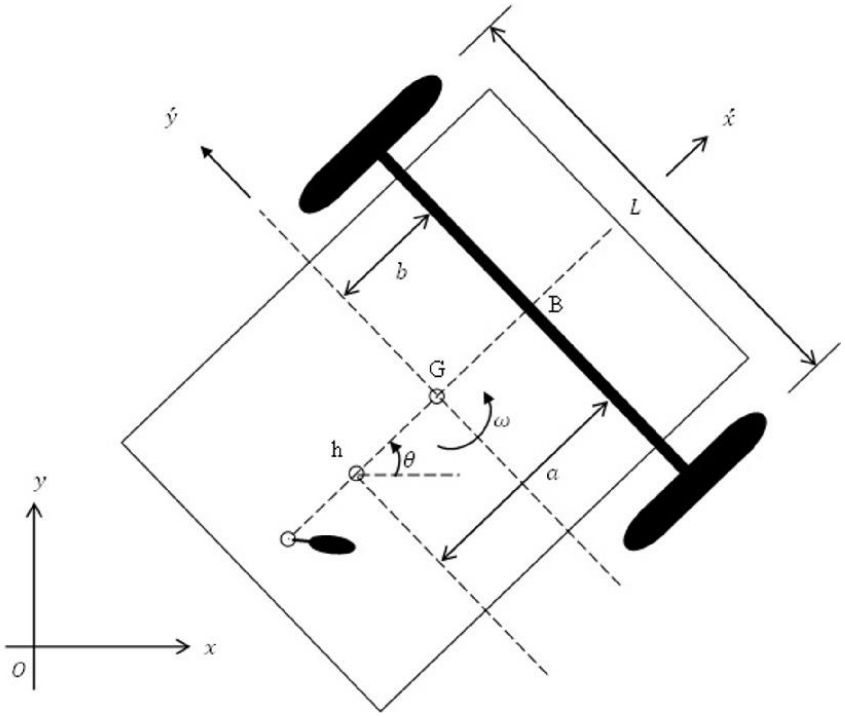


Figura 3.1. Modelo cinemático diferencial [26]**.**

Para describir el movimiento del robot en un plano bidimensional, los sistemas de coordenadas se seleccionan como se muestra en la Figura 3.1. El $x - O - y$ es el sistema de coordenadas global, y el $x' - G - y'$ es el sistema de coordenadas local fijado en el robot móvil con el origen en el centro de gravedad (CG) del robot, donde x′ en la dirección del rumbo e y′ en la dirección lateral. La Figura 3.1 también muestra los parámetros necesarios para obtener el modelo. Los puntos $\boldsymbol{G}$, $\boldsymbol{B}$ y $h = (x, y)$ representan el CG, el centro de la línea base de la rueda y el punto que se requiere para seguir una trayectoria, respectivamente.

La plataforma del robot móvil tiene dos ruedas motrices y una rueda pasiva que se agrega con fines de estabilidad, ya que el robot requiere al menos tres puntos de contacto con el suelo para ser estáticamente estable. Las dos ruedas motrices tienen el mismo radio que se denota por r. La distancia entre las dos ruedas se denota por L. La postura del robot móvil se representa mediante un vector $p = [x\ y\ \theta]^T$, donde x e y son las coordenadas del punto h según el marco global, y θ es la orientación del robot móvil, que es el ángulo entre el eje x del marco global y el eje x' del marco del robot. Las dos ruedas motrices pueden girar libremente sobre sus ejes de rotación, que coinciden con el eje y de la estructura del robot (y'). La velocidad lineal del robot (v) es ortogonal al eje de

rotación de las dos ruedas motrices. De acuerdo con las restricciones no holonómicas, se supone que el robot no tiene velocidad lateral, lo que significa que las ruedas giran puramente sin deslizamiento.

Suponiendo que $a = b$, el modelo cinemático del sistema de robot móvil de accionamiento diferencial es:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Donde ω y v son las velocidades de rotación y traslación (lineal) del robot. El objetivo del diseño de control es seguir una trayectoria de referencia definida por

$$\boldsymbol{p}_r = [x_r \quad y_r \quad \theta_r]^T$$

Donde $\boldsymbol{p}_r$ es la trayectoria de referencia del robot, que es suave. El objetivo es diseñar un controlador para lograr el seguimiento de este camino.

El error de posicionamiento $\boldsymbol{e}(t)$, que es la diferencia entre la postura de referencia y la real es

$$\boldsymbol{e}(t) = \boldsymbol{p}_r(t) - \boldsymbol{p}(t)$$

$$= [x_r \quad y_r \quad \theta_r]^T - [x \quad y \quad \theta]^T$$

El error $\boldsymbol{e}(t)$ se da en el sistema de coordenadas global. Es más adecuado diseñar un controlador después de transformar el modelo de robot al sistema de coordenadas local. Si $x_e(t)$, $y_e(t)$ y $\theta_e(t)$ son errores en el sistema de coordenadas local, entonces x_e da el error en la dirección de conducción, y_e da el error en la dirección lateral y θ_e da el error en orientación. El error de postura local $\boldsymbol{e}_l(t) = [x_e(t) \quad y_e(t) \quad \theta_e(t)]^T$ se determina transformando el error de postura global $\boldsymbol{e}(t)$ al sistema de coordenadas local multiplicándolo por una matriz de rotación.

$$\begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} \tag{3.3}$$

La ecuación de espacio de estado de la cinemática del robot móvil con el vector de estado $\boldsymbol{e} = [x_e \quad y_e \quad \theta_e]^T$ es:

$$\begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} = \begin{bmatrix} y_e\omega - v + v_r \cos\theta_e \\ -x_e\omega + v_r \sin\theta_e \\ \omega_r - \omega \end{bmatrix} \tag{3.4}$$

Donde v_r es la velocidad lineal de referencia y ω_r es la velocidad angular de referencia de la trayectoria.

3.1.2 Modelo dinámico

El conjunto de rueda motriz consta de rueda, caja de cambios y motor eléctrico. La relación voltaje-corriente del motor se describe mediante una ecuación diferencial de primer orden, que cuando se combina con las ecuaciones de movimiento da como resultado un sistema dinámico gobernado por una ecuación diferencial de tercer orden.

Los modelos de motor obtenidos despreciando el voltaje en las inductancias son:

$$\begin{aligned} \boldsymbol{\tau_R} &= \frac{\boldsymbol{k_a}(\boldsymbol{u_R} - \boldsymbol{k_b}\boldsymbol{\omega_R})}{\boldsymbol{Ra}} \\ \boldsymbol{\tau_L} &= \frac{\boldsymbol{k_a}(\boldsymbol{u_L} - \boldsymbol{k_b}\boldsymbol{\omega_L})}{\boldsymbol{Ra}} \end{aligned} \tag{3.5}$$

Donde ω_R y ω_L corresponde con la velocidad angular de la rueda derecha e izquierda, u_R y u_L es el voltaje de entrada aplicado al motor derecho e izquierdo respectivamente; k_b es igual a la multiplicación de la constante de tensión y la relación de transmisión; R_a es la resistencia eléctrica; τ_R y τ_L son los pares de los motores derecho e izquierdo multiplicados por la relación de transmisión; y k_a es la constante de par multiplicada por la relación de transmisión.

Las ecuaciones de espacio de estado del modelo dinámico con vector de estado $[v \quad \omega]^T$ y vector de entrada $[\tau_R \quad \tau_L]^T$ son

$$\begin{aligned} \dot{v} &= \frac{r(\tau_R + \tau_L) - 2B_e v}{mrR + 2I_e} \\ \dot{\omega} &= \frac{rd(\tau_R + \tau_L) - d^2 B_e \omega}{2I_z rR + d^2 I_e} \end{aligned} \tag{3.6}$$

Donde I_e y B_e son el momento de inercia y el coeficiente de fricción viscosa del rotor del motor, la caja de cambios y la rueda combinados, R es el radio nominal del neumático, m es la masa del robot e I_z es el momento de inercia del robot con respecto al eje vertical ubicado en G. En la Tabla 3.1 se indican los parámetros del robot.

Tabla 3.1. Parámetros para el modelo del robot Khepera IV

Parámetro	Descripción	Unidad
k_a	Constante de par	$N \cdot m/A$
L_a	Inductancia de la armadura	H
R_a	Resistencia de la armadura	Ω
r	Radio de la rueda	m
m	Masa	Kg
I_z	Inercia	$Kg \cdot m^2$
L	Distancia entre las ruedas	m
k_b	Constante contraelectromotriz	$V/(rad/s)$
I_e	Coeficiente del momento de inercia	$Kg \cdot m^2$
B_e	Coeficiente de fricción viscosa	$N \cdot m \cdot s$

El Khepera IV tiene 2 motores de corriente continua para impulsar sus dos ruedas. Los motores tienen una potencia nominal de 1,96W. El engranaje integrado tiene una relación de reducción de 19:1 y una eficiencia del 78%. hay otra caja reductora dentro del cárter del bloque motor, con una relación de 2:1 y una eficiencia de 85%. La relación total es entonces 38:1 y la eficiencia es 66,3%, lo que significa que hay 1,3W de potencia mecánica utilizable por rueda.

$$\tau = 1{,}3\ \mathrm{Nm}$$

$$u = 7{,}4\ \mathrm{voltios}$$

Entonces

$$K = \frac{\tau}{u} = \frac{1{,}3}{7{,}4} = 0.1757[N \cdot m/V]$$

La masa del robot es

$$M = 0.6\,Kg$$

El modelo obtenido se tiene $\dot{v}$, $\ddot{\theta}$ para obtener el desplazamiento se procede a integrar las salidas.

Con el cambio de variables, se tiene:

$$\begin{bmatrix} v \\ \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{3.7}$$

Y su derivada:

$$\begin{bmatrix} \dot{v} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} \tag{3.8}$$

3.2 Modelo Fraccionario

Como se indicó previamente, el cálculo fraccionario es una extensión de los conceptos de cálculo tradicionales a órdenes no enteros. Se trata de derivadas e integrales de orden fraccionario, que se han utilizado para describir sistemas con memoria y dependencias de largo alcance. Las derivadas de orden fraccionario se definen como se indica a continuación:

Los FOM son modelos matemáticos que utilizan cálculo fraccionario para representar la dinámica de un sistema. El FOM es una generalización del modelo de espacio de estado de orden entero y se define de la siguiente manera:

$$D^{\alpha} y(t) = \frac{1}{\Gamma(n-\alpha)} \frac{d^n}{dt^n} \int_0^t \frac{y(\tau)}{(t-\tau)^{\alpha+1-n}} d\tau \tag{3.9}$$

Donde α es el orden de la derivada fraccionaria, n es el entero más pequeño mayor que α y Γ es la función Gamma de Euler.

$$\begin{aligned} D^{\alpha} x(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{3.10}$$

Donde A, B, C y D son las matrices del sistema y α es el orden de la derivada fraccionaria.

Para proponer la estructura del modelo para el MPC, se utiliza la derivada del control como su señal de entrada, manteniendo la misma salida. Como resultado, se incluye un integrador en el modelo diseñado. Además, debido al principio de control del horizonte de retroceso, donde se requiere la salida de la planta para la predicción del control, asumimos que la entrada no puede afectar la salida al mismo tiempo y, por lo tanto, la matriz $D = 0$ en el modelo de planta (3.7). Tal modelo LTI de orden fraccional aumentado del sistema se puede representar mediante el siguiente modelo de espacio de estado:

$$D^{\alpha}x(t) = Ax(t) + B_0 D_t^{\lambda}\, u(t)$$
$$y(t) = Cx(t) \tag{3.11}$$

El modelo de espacio de estado aumentado de orden fraccionario (3.11) también contiene la derivada fraccionaria de orden λ de la señal de control como su entrada, y su salida sigue siendo la misma cómo se puede apreciar en la Figura 3.2.

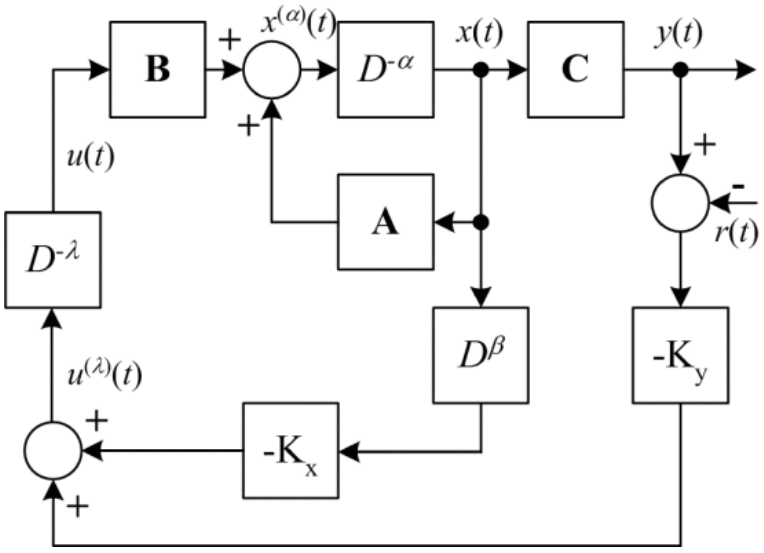


Figura 3.2. Estructura Básica del FOMPC

3.2.1 Modelo de espacio de estados no mínimo extendido

Por simplicidad, si se considera un sistema SISO con tiempo de retardo y se aplica la ecuación (2.7) se obtiene el modelo indicado en (3.12).

$$G(s) = \frac{Ke^{-\tau s}}{Ts^{\alpha} + 1} \tag{3.12}$$

Donde α es el orden no entero del proceso modelado, τ es el tiempo de retardo, K y T son el coeficiente de proporcionalidad y la contante de tiempo del proceso respectivamente.

Como se aprecia en el sistema de (3.12) los métodos tradicionales de discretización no pueden ser utilizados, por lo que se debe recurrir al método numérico aproximado de GL para obtener la expresión discreta del modelo de orden fraccional. Para discretizar el operador de orden fraccional se debe recurrir a la función generadora discretizada y a la expansión en serie de potencias. El resultado se representa en una estructura ARX. Del mismo modo, la salida $y(k)$ será derivada en la base de la entrada $u(k-1)$. Entonces el modelo discreto de orden fraccional puede ser representado como se indica en (3.13).

$$y(k) + \mu \sum_{l=1}^{L_s} \omega_l^{\alpha}\, y(k-l) = Hu(k-d-1) \tag{3.13}$$

Donde:

$$\mu = \frac{T}{T_s^{\alpha}}\left(1 + \frac{T}{T_s^{\alpha}}\right)^{-1}$$

$$H = K\left(1 + \frac{T}{T_s^{\alpha}}\right)^{-1}$$

$$\omega_0^{\alpha} = 1$$

$$\omega_l^{\alpha} = \left(1 - \frac{\alpha + 1}{l}\right)\omega_{l-1}^{(\alpha)} \quad l = 1, 2, \cdots, L_s$$

En el tiempo de retardo $d = \tau/T_s$, T_s es el tiempo de muestreo y L_s es la longitud aproximada de memoria de los operadores de orden fraccional. $y(k)$ y $u(k)$ son la salida y la entrada respectivamente del sistema de orden fraccional [22].

El operador de diferencia Δ se añade al modelo en (3.13). Si se considera $F_l = \mu\omega_l^{\alpha}$ el modelo se representa por (3.14).

$$\Delta y(k) = -F_1\Delta y(k-1) - F_2\Delta y(k-2) - \cdots - F_{L_s}\Delta y(k-L_s) + H\Delta u(k-1-d) \tag{3.14}$$

La variable de estados es (3.15).

$$\Delta x_m(k) = [\Delta y(k), \Delta y(k-1), \cdots, \Delta y(k-L_s), \Delta u(k-1), \Delta u(k-2), \cdots, \Delta u(k-d)]^T \tag{3.15}$$

Por lo que el modelo de espacio de estados considerando (3.14) y (3.15) es obtenido en (3.16).

$$\begin{aligned} \Delta x_m(k+1) &= A_m\Delta x_m(k) + B_m\Delta u(k) \\ \Delta y(k+1) &= C_m\Delta x_m(k+1) \end{aligned} \tag{3.16}$$

Donde

$$A_m = \begin{bmatrix} -F_1 & -F_2 & \cdots & -F_{L_s-1} & -F_{L_s} & 0 & \cdots & 0 & H \\ 1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & 0 & 1 & 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & 1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & \cdots & 0 & \cdots & 1 & 0 \end{bmatrix}$$

$$B_m = [0 \quad \cdots \quad 0 \quad 1 \quad 0 \quad \cdots \quad 0]^T$$

$$C_m = [1 \quad 0 \quad 0 \quad \cdots \quad 0]^T$$

Para extender el modelo de espacio de estados de orden fraccional, la variable de estados Δx_m y el error de seguimiento de la salida $e(k)$ son considerados en el modelo (3.17).

$$z(k+1) = Az(k) + B\Delta u(k) + C\Delta r(k+1) \tag{3.17}$$

Donde

$$e(k+1) = e(k) + C_mA_m\Delta x_m(k) + C_mB_m\Delta u(k) - \Delta r(k+1)$$

$$z(k+1) = \begin{bmatrix} e(k+1) \\ \Delta x_m(k+1) \end{bmatrix}$$

$$z(k) = \begin{bmatrix} e(k) \\ \Delta x_m(k) \end{bmatrix}$$

$$e(k) = y(k) - r(k)$$

$$A = \begin{bmatrix} 1 & C_m A_m \\ \mathbf{0} & A_m \end{bmatrix}$$

$$B = \begin{bmatrix} C_m B_m \\ B_m \end{bmatrix}$$

$$C = \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix}$$

En donde $r(k)$ es el set-point de la trayectoria de seguimiento, $e(k)$ es el error de seguimiento entre la salida del proceso y el set-point, $\mathbf{0}$ es un vector de dimensión $(L_s + d) \times 1$.

3.2.2 Acción de control de orden fraccional

En lugar de modelar la señal de control $u(t)$, el MPC de tiempo continuo apuntará a la derivada de orden fraccional de la señal de control, que satisfaga la propiedad:

$$\int_0^\infty D_t^\lambda \, u(t)^2 dt < \infty$$

Para señales de entrada constantes externas. Usando el modelo de espacio de estado de orden fraccional aumentado, donde la entrada es $D_t^\lambda u(t)$, el sistema de control de lazo cerrado es:

$$D^\alpha x(t) = \left(A - BK_{mpc}\right) x(t)$$

Se pueden calcular los valores propios de lazo cerrado del sistema de control predictivo. El control de retroalimentación de estado se da entonces como:

$$D_t^\lambda u(t) = -K_{mpc} x(t) = [K_x \quad K_y] \begin{bmatrix} D_t^{\beta} x(t) \\ y(t) - r(t) \end{bmatrix}$$

Donde $r(t)$ es una señal de punto de referencia y la señal de error es $y(t) - r(t)$. El orden fraccionario $\beta \in R$ es un parámetro de controlador adicional junto con las constantes K_x y K_y. Para obtener la ley de control y la acción del control integral, necesitamos integrar el resultado de la siguiente manera:

$$u(t) = D_t^{-\lambda}\left(-K_{mpc}x(t)\right)$$

3.3 Diseño del controlador FOMPC

El diseño FOMPC implica definir una función objetivo, restricciones y optimizar las entradas de control en un horizonte de tiempo finito. La función objetivo es una medida del rendimiento del sistema y normalmente se define como una suma ponderada del error de seguimiento y el esfuerzo de control. Las restricciones generalmente se definen para garantizar que las variables de estado y de entrada permanezcan dentro de los límites permitidos.

El diseño FOMPC implica resolver un problema de optimización en cada paso de tiempo. El problema de optimización, en tiempo continuo se puede formular de la siguiente manera:

$$\underset{U}{\operatorname{argmin}} J = \sum_{k=0}^{N-1}\left[\|y(k) - r(k)\|_Q^2 + \|u(k)\|_R^2\right] \tag{3.18}$$

Sujeto a:

$$x(k+1) = Ax(k) + Bu(k)$$

$$y(k) = Cx(k)$$

$$u_{min} \leq u(k) \leq u_{max}$$

$$x_{min} \leq x(k) \leq x_{max}$$

$$y_{min} \leq y(k) \leq y_{max}$$

$$x(0) = x_0$$

Donde U es la secuencia de entrada de control, J es la función objetivo, N es el horizonte de predicción, Q y R son las matrices de ponderación para el error de seguimiento y el esfuerzo de control, respectivamente, r es la trayectoria de referencia, y u_{min}, u_{max}, x_{min}, x_{max}, y_{min}y y_{max} son las restricciones en las variables de entrada y de estado.

Para tiempo discreto, el desarrollo de la estrategia de FOMPC utiliza el modelo (3.17), por lo tanto, las variables de estado futuras se representan por (3.19).

$$Z = Gz(k) + S\Delta\mathrm{U} + \Psi\Delta\mathrm{R} \tag{3.19}$$

Donde

$$Z = \begin{bmatrix} z(k+1) \\ z(k+2) \\ \vdots \\ z(k+P) \end{bmatrix}$$

$$S = \begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ A^{P-1}B & A^{P-2}B & \cdots & A^{P-M}B \end{bmatrix}$$

$$G = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^P \end{bmatrix}$$

$$\Psi = \begin{bmatrix} C & 0 & \cdots & 0 \\ AC & C & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ A^{P-1}C & A^{P-2}C & \cdots & A^{P-M}C \end{bmatrix}$$

$$\Delta U = [\Delta u(k) \quad \Delta u(k+1) \quad \cdots \quad \Delta u(k+M-1)]^T$$

$$\Delta R = [\Delta r(k+1) \quad \Delta r(k+2) \quad \cdots \quad \Delta r(k+P)]^T$$

$$r(k+i) = \lambda^i y(k) + \left(1-\lambda^i\right)c(k), \;\; i = 1,2,\cdots,P$$

Donde $c(k)$ es el set-point en el tiempo actual k, λ es el factor de suavizado, P y M son el horizonte predictivo y de control.

Para añadir precisión a la descripción del proceso se añaden operadores fraccionales a la función de costo como se indica en (3.20) [25].

$$J = {}^{\gamma_1}I_{T_s}^{PT_s} z(t)^T z(t) + {}^{\gamma_2}I_{T_s}^{MT_s} \Delta u(t-1)^2$$

$$J = \int_{T_s}^{PT_s} D^{1-\gamma_1} z(t)dt + \int_{T_s}^{MT_s} D^{1-\gamma_2} \Delta u(t-1)^2 dt \tag{3.20}$$

Para ajustar los valores de ponderación de las variables de estado y del error, las matrices Q_j y R deben aplicarse a la función de costo, que puede ser discretizada utilizando la definición de GL.

$$J = Z^T \Lambda_1 Z + \Delta U^T \Lambda_2 \Delta U \tag{3.21}$$

Donde

$$\Lambda_1 = block\ diag\left[T_s^{\gamma_1}(\omega_{P-1}Q_1, \omega_{P-2}Q_2, \cdots, \omega_1 Q_{P-1}, \omega_0 Q_P)\right]$$
$$\Lambda_2 = T_s^{\gamma_2}\ diag\left[T_s^{\gamma_2}(r_1\omega_{M-1}, r_2\omega_{M-2}, \cdots, r_{M-1}\omega_1, r_M\omega_0)\right]$$
$$\omega_q = \omega_q^{(-\gamma_\varepsilon)} - \omega_{q-(X-1)}^{(-\gamma_\varepsilon)}$$
$$\omega_0^{(-\gamma_\varepsilon)} = 1, \omega_q^{(-\gamma_\varepsilon)} = \left(1 - \frac{(1-\gamma_\varepsilon)}{q}\right)\omega_{q-1}^{(-\gamma_\varepsilon)}\ \ \forall q > 0$$
$$\omega_q^{(-\gamma_\varepsilon)} = 0\ for\ q < 0, X = P\ for\ \varepsilon = 1 \wedge X = M\ for\ \varepsilon = 2$$
$$Q_j = diag\left(q_\varepsilon, q_{y_1}, q_{y_2}, \cdots, q_{yL_s}, q_{u_1}, q_{u2}, \cdots, q_{ud}\right)$$
$$R = diag(r_1, r_2, \cdots, r_M)$$

Por lo tanto, minimizando (3.21) se obtiene la variable manipulada en (3.23).

$$\Delta U = -(S^T \Lambda_1 S + \Lambda_2)^{-1} S^T \Lambda_1 (Gz(k) + \Psi \Delta R) \tag{3.22}$$

$$u(k) = [1 \quad 0 \quad \cdots \quad 0]\Delta \mathrm{U} + u(\mathrm{k}-1) \tag{3.23}$$

Se ha demostrado que las técnicas de optimización basadas en cálculo fraccionario tienen algunas ventajas sobre los métodos de optimización clásicos, especialmente para sistemas con efectos de memoria. Los métodos de optimización basados en cálculo fraccional tienen en cuenta la dinámica de orden fraccional del sistema y pueden mejorar el rendimiento de este.

El diseño FOMPC implica diseñar un modelo de orden fraccional del sistema y usarlo en la formulación del problema de optimización. El modelo de orden fraccional es una representación más precisa de la dinámica del sistema en comparación con el modelo de orden entero. El algoritmo FOMPC implica resolver el problema de optimización en cada paso de tiempo usando el modelo de orden fraccional y usando la entrada de control obtenida para el sistema.

3.4 Integración CoppeliaSim – MATLAB

CoppeliaSim es un potente simulador de robots en 3D que proporciona una gran variedad de funciones y capacidades integradas. Ofrece varias interfaces, como API remota, ROS y más. Tiene sus propios lenguajes de programación (basados en Lua), pero también proporciona una API para interactuar con otros lenguajes de programación populares como Python y MATLAB.

Sin embargo, debido a la versatilidad del programa MATLAB para su uso en investigación, se ha decidido utilizar este para realizar la programación del modelo fraccionario y del controlador MPC en el robot Khepera IV. Estos resultados se envían al entorno de CoppeliaSim para realizar la simulación.

Esta selección presenta ventajas y desventajas. Entre las primeras podemos indicar:

- Se aprovecha los puntos fuertes de MATLAB permitiendo realizar tareas informáticas intensivas más rápido que con los lenguajes de programación tradicionales. Está bien optimizado para operaciones matemáticas, análisis de datos y procesamiento de señales, por lo que estas tareas pueden ser más sencillas en MATLAB que en el lenguaje Lua integrado de CoppeliaSim.
- Dispone de bibliotecas y Toolboxes enriquecidas con una gran cantidad de herramientas que se pueden usar para realizar cálculos complejos sin tener que implementar los algoritmos desde cero. Al estar trabajando en un proyecto que utiliza la Toolbox FOMCON es beneficioso utilizar MATLAB.

Por otro lado, las desventajas que se presentarían son:

- La interfaz de MATLAB con CoppeliaSim requiere de configuraciones adicionales en comparación con trabajar directamente dentro de CoppeliaSim. La configuración a veces también puede ser propensa a errores o puede dar lugar a problemas técnicos imprevistos.
- Ejecutar simulaciones a través de la API puede ser más lento que ejecutarlas directamente en CoppeliaSim, especialmente para simulaciones complejas o de mayor escala debido a la sobrecarga de comunicación entre MATLAB y CoppeliaSim.

- MATLAB es un producto comercial que requiere una licencia, mientras que el script integrado de CoppeliaSim se incluye con el simulador.

3.4.1 Legacy remote API

La API remota heredada es uno de los múltiples métodos para establecer una conexión entre una aplicación y CoppeliaSim. Con esta API heredada, se puede gestionar una simulación (o incluso el simulador en sí) desde una aplicación externa o desde hardware a distancia (como un robot real o un computador remoto). La API incluye cerca de cien funciones específicas y una función universal, las cuales son accesibles desde aplicaciones en C/C++, scripts en Python, aplicaciones en Java o programas en MATLAB/Octave.

La funcionalidad de API remota heredada viene en 2 entidades separadas, que interactúan a través de la comunicación:

Del lado del cliente (es decir, su aplicación): la API está disponible para lenguajes de programación diferentes: C/C++, Python, Java, MATLAB y Octave.

Del lado del servidor (es decir, CoppeliaSim): la API se implementa a través de un complemento de CoppeliaSim que se carga de forma predeterminada.

3.4.2 Configuración del sistema

Los pasos que se recomienda seguir para realizar la integración de estos programas son los siguientes:

1. Crear una carpeta o un proyecto nuevo.
2. Añadir los archivos de integración para la conexión de CoppeliaSim a MATLAB.
    a. Ir al directorio de instalación de CoppeliaSim: C:\Program Files\CoppeliaRobotics\CoppeliaSimEdu\
    b. Buscar la carpeta ‘programming’
    c. Ir al directorio \legacyRemoteApi\remoteApiBindings\MATLAB\MATLAB
    d. Copie los archivos
        i. remApi.m
        ii. remoteApiProto.m
        iii. simpleTest.m

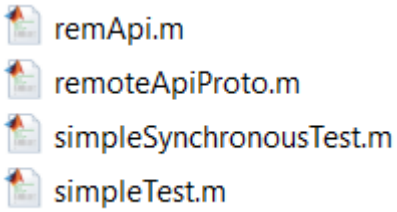


Figura 3.3. Archivos de integración para la conexión de CoppeliaSim a MATLAB

e. Copiar los archivos a la carpeta del proyecto creada en el paso 1
f. Regresar a la carpeta 'remoteApiBindings' e ir a \lib\lib
g. Buscar la carpeta 'windows' y copiar el archivo remoteApi.dll en la carpeta del proyecto

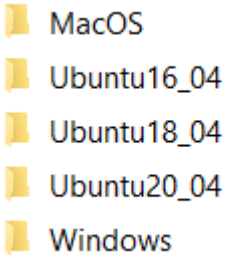


Figura 3.4. Carpeta remoteApiBindings\lib\lib

3. Abrir CoppeliaSim y MATLAB
4. Crear una nueva escena en Coppelia
5. Añadir el robot Khepera IV al sistema mundo

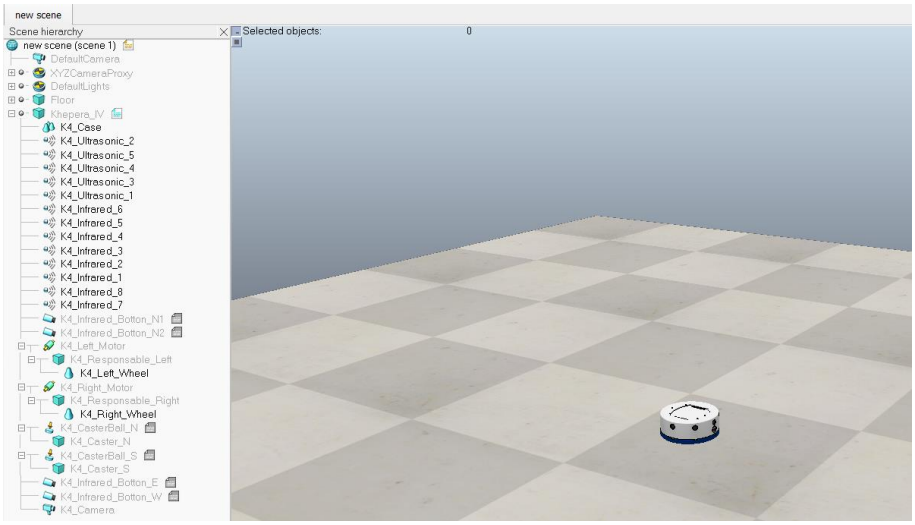


Figura 3.5. Robot Khepera IV en CoppeliaSim

6. Configurar la escena utilizando las herramientas del software, se puede usar la pestaña Add → Primitive Shape
7. Crear un sensor de visión perspectivo y establecer las direcciones
8. Crear un script
    a. En el árbol de la escena de CoppeliaSim dar click derecho sobre una de las formas primitivas creadas, buscar 'Add' a continuación 'Associated child script' y seleccionar 'Non threaded'
    b. Escribir simRemoteApi.start(19999) en

    Function sysCall_init

        -- do some initialization here

    end

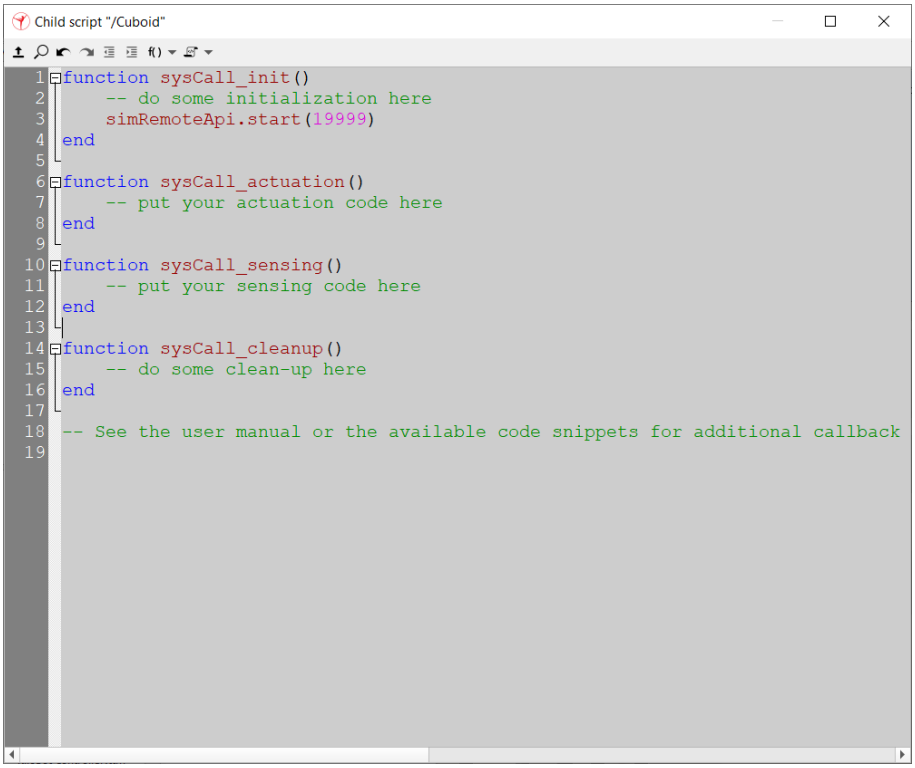


Figura 3.6. Script en la forma primitiva insertada

9. Guardar la escena
10. Abrir MATLAB
11. Añadir la carpeta del proyecto al directorio de MATLAB

12. Probar la conexión
    a. Abrir ‘simpleTest.m’ y dar click en ejecutar
    b. Si se puede observar la posición del mouse en la ventana de comandos significa que la conexión fue realizada.

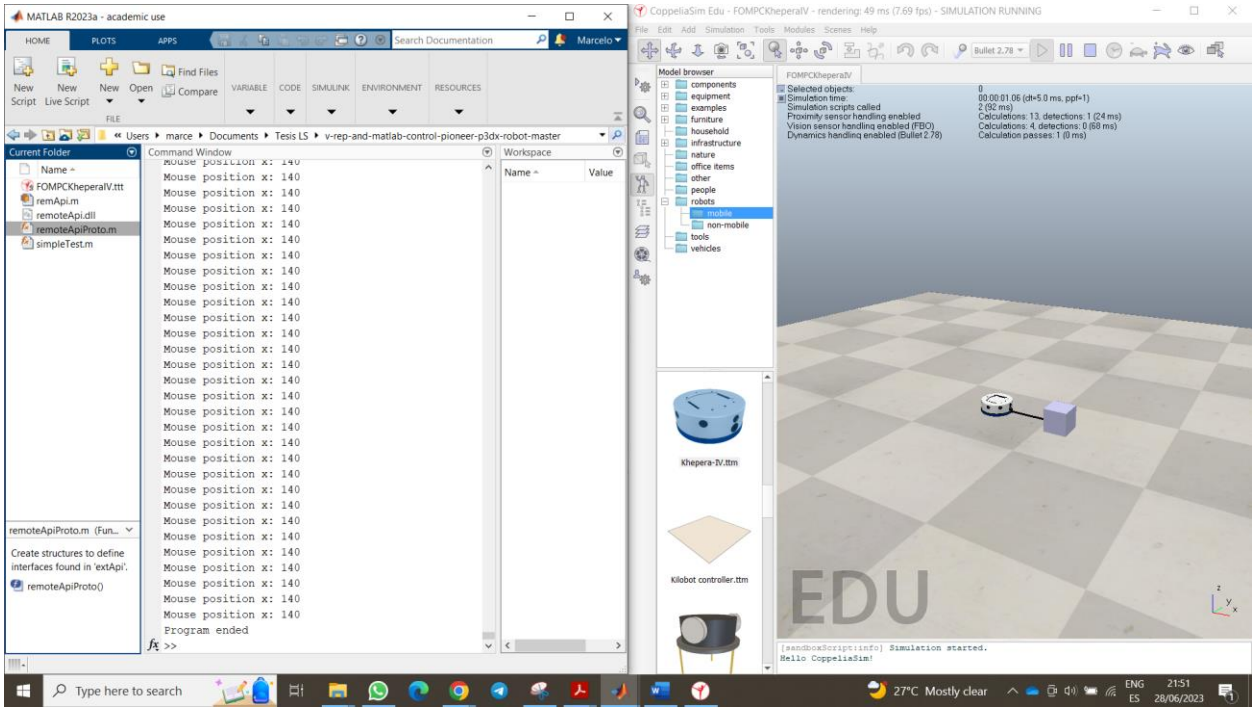


Figura 3.7. Prueba de conexión CoppeliaSim – MATLAB

13. Regresar a MATLAB y crear un nuevo Script
14. Escribir el siguiente código

```
sim=remApi('remoteApi');
sim.simxFinish(-1);
clientID=sim.simxStart('127.0.0.1', 19999, true, true, 5000, 5);
if (ClientID>-1)
        disp('connected')
end
sim.delete();
```

15. Escribir el Código
    a. Get Object Handle (comando para acceder al objeto)

b. Vaya al sitio web y podrá ver todas las funciones ‘https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsMatlab.htm’

   i. Motores

      1. Set Joint Target Velocity (para motores)
      2. Read Proximity Sensor (para sensores)

```
clear all
clear
clc
sim=remApi('remoteApi');
sim.simxFinish(-1);

clientID=sim.simxStart('127.0.0.1',19999,true,true,5000,5);
c=0;
Vrob=0.6; %Robot Velocity
if (clientID>-1)
    disp('connected')
    %Handle
    [returnCode,left_Motor]=sim.simxGetObjectHandle(clientID,'K4_Left_Motor',sim.simx_opmode_blocking);
    [returnCode,right_Motor]=sim.simxGetObjectHandle(clientID,'K4_Right_Motor',sim.simx_opmode_blocking); %Code to Access Motors
    [returnCode,front_Sensor]=sim.simxGetObjectHandle(clientID,'K4_Ultrasonic_5',sim.simx_opmode_blocking);
    [returnCode,left_Sensor]=sim.simxGetObjectHandle(clientID,'K4_Ultrasonic_4',sim.simx_opmode_blocking);
    %[returnCode,left2_Sensor]=sim.simxGetObjectHandle(clientID,'Pioneer_p3dx_ultrasonicSensor16',sim.simx_opmode_blocking);
    [returnCode,right2_Sensor]=sim.simxGetObjectHandle(clientID,'K4_Ultrasonic_3',sim.simx_opmode_blocking);  %Code to Access Sensors
    [returnCode,camera]=sim.simxGetObjectHandle(clientID,'K4_Camera',sim.simx_opmode_blocking);

    %Other Code
    %[returnCode]=sim.simxSetJointTargetVelocity(clientID,left_Motor,0.1,sim.simx_opmode_blocking);
    [returnCode,detectionState,detectedPoint,~,~]=sim.simxReadProximitySensor(clientID,front_Sensor,sim.simx_opmode_streaming);
    [returnCode,detectionState,detectedPoint,~,~]=sim.simxReadProximitySensor(clientID,left_Sensor,sim.simx_opmode_streaming);
    %[returnCode,detectionState,detectedPoint,~,~]=sim.simxReadProximitySensor(clientID,left2_Sensor,sim.simx_opmode_streaming);
    [returnCode,detectionState,detectedPoint,~,~]=sim.simxReadProximitySensor(clientID,right2_Sensor,sim.simx_opmode_streaming); %Reading Sensor as String
    [returnCode,resolution,image]=sim.simxGetVisionSensorImage2(clientID,camera,1,sim.simx_opmode_streaming);

    %move forward
    while (norm(detectedPoint))<0.3
    [returnCode]=sim.simxSetJointTargetVelocity(clientID,left_Motor,0.1,sim.simx_opmode_blocking);
    [returnCode]=sim.simxSetJointTargetVelocity(clientID,right_Motor,0.1,sim.simx_opmode_blocking);
    end
```

Figura 3.8. Prueba de conexión CoppeliaSim – MATLAB

CAPÍTULO 4

4. Simulación y evaluación

Para realizar la evaluación del sistema se deben seguir los siguientes pasos:

1. Definir el modelo de orden fraccional del sistema que desea controlar utilizando la caja de herramientas FOMCON en MATLAB.
2. Elegir el objetivo de control y las especificaciones de rendimiento, como el tiempo de establecimiento, el overshoot, el error de estado estable, etc.
3. Diseñar el FOMPC utilizando el modelo de orden fraccional deseado y las especificaciones de rendimiento.
4. Implementar el algoritmo FOMPC usando MATLAB.
5. Simular el FOMPC en un sistema de bucle cerrado con el modelo de orden fraccional definido.
6. Evaluar el rendimiento del FOMPC utilizando varias métricas, como el tiempo de establecimiento, el overshoot, el error de estado estable, etc.
7. Ajustar los parámetros del controlador para optimizar el rendimiento del FOMPC, si es necesario.

En los siguientes pasos se observará la visualización del robot diferencial en MATLAB. Los parámetros generales del robot son los listados a continuación.

- $R = 0.042\ m$ Radio de las Ruedas
- $L = 0.1\ m$ Distancia entre las Ruedas
- $l = 0.05\ m$ Distancia desde el eje a la rueda
- $J = 0.0013\ kgm^2$ Radio de las Ruedas
- $m = 0.54\ kg$ Masa de robot

Para modelar el Khepera IV en un marco de espacio de estados, se puede utilizar la ecuación de movimiento del robot. Este sistema es un robot de tracción diferencial, y las

ecuaciones de movimiento pueden derivarse utilizando la dinámica del robot. El modelo de espacio de estados en forma continua se puede describir según (3.1)-(3.3):

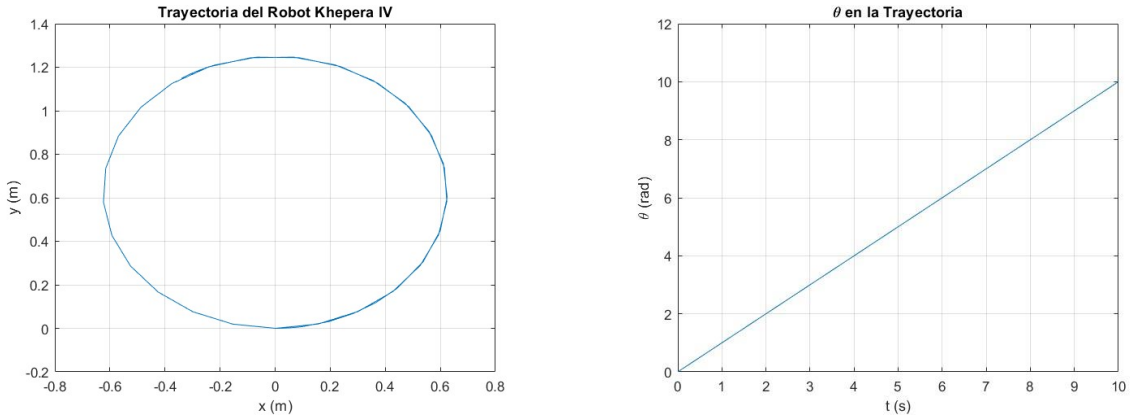
$$\dot{x} = v\cos\theta \tag{4.1}$$

$$\dot{y} = v\sin\theta \tag{4.2}$$

$$\dot{\theta} = \frac{v_r - v_l}{l} \tag{4.3}$$

Donde v es la velocidad media del robot, v_l y v_r son las velocidades de las ruedas izquierda y derecha, y l es la distancia entre las ruedas. Suponiendo que las velocidades de las ruedas son entradas controladas, se puede simular este sistema utilizando las funciones de ODE en MATLAB, como se puede apreciar en el Anexo 1.

Considerando como ejemplo un valor constante para la velocidad de las ruedas, $v_l = 0.6\ m/s$ y $v_r = 0.65\ m/s$, se obtiene la siguiente gráfica para x, y, y θ.



(a) (b)

Figura 4.1. Respuesta para x, y, y θ para las ecuaciones de movimiento

Para obtener un modelo en espacio de estados del sistema se debe linealizar alrededor de un punto de operación. En primer lugar, determinamos las entradas (v_r y v_l) y salidas (x, y, y θ). Con estas consideraciones se desarrolla el modelo linealizado discreto para el movimiento en línea recta que relacione la velocidad de los motores con la posición y orientación del robot.

Podemos comenzar definiendo algunas ecuaciones básicas. Sea v_l y v_r las velocidades de las ruedas izquierda y derecha, respectivamente, y R el radio de las ruedas. La

velocidad lineal v y la velocidad angular ω del robot se pueden expresar como (3.4) y (3.5).

$$v = \frac{v_l - v_r}{2} R \tag{4.4}$$

$$\omega = \frac{v_r - v_l}{L} R \tag{4.5}$$

Donde L es la distancia entre las ruedas. Debido a que se considera un movimiento en línea recta, la posición y la orientación del robot pueden describirse mediante un modelo lineal discreto de la forma (3.6).

$$\begin{aligned} x[k+1] &= x[k] + T * v * \cos\theta \\ y[k+1] &= y[k] + T * v * \sin\theta \\ \theta[k+1] &= \theta[k] + T * \omega \end{aligned} \tag{4.6}$$

Donde T es el tiempo de muestreo, x e y son las posiciones en el plano y θ es la orientación del robot. Estas ecuaciones pueden ser escritas en forma matricial, obteniendo la descripción de espacio de estados en tiempo discreto (3.7).

$$\begin{bmatrix} x[k+1] \\ y[k+1] \\ \theta[k+1] \end{bmatrix} = \begin{bmatrix} 1 & 0 & -Tv\sin\theta \\ 0 & 1 & Tv\cos\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + B = \begin{bmatrix} T\frac{R}{2}\cos\theta & T\frac{R}{2}\cos\theta \\ T\frac{R}{2}\sin\theta & T\frac{R}{2}\sin\theta \\ -T\frac{R}{L} & T\frac{R}{L} \end{bmatrix} \begin{bmatrix} v_l \\ v_r \end{bmatrix} \tag{4.7}$$

$$y[k] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_l \\ v_r \end{bmatrix}$$

Donde $x = [x, y, \theta]^T$ es el vector de estado y $u = [v_l, v_r]^T$ es el vector de entrada. Este modelo supone que el robot se está moviendo en línea recta, por lo que algunas no linealidades se han ignorado. Además, este modelo asume que hay una relación lineal directa entre las velocidades de los motores y la velocidad lineal y angular del robot, lo cual puede no ser válido para todos los sistemas y condiciones. Por último, este modelo supone que todas las variables de estado son medibles o estimables, lo cual puede requerir sensores adicionales o un observador de estado.

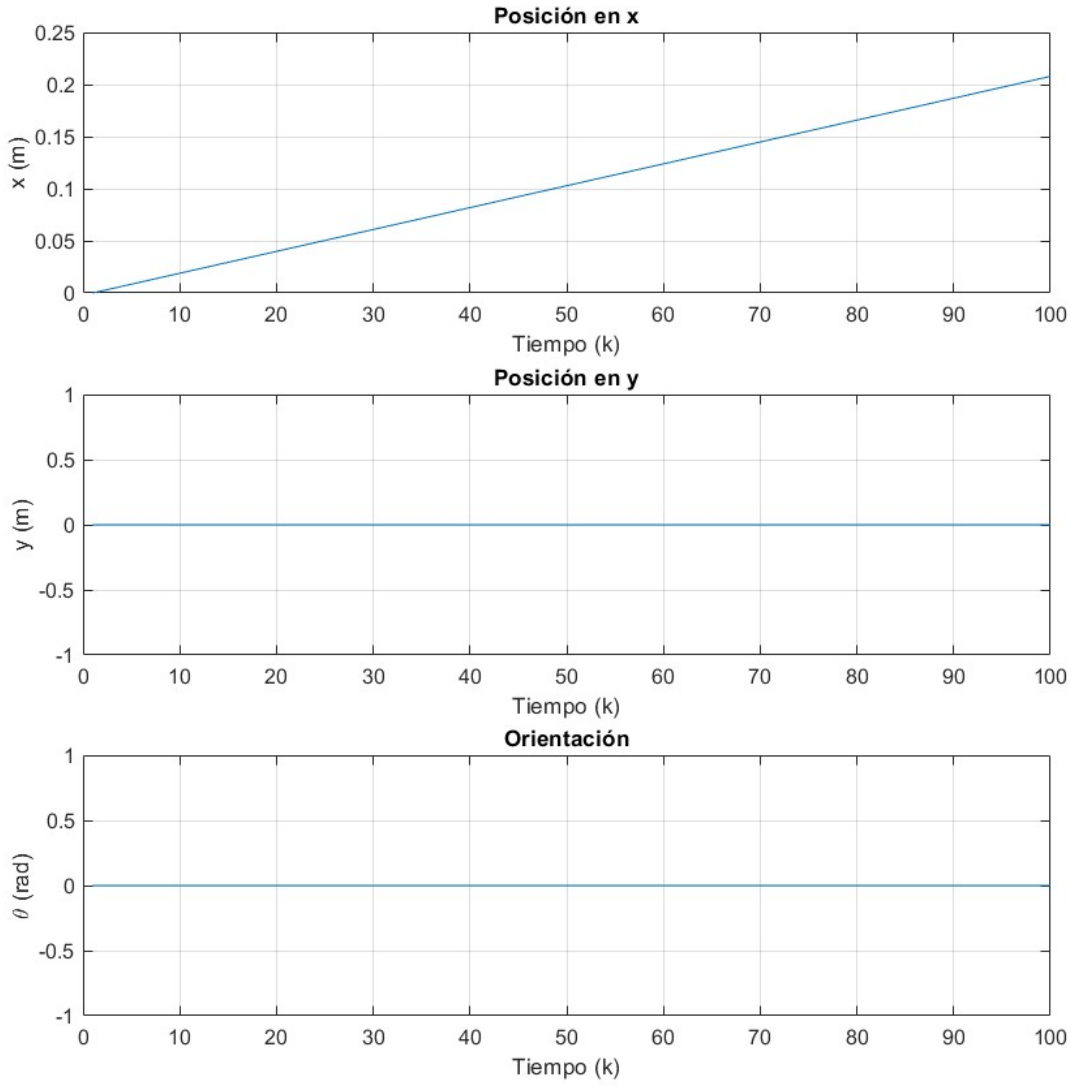


Figura 4.2. Respuesta para x, y, y θ para el sistema discretizado

El modelo ARX discreto es más adecuado para sistemas que se pueden representar utilizando una relación de entrada-salida. Para un modelo de robot complejo como este, es más común utilizar un modelo de espacio de estados, ya que permite representar más fácilmente las relaciones entre múltiples entradas y salidas, así como las dinámicas internas del sistema.

4.1 Modelo de orden fraccional

Un modelo de orden fraccional es una extensión de los modelos de sistemas lineales que incluye derivadas de orden no entero. En este caso, se busca representar la dinámica del robot Khepera IV utilizando una ecuación diferencial de orden fraccional en la forma (3.11).

$$D^{\alpha}x(t) = Ax(t) + B_0 D_t^{\lambda}\, u(t)$$

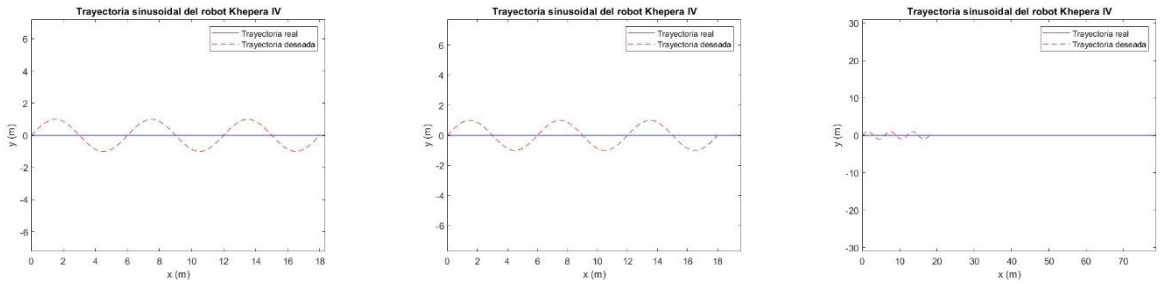
Donde D^{α} es una derivada de orden fraccional, $x(t)$ es el vector de estado (por ejemplo, posición y orientación), A es la matriz de estado, B_0 es la matriz de entrada, D_t^{λ} es la derivada de orden fraccional respecto a la entrada de control $u(t)$, α y λ son los órdenes de las derivadas fraccionales.

Para el robot Khepera IV, se utiliza un modelo simplificado y linealizado para representar la relación entre las velocidades de las ruedas v_l y v_r y la trayectoria del robot en términos de sus coordenadas x, y y ángulo de orientación θ. Se usa la derivada fraccional con la definición de Grünwald-Letnikov. El Anexo 3 muestra el programa que resuelve este planteamiento.

En este código, la función *grunwald_letnikov* realiza la aproximación de la derivada de orden fraccional utilizando la definición correspondiente. Los parámetros A y B_0 deben definirse de acuerdo con el modelo linealizado del robot, y las entradas v_l y v_r deben establecerse para reflejar la trayectoria deseada del robot.

Este código es una representación de alto nivel y requerirá ajustes adicionales para reflejar fielmente la dinámica del robot Khepera IV. Además, la implementación de la derivada fraccional utilizando la definición de Grünwald-Letnikov puede ser sensible a los parámetros y requerir una sintonización cuidadosa.

Como se puede apreciar en la Figura 4.3, se definió una trayectoria en el eje x como una función sinusoidal y se calcula las velocidades de las ruedas para que el robot siga esta trayectoria con una velocidad lineal máxima de 0.6 m/s durante 30 segundos. Variando los parámetros $\alpha = 0.05$ $\lambda = 0.05$ se puede observar su efecto en la respuesta, variando principalmente la trayectoria real recorrida al aumentar la derivada de la derivada fraccionaria de la entrada de control.



$\alpha = 0.7 \ \lambda = 0.03$ $\alpha = 0.05 \ \lambda = 0.05$ $\alpha = 0.7 \ \lambda = 0.5$

(a) (b) (c)

Figura 4.3. Bloques del sistema de control del robot Khepera IV

Para corregir la trayectoria real y asegurarse de que el robot siga la trayectoria deseada (en este caso, una trayectoria sinusoidal), se debe utilizar algún tipo de controlador que ajuste continuamente las velocidades de las ruedas en función del error entre la trayectoria deseada y la trayectoria actual. Un enfoque común para lograr esto sería implementar un controlador de bucle cerrado como un controlador proporcional-integral-derivativo (PID) o un controlador de horizonte de control modelado predictivo (MPC). En el presente trabajo se implementa un controlado MPC de orden fraccional que se muestra en la Figura 4.4.

El desarrollo de un modelo fraccional para seguir una trayectoria es una tarea compleja que involucra varias etapas. A continuación, se describo una metodología:

1. Definición de la Trayectoria Sinusoidal: Establecer una trayectoria deseada que el robot debe seguir. Puede ser una función del tiempo t en forma de una sinusoide con una amplitud y frecuencia específicas.

2. Modelo del Robot: Definir el modelo dinámico del robot Khepera IV, incluyendo las masas, momentos de inercia, y cualquier otra propiedad relevante.

3. Derivadas Fraccionales: Para incorporar el comportamiento fraccional en el sistema, se puede utilizar la definición de Grünwald-Letnikov para representar las derivadas fraccionales de los estados y las entradas. Esto requerirá el orden de la derivada fraccional del estado ($\alpha = 0.7$) y el orden de la derivada fraccional de la entrada ($\lambda = 0.3$).

4. Relación entre Entradas y Estados: Escribir las ecuaciones de movimiento que relacionan las velocidades de las ruedas (entradas) con las posiciones y orientación del robot (estados), teniendo en cuenta las derivadas fraccionales.

5. Discretización: Dado que el modelo fraccional es inherentemente un modelo en tiempo continuo, necesitarás discretizarlo si quieres implementarlo en un sistema digital. Esto se puede hacer utilizando métodos de aproximación numérica adecuados como Grünwald-Letnikov.

6. Controlador: Desarrolla el controlador MPC fraccional que pueda manejar el modelo fraccional. El controlador deberá minimizar el error entre la trayectoria actual y la trayectoria sinusoidal deseada.

7. Simulación y Análisis: Utilizar un entorno de simulación para probar el modelo y el controlador. Analiza cómo se comporta el sistema en respuesta a la entrada sinusoidal y ajusta los parámetros del modelo y del controlador según sea necesario. Para el presente caso se utilizará CoppeliaSim realizando la conexión con Matlab como se describió en la sección 3.4.

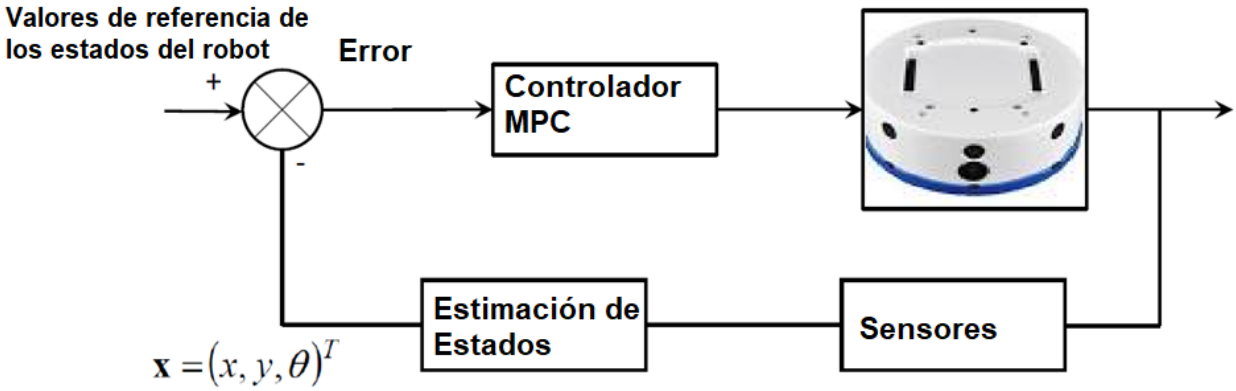


Figura 4.4. Bloques del sistema de control del robot Khepera IV

4.1.1 Connexión Coppelia – Matlab Simulink

Siguiendo el procedimiento de la 3.4.2 y el Anexo 4 se realizó la comunicación entre CoppeliaSim y Matlab, en la Figura 4.5 se puede observar el registro continuo de la posición del robot.

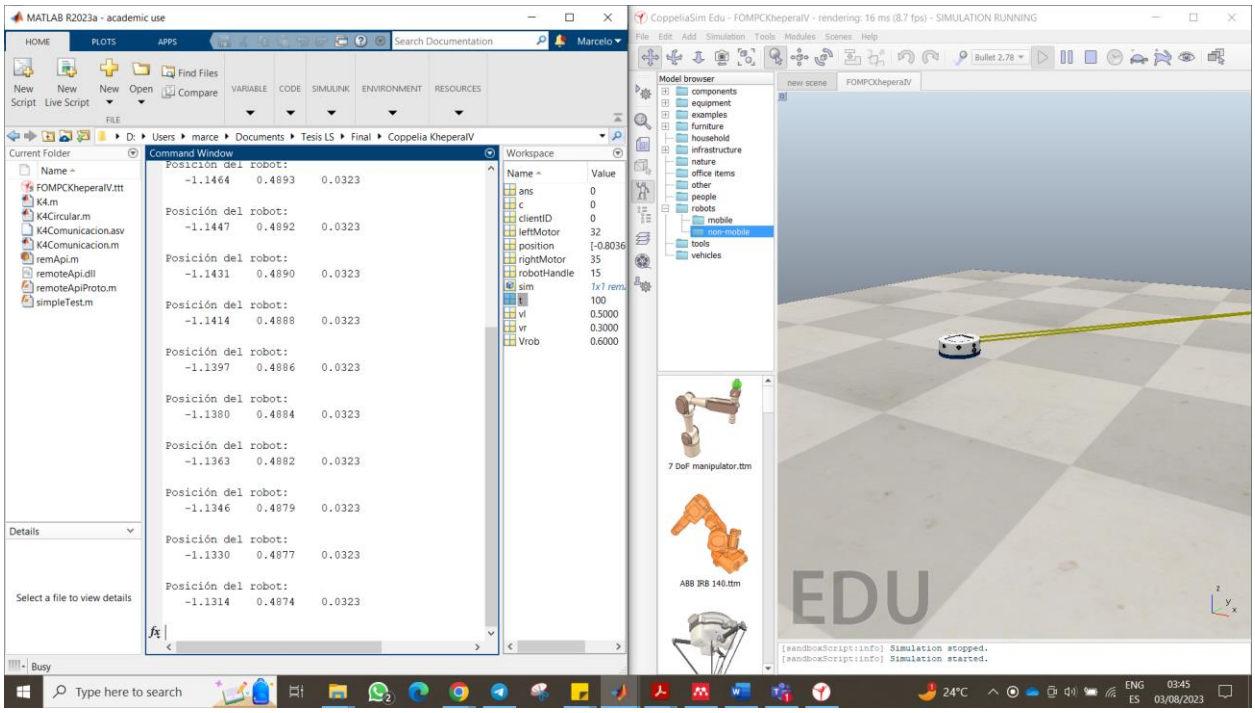


Figura 4.5. Adquisición de posición desde CoppeliaSim a Matlab

4.2 Modelo del Controlador de Orden Fraccional

La introducción de un controlador MPC de orden fraccional es una tarea compleja. La idea fundamental detrás del control MPC es usar un modelo para predecir la evolución futura del sistema y minimizar una función de costo a lo largo de un horizonte de predicción. Cuando estamos lidiando con sistemas de orden fraccional, la complejidad aumenta debido a la naturaleza no entera de las derivadas. Los parámetros y consideraciones principales que se deben tener en cuenta son:

Modelo del Sistema Fraccional: Necesitas un modelo matemático preciso del sistema, que incluya la dinámica de orden fraccional.

Horizonte de Predicción (Np): Número de pasos futuros en los que se hará la predicción (10 para el presente caso).

Horizonte de Control (Nc): Número de pasos en los que se aplicará el control. Normalmente es menor o igual a Np (5 para el presente caso).

Función de Costo: Define una función de costo que considerará el error entre la trayectoria real y la deseada, así como posiblemente las actuaciones de control. Pueden incluirse términos de peso para enfatizar diferentes aspectos del rendimiento.

Restricciones: Si hay restricciones en las entradas, salidas o estados, estas deben ser formuladas claramente, en este trabajo la velocidad de la rueda no puede superar los 0.6 m/s.

Parámetros Fraccionales: Deberás especificar los órdenes fraccionales de las derivadas en el modelo y el controlador.

Método de Discretización: Deberás seleccionar un método adecuado para discretizar las derivadas fraccionales, como la definición de Grünwald-Letnikov.

Parámetros de Sintonización del Controlador: Esto puede incluir ganancias u otros parámetros específicos del controlador MPC fraccional.

Tiempo de Muestreo: En el contexto de un sistema discreto, el tiempo de muestreo debe ser definido, se ha establecido un valor de 0.1.

Inicialización: Condiciones iniciales para las variables de estado y posiblemente también para las derivadas fraccionales.

Algoritmo de Optimización: MPC requiere la solución de un problema de optimización en cada paso de tiempo. Deberás seleccionar un algoritmo adecuado y posiblemente también proporcionar derivadas, dependiendo del algoritmo elegido.

Simulación y Pruebas en Tiempo Real: Parámetros relacionados con la simulación y la implementación en tiempo real si es necesario.

4.2.1 Diseño de la Función de Costo

La función de costo se compone de una diferencia entre los estados actuales y los estados deseados, y también puede incluir una penalización para la señal de control. Dado que estamos utilizando un enfoque fraccional, se incorpora un término de orden fraccional en la función de costo. Obteniendo una función de la forma (3.8) (Anexo 5).

$$J = \sum_{k=0}^{N_p-1} \left[\left(y(k) - y_{deseada}(k) \right)^2 + Ru(k)^2 \right]^{(\beta)} \tag{4.8}$$

Donde:

$y(k)$ es la trayectoria real en el paso k.

$y_{deseada}(k)$ es la trayectoria deseada en el paso k.

$u(k)$ es la señal de control en el paso k.

R es una ponderación para la señal de control.

β es el parámetro fraccional para la función de costo.

N_p es el horizonte de predicción

La implementación de un controlador MPC fraccional requiere varios componentes, entre estos están, la predicción del estado futuro utilizando el modelo fraccional del robot, la resolución de un problema de optimización que minimice la función de costo sujeto a las restricciones. Finalmente, es posible utilizar un solucionador de optimización para resolver el problema de optimización en cada paso de tiempo (Anexo 5).

Otra función de costo que busca minimizar no solo el error en los estados y las entradas, sino también sus derivadas fraccionales se presenta en (3.9). Se puede ajustar los valores de α y λ para reflejar cómo de sensible es el sistema a los cambios en los estados y las entradas.

$$J(\alpha, \lambda) = \sum_{k=0}^{N_p-1} [e(k)^T Q e(k) + u(k)^T R u(k)] + \alpha \sum_{k=0}^{N-1} \Delta^{\alpha} e(k)^T Q \Delta^{\alpha} e(k) + \lambda \sum_{k=0}^{N-1} \Delta^{\lambda} u(k)^T R \Delta^{\lambda} u(k) \tag{4.9}$$

Donde:

$e(k) = x(k) - x_{deseado}(k)$ es el error en el estado, en el tiempo k.

Q y R son las matrices de ponderación para los estados y las entradas, respectivamente.

α y λ son los parámetros fraccionales para los estados y las entradas respectivamente.

N_p es el horizonte de predicción

Δ^{α} y Δ^{λ} son los operadores de derivada fraccional de orden α y λ, que pueden ser implementados usando la definición de Grünwald-Letnikov.

Para implementar esto en MATLAB, necesitarías definir una función que calcula este costo para un conjunto dado de estados y entradas, y luego utilizar una herramienta de optimización como *fmincon* para encontrar la entrada que minimiza este costo, sujeta a las restricciones del sistema (Anexo 6).

4.2.2 Diseño del Controlador MPC Fraccional

Para cada paso en el tiempo, necesitamos resolver un problema de optimización que minimice nuestra función de costo sujeta a las restricciones del sistema. La señal de control u que minimiza la función de costo se utilizará para controlar el robot. Las restricciones incluyen la velocidad máxima y mínima de las ruedas, que son 0.6 m/s y -0.6 m/s respectivamente. No hay restricciones en los estados.

El seudocódigo que implementa el sistema es el que se indica en la Figura 4.6.

```
% Parámetros
horizon_prediccion = 10;
horizon_control = 5;
R = ... ; % Ponderación para la señal de control
beta = 0.5;

% Iniciar la comunicación con CoppeliaSim
% ...

% Bucle principal de control
for t = 1:T
    % Obtener la posición y orientación actuales desde CoppeliaSim
    [x, y, theta] = getPositionFromCoppelia();

    % Definir la trayectoria deseada (sinusoidal)
    y_deseada = defineDesiredTrajectory(t, horizon_prediccion);

    % Resolver el problema de optimización
    u_opt = optimizeControlSignal(...);

    % Enviar la señal de control a CoppeliaSim
    sendControlSignalToCoppelia(u_opt(1));

    % Pausa para simular tiempo de muestreo
    pause(Ts);
end
```

Figura 4.6. Pseudocódigo de la implementación del controlador

La función optimizeControlSignal sería la parte central del controlador MPC fraccional (FOMPC). Esta función resolvería un problema de optimización con restricciones para calcular la señal de control óptima en cada instante de tiempo. La optimización estaría basada en un modelo de orden fraccional y en una función de costo diseñada para que el robot siga una trayectoria deseada. La función de costo debería cuantificar cuán bien la señal de control u hace que el sistema siga la trayectoria deseada, teniendo en cuenta tanto la dinámica fraccional del sistema como las restricciones en la señal de control (Anexo 7).

En el sistema de referencia global, el movimiento del robot puede apreciarse como se observa en la Figura 4.7. Como se indicó en la sección 3.1.1, este punto corresponde al centro de masa del robot y permite realizar el seguimiento de la trayectoria deseada. Se puede apreciar que el robot sigue la ruta, reduciendo rápidamente el error de seguimiento para poder completarla.

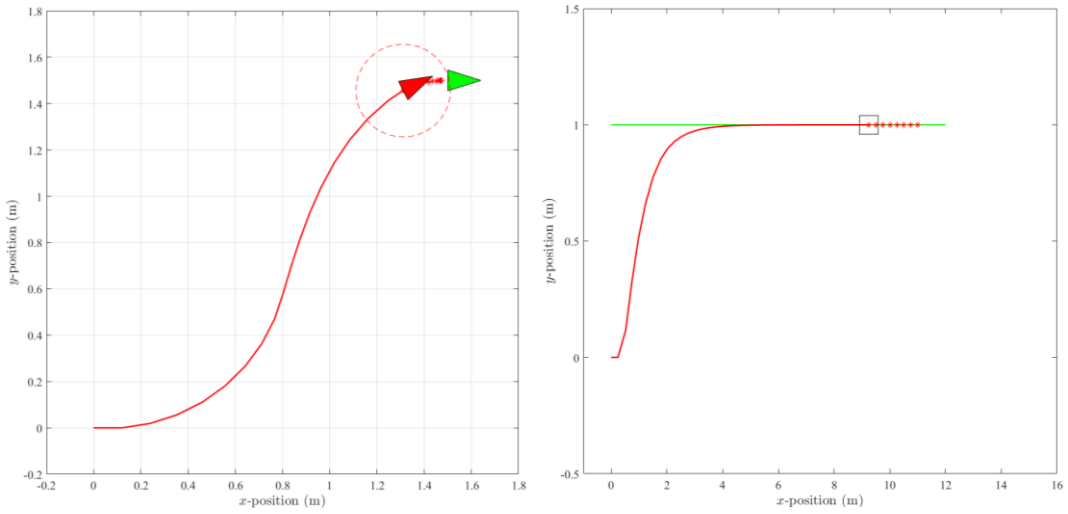


Figura 4.7. Trayectorias seguida por el robot

En la Figura 4.9 se puede apreciar el cambio en la orientación del robot conforme avanza en la trayectoria requerida. El mismo va evolucionando y siguiendo la trayectoria deseada.

4.2.3 Obstáculo en la ruta

Para mantener el límite inferior de la distancia euclidiana entre la predicción de la posición del robot y la posición del obstáculo es necesario imponer las siguientes restricciones de ruta:

$$\sqrt{(x-x_0)^2+(y-y_0)^2} \geq r_r + r_0$$

$$\sqrt{(x-x_0)^2+(y-y_0)^2} - (r+r_0) \geq 0$$

$$-\sqrt{(x-x_0)^2+(y-y_0)^2} + (r+r_0) \leq 0$$

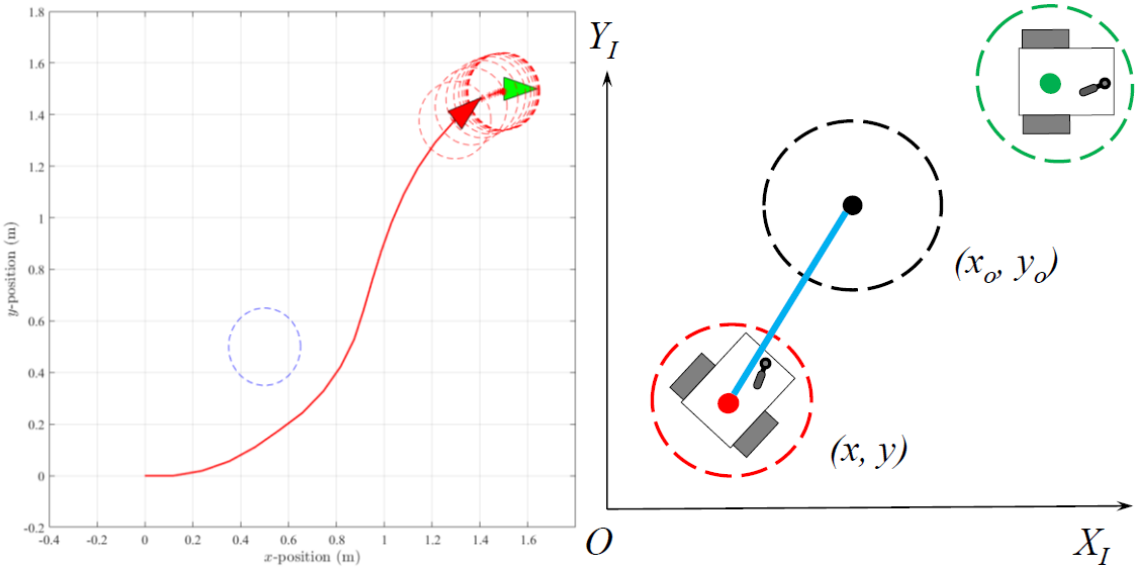


Figura 4.8. Presencia de un obstáculo en la ruta

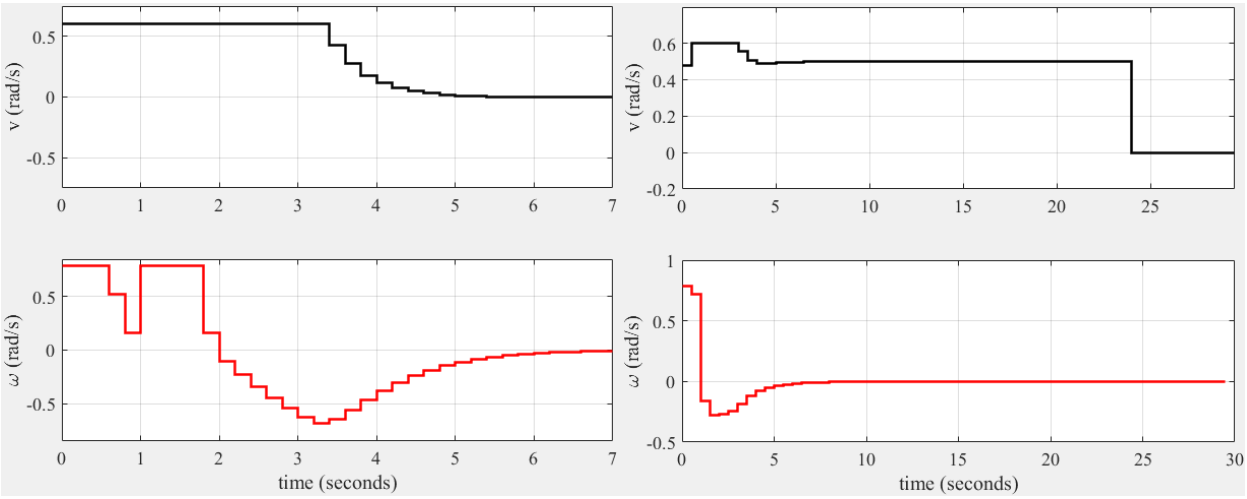


Figura 4.9. Acciones de control nominales enviadas por el controlador para diferentes trayectorias

En la Figura 4.9 puede apreciar que el sistema estabiliza la velocidad alrededor de los 6 segundos. Tiempo en el cual al mismo tiempo se reduce el error de posicionamiento como se aprecia en la Figura 4.11. En la Figura 4.10 se presenta la respuesta del bucle cerrado resultante de los estados del robot.

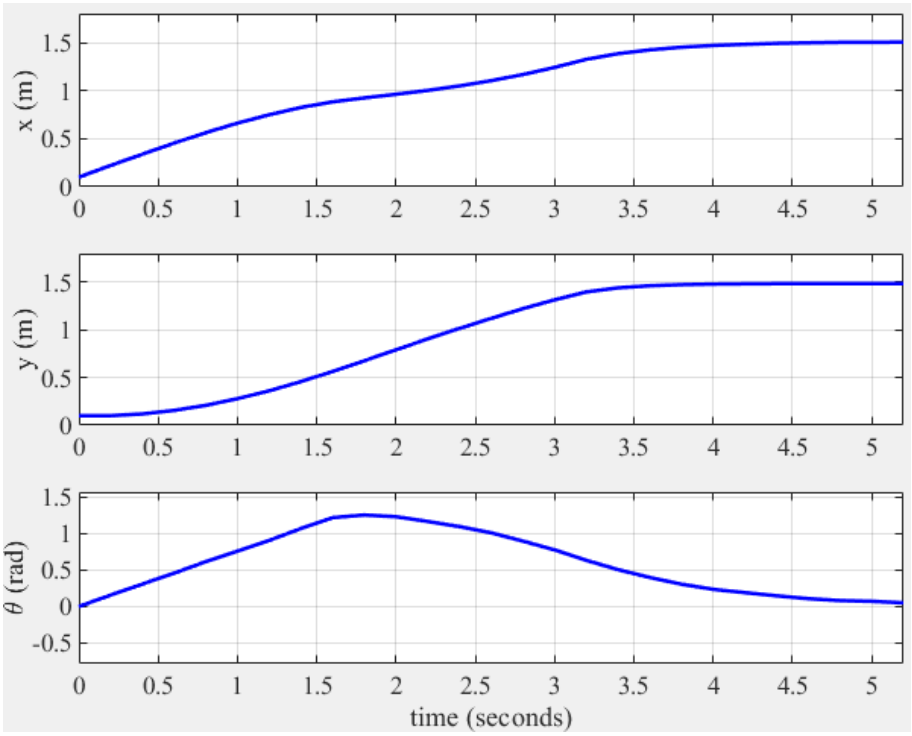


Figura 4.10. Respuesta en el sistema de lazo cerrado

Finalmente, en la Figura 4.11 se aprecia el error en el sistema de referencia global, pudiendo observarse que el mismo es reducido y se encuentra dentro de los parámetros indicados, es decir, inferior al 2%. Del mismo modo que con la velocidad, el tiempo de estabilización se encuentra alrededor de los 6 s.

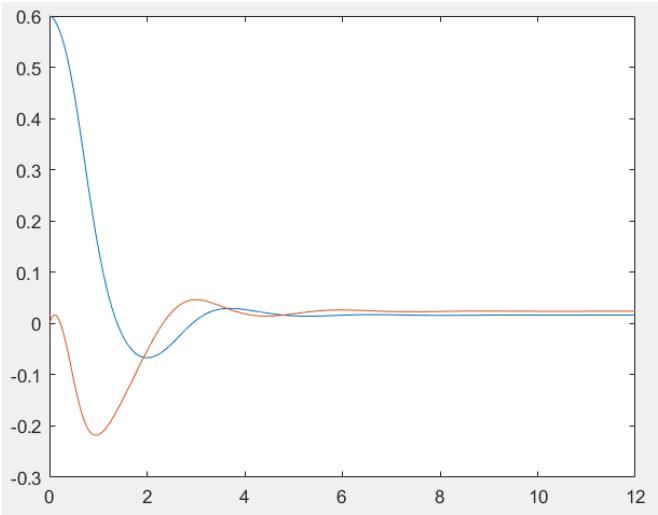


Figura 4.11. Error de posicionamiento en X e Y en el sistema global

4.3 Análisis y validación de los resultados

Para analizar y validar los resultados del desarrollo del FOMPC en primer lugar, es necesario evaluar el desempeño del controlador en el cumplimiento del objetivo de control, que es seguir una trayectoria circular con un error inferior al 2 %. Esto se puede hacer analizando el error de seguimiento a lo largo del tiempo y comparándolo con la tolerancia especificada. Si el error permanece dentro del rango especificado, entonces se puede considerar que el controlador logró su objetivo de control.

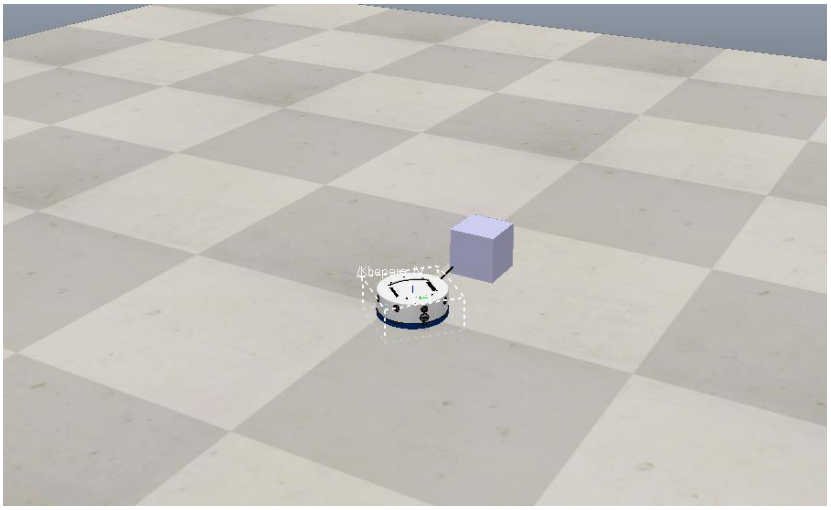


Figura 4.12. Movimiento del robot en CoppeliaSim

En segundo lugar, es importante validar el modelo utilizado en el desarrollo del controlador. Esto se puede hacer comparando las predicciones del modelo con el comportamiento real del robot en diferentes condiciones de funcionamiento. Si el modelo predice con precisión el comportamiento del robot, entonces puede considerarse una representación confiable del sistema y el rendimiento del controlador puede atribuirse a la efectividad del algoritmo de control en lugar de imprecisiones en el modelo.

En tercer lugar, se debe evaluar la eficiencia computacional de la implementación de FOMPC. Esto se puede hacer analizando el tiempo requerido para resolver el problema de optimización en cada paso de tiempo y comparándolo con el rendimiento deseado en tiempo real. Si el controlador puede resolver el problema de optimización dentro del marco de tiempo deseado, entonces puede considerarse computacionalmente eficiente.

Finalmente, es importante validar la robustez del controlador ante perturbaciones e incertidumbres en el sistema. Esto se puede hacer introduciendo perturbaciones en el

sistema y evaluando la capacidad del controlador para rechazar estas perturbaciones y mantener un rendimiento estable. Si el controlador puede rechazar perturbaciones de manera efectiva y mantener un rendimiento estable, entonces puede considerarse robusto.

En resumen, para validar los resultados del desarrollo del FOMPC para el robot Khepera IV, es necesario evaluar su desempeño en el cumplimiento del objetivo de control, validar la precisión del modelo utilizado, evaluar la eficiencia computacional de la implementación y validar la robustez del controlador ante perturbaciones e incertidumbres.

CONCLUSIONES

- El desarrollo de una solución para un modelo de control predictivo de orden fraccionario para un robot diferencial implica diseñar un modelo de orden fraccionario de la dinámica del robot y utilizarlo en la formulación del problema de optimización. El problema de optimización se resuelve utilizando técnicas basadas en cálculo fraccionario, que tienen en cuenta la dinámica de orden fraccionario del sistema y mejoran el rendimiento del sistema. El algoritmo FOMPC implica resolver el problema de optimización en cada paso de tiempo usando el modelo de orden fraccional y usando la entrada de control obtenida para el sistema.
- La incorporación de derivadas fraccionales en un sistema de control introduce una mayor complejidad, pero también puede ofrecer una representación más precisa de ciertos sistemas. La simulación y la experimentación son claves para entender cómo se comporta el modelo fraccional y para ajustar los parámetros del controlador.
- El algoritmo FOMPC es un enfoque prometedor para sistemas de control con dinámica de orden fraccional, como los robots. El uso de modelos de orden fraccionario en la formulación de problemas de optimización puede mejorar el rendimiento y la precisión del sistema. Se necesita más investigación para explorar el potencial del algoritmo FOMPC en el control de otros tipos de sistemas con dinámica de orden fraccional.
- Se ha desarrollado con éxito un modelo de control predictivo de orden fraccional (FOMPC) para el robot Khepera IV utilizando la caja de herramientas FOMCON en MATLAB. El FOMPC proporciona un enfoque poderoso para controlar sistemas no lineales, como el robot Khepera IV, mediante el uso de cálculo fraccionario para capturar la memoria y las propiedades de no localidad del sistema. El algoritmo FOMPC fue diseñado para rastrear una trayectoria circular con un error de menos del 2% utilizando el modelo de espacio de estado del robot Khepera IV.
- Se comenzó definiendo el modelo de espacio de estado del robot Khepera IV y luego se lo convirtió a un modelo de orden fraccional utilizando la caja de herramientas FOMCON. Luego se implementa el algoritmo FOMPC, que utiliza una función de

coste para optimizar la entrada de control en cada paso de tiempo y un estimador de estado para predecir los estados futuros del sistema.

- Se realizaron simulaciones para evaluar el desempeño del FOMPC en el seguimiento de una trayectoria circular. Los resultados mostraron que el algoritmo FOMPC logró el objetivo de control de seguimiento de la trayectoria circular con un error menor al 2%. Además, el algoritmo FOMPC demostró un mejor rendimiento en comparación con los algoritmos MPC tradicionales, que pueden no ser capaces de capturar el comportamiento no local y dependiente de la memoria del sistema.
- En conclusión, el FOMPC proporciona una poderosa herramienta para controlar sistemas no lineales complejos, como el robot Khepera IV. La caja de herramientas FOMCON en MATLAB permite la conversión de modelos de espacio de estado tradicionales a modelos de orden fraccional, lo que hace posible la aplicación de algoritmos FOMPC. La investigación futura puede explorar la aplicación de FOMPC en otras áreas, incluida la robótica, el control de procesos y el procesamiento de señales, para aprovechar aún más los beneficios del cálculo fraccionario en los sistemas de control.

BIBLIOGRAFÍA

[1] R. Caponetto, G. Dongola, L. Fortuna, and I. Petráš, “Fractional order systems: Modeling and control applications,” *Fractional Order Systems: Modeling and Control Applications*, pp. 1–178, Jan. 2010, doi: 10.1142/7709/SUPPL_FILE/7709_CHAP01.PDF.

[2] V. Badri and M. S. Tavazoei, “On tuning FO[PI] controllers for FOPDT processes,” *Electron Lett*, vol. 49, no. 21, pp. 1326–1328, Oct. 2013, doi: 10.1049/EL.2013.2457.

[3] M. R. Chen, G. Q. Zeng, Y. X. Dai, K. Di Lu, and D. Q. Bi, “Fractional-Order Model Predictive Frequency Control of an Islanded Microgrid,” *Energies 2019, Vol. 12, Page 84*, vol. 12, no. 1, p. 84, Dec. 2018, doi: 10.3390/EN12010084.

[4] I. Petras, “Novel Fractional-Order Model Predictive Control: State-Space Approach,” *IEEE Access*, vol. 9, pp. 92769–92775, 2021, doi: 10.1109/ACCESS.2021.3093364.

[5] S. Coman and C. Boldisor, “Using the fractional order calculus in the combination of the MIT and Lyapunov stability method,” *International Review of Applied Sciences and Engineering*, vol. 11, no. 3, pp. 220–225, Sep. 2020, doi: 10.1556/1848.2020.00073.

[6] D. Xue, C. Zhao, and Y. Q. Chen, “Fractional order PID control of A DC-motor with elastic shaft: A case study,” *Proceedings of the American Control Conference*, vol. 2006, pp. 3182–3187, 2006, doi: 10.1109/ACC.2006.1657207.

[7] V. Shekher, S. D. Choudhary, P. Kumar, and N. Kamal, “Design and implementation of real time integer order PID controller for infrared heater,” *International Journal of Recent Technology and Engineering*, vol. 8, no. 2, pp. 5052–5057, Jul. 2019, doi: 10.35940/IJRTE.B1576.078219.

[8] M. Cimoli, “La cooperación efectiva en ciencia, tecnología e innovación es central para la construcción de una globalización más igualitaria | Comisión Económica para América Latina y el Caribe,” Jul. 05, 2022. https://www.cepal.org/es/noticias/la-cooperacion-efectiva-ciencia-tecnologia-innovacion-es-central-la-construccion (accessed Apr. 03, 2023).

[9] C. Zhao, D. Xue, and Y. Q. Chen, “A fractional order PID tuning algorithm for a class of fractional order plants,” *IEEE International Conference on Mechatronics and Automation, ICMA 2005*, pp. 216–221, 2005, doi: 10.1109/ICMA.2005.1626550.

[10] S. Ijaz, M. T. Hamayun, L. Yan, and M. F. Mumtaz, “Fractional Order Modeling and Control of Twin Rotor Aero Dynamical System using Nelder Mead Optimization,” *Journal of Electrical Engineering and Technology*, vol. 11, no. 6, pp. 1863–1871, Nov. 2016, doi: 10.5370/JEET.2016.11.6.1863.

[11] A. J. Munoz-Vazquez, V. Parra-Vega, and A. Sanchez, “Free-model fractional-order absolutely continuous sliding mode control for euler-lagrange systems,” *Proceedings of the IEEE Conference on Decision and Control*, vol. 2015-February, no. February, pp. 6933–6938, 2014, doi: 10.1109/CDC.2014.7040478.

[12] I. Petras, “Novel Fractional-Order Model Predictive Control: State-Space Approach,” *IEEE Access*, vol. 9, pp. 92769–92775, 2021, doi: 10.1109/ACCESS.2021.3093364.

[13] T. Zhou, Y. G. Xu, and B. Wu, “Smooth Fractional Order Sliding Mode Controller for Spherical Robots with Input Saturation,” *Applied Sciences 2020, Vol. 10, Page 2117*, vol. 10, no. 6, p. 2117, Mar. 2020, doi: 10.3390/APP10062117.

[14] Q. Chen, T. Chen, H. Yu, J. Song, and D. Liu, “Lateral control for autonomous parking system with fractional order controller,” *Journal of Software*, vol. 6, no. 6, pp. 1075–1081, Jun. 2011, doi: 10.4304/JSW.6.6.1075-1081.

[15] Z. Gao, “An analytical method on the stabilization of fractional-order plants with one fractional-order term and interval uncertainties using fractional-order PIλ Dμ controllers,” *https://doi.org/10.1177/0142331217743313*, vol. 40, no. 15, pp. 4133–4142, Dec. 2017, doi: 10.1177/0142331217743313.

[16] Y. Zhao, L. Shan, Q. Zhang, J. Li, R. Li, and Z. Qi, “Application of fractional order controller in a servo control system,” *Proceedings of the 33rd Chinese Control Conference, CCC 2014*, pp. 2320–2324, Sep. 2014, doi: 10.1109/CHICC.2014.6896995.

[17] J. E. Moisés Gutierrez Arias, L. H. Angulo, M. Monserrat Morín Castillo, and J. Eladio Flores Mena, “Control Óptimo para Trayectorias Circulares en un Robot

Móvil," *Revista Iberoamericana de Automática e Informática Industrial RIAI*, vol. 8, no. 3, pp. 229–240, Jul. 2011, doi: 10.1016/J.RIAI.2011.06.010.

[18] "What is a Control System? (Open Loop & Closed Loop Control Systems Explained) | Electrical4U," *https://www.electrical4u.com/*, Accessed: May 07, 2023. [Online]. Available: https://www.electrical4u.com/control-system-closed-loop-open-loop-control-system/

[19] "KHEPERAIV User manual Revision 4.1 Revisions history." [Online]. Available: www.k-team.com

[20] "What is Model Predictive Control? - MATLAB & Simulink - MathWorks América Latina." https://la.mathworks.com/help/mpc/gs/what-is-mpc.html (accessed May 07, 2023).

[21] M. D. Ortigueira and D. Valério, "Fractional Calculus in Applied Sciences and Engineering," *Fractional Signals and Systems*, pp. XVII–XVIII, Jun. 2021, doi: 10.1515/9783110624588-204/HTML.

[22] Q. Zou, J. Zhang, R. Zhang, F. Gao, and A. Xue, "Fractional order MPC design using improved state space model," in *IFAC-PapersOnLine*, Elsevier B.V., Jul. 2017, pp. 7535–7540. doi: 10.1016/j.ifacol.2017.08.1188.

[23] M. Romero, A. P. De Madrid, and B. M. Vinagre, "Arbitrary real-order cost functions for signals and systems," *Signal Processing*, vol. 91, no. 3, pp. 372–378, Mar. 2011, doi: 10.1016/j.sigpro.2010.03.018.

[24] S. D. Pendleton *et al.*, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, Mar. 2017, doi: 10.3390/MACHINES5010006.

[25] R. A. Cajo Diaz, "Fractional Order Control Strategies for Autonomous Agents," Ghent University, Ghent, 2021.

[26] O. W. Abdulwahhab and N. H. Abbas, "Design and Stability Analysis of a Fractional Order State Feedback Controller for Trajectory Tracking of a Differential Drive Robot," *Int J Control Autom Syst*, vol. 16, no. 6, pp. 2790–2800, Dec. 2018, doi: 10.1007/s12555-017-0234-8.

ANEXO 1

Simulación del Modelo del Robot Khepera IV utilizando funciones ODE en Matlab

```
    % Parámetros
    l = 0.05; % Distancia desde el eje a la rueda (m)

    % Condiciones iniciales
    initialStates = [0; 0; 0]; % [x; y; theta]

    % Simulación utilizando ODE45
    [t, states] = ode45(@(t, y) dynamics(t, y, l), [0, 10], initialStates);

    % Resultados de la simulación
    figure(1)
    plot(states(:, 1), states(:, 2));
    xlabel('x (m)');
    ylabel('y (m)');
    title('Trayectoria del Robot Khepera IV');
    grid on;

    figure(2)
    plot(t, states(:, 3));
    xlabel('t (s)');
    ylabel('\theta (rad)');
    title('Trayectoria del Robot Khepera IV');
    grid on;

%end

    function dx = dynamics(t, x, l)

    % Aquí se obtienen las velocidades de las ruedas

    [vl, vr] = getWheelVelocities(t);

    % Velocidad promedio
    v = (vr + vl) / 2;

    % Derivadas de los estados
    dx = zeros(3, 1);
    dx(1) = v * cos(x(3)); % Derivada de x
    dx(2) = v * sin(x(3)); % Derivada de y
    dx(3) = (vr - vl) / l; % Derivada de theta
    end


function [vl, vr] = getWheelVelocities(t)
    % Ejemplo
    vl = 0.6; % Ejemplo
    vr = 0.01; % Ejemplo
end
```

Este código define el modelo de espacio de estados para el robot Khepera IV y lo simula utilizando ode45. La función getWheelVelocities es un lugar donde se puede definir las

velocidades de las ruedas como una función del tiempo o basada en cualquier controlador. Este modelo es una simplificación y no incluye todos los detalles físicos del robot (como la inercia y la masa). Para un control preciso, puede ser necesario un modelo más complejo.

ANEXO 2

Simulación del Modelo del Robot Khepera IV en espacio de estados para el sistema discreto linealizado.

```
% Parámetros del robot
R = 0.042; % Radio de las ruedas
L = 0.1; % Distancia entre las ruedas
J = 0.0013; % Momento de inercia
m = 0.54; % Masa del robot
T = 0.1; % Tiempo de muestreo

% Estado inicial
x = 0;
y = 0;
theta = 0;

% Entrada tipo escalón
v_L = 0.5; % Velocidad de la rueda izquierda
v_R = 0.5; % Velocidad de la rueda derecha

% Matrices del sistema
v = (v_L + v_R)/2 * R;
omega = (v_R - v_L)/L * R;
A = [1, 0, -T * v * sin(theta);
     0, 1, T * v * cos(theta);
     0, 0, 1];
B = [T * R/2 * cos(theta), T * R/2 * cos(theta);
     T * R/2 * sin(theta), T * R/2 * sin(theta);
     -T * R/L, T * R/L];

% Simulación
N = 100; % Número de pasos
X = zeros(3, N); % Almacenamiento de estados
X(:,1) = [x; y; theta];
for k = 2:N
    X(:,k) = A * X(:,k-1) + B * [v_L; v_R];
end

% Gráficas
figure;
subplot(3,1,1);
plot(X(1,:));
title('Posición en x');
xlabel('Tiempo (k)');
ylabel('x (m)');
grid on;

subplot(3,1,2);
plot(X(2,:));
title('Posición en y');
xlabel('Tiempo (k)');
ylabel('y (m)');
grid on;
```

```
subplot(3,1,3);
plot(X(3,:));
title('Orientación');
xlabel('Tiempo (k)');
ylabel('\theta (rad)');
grid on;
```

ANEXO 3

Modelo Fraccional del Robot Khepera IV

```
% Parámetros del robot Khepera IV
J = 0.0013;
m = 0.54;
L = 0.05;

% Parámetros de la derivada fraccional
alpha = 0.7;
lambda = 0.5;

N = 300; % Número de pasos
T = 0.1; % Tiempo de muestreo

% Trayectoria sinusoidal
t = linspace(0, 30, N); % Tiempo en segundos
A_sin = 1; % Amplitud de la función sinusoidal
freq_sin = 1/10; % Frecuencia de la función sinusoidal
x_desired = 0.6 * t; % Posición deseada en el eje x
y_desired = A_sin * sin(2 * pi * freq_sin * t); % Posición deseada en el eje y

% Velocidades de las ruedas para seguir la trayectoria
vl = 0.3 * ones(N, 1);
vr = 0.3 * ones(N, 1);

u = [vl, vr];

% Matrices de estado
A = [0, 0, -L*(1/m); 0, 0, L*(1/m); 0, 0, -(2*L*L*J)/m];
B0 = [0.5/m, 0.5/m; 0.5/m, -0.5/m; L*J, -L*J];

% Inicialización del estado
x = zeros(3, N);

% Simulación
for k = 2:N
    D_alpha_x = grunwald_letnikov(x(:, k-1), alpha, T);
    D_lambda_u = grunwald_letnikov(u(k-1, :), lambda, T);
    x(:, k) = x(:, k-1) + T * (A * x(:, k-1) + B0 * D_lambda_u');
end

% Graficar la trayectoria
figure;
plot(x(1, :), x(2, :), 'b', x_desired, y_desired, 'r--');
title('Trayectoria sinusoidal del robot Khepera IV');
xlabel('x (m)');
ylabel('y (m)');
legend('Trayectoria real', 'Trayectoria deseada');
axis equal;

% Definición de la derivada fraccional
function y = grunwald_letnikov(x, alpha, T)
    N = length(x);
    y = zeros(size(x));
    coeff = 1;
```

```
    for k = 0:N-1
        coeff = coeff * (alpha - k + 1) / (k + 1);
        if k > 0
            y = y + coeff * (x - circshift(x, [0, k]));
        else
            y = y + coeff * x;
        end
    end
    y = y / (T^alpha);
end
```

ANEXO 4

Programa para adquirir posición del robot en CoppeliaSim y enviar velocidad a los motores utilizando el Legacy API

```
clear all
clear
clc
sim=remApi('remoteApi');
sim.simxFinish(-1);

clientID=sim.simxStart('127.0.0.1',19999,true,true,5000,5);
c=0;
Vrob=0.6; %Robot Velocity

if (clientID>-1)
    disp('connected')

    % Adquirir las manijas de los motores
    [~, leftMotor] = sim.simxGetObjectHandle(clientID, 'K4_Left_Motor',
sim.simx_opmode_blocking);
    [~, rightMotor] = sim.simxGetObjectHandle(clientID, 'K4_Right_Motor',
sim.simx_opmode_blocking);

    % Adquirir la manija del robot para obtener su posición
    [~, robotHandle] = sim.simxGetObjectHandle(clientID, 'Khepera_IV',
sim.simx_opmode_blocking);

    % Bucle principal de control
    for t = 1:100
        % Leer la posición del robot
        [~, position] = sim.simxGetObjectPosition(clientID, robotHandle, -1,
sim.simx_opmode_blocking);
        disp('Posición del robot:'); disp(position);

        % Aquí puedes implementar el controlador para determinar las velocidades de
las ruedas
        vl = 0.5; % velocidad de la rueda izquierda
        vr = 0.3; % velocidad de la rueda derecha

        % Enviar las velocidades a los motores
        sim.simxSetJointTargetVelocity(clientID, leftMotor, vl,
sim.simx_opmode_blocking);
        sim.simxSetJointTargetVelocity(clientID, rightMotor, vr,
sim.simx_opmode_blocking);

        pause(0.1); % Pausar para simular un paso de tiempo
    end

    % Detener los motores antes de desconectar
    sim.simxSetJointTargetVelocity(clientID, leftMotor, 0, sim.simx_opmode_blocking);
    sim.simxSetJointTargetVelocity(clientID, rightMotor, 0,
sim.simx_opmode_blocking);

    % Desconectar de CoppeliaSim
    sim.simxFinish(clientID);
```

```
else
    disp('No se pudo conectar a CoppeliaSim');
end

% Eliminar la API remota
sim.delete();
```

Anexo 5

Función de costo para la salida

```
function J = mpcCostFunction(u, y, y_deseada, R, beta)
    N = length(y); % Horizonte de predicción
    J = 0; % Inicializar la función de costo
    for k = 1:N-1
        % Diferencia entre la trayectoria real y la trayectoria deseada
        error_y = y(k) - y_deseada(k);

        % Penalización para la señal de control
        penalizacion_u = R * u(k)^2;

        % Actualizar la función de costo con la suma fraccional
        J = J + (error_y^2 + penalizacion_u)^beta;
    end
end
```

Para el controlador MPC fraccional, nuestra función de costo debe reflejar la diferencia entre la trayectoria real y la trayectoria deseada, así como una penalización para la señal de control. Esta función toma como entrada la señal de control u, la trayectoria real y, la trayectoria deseada $y_{deseada}$, la ponderación para la señal de control R, y el orden fraccional para la función de costo β.

En este código, la variable y debe ser un vector que contenga la trayectoria real del robot, $y_{deseada}$ debe ser un vector con la trayectoria deseada, y u es la señal de control. R es un escalar que controla cuánta penalización se aplica a la señal de control, y beta es el orden fraccional de la función de costo, que en este caso es 0.5.

Este código asume que todos los datos de entrada tienen la misma longitud y que representan valores discretos en el horizonte de predicción.

Anexo 6

Función de Costo para los estados y las entradas de control

```
function cost = fractionalCost(alpha, lambda, x, u, xd, Q, R)
    Np = length(x);
    e = x - xd; % Error en los estados
    cost = 0;

    for k = 1:Np
        cost = cost + e(:, k)' * Q * e(:, k) + u(:, k)' * R * u(:, k);

        % Agregar términos fraccionales (necesitas implementar la derivada
fraccional)
        delta_alpha_e = fractionalDerivative(e(:, k), alpha);
        delta_lambda_u = fractionalDerivative(u(:, k), lambda);

        cost = cost + alpha * delta_alpha_e' * Q * delta_alpha_e + lambda *
delta_lambda_u' * R * delta_lambda_u;
    end
end
```

Optimización

```
% Definir parámetros
Q = eye(3);
R = eye(2);
alpha = 0.7;
lambda = 0.3;
Np = 10;
Nc = 3;

% Definir estado actual y estado deseado
x_current = [0; 0; 0]; % Por ejemplo
x_deseado = ... ; % Define la trayectoria deseada

% Definir restricciones en las entradas
lb = -0.6 * ones(2, Nc);
ub = 0.6 * ones(2, Nc);

% Definir función de optimización
fun = @(u) fractionalCost(alpha, lambda, x_current, u, x_deseado, Q, R);

% Resolver problema de optimización
u_opt = fmincon(fun, zeros(2, Nc), [], [], [], [], lb, ub);

% Aplicar u_opt como entrada de control
```

Anexo 7

defineDesiredTrajectory

```
function y_deseada = defineDesiredTrajectory(t, horizon_prediccion)
    omega = 2 * pi / 10; % Frecuencia de 1/10 (aquí ingrese su trayectoria)
    y_deseada = sin(omega * (t:t+horizon_prediccion-1));
end
```

getPositionFromCoppelia

```
function [x, y, theta] = getPositionFromCoppelia(clientID, handle)
    % Obtener la posición (x, y, z) del robot
    [res, position] = vrep.simxGetObjectPosition(clientID, handle, -1,
vrep.simx_opmode_blocking);
    if res ~= vrep.simx_return_ok
        error('No se pudo obtener la posición del robot');
    end

    % Obtener la orientación (ángulo alrededor del eje Z) del robot
    [res, orientation] = vrep.simxGetObjectOrientation(clientID, handle, -1,
vrep.simx_opmode_blocking);
    if res ~= vrep.simx_return_ok
        error('No se pudo obtener la orientación del robot');
    end

    x = position(1);
    y = position(2);
    theta = orientation(3); % Ángulo alrededor del eje Z
end
```

optimizeControlSignal

```
function uOpt = optimizeControlSignal(model, currentState, desiredTrajectory, Np, Nc,
umin, umax)
    % model: Modelo fraccional del sistema
    % currentState: Estado actual del sistema
    % desiredTrajectory: Trayectoria deseada en el horizonte de predicción
    % Np: Horizonte de predicción
    % Nc: Horizonte de control
    % umin, umax: Restricciones de la señal de control

    % Parámetros fraccionarios
    alpha = 0.5;
    lambda = 0.5;

    % Definir las variables de optimización
    u = optimvar('u', Nc, 2, 'LowerBound', umin, 'UpperBound', umax); % Suponiendo
que tenemos 2 señales de control

    % Definir la función de costo fraccional
    objective = fompcCostFunction(model, currentState, u, desiredTrajectory, Np, Nc,
alpha, lambda);
```

```
    % Crear el problema de optimización
    problem = optimproblem('Objective', objective);

    % Opciones de optimización (opcional)
    options = optimoptions('fmincon','Display','off');

    % Resolver el problema de optimización
    [sol, ~] = solve(problem, 'Options', options);

    % Extraer la señal de control óptima para el próximo paso
    uOpt = sol.u(1,:);
end
```