BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA ELECTRÓNICA

MAESTRÍA EN CIENCIAS DE LA ELECTRÓNICA
OPCION EN AUTOMATIZACIÓN

Control predictivo para la evasión de obstáculos en cobots

T E S I S

PRESENTADA PARA OBTENER EL TITULO DE:

MAESTRO EN CIENCIAS DE LA ELECTRONICA
OPCION EN AUTOMATIZACION

PRESENTA

ING. RAÚL DÁVILA RUIZ

DIRECTORES DE TESIS

DR. JORGE DIONICIO FIERRO ROJAS

DR. JOSÉ FERNANADO REYES CORTÉS

Puebla, Puebla, Diciembre 2025

BUAP

Agradecimientos

Quiero agradecer a mi familia.
Quienes siempre estuvieron a mi lado apoyándome, procurando mi bienestar al asegurarse de que comiera adecuadamente, procurando que durmiera un mínimo de horas necesarias para no afectar mi desempeño cognitivo, procurandome los días de lluvia para que no me mojase, así como llevándome y en varias ocasiones recogiéndome de la universidad para eficientemente los tiempos de traslado.
A mi padre que me apoyaba realizando tareas de la casa para que tuviera mas tiempo disponible, a mi madre que me esperaba en casa para comer juntos, a mi hermana Pao con la que cocinaba en las noches para tener comida saludable y rica para el lunch de cada día, a mi hermana Cristi con quien me divertí y bromee varias ocaciones en estos 2 años, a mi sobrina Helena quien siempre me a demostrado un gran cariño, muchas gracias y disculpen mis malos momentos a lo largo de estos 2 años.

Agradezco a mis asesores de tesis el Dr. Jorge Dionisio Fierro Rojas y al Dr. José Fernando Reyes Cortés por toda su guía, preparación, paciencia y apoyo. Quiero agradecer a todos mis profesores durante mi preparación en maestría, al Dr. Moises , la Dra. Amparo, al Dr. Sergio, la Dra. Aurora, Al Dr. Zemliak y a la Dra. Josefina, muchas gracias por compartir su tiempo y todos los conocimientos que me han brindado, no solo aquellos relacionados con la maestría, también los que ayudan a mi desarrollo profesional y como persona. Estoy seguro me servirán para toda la vida.

A mis amigos quienes a lo largo de estos 2 y medio años pasamos por muchas cosas y vivimos muchos momentos agradables, nunca olvidare los momentos de ir por comida y comer todos juntos o esos instantes después de acabar los avances de tesis en donde sentíamos que por fin podíamos relajarnos. Como en los primeros semestres realizábamos nuestras sesiones de estudio e intentábamos resolver los problemas que creíamos serian los que vendrían en el examen. Si bien el siguiente año estuvimos mas enfocados en realizar nuestras tesis, siempre buscábamos la oportunidad de convivir, principalmente los viernes de chilaquiles. Agradezco el haberlos conocido y crecido a su lado.

Quiero agradecer a CONAHCyT por la beca recibida para poder realizar mis estudios de Maestría, ya que sin su apoyo este estudio no podría haberse llevado a cabo.

Finalmente, agradezco a dios por la salud y virtud que me da para hacer lo que más me gusta en este mundo.

Resumen

La presente tesis aborda la creación de una plataforma computacional basada en Python para realizar la simulación de evasión de obstáculos. En la cual se cuenta con 2 robots tipo brazo articulado, el primero del tipo revoluta-revoluta (RR) con un movimiento horizontal, que toma el papel de nuestro obstáculo dinámico, mientras que el otro robot, un cobot de 6 grados de libertad, basado en el Robot UR5 de Universal Robot, donde solo se controla la posición del robot convirtiéndolo en un robot de 3 grados. Dotado de un algoritmo de Control Predictivo Basado en Modelos (MPC) con la finalidad de evitar las posibles colisiones con el robot RR. Con la finalidad de simular una interacción real entre ambos robots, se les asigna una tarea de pick-and-place (tomar y colocar), se realizó una planificación de trayectorias para ambos robots con la finalidad de que se intercepten en un punto y momento especifico, donde solo el control MPC del UR5 tiene la capacidad de anticiparlo con el tiempo suficiente para evitar la colisión. Mientras el Robot RR,cuenta con un control PID para seguir la trayectoria establecida.
El MPC dota al cobot de la capacidad de evadir al Robot RR moviéndose a diferentes velocidades, esta capacidad permite incorporar distintos objetos como el obstáculo que se desea evitar. .
Además de la evasión de obstáculos para el robot UR5, se implementó un controlador predictivo (MPC) para el seguimiento de trayectorias del robot RR. Como paso para validar el algoritmo antes de una implementación en un robot físico, el modelo y el controlador se integraron en un sistema embebido: una tarjeta Raspberry Pi 4B. Esta etapa intermedia permitió desarrollar y probar el control en un entorno de hardware dedicado, cerrando la brecha entre la simulación pura y el control de un sistema robótico real.

Índice general

Capítulo 1

Introducción

Los cobots, también conocidos como robots colaborativos, son una de las tecnologías que están adquiriendo una gran atención desde la introducción de Industria 4.0 debido a las ventajas que éstos aportan con respecto a los robots tradicionales. Los cobots permiten la colaboración con los humanos, además de ser más fáciles de usar, tener una mayor flexibilidad y adaptabilidad y tener un mayor número de aplicaciones. Los cobots están diseñados para aumentar las capacidades humanas, no para reemplazarlas. Los cobots en comparación con los robots industriales no necesitan contar con sistemas de protección (en muchos casos jaulas), que los separen del resto de áreas de trabajo. Como se muestra en la Figura 1.1.

Figura 1.1: Comparación de robots industriales y cobots obtenido de ”https://gibas.nl/en/gibas-automation/production-automation/robot-and-cobot/”

Una de las principales características de los cobots, es procurar la seguridad de las personas, esto va de la mano con evitar que el cobot colisione con las personas con las que trabaja.
La evasión de obstáculos es la capacidad de un robot para evitar colisionar con un obje-

to estático o en movimiento. En los últimos años este tema ha tomado gran relevancia, ya que cada vez se busca dotar de mayor autonomía a los robots, salvaguardar la seguridad de las personas y procurar la integridad del robot, asi como la de los objetos manipulados. Para poder llevar a cabo el proceso de evasión, el robot debe de contar con algún sistema que le permita obtener la información de su entorno, por lo que se debe dotar al mismo de sensores, como cámaras estereoscópicas o sensores infrarrojos. Estos le permiten crear un mapa de su entorno para determinar su posición y la posición de los objetos que lo rodean.
Una vez se tiene el mapeo del entorno, se realiza una planificación de trayectoria, la cual consiste en utilizar algoritmos para calcular la mejor trayectoria a su objetivo, con restricciones de tiempo, evitando colisiones. Para poder evadir los obstáculos el robot puede decidir si modifica su velocidad, su trayectoria o incluso si se detiene.
Dos características importantes que se deben considerar en el desempeño de un sistema de control de evasión de obstáculos son que la evasión debe ser precisa y rápida para garantizar la seguridad de las personas y del robot, así como ser robusto para distintos tipos de obstáculos y entornos. Estas características dependen de la capacidad de los sensores para poder detectar adecuadamente su entorno, el método de detección empleado y de la capacidad computacional para poder procesar la información de los sensores en tiempo real.

Unos de los métodos que se utilizan para realizar la detección de obstáculos son el método de Campos Potenciales Artificiales, Mapa de Colisiones Avanzado y Redes Neuronales Artificiales. Tradicionalmente, se emplean métodos reactivos como el control PID o la asignación de polos, los cuales modifican la trayectoria al conocer el error del estado actual del sistema.
Sin embargo, el Control Predictivo Basado en Modelos (MPC), por sus siglas en ingles, representa un enfoque predictivo fundamentalmente diferente. A diferencia de los métodos reactivos que requieren un estímulo inmediato para alterar la trayectoria, el MPC utiliza un modelo interno del sistema para predecir continuamente estados futuros y posibles colisiones.
El MPC, es un método de control basado en el uso de un modelo matemático del sistema, se utiliza para predecir su comportamiento futuro y poder evaluar diferentes secuencias de control, eligiendo aquella que permita minimizar una función de costo que cumpla con las restricciones y objetivos del sistema. Este método de control opera en un horizonte de tiempo de dos pasos: predicción y control. En el paso de control se recalcula el plan de control basado en las mediciones y la predicción a corto plazo.
La función de costo, considera el objetivo que se busca optimizar, como minimizar el error, maximizar la velocidad de la tarea o mantener en un rango determinado los parámetros, además de permitir establecer restricciones en las variables del estado y las entradas.
Para poder realizar parte de la simulación y análisis es necesario recurrir a un prototipado virtual, una técnica en donde se crea y evalúa un prototipo digital de un producto, máquina o sistema. Este prototipo es capaz de reproducir la dinámica de movimiento del sistema real con un grado de exactitud aceptable, utilizando herramientas de diseño asistido por computadora (CAD) y simulaciones de ingeniería (CAE), para diseñar, anali-

zar y validar productos o sistemas antes de construir un prototipo físico. En esta tesis se planea emplear el prototipado virtual para realizar un análisis dinámico y cinemático, sin emplear el modelo analítico.
Para abordar el problema de evasión de obstáculos, es conveniente separar el modelo virtual del control y trabajarlos por separado, esto con la finalidad de aligerar los requerimientos de contar con el modelo matemático del sistema estudiado, se opta por utilizar la cosimulación. La cosimulación, la cual consiste en emplear herramientas computacionales de simulación independientes, ejecutadas simultáneamente, cada una modelando una parte diferente de un sistema complejo y compartiendo datos en tiempo real.
Se ha empleado principalmente en la simulación y diseño de sistemas de control, en el análisis de rendimiento de software, en el modelado y simulación de fenómenos climáticos, sistemas ecológicos y procesos biológicos por mencionar algunos.
En este trabajo de tesis se realiza una plataforma virtual a través de la integración de el entorno de software, Adams View, donde se modelarán los robots con el objetivo principal de obtener su modelo dinámico y cinemático, y Spyder, un entorno de desarrollo integrado (IDE) para Python dentro de la distribución Anaconda con las herramientas para escribir, depurar y ejecutar el lenguaje Python. en un entorno de co-simulación. El objetivo de esta plataforma será la simulación de un cobot de 6 gdl y un robot RR, que deberán realizar una tarea de pick and place, para ambos robots. Al cobot se le dotará de un algoritmo de control MPC para poder llevar a cabo la tarea de evasión de obstáculos, el robot RR realizara el papel de obstáculo dinámico, para simular la presencia de una persona que ingresa al espacio de trabajo del cobot.

1.1. Antecedentes

El problema de evasión de obstáculos en el control de robots manipuladores se ha abordado extensamente con diferentes estrategias, como Redes Neuronales Artificiales [1], Mapas de Colisión, Campos de Potencial Artificial, por citar algunas. Particularmente, el uso del control óptimo en el esquema MPC se ha reportado en varios foros, destacando el enfoque de dos robots en una tarea cooperativa [2]. En esa misma línea, el proyecto de investigación propuesto considera condiciones de cooperación entre dos robots manipuladores, con la diferencia de que cada robot cuenta con una estrategia de control diferente, MPC y PID, respectivamente.

1.1.1. Interacción cobots - humano

Los cobots fueron creados en 1996 por los profesores Edward Colgate y Michael Peshkin como una iniciativa de General Motors. Su patente definió sus cobots como un aparato y método para la interacción física entre una persona y un manipulador de propósito general controlado por una computadora (Figura 1.2). En un principio fueron llamados máquinas de restricción programable resaltando la pasividad y seguridad que éstos garantizaban a los humanos.

La ISO/TC 299 define al cobot como un robot diseñado para interactuar directamente con los humanos en un espacio de trabajo colaborativo.
Basándose en [3] los cobots o robots colaborativos son una categoría de los sistemas robóticos diseñados para trabajar junto a humanos en un espacio compartido. Los cobots están diseñados para colaborar con trabajadores humanos mejorando la eficiencia y la seguridad en el lugar de trabajo.
También se puede definir a los cobots basándose en las regulaciones que dictan y definen su comportamiento, para garantizar un entorno seguro para el ser humano. Las normas ISO 10218-1:2011 e ISO 10218-2:2011 especifican los requisitos y directrices para el diseño seguro, medidas de protección e información para el uso de robots industriales. También describen los peligros asociados con los robots y brindan los requisitos para eliminar o reducir estos peligros [4].
Los principios de seguridad que proponen las normas ISO 10218-1:2011 e ISO 10218-2:2011 se pueden tomar como fundamentos para establecer condiciones seguras en robots no industriales. De esta norma se pueden distinguir 4 tipos de colaboración humano-robot:

1. Parada supervisada por seguridad. En este caso, si el operador entra dentro de un área designada para el robot, el robot se detendrá hasta que el operador salga del área. Tanto el operador como la máquina pueden trabajar en el mismo espacio, pero no al mismo tiempo.

2. Supervisión de velocidad y separación, se emplean escáneres o sistemas de visión para inspeccionar las diferentes zonas en las que se divide el área de trabajo. Se asigna un área en donde el robot se detiene si el operador ingresa, se asigna un área más alejada donde el robot opera con velocidad reducida, si detecta la presencia de un operador, si ambas áreas están libres, el robot trabaja con su parámetros al máximo.

3. Limitación de potencia y fuerza. Este tipo de colaboración se limita a cobots con codificadores con alta resolución, que permitan controlar la velocidad y posición del robot con alta precisión. Se miden y evalúan las fuerzas y los pares con sensores sensibles de par, se mide la energía consumida para identificar colisiones y reaccionar.

4. Guía manual, es cuando el operador desplaza una carga con ayuda de la maquina, el operador puede estar en contacto directo con la máquina, pero el el robot no inicia el movimiento por su cuenta. Este tipo de colaboración, no es exclusivo de los cobots, pero el robot debe contar con algún sensor que detecte cargas externas, además de reducir la velocidad y brindarle una función se seguridad.

Es indispensable que el cobot cuente con sensores, sistemas de control y equipos periféricos para poder cumplir con las condiciones de seguridad.
Por otro lado, la norma ISO/TS 15066:2016 especifica los requisitos de seguridad para sistemas de robots industriales colaborativos y el entorno de trabajo, así como conceptos de colaboración y los requisitos necesarios para lograrlo. Incluso abarca los resultados de un estudio de investigación acerca del umbral de dolor comparado con la velocidad

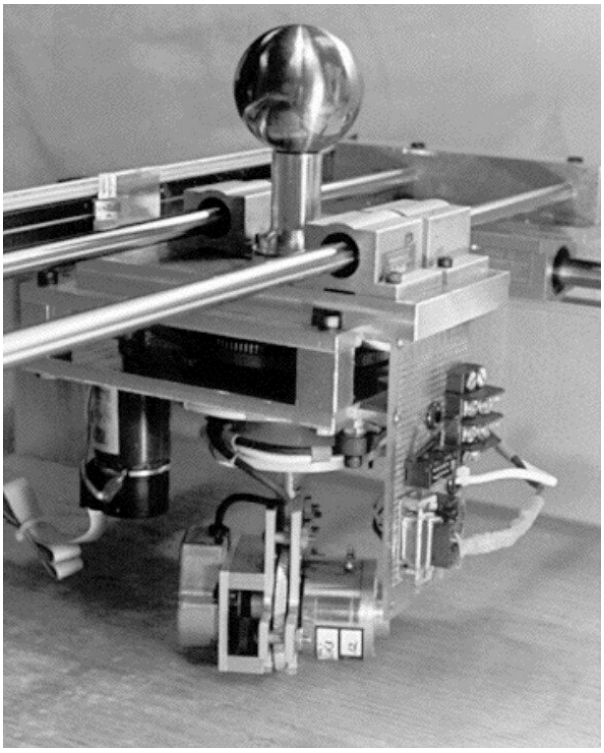


Figura 1.2: Cobot de Edward Colgate y Michael Peshkin obtenido de "$https$: $//peshkin.mech.northwestern.edu/publications/1999_Peshkin_Cobots.pdf$"

del robot, la presión y el impacto en las partes especificas del cuerpo, complementando así los requerimientos planteados en la norma ISO 10218-1 e ISO 10218-2.
Algunas ventajas que ofrece el uso de cobots con respecto a los robots convencionales son:

- Son más fáciles de programar.
- Cuentan con interfaces interactivas.
- No requieren personal especializado.
- Fácil de re-programar.
- Mayor flexibilidad de las tareas que puede realizar.
- Son más pequeños, lo que evita una inversión mayor para adecuar el espacio de trabajo para el cobot.
- Mas fáciles de transportar y manipular.

Tomando en cuenta las ventajas que nos brindan los cobots en la que nos centraremos es la seguridad que le brinda al aperador.

De acuerdo con [5], algunas de las aplicaciones actuales de los cobots son:

- Aplicaciones industriales, como los robots colaborativos de Universal Robots en la linea de producción de BMW.
- Colaboración humano-robot en la linea de ensamblaje de Audi, basado en el uso del robot ”PART4you”, que cuenta con una cámara y una ventosa de succión para hacer más ergonómico el proceso de montaje.
- El uso de cobots en la fábrica de ŠKODA para la inserción del pistón del actuador de cambio de marchas, en colaboracion con otros robots.
- El uso de brazos robóticos de Universal Robots en las fábricas de Volkswagen para la manipulación de componentes delicados, donde el operador puede terminar el proceso de ensamble con ayuda del robot.

Con el paso del tiempo, cada vez más empresas están desarrollando nuevos modelos de cobots que se adaptan a las necesidades de las industrias, para reducir las áreas de trabajo que emplean los robots tradicionales y mejorar su producción.
Actualmente los robots industriales son capaces de realizar tareas de alta precisión, ayudar a mejorar y aumentar la productividad, así como reducir la carga de trabajo de los operadores, pero están limitados por su programación. Por su parte la integración de los cobots a la industria y su flexibilidad para realizar distintas aplicaciones, han aumentado las áreas en donde se emplean los robots. Por ello, una de las áreas de desarrollo en las que se esta trabajando es la de crear cobots ”inteligentes”, capaces de analizar y planificar las tareas a desempeñar para poder realizar su trabajo de la manera más eficiente.

1.1.2. Métodos de detección de obstáculos

Existen diferentes formas para poder detectar los objetos que se encuentran en el espacio de trabajo del robot y clasificarlos como obstáculos, entre las más importantes se encuentran:

- Método de campos potenciales artificiales (APFM, por sus siglas en ingles), el cual consiste en llenar un campo potencial artificial en el espacio de trabajo del robot. El punto objetivo genera un campo atractivo, y los obstáculos un campo repulsivo, debido a los efectos atractivo y repulsivo que los campos generan sobre el efector final del robot, este puede llegar a su objetivo, evitando los obstáculos [6]. Al no ser un método tan preciso, ya existen estrategias híbridas para mejorar su desempeño.
- Se genera un mapa 2D que identifica las posibles colisiones entre los efectores finales de dos robots. Cada robot ejecuta trayectorias lineales PTP (point-to-point) entre puntos definidos, con movimientos de velocidad constante y características invariantes en el tiempo. Mediante la discretización temporal en intervalos, se calcula la posición de cada efector en cada instante. La detección de colisiones ocurre cuando la distancia entre efectores es inferior a un umbral predefinido, transformándose estos instantes de colisión a coordenadas espaciales utilizando

la longitud total de la trayectoria. Finalmente, se genera un gráfico 2D que superpone la trayectoria del robot en movimiento con los puntos de colisión proyectados sobre esta [7] y [8].

- Uso de sensores de proximidad. La integración de sensores de proximidad capacitivos constituye una solución reactiva para la detección de obstáculos. Estos sensores permiten detectar un obstáculos en el entorno cercano, generando una respuestas correctivas cuando el obstáculo está dentro del rango de detección. Este enfoque permite frenar y ajustar la trayectoria como reacción a la presencia detectada, mejorando los tiempos de respuesta [9].

- Método de aprendizaje por reforzamiento, el cual requiere establecer un algoritmo de control, un modelo para llevar al objetivo deseado y se realiza una simulación a través de casos específicos. Se genera una función de recompensa, la cual brinda una retroalimentación positiva o negativa al robot para modificar su comportamiento [10].

- Sistema de visión. Se integran sensores, como cámaras, sensores ultrasónicos o sensores láser, que le brindan al robot información sobre su entorno. Los sensores ultrasónico y láser le brindan información al robot por medio de un proceso de reflexión, mientras la cámara obtiene dos imágenes que se comparan por medio de método de procesamiento de imágenes digitales para obtener la distancia a la que se encuentra el objetivo y los obstáculos, dependiendo de la calidad de los datos obtenidos y la proximidad al objeto, el robot puede reducir el riesgo de colisión [11].

Para poder determinar cuál es el método que nos resulta más útil para nuestra situación, se debe tener en cuenta las restricciones con las que contamos, como espacio, costo computacional, parámetros que nos brinda el sistema, etc. Para esta tesis se propuso implementar una método similar a los sensores de proximidad, al conocer la posición del obstáculo, se mide la distancia del obstáculo relativa al robot para determinar la dirección y velocidad con la que se desplaza el obstáculo hacia el robot.

1.1.3. Control predictivo basado en modelos

De acuerdo con Lee y Marcus 1967, MPC es una técnica que se utiliza para la síntesis de un control de lazo cerrado a partir del conocimiento de los controladores de lazo abierto, para medir el estado actual del proceso y calcular una función de control para este lazo abierto. El primer valor obtenido de esta función es usada brevemente, después se vuelve a realizar la medición del proceso y se calcula una nueva función de control. [12]
Una interpretación más actual es que el MPC es un esquema de control en el que se utiliza un modelo para predecir el comportamiento futuro del sistema en una ventana de tiempo finito u horizonte. Con base en estas predicciones y el estado actual del sistema medido o estimado, se calculan las entradas óptimas de control para un objetivo de control definido y sujetas a la restricción del sistema. El proceso de medición, estimación y

cálculo se repite cada determinado intervalo de tiempo u horizonte desplazado.
De acuerdo con el artículo [12], el MPC surge de la idea de emplear un modelo de la planta a controlar para predecir el comportamiento futuro de la salida. Esta capacidad de predicción permite resolver problemas de control óptimos en línea, donde el error de seguimiento, es decir, la diferencia entre la salida predicha y la referencia deseada, se minimiza en un horizonte futuro, posiblemente sujeto a restricciones en las entradas, salidas y estados manipulados. Si el modelo es lineal, el limite de desempeño se expresa a través de la norma L_1, si es no lineal se expresa a través de las normas $L_1|L_\infty$ [13]. El resultado de la optimización es aplicado en tiempo t solo a la primera entrada de la secuencia óptima de comando a la planta. Las entradas óptimas restantes se descartan y se resuelve un nuevo problema de control para t + 1. Para cada tiempo t se obtienen nuevas medidas de la planta usando el mecanismo de horizonte en retroceso, se proporciona al controlador las características de retroalimentación deseadas. [14]
MPC ha tenido éxito ya que puede manejar restricciones para las variables controladas y manipuladas sistemáticamente, maneja problemas de control multivariable, es un método fácil de ajustar y es una metodología abierta basada en principios básicos [15]. De acuerdo con [16], un metodo para lograr estabilidad en con un horizonte finito consiste en añadir una restricción terminal que establezca que el estado sea cero al final del horizonte.

Componentes

Todos los algoritmos MPC poseen elementos comunes y elementos opcionales que se pueden seleccionar para variar el algoritmo. Los elementos comunes son:

1. Modelo predictivo (modelo de proceso y modelo de perturbación).
2. Función objetivo.
3. Ley de control.

La estructura básica de un MPC se muestra en la Figura 1.3. El modelo es lo más importante del MPC y puede ser modelo de perturbación o de procesos. Los modelos de procesos pueden ser:

1. Respuesta al impulso.
2. Respuesta escalonada.
3. Función de transferencia.
4. Espacio de Estados.

Mientras que el modelo de perturbación comúnmente utilizado es el Autorregresivo Controlado y la Media Móvil Integrada (CARIMA)

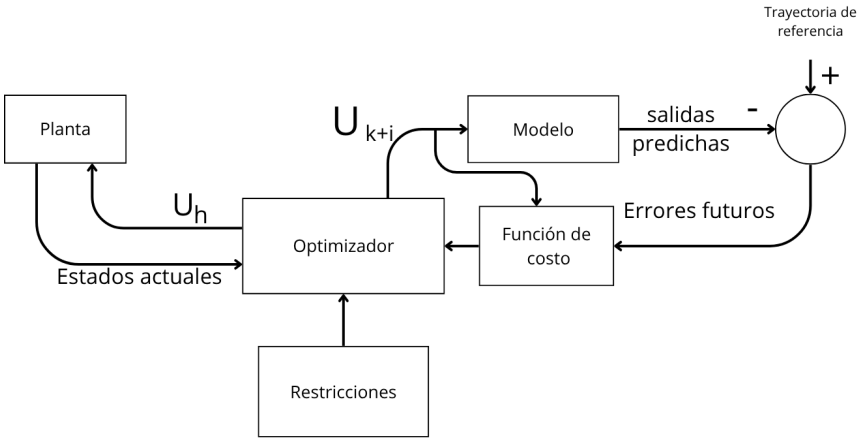


Figura 1.3: Estructura básica del MPC

Características del MPC

1. Emplea modelo explícito: Utiliza un modelo matemático del sistema para predecir la evolución futura de los estados de la planta sobre un horizonte de predicción, permitiendo acciones de control anticipadas.

2. Utiliza optimización basada en función de costo: Formula el problema en el dominio del tiempo mediante una función de costo que penaliza las desviaciones del estado deseado y el esfuerzo de control.

3. Implementa control de horizonte móvil: Actualiza la función de costo en cada instante de muestreo, recalculando la trayectoria óptima basándose en el estado actual del sistema minimizando los errores predichos.

4. Incorpora restricciones explícitas: Integra en la formulación del problema limitaciones operativas como restricciones de estado y de control.

Como en la mayoría de los sistemas, el MPC se divide en 2 clases que se detallan a continuación:
MPC lineal:

- Modelo lineal : $\dot{x}$ = *Ax* + *Bu.*
- Función de costos cuadrática: *F* = x^t*Qx* + u^t*Ru.*
- Restricciones lineales : *Hx* + *Gu ¡0.*
- Programa cuadrático : $\min_v$ 0.5 v^T $\tilde{H}v$ + $c^T v$ sujeta a la restricción: L$v \leq$ b

MPC no lineal:

- Modelo no lineal: $\dot{x}$ = *f(x,t).*
- Función de costos que puede ser no cuadrática: *F = (x,u).*

- Restricciones no lineales: $h(x,u) < 0$
- Programa no lineal.

El MPC se puede subcategorizar en 4 modelos de control.

1. MPC robusto, garantiza la viabilidad y estabilidad del control.
2. MPC de retroalimentación, mitiga el encogimiento de la región factible.
3. MPC pre-computarizado, solución lineal por pasos, almacenado en una base de datos o solucionado fuera de lineal utilizando parámetros.
4. MPC descentralizado.

El MPC se basa en 3 ideas:

- Uso de un modelo para predecir el resultado del proceso a lo largo de un horizonte de tiempo futuro.
- Cálculo de una secuencia de control para optimizar un índice de rendimiento.
- Una estrategia de horizonte en retroceso, lo que implica la aplicación de la primera señal de control de la secuencia calculada en cada paso.

La metodología de todos los controladores de la familia de los MPC se caracteriza por la estrategia de control presentada en la Figura 1.4 consistente de 3 pasos:
Paso 1: Las salidas predictivas yb(t + k—t), k= 1,2,...,N , para el horizonte de predicción son calculados para cada instante t, utilizando el modelo del proceso. Estos dependen de los valores conocidos hasta el momento t (entradas y salidas pasada), incluyendo las salidas actuales (condiciones iniciales) y(t) y las señales de control futuras para ser calculadas.

Paso 2: Las secuencias de las futuras señales de control son calculadas para optimizar el rendimiento.

Paso 3: Se transmite la señal de control actual u(t—t) al proceso. Se mide y(t+1) en el siguiente muestreo y se repite el paso 1. U(t+1||t+1) se calcula utilizando el concepto de horizonte en retroceso.

1.1.4. Virtualización

Basándose en [17], cuando el sistema que se desea modelar resulta muy complejo para representarlo en un único entorno, se opta por la virtualización integral del sistema. Este enfoque permite crear una réplica digital mediante la integración de diversos componentes modelados con diferentes herramientas, técnicas, algoritmos o programas especializados, cada uno dedicado a aspectos específicos del sistema.
La virtualización surge como una solución a estos problemas, ya que permite crear un prototipo digital unificado que replica el comportamiento del sistema real. A diferencia

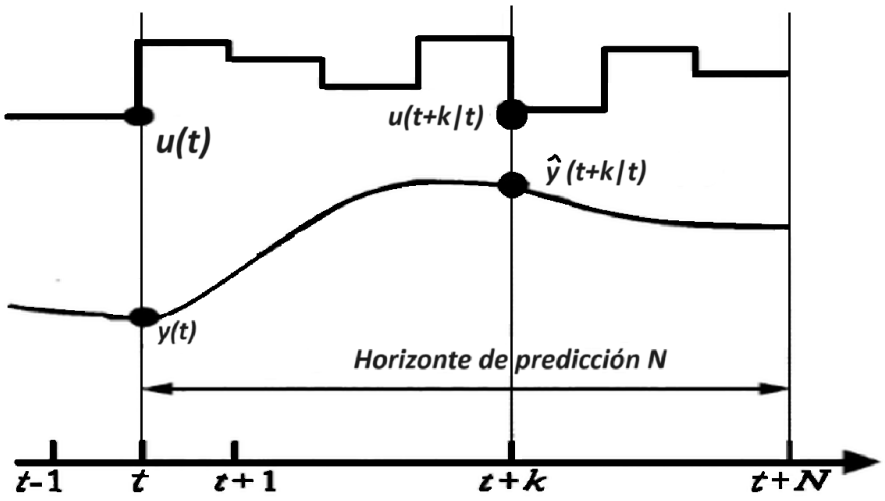


Figura 1.4: Estrategia de Control de Modelos Predictivos, tomado de [12]

de enfoques fragmentados, la virtualización integra coherentemente todos los subsistemas en un entorno único, facilitando la simulación conjunta y el análisis de interacciones.

De acuerdo con la tesis [18], la Ingeniería Asistida por Computadora (CAE) se relaciona con la virtualización al permite crear prototipos virtuales. En esta tesis se plantea el uso de ADAMS View un software que trabaja con prototipos virtuales de sistemas mecánicos, lo que permite validar su funcionalidad y comportamiento sin la necesidad de construir prototipos físicos. Además, cuenta con ADAMS/Control que facilita la integración entre la simulación de movimiento y el diseño de sistemas de control en un entorno virtual unificado.

Para implementar la virtualización se recomienda que las herramientas utilizadas admitan estándares abiertos, permitiendo a fabricantes y equipos de desarrollo intercambiar e integrar modelos dentro de un ecosistema virtual coherente.

1.2. Definición del problema

La implementación del Control Predictivo basado en Modelo (MPC) para un sistema robótico, requiere superar varios desafíos fundamentales. Estos desafíos se pueden categorizar en tres aspectos críticos e interrelacionados:

1. **Modelado preciso y eficiente del sistema y su entorno.**
2. **Selección y diseño de una función de costos que refleje el comportamiento deseado.**

3. **Selección de un optimizador adecuado para resolver el problema de optimización en línea.**

A continuación, se detallan las dificultades específicas y las consideraciones dentro de cada aspecto.

1.2.1. Desafíos en el Modelado del Robot y el Entorno

El modelado es la base del MPC, ya que la capacidad predictiva del controlador depende directamente de la precisión del modelo. Para el caso de un robot UR5 y su entorno, esto presenta varias dificultades:

- Complejidad Cinemática y Dinámica: Un robot como el UR5, posee una cadena cinemática compleja. Su modelado dinámico, que considera masas, inercias y fuerzas de fricción, resulta en ecuaciones no lineales y acopladas. La linealización de este modelo para su uso en MPC puede introducir errores.
- Modelado virtual: La creación de un modelo virtual en un entorno de simulación (Adams View) presenta los siguientes retos:
  - Es difícil crear un modelo al no estar diseñado para ese propósito.
  - Al importar el modelo desde un software especializado en modelado, es necesario verificar que se respeten las restricciones del modelo y que no causen problemas en el entorno de simulación.
  - Verificar que las condiciones de simulación sean las mismas que las planteadas en las pruebas teóricas.
- Detección y Modelado de Obstáculos: La capacidad de evitar colisiones es fundamental. La selección del método de detección de obstáculos es un compromiso entre la precisión y la carga computacional.
- Sincronización de Múltiples Robots: Un objetivo de este trabajo es la sincronización de dos modelos de robot distintos: un robot RR (planar de 2 GDL) y el UR5 (de 3 GDL). Esto añade una capa de complejidad al modelado, ya que es necesario:
  - Desarrollar un modelo cinemático y dinámico acoplado del sistema multirobot.
  - Establecer un esquema de comunicación y sincronización temporal entre los controladores de ambos robots, ya sea mediante un controlador centralizado o uno distribuido con comunicación de estados.

1.2.2. Selección de la Función de Costos

La función de costos es el criterio que el MPC busca minimizar para lograr el comportamiento deseado. Su selección es importante y debe equilibrar múltiples objetivos, a menudo conflictivos:

- Seguimiento de Trayectoria: Penalizar la desviación del robot de la trayectoria de referencia deseada.
- Suavidad del Movimiento: Incluir términos que penalicen cambios bruscos en las velocidades o aceleraciones de las articulaciones, para reducir los torques requeridos.
- Restricciones: Llevar a cabo la evitación de obstáculos

La elección de los pesos relativos de cada término en la función de costos es un proceso no trivial y determina en gran medida el rendimiento final del sistema.

1.2.3. Selección del Optimizador

El problema de optimización se puede considerar que consiste en encontrar una $\mathbf{U}$ por medio de una función de costos. Siendo $\mathbf{U}$ un vector que contiene las entradas de control optimizadas para un horizonte de predicción $\mathbf{N}$ de la forma:

$$U = [u(t), u(t+1), u(t+2), \ldots, u(t+N-1)] \tag{1.1}$$

Cada u(t+k) es la acción de control en el instante t+k que queremos optimizar para lograr el mejor seguimiento de la trayectoria deseada o minimizar el error. Mientras que la función de costo de forma generar se puede considerar como:

$$J = \sum_{k=0}^{N-1} \|x(t+k+1) - x_d(t+k+1)\|_Q^2 + \|u(t+k)\|^2 \tag{1.2}$$

Donde x(t+k+1), es el estado del sistema predicho en el futuro.
x_d (t+k+1) es el estado deseado en el futuro.
$\|\cdot\|_Q^2$ es la penalización al error de estado (Q es una matriz positiva definida).
$\|\cdot\|_R^2$ Penalización al esfuerzo de control (R también es positiva definida).

Dada la naturaleza no lineal de los robots y la posible no convexidad introducida por las restricciones de colisión, la selección del optimizador es crítica para la viabilidad del MPC en tiempo real. Dado que el modelo no se linealizará, es necesario emplear métodos de Programación No Lineal (NLP).

Para este trabajo, se utilizará la biblioteca minimize de SciPy en Python, lo que plantea el desafío de seleccionar y configurar el algoritmo más adecuado entre los disponibles:

- CG (Conjured Gradient): Método iterativo eficaz para resolver problemas de optimización no restringida.
- Dogleg: Método para resolver problemas de cuadrados mínimos no lineales.
- L-BFGS-B (Limited-memory Broyden–Fletcher–Goldfarb–Shanno with Bounds): Este método es muy eficiente para problemas de optimización a gran escala con restricciones de caja (límites simples en las variables).

- COBYQA (Constrained Optimization By Quadratic Approximations): Optimización basada en modelos cuadráticos para problemas con restricciones no lineales, requiere pocas evaluaciones.
- Newton-CG: Método newtoniano que usa gradiente conjugado para resolver el sistema lineal, requiere hessiana definida positiva
- BFGS (Broyden-Fletcher-Goldfarb-Shanno): Método iterativo para resolver problemas de optimización no lineal sin restricciones.
- SLSQP (Sequential Least Squares Programming): Este algoritmo es una opción robusta para problemas con restricciones generales de igualdad o desigualdad.

En conclusión el problema principal de esta tesis consiste, por tanto, en integrar y resolver satisfactoriamente estos tres aspectos interconectados. Esto implica: desarrollar un modelo analítico y virtual preciso y no lineal del UR5 y el sistema multi-robot; seleccionar un método eficiente para la detección y modelado de obstáculos; diseñar una función de costos que capture todos los objetivos de control; y elegir, configurar y validar el rendimiento de los optimizadores no lineales SLSQP y L-BFGS-B de la biblioteca SciPy para la implementación de un esquema de MPC eficaz para el control y sincronización de robots con capacidades de evitación de colisiones.

1.3. Objetivos.

1.3.1. Objetivo general

Desarrollar un modelo virtual de una estación de trabajo de ensamble robotizado que integra un cobot articulado de 6 GDL y un robot RR de movimiento planar, cada uno con su respectivo algoritmo de control codificado en Python, un controlador predictivo (MPC) para el cobot, diseñado para garantizar la evasión de colisiones con el robot RR, y un controlador PID para el robot RR.

1.3.2. Objetivos específicos.

1. Revisión del estado del arte en MPC
   - *a*) Estudiar el estado del arte en Control Predictivo Basado en Modelos MPC.
   - *b*) implementar en Python el control PID de un sistema dinámico simple.
2. Desarrollo del modelo virtual de la plataforma de ensamble robotizada
   - *a*) Estudiar la cinemática directa e inversa de robots manipuladores.
   - *b*) Estudiar la dinámica de robots manipuladores.
   - *c*) Obtener modelos CAD del cobot y del robot RR
   - *d*) Implementar el modelo virtual en Adams View.

3. Implementación de la simulación sin obstáculos

   a) Programar en Python el algoritmo de control PID para el robot RR

   b) Programar en Python el algoritmo de control PID para el cobot.

   c) Implementar la simulación sin obstáculos para los diferentes robots

4. Implementación de la simulación con obstáculos

   a) Planificar las trayectorias para la tarea de pick-and-place a realizar por el cobot.

   b) Planificar la trayectoria que deba seguir el robot RR a fin de intersectar repetidas veces la trayectoria del cobot.

   c) Diseñar el MPC para el cobot, a fin de que evada las inminentes colisiones y complete la tarea de pick-and-place.

5. Estudio de diferentes estrategias de control de movimiento de cobots sujetas a condiciones de colisión.

   a) Estudiar el desempeño del control PID del cobot sujeto a colisiones

   b) Estudiar el desempeño del control MPC del cobot sujeto a colisiones

   c) Realizar un análisis comparativo del desempeño del cobot con MPC con el del cobot con PID.

   d) Implementar en una tarjeta Raspberry Pi el control MPC en tiempo real.

   e) Publicar los resultados.

1.3.3. Aportación de la Tesis

La realización del proyecto de tesis propuesto derivará en algunos beneficios para la comunidad académica e industrial, entre los que destacan:

- Disponer de una plataforma tecnológica que facilita el estudio de:
  - Desempeño del MPC en el control de movimiento de cobots.
  - Control de movimiento de cobots con requerimientos de evasión de obstáculos.
  - Comparación del desempeño de algoritmos de control de movimiento de cobots.
- La documentación de los resultados constituirá una base de conocimiento para estudiantes e investigadores que aborden el estudio de la dinámica y control de robots manipuladores bajo un esquema de control óptimo MPC.

1.4. Justificación

1.4.1. Uso de Control Predictivo Basado en Modelos

El control PID permite verificar que el robot siga una trayectoria deseada, su desempeño depende críticamente de una sintonización precisa de las ganancias y de la densidad de puntos de interpolación. Esta dependencia se debe a su naturaleza reactiva, donde las acciones de control solo se actualizan en el instante posterior a la detección del error, lo que resulta en una respuesta tardía ante perturbaciones o cambios dinámicos.
Frente a estas limitaciones, el control predictivo basado en modelos se presenta como una alternativa debido a las siguientes características:

- Es apropiado para tratar con la evasión de obstáculos al aplicar acciones de control que se anticipan a eventos perturbadores.
- Es bien conocido en el ámbito industrial en control de procesos.
- Puede usarse para optimizar el consumo de energía de un robot manipulador.
- Es bien aceptado en la industria, debido a que es capaz de:
  - Manejar problemas de control multivariable.
  - Tomar en cuenta las limitaciones del actuador.
  - Aumentar las ganancias, al permitir una operación más cercana a las restricciones del sistema.
  - Manejar cambios estructurales.
  - Hacer cálculos en linea con antelación.
  - Permitir la sincronización del control con relativa facilidad.

Estas ventajas posicionan al MPC como una herramienta idónea para mejorar la precisión, eficiencia y adaptabilidad del control en robots.

1.4.2. Uso de Python

Python, lenguaje de programación de alto nivel de propósito general, representa una opción profesional para el análisis dinámico de la plataforma robotizada sujeta a algoritmos de control. Es de fuente abierta, gratuito y multiplataforma. Cuenta con un conjunto vasto de bibliotecas que facilitan la implementación de algoritmos de control de sistemas dinámicos con modelos matemáticos descritos por ecuaciones diferenciales ordinarias. Tiene funciones y herramientas para la exposición gráfica de resultados y cuenta con un sinnúmero de recursos de apoyo impreso y digital, como foros de discusión, blogs y vídeos [19].

1.4.3. Trabajar con cobots

Se eligió trabajar con un cobot en lugar de un robot manipulador clásico porque los cobots son parte de las tecnologías más recientes en la llamada manufactura inteligente propia del paradigma de I4.0 [2], y porque los cobots son por definición robots que interactúan o colaboran con los humanos, por lo que tiene mayor relevancia hablar de una posible colisión entre ambos.

1.4.4. Modelo virtual en Adams View

Al ser Adams View una opción ideal para el estudio de la dinámica de sistemas mecánicos representados por modelos virtuales [20], se cuentan algunas de las ventajas con respecto a otros modelos no virtuales:

- El desarrollo de un modelo virtual de los robot manipulador facilitará la visualización de su desempeño en la realización de movimientos.
- No requiere del modelo analítico del robot para el análisis de su dinámica, aspecto que permite modificar la geometría del robot, los materiales constituyentes, e integrar modelos de objetos físicos para el estudio de interacciones entre ellos.
- Es posible realizar un modelo virtual funcional del robot en relativamente corto tiempo dependiendo de la complejidad del modelo.
- Permite, mediante un procedimiento iterativo, el ajuste de parámetros de diseño y luego evaluar el desempeño dinámico con notable facilidad.
- Existen licencias gratuitas para estudiantes disponibles en el sitio Web de MSC software (HEXAGON) [21].

1.4.5. Técnica de Virtualización

El recurso de la virtualización del sistema presenta algunas ventajas:

- Permite trabajar con la dinámica robótica y los algoritmos de control dentro de un entorno unificado, aprovechando las capacidades especializadas de cada herramienta de software.
- Facilita la simulación de dinámica del robot manipulador junto con los algoritmos de control en un esquema de lazo cerrado dentro del entorno virtual.
- Gracias a la integración de los componentes, los resultados de la virtualización se aproximan a los obtenidos con un cobot real controlado por computadora.

Aprovechando, entonces, los beneficios de Adams View y de Python la virtualización se consolida como un enfoque eficiente para implementar un prototipo virtual confiable y en un tiempo razonable.

Capítulo 2

Estado del arte

En el trabajo [22] se hizo uso del control predictivo basado en modelos para generar un espacio de trabajo en donde interactúan dos brazos robóticos para llevar a cabo una tarea de pick-and-place evitando la colisión. Para llevar a cabo esta tarea se planteó un algoritmo de control híbrido de dos capas, la capa superior se ocupa del problema de planificación, mientras la capa inferior tiene un algoritmo MPC que se ejecuta sobre una ley de control de par calculado. El objetivo de control es la asignación de los robots a los objetos, la secuencia para recoger el objeto y calcular las correspondientes trayectorias libres de colisión para los robots, buscando minimizar el tiempo requerido para realizar la acción. Se pudo establecer que el uso de la estructura jerárquica permite la interacción de un algoritmo de programación con un algoritmo MPC de tiempo óptimo.

El articulo [23] propone un esquema de control de arquitectura compartida basado en MPC para la prevención de colisiones en entornos de conducción colaborativa. Este esquema se estructura en dos capas: una superior que evalúa los riesgos de forma no lineal y una inferior que implementa un control lineal computacionalmente eficiente. La propuesta demuestra la eliminación de colisiones en escenarios complejos, como cambios de carril a alta velocidad, logrando un equilibrio entre la asistencia en la evasión, el control del conductor y la estabilidad del vehículo.

En el artículo [24] se emplea el MPC en un robot móvil, para evadir obstáculos estáticos, mientras este sigue un objetivo en movimiento. El problema de control se planteó como un problema de programación cuadrática con restricciones para encontrar el vector de acción de control incremental, utilizado para calcular la entrada de control óptima, al cual se le sumó un algoritmo de conmutación y se aplicaron algunos cambios en la definición de la orientación deseada para incluir la evasión de obstáculos. Al utilizar la trayectoria tangente a un circulo en cada iteración para la evasión de obstáculos, se pudo demostrar por medio de simulaciones que el control satisface ambos objetivos.

Para conocer un poco más sobre la utilidad del MPC, se revisó el artículo [25] en donde se comparan 3 metodos de control basados en el MPC para un robot de 6 grados de libertad en un ensamble colaborativo. Se utilizan los parámetros de Denavit-Hartenberg para mapear el espacio articular en el espacio cartesiano, y se utiliza el jacobiano pa-

ra relacionar la velocidad del efector final y la velocidad de las articulaciones. El MPC lineal emplea el modelo de una planta lineal o linealizado al rededor de unn punto. El MPC no lineal utiliza el modelo no lineal de la planta descrito por sus ecuaciones de estados y salidas. Para el MPC adaptivo se linealiza y discretiza el modelo de la planta, que se actualiza para cada tiempo de muestreo.
Como resultado el MPC lineal es el controlador más rápido pero con peor precisión, el MPC no lineal es el más preciso, pero requiere muchos recurso. Por su parte el MPC adaptivo tiene alta precisión y requiere pocos recursos.

El control predictivo basado en modelos no es solo útil para el control de robots. Como se ve en los artículos [26] y [27] el MPC se utilizó para el control de la tracción un vehículo eléctrico y brindar un control robusto para un respirador. En ambos casos se busca maximizar la seguridad de las personas; en el primero al controlar el torque aplicado a las llantas a la hora de frenar para evitar que el coche derrape, y en el segundo al delimitar adecuadamente el volumen de oxigeno suministrado por minuto.

En este documento se ha mencionado que existen diferentes formas en la que los cobots pueden trabajar o colaborar con las personas, pero en cualquiera de los casos el robot debe detener su operación o limitar sus parámetros para seguir realizando su trabajo con el operador. En el artículo [1] se aborda la reducción del rendimiento debido a que el robot trabaja con parámetros limitados y esto se puede mitigar con una adecuada estrategia de evasión de obstáculos. Se utilizó un sistema de captura de movimiento basado en una red neuronal de convolución, que permite generar un algoritmo de aprendizaje para reconocer a las personas en un encuadre y estimar su posición entre diferentes cuadros. Se realiza una superposición de los puntos para poder reconstruir al operador en un espacio 3D.
En este trabajo se abordaron diferentes escenarios de colaboración, desde la coexistencia del operador en el área de trabajo hasta diferentes grados de colaboración. El artículo demuestra que es posible generar un ambiente con las medidas adecuadas para una colaboración humano-robot, sin embargo, resalta la importancia del monitoreo en tiempo real del operador.

Con base en los artículos revisados se tienen las siguientes consideraciones:

- A pesar de que en la tesis se planea emplear 2 robots como en el articulo [22], solo se dotará de control NMPC a uno de ellos, a diferencia de los 2 robots presentes en el articulo, el enfoque que se tiene es que el proceso de evasión de obstáculos dependa enteramente del robot UR5.
- El problema de evasión de obstáculos presenta un obstáculo dinámico, debido a que la posición del robot RR (obstáculo), varia en función del tiempo.
- En comparación con el articulo [23] donde el MPC se emplea como asistencia para la toma de decisiones, en este trabajo el MPC es el encargado de llevar a cabo las acciones de evasión, sin la intervención de una persona.

- Al igual que en los artículos [26] y [27], en esta tesis se priorizar la seguridad de las personas. Se trabaja a nivel simulación y se plantea como trabajo futuro la implementación en un cobot real.
- El objetivo en la evasión de obstáculos de esta tesis es evitar que el cobot se detenga, para realizar ello se plantea variar la velocidad del cobot.
- Al trabajar en un entorno de simulación no se requiere la integración de sistemas de visión o sensores para identificar la posición del obstáculo; sin embargo se obtiene un resultado similar al predecir la posición y velocidad relativa al cobot.

2.1. Alcance y limitaciones

Con base en lo planteado anteriormente, se puede establecer que los alcances de la tesis son:

1. Implementación de Control Predictivo Basado en Modelos no lineal (NMPC):
   - Diseño e implementación de un controlador NMPC exclusivamente para el cobot UR5, con el objetivo de evadir obstáculos dinámicos (el robot RR) sin detener su operación.
   - El MPC se diseñará para funcionar en un entorno de simulación, utilizando un modelo dinámico del robot y del obstáculo.
2. Comparativa de estrategias de control:
   - Evaluación del desempeño del sistema bajo dos esquemas de control para el cobot: un controlador PID y el controlador NMPC .
3. Implementación en tiempo real:
   - Portar el algoritmo de control NMPC a una Raspberry Pi para demostrar su funcionamiento en un sistema embebido y evaluar su viabilidad para operar en tiempo real en un entorno futuro con un robot físico.
4. Obstáculo Dinámico:
   - La evasión de obstáculos se probará frente a un obstáculo dinámico (robot RR) cuya trayectoria es conocida o puede ser predicha con precisión, simulando un control en tiempo real.
5. Seguimiento de trayectoria:
   - Por el lado del robot UR5, el control NMPC además de realizar la evasión de obstáculos contará con la capacidad de seguir la trayectoria establecida para cumplir con la tarea de tomar y colocar.
   - El robot RR contará con un control PID, para asegurar el seguimiento de su propia trayectoria establecida.

Por otro lado, esta misma investigación estará sujeta a las limitaciones:

- Solo el cobot UR5 estará dotado de capacidad de evasión activa (NMPC). El robot RR seguirá una trayectoria preestablecida con control PID, sin capacidad de reaccionar al cobot. No existirá comunicación bidireccional entre los robots. Toda la responsabilidad de evitar la colisión recae en el cobot.
- Se trabajará exclusivamente en un entorno de simulación, por lo que perturbaciones como ruido en sensores, imperfecciones mecánicas o retardos de comunicación no serán considerados.
- El escenario de evasión se limitará a un obstáculo dinámico (el robot RR) en un entorno espacial definido.
- La complejidad computacional del algoritmo NMPC y las características del hardware utilizado limita la frecuencia de control y la longitud del horizonte de predicción.
- Los resultados y conclusiones sobre el desempeño del NMPC serán específicos para el modelo de simulación y las condiciones establecidas. Su extrapolación a otros tipos de robots o entornos operativos requeriría estudios adicionales.

2.2. Relevancia

Esta tesis busca brindar una plataforma computacional donde se pueda simular la evasión de un obstáculo dinámico (robot RR), al dotar de un control NMPC a un cobot (robot UR5), sobre el cual recae toda la responsabilidad de evitar el obstáculo. Así como la integración del control NMPC en un sistema embebido para la implementación del control en tiempo real.

Capítulo 3

Metodología

El planteamiento general para llevar a cabo el desarrollo de la tesis fue partir de un punto especifico a lo general.
En primera instancia se realizó el modelado de los robots, comenzando por obtener la cinemática directa de un péndulo, así como su dinámica y un control PID.
Con los conocimientos adquiridos, se obtuvieron los modelos de cinemática directa e inversa del robot RR y se implementó un control de posición empleando un controlador PID. Después se realizó la planeación de trayectorias para el robot RR y se generó un código para llevar a cabo el control de seguimiento de trayectorias con un control PID.
Al conocer el comportamiento del robot RR bajo un esquema de control, se realizó un código para implementar el control NMPC. En este punto, se ponderó la importancia de obtener un modelo preciso para la eficacia de este control. En esta fase se realizó el modelo solo tomando en cuenta las matrices de Inercia y Coriols, y el vector de pares gravitacionales. Después se agregó un termino de fricción viscosa siguiendo el principio de pasividad, el cual establece que los sistemas tienden a disipar la energía si existe algún fenómeno que frene el movimiento del mismo, y se observó que la precisión del control NMPC mejoró moderadamente. Siguiendo este procedimiento, se realizó el control predictivo para el robot UR5, tomando en consideración el aprendizaje logrado hasta ese momento.
Con la intención de entender cómo integrar la evasión de obstáculos al control NMPC se modeló un obstáculo simple, el cual consiste de un objeto que se mueve en una linea recta y pasa por la trayectoria establecida para el robot RR. Considerando que el efector final se modeló como un punto que se desplaza en el espacio de trabajo con respecto al tiempo, fue necesario modelar al robot como una serie de prismas unidos y no solo como un punto, permitiendo si se generan colisiones con los eslabones del robot y no solo con el efector final.
Para incluir simultáneamente ambos robots y simularlos en un mismo espacio, se generaron Clases en Python para cada uno de los robots. Esto se debe a que se generaron 2 diferentes instancias de control, un control PID para el seguimiento de trayectorias corriendo antes y después del periodo de tiempo que el robot se encuentra en riesgo de colisión, y un control NMPC para la evasión de obstáculos durante el periodo que existe

riesgo de colisión.
De cada uno de las simulaciones se obtuvieron gráficas en donde se observa la precisión con la que cada uno de los controles lleva a cabo la tarea de seguimiento de trayectorias. De igual manera, en los códigos de control NMPC se registraron los tiempos de optimización para cada instante de tiempo, así como los tiempos de simulación para los códigos de los robots.
Debido a la complejidad del problema principal, se tomó la decisión de seguir esta metodología con modelos simples para identificar los factores a tomar en cuenta en las fases posteriores. Al realizar esta metodología se puede descomponer el problema principal en pasos de forma SMART, (Específico, Medible, Alcanzable, Relevante y con plazo definido).

3.1. Ruta crítica

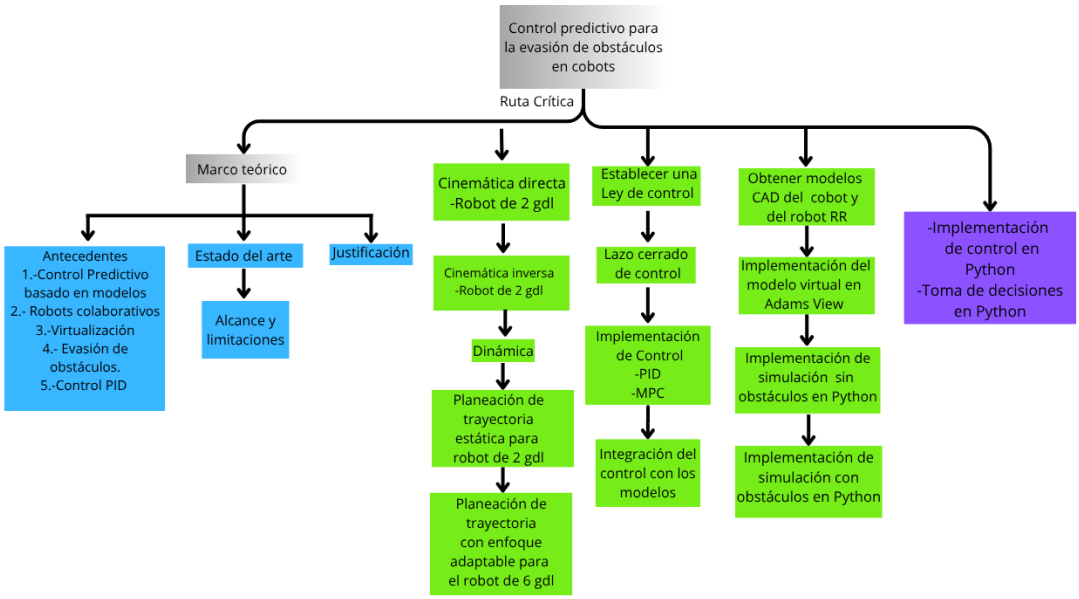


Figura 3.1: Actividades indispensables a realizar para cada semestre

Como se ilustra en el diagrama, la ruta crítica comprende el conjunto de actividades esenciales para la ejecución de este trabajo.
Los elementos en azul corresponden al marco teórico, el cual se estructura en tres componentes principales:

- Antecedentes: Fundamentos teóricos necesarios para el desarrollo de la investigación.
- Estado del arte: Revisión bibliográfica de trabajos afines que permiten establecer los alcances y limitaciones de la tesis.

- Justificación: Razones que motivaron la elección de los temas y herramientas utilizadas.

Por su parte, los recuadros en verde representan la fase práctica, organizada en tres ejes:

1. Desarrollo teórico: Obtención de la cinemática y dinámica de ambos robots, y planificación de trayectorias.
2. Implementación de control: diseño de la ley de control, configuración del lazo cerrado y desarrollo de los esquemas PID y MPC.
3. Modelado virtual: generación de modelos CAD, verificación en Adams y creación de la plataforma virtual para simulación.

Finalmente, se seleccionó Python como lenguaje de programación debido a su sintaxis accesible, naturaleza de código abierto y amplio ecosistema de bibliotecas, lo que facilita la integración de funciones de otros lenguajes y expande las capacidades para cumplir con los objetivos del proyecto.

Capítulo 4

Cinemática

De acuerdo con [28], la cinemática de robots manipuladores se refiere al estudio analítico del movimiento del robot con respecto a un sistema de referencia cartesiano fijo (x, y, z) relacionando la dependencia de que existe entre las coordenadas articulares o generalizadas q $\in \mathbb{R}^n$, sus parámetros geométricos y las coordenadas cartesianas [x, y, z]$^T \in \mathbb{R}^3$ y de orientación [θ , ϕ, ψ]$^T \in \mathbb{R}^3$ del efector final . La cinemática de manipuladores se puede dividir en 2 clases.

4.1. Cinemática directa

La resolución del problema cinemático directo permite conocer cuál es la posición y orientación que adopta el extremo del robot cuando cada una de las variables que fijan la posición u orientación de sus articulaciones toma valores determinados. Es decir, si conocemos la posición de las articulaciones y su relación con los eslabones que conforman el robot, podemos determinar la orientación y posición del efector final.
Para obtener el modelo cinemático directo se pueden emplear 2 enfoques denominados métodos gráficos y métodos basados en cambio de referencia del sistema.
Los primeros son adecuados para casos simples, pero al no ser sistemáticos, su uso está limitado a robots de pocos grados de libertad. Por otro lado, los métodos basados en cambio de sistema de referencia permiten de una manera sistemática abordar la obtención del modelo cinemático directo del robot para robots de n grados de libertad.
El método gráfico busca obtener la relación entre las coordenadas articulares y la posicion y orientación del extremo del robot por simples consideraciones geométricas, como se muestra en la figura 4.1 .
Dado que un robot se puede considerar como una cadena cinemática abierta formada por objetos rígidos o eslabones unidos entre sí mediante articulaciones, se puede establecer un sistema de referencia fijo situado en la base del robot y describir la localización de cada uno de los eslabones con respecto a dicho sistema de referencia. De esta forma, el problema cinemático directo se reduce a encontrar una matriz de transformación homogénea **T** que relacione la posición y orientación del extremo del robot respecto del sistema de referencia fijo situado en la base del mismo.
Esta matriz se puede considerar una matriz aumentada de la siguiente forma y se le

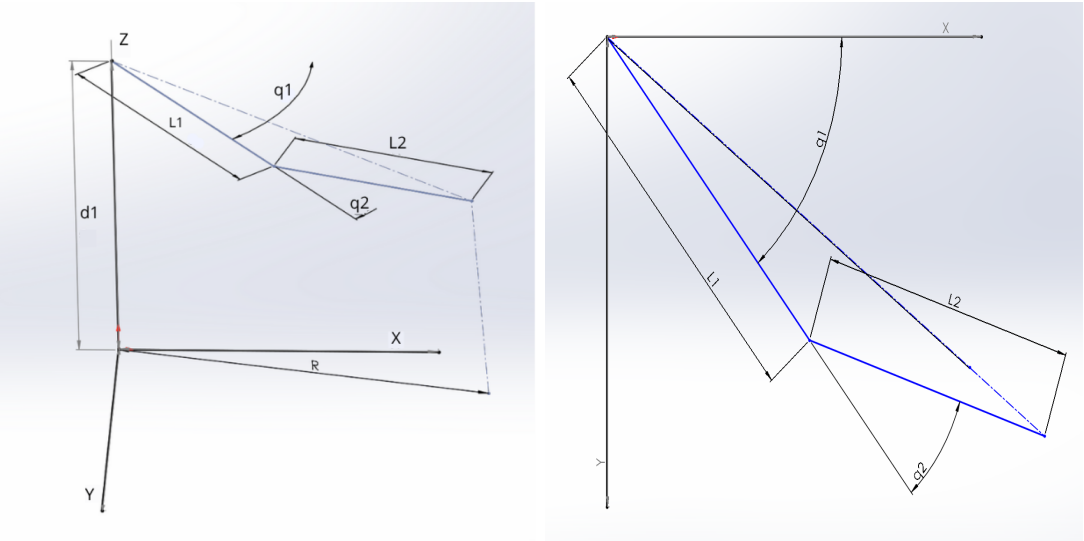


Figura 4.1: Robot de 2 grados de libertad

denomina matriz de transformación homogénea (**T**).

$$\begin{aligned} T &= \begin{bmatrix} R_{3x3} & P_{3x1} \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \text{Rotación} & \text{Traslación} \\ 0 & 1 \end{bmatrix} \end{aligned} \tag{4.1}$$

Donde R_{3x3} es una matriz ortogonal de 3x3, P es un vector de 3x1, 0 es un vector de ceros de 1x3, también conocido como vector de perspectiva y el número 1 representando el escalamiento, estos últimos 2 se emplean para propósito de acoplamiento de dimensiones.

Con base en lo expuesto en [29], un robot de n grados de libertad está conformado por n eslabones unidos a n articulaciones, de forma que cada par articulación-eslabón constituye un grado de libertad. Para poder encontrar la relación del efector final con la base, se le asigna a cada eslabón un sistema de referencia solidario a éste y se emplean matrices de transformación homogénea para representar las rotaciones y traslaciones relativas entre los sistemas asociados a dos eslabones consecutivos y normalmente se suele denominar matriz ${}^{i-1}A_i$. De esta forma, se pude expresar la relación de la base con el primer eslabón como ${}^0A_1$, ${}^1A_2$ la relación del primer eslabón con el segundo y así consecutivamente hasta el n-simo eslabón del robot, donde la matriz de transformación homogénea para este caso se expresa como ${}^0A_n$.
Por ejemplo para un robot de 4 grados de libertad al representar la cinemática parcial hasta el segundo eslabón se obtiene una ecuación como (4.2), mientras si se desea representar de forma completa, la ecuación resultante es como (4.3).

$${}^0A_2 = {}^0A_1\,{}^1A_2 \tag{4.2}$$

$${}^0A_4 = {}^0A_1\,{}^1A_2\,{}^2A_3\,{}^3A_4 \tag{4.3}$$

Cuando se consideran todos los grados de libertad, a la matriz ${}^0A_n$ se le suele denominar **T**, así pues la orientación y dirección del efector final de un robot de 6 grados de libertad esta dada por la matriz **T**.

$$T = {}^0A_6 = {}^0A_1\,{}^1A_2\,{}^2A_3\,{}^3A_4\,{}^4A_5\,{}^5A_6 \tag{4.4}$$

Cada una de estas matrices nos permite pasar del sistema asociado al eslabón i-1 al asociado a i; además éstas dependen de constantes geométricas propias del eslabón y del valor de q_i, reescribiendo la ecuación 4.4 se obtiene:

$$T(q_1, ...q_n) = {}^0A_1(q_1) \cdot {}^1A_2(q_2) \ldots {}^{n-1}A_n(q_n) \tag{4.5}$$

Para solucionar el problema cinemático directo es necesario hacer coincidir la matriz T obtenida con la matriz de transformación homogénea **[noap]** correspondiente a la localización en la que se desea posicionar al robot. Siendo **[noap]** un nemotecnia para referirse a los vectores columna que corresponden a la matriz de transformación. Siendo el vector **n** orientación **normal**, representa la dirección del eje X del sistema de coordenadas adjunto al efector final del robot, el vector **o** de **orientación**, representa la dirección del eje Y del sistema de coordenadas del efector final, el vector **a** de **acercamiento**, representa la dirección del eje Z del sistema de coordenadas del efector final y el vector **p** de **posición**, representa las coordenadas del origen del sistema de coordenadas del efector final con respecto al sistema de coordenadas de referencia (base del robot).
Debido a que es posible describir la relación existente entre 2 eslabones consecutivos de diversas maneras al emplear cualquier sistema de referencia asociado a cada eslabón, no es posible obtener resultados iguales aunque su equivalentes al momento de realizar el análisis de la cadena cinemática. Para evitar que existan discrepancias en la interpretación de la geometría de los eslabones y cómo se relacionan, metodología Denavit-Hartenberg.

4.2. Metodologia Denavit-Hartenberg

La metodología que se propuso es conocida como metodología de Denavit-Hartenberg, la cual establece la localización que debe tomar cada sistema de coordenadas S_i ligado a cada eslabón i de una cadena articulada para poder sistematizar la obtención de las ecuaciones cinemáticas de la cadena completa. Gracias a esta metodología es posible pasar de un sistema de coordenadas al siguiente mediante 4 transformaciones básicas que dependen exclusivamente de las características geométricas del eslabón.
Estas 4 transformaciones básicas consisten en una sucesión de rotaciones y traslaciones que permiten relacionar el sistema de referencia del eslabón i–1 con el sistema del eslabón i. Las transformaciones en cuestión son las siguientes:

1. Ángulo θ_i dado por una rotación a rededor de z_{i-1}. Que corresponde a la matriz de rotación $\text{Rot}_{z_{i-1}}$ (θ_i)

2. Traslación a lo largo de z_{i-1} una distancia d_i. Representado mediante el vector de traslación d_i = (0,0,d_i), que corresponde a una matriz de traslación $\text{Trans}_{z_{i-1}}$ (d_i).

3. Traslación a lo largo de x_i una distancia a_i. Representado mediante el vector de traslación a_i = (a_i,0,0), que corresponde a una matriz de traslación Trans_{x_i} (a_i).

4. Ángulo α_i dado por una rotación a rededor de x_i. Que corresponde a la matriz de rotación Rot_{x_i} (α_i)

Donde cada una de estas transformaciones se representa mediante una submatriz de transformación. La multiplicación sucesiva de estas matrices es necesaria porque cada transformación modifica el sistema de referencia respecto al cual se aplica la siguiente transformación.
Cada submatriz de transformación representa un cambio de sistema de coordenadas, y la multiplicación matricial es el mecanismo matemático para componer estas transformaciones secuenciales. El producto resultante encapsula en una sola matriz el efecto acumulado de los cuatro movimientos básicos, proporcionando la transformación completa entre dos sistemas consecutivos de la cadena cinemática.
Dado que el producto de matrices no es conmutativo, las transformaciones se han de realizar en el orden indicado. De este modo se tiene que:

$$^{i-1}A_i = Rotz(\theta_i)\,\mathbf{T}(0,0,d_i)\,\mathbf{T}(a_i,0,0)\,Rotx(\alpha_i) \tag{4.6}$$

Que representa el orden especifico en el que se multiplica cada una de las submatrices de transformación (de derecha a izquierda) para relacionar el sistema de referencia del eslabón actual con el del eslabón siguiente. En su forma matricial es:

$$\begin{aligned} ^{i-1}A_i &= \begin{bmatrix} C\theta_i & -S\theta_i & 0 & 0 \\ S\theta_i & C\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha_i & -S\alpha_i & 0 \\ 0 & S\alpha_i & C\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{4.7}$$

Como se ha indicado, para que la matriz $^{i-1}A_i$, definida en 4.7 relacione los sistemas S_{i-1} y S_i, es necesario que los sistemas se hayan escogido de acuerdo a unas determinadas reglas. Éstas, junto con la definición de los 4 parámetros θ_i, d_i, a_i y α_i conocidos como parámetros de Denavit-Hartenberg, conforman el algoritmo para la resolución del problema cinemático directo, de acuerdo con [30], este algoritmo puede ser de la siguiente forma:

Algoritmo D-H para la convención clásica

Paso 1 Identificar y enumerar los enlaces que comienzan con la base (0) y terminan con el efector final (n).

Paso 2 Seleccionar la ubicación del marco con respecto a una base es arbitraria, se elige el eje x_0 perpendicular a z_0 y y_0 siendo $y_0 = x_0 \times z_0$.

Paso 3 Ubicar el origen o_i en la normal común a z_i y z_{i-1}.

Paso 4 Establecer x_i a lo largo de la normal común entre z_i y z_{i-1} hasta o_i.

- Si los ejes z de 2 uniones se intersectan, no existe plano normal entre ellos (o su longitud es 0), se asigna el eje x a lo largo de la línea perpendicular al plano formado por los ejes z.
- Si 2 ejes z son paralelos, se elige la normal común que esté alineada con la normal previa.
- Si los ejes z_i y z_{i-1} coinciden, el origen se coloca en el eje común. Si la unión es de revolución el eje x_i coincide con x_{i-1}, si la unión es prismática x_i se elige paralelo a x_{i-1} y el origen se ubica al final de la unión i.

paso 5 El eje y_i es seleccionado de acuerdo a la regla de la mano derecha, $y_i = z_i \times x_i$.

Paso 6 Se establece el sistema de referencias del efector final, ubicando el eje z co-lineal al eje de aproximación del efector, el eje x se establece perpendicular al plano del efector final y el eje y se determina por el producto vectorial $\hat{y} = \hat{z} \times \hat{x}$, asegurando un sistema de coordenadas ortogonal derecho.. El origen se coloca preferentemente en el centro del efector.

Paso 7 Se crea una tabla de parámetros para 2 eslabones, como la tabla 4.1, para obtener los parámetros de Denavit-Hartenberg (θ_i, d_i, a_i y α_i).

Link, i	a_i	α_i	d_i	θ_i
1	a_1	α_1	d_1	θ_1
2	a_2	α_2	d_2	θ_2

Tabla 4.1: Parámetros de eslabones

Paso 8 Se forma la matriz de transformación homogénea ${}^{i}T_{i-1}$ sustituyendo los valores en la ecuación (4.7).

Paso 9 De ${}^{0}T_n = {}^{0}T_1 \quad {}^{1}T_2 \quad {}^{2}T_3 \ldots {}^{n-1}T_n$. Esto proporciona la posición y orientación del marco efector final expresadas en las coordenadas base.

El algoritmo anterior contempla la convención clásica de Denavit-Hartenberg, sin embargo esta no es el único algoritmo, también se cuenta con una convención modificada para este algoritmo.
Las principales características de la convención modificada se presentan a continuación:

Convención modificada

- Ubicación sistemática de sistemas de coordenadas: Cada sistema S_i se sitúa en la articulación $i+1$ en lugar de en la articulación i, con el eje z_i alineado con el eje de la articulación $i+1$.
- Parámetros simplificados: Utiliza solo cuatro parámetros por eslabón (θ_i, d_i, a_i, α_i), pero con interpretaciones geométricas diferentes a la convención clásica.
- Reglas estandarizadas para ejes paralelos: Establece criterios precisos para la colocación de sistemas de coordenadas cuando ejes consecutivos son paralelos, eliminando ambigüedades.
- Orientación consistente del eje x_i: Define x_i perpendicular a ambos ejes z_{i-1} y z_i, siguiendo la regla $x_i = \pm(z_{i-1} \times z_i)/\|z_{i-1} \times z_i\|$ cuando los ejes no son paralelos.
- Matriz de transformación homogénea unificada: Mantiene la misma estructura matricial que la convención clásica, pero con diferentes interpretaciones de los parámetros en la cadena cinemática.

Esta convención modificada ofrece ventajas en implementaciones computacionales y proporciona una metodología más sistemática para robots con configuraciones cinemáticas complejas o múltiples ejes paralelos. sin embargo no se profundizará más debido a que no será empleada en esta tesis.
Con los conocimientos obtenidos se realiza una aplicación practica de la obtención de la cinemática directa por medio de la metodología de Denavit-Hartenberg clásica, para el robot de 2 grados de libertad que se emplea en esta tesis.

4.2.1. Cinemática inversa

El objetivo del problema cinemático inverso consiste en encontrar los valores que deben adoptar las coordenadas articulares del robot q = [q_1, q_2, ..., q_n]^T para que su extremo se posicione y oriente según una determinada localización espacial **(p, [n, o, a]).** A diferencia del proceso para resolver el problema cinemático directo, el cual es sistemático a partir de la utilización de matrices de transformación homogénea y esté no depende de la configuración del robot. Para la solución del problema cinemático inverso, al igual que con el directo, la configuración del robot es fuertemente importante. El problema cinemático inverso se puede resumir en encontrar una razón matemática de la forma:

$$\begin{aligned} q_k &= f_k(x, y, z, \phi, \theta, \psi) \\ k &= 1...n\,(GDL) \end{aligned} \tag{4.8}$$

Al contrario de lo que ocurría en el problema cinemático directo, la solución del problema cinemático inverso no es única, existiendo diferentes n-tuplas [q_1, ..., q_n]^T que posicionan y orientan el extremo del robot del mismo modo. La mayoría de los robots poseen cinemáticas relativamente simples que facilitan en cierta medida la resolución

de su problema cinemático inverso. Algunas de las geometrías que facilitan este análisis son:

- Estructura planar del robot (eslabones comprendidos en un plano).
- Últimos 3 grados dedicados a orientar el extremo del robot, corresponden a giros sobre ejes que se cortan en un punto.

Los métodos geométricos permiten obtener los valores de las primeras variables articulares, que son las que consiguen posicionar el robot (prescindiendo de la orientación de su extremo). Para ello, utilizan relaciones trigonométricas y geométricas sobre los eslabones del robot. Se suele recurrir a la resolución de triángulos formados por los eslabones y articulaciones del robot.
También se puede resolver el problema por medio de la relación:

$$\begin{bmatrix} \mathbf{n} & \mathbf{o} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} = [\mathbf{t}_{ij}] \tag{4.9}$$

donde los eslabones t_{ij} son función de las coordenadas articulares $[q_1, ..., q_n]^T$, es posible pensar que mediante ciertas combinaciones de las 12 ecuaciones escalares (correspondientes a los elementos no nulos de las matrices homogéneas), las cuales relacionan los parámetros de orientación (**n, o, a**) y posición (**p**) con las variables articulares q_i. La estrategia de solución consiste en seleccionar combinaciones específicas de estas ecuaciones que permitan despejar iterativamente cada variable articular, comenzando típicamente por aquellas que afectan principalmente la posición del efector.
Este procedimiento sistemático permite desacoplar los grados de libertad y resolverlos sucesivamente, simplificando la solución completa del problema de cinemática inversa.

Cinemática inversa por métodos geométricos

El procedimiento se basa en encontrar suficiente número de relaciones geométricas en las que intervienen las coordenadas del extremo del robot, sus coordenadas articulares y las dimensiones físicas de sus eslabones. Por esto, mientras mayor sea el numero de las variables articulares que se desean conocer más complicado será su análisis. Para la ejemplificación de este método se realizará la solución del problema de cinemática inversa en la sección de aplicación práctica.

Cinemática inversa a partir de la matriz de transformación homogénea

Se busca obtener el modelo cinemático inverso de un robot a partir del conocimiento de su modelo directo. Es decir, conociendo las relaciones que expresan el valor de la posición y orientación del extremo del robot en función de sus coordenadas articulares, se busca obtener las relaciones inversas.
En la práctica esto no resulta una tarea fácil, y en muchas ocasiones es tan complejo que se descarta esta tarea, optando por restringir el robot a movimientos preprogramados o enseñados punto a punto o emplear algoritmos iterativos. En muchas ocasiones el

número de ecuaciones obtenidas por el método de transformación homogénea supera el número de incógnitas que se desea conocer, generando así cierta dependencia entre la ecuaciones obtenida (debido a la ortonormalidad de **n o a**) brindando mucha importancia a la ecuación (4.9). En la sección de aplicación práctica se generará un ejemplo de este método.

4.3. Planificación de trayectorias

De acuerdo con [31], una trayectoria para un manipulador es un historial en el tiempo de la posición, velocidad y aceleración de cada grado de libertad. Para poder facilitar la descripción del movimiento de un manipulador de un sistema de robot, lo que se debe hacer es proporcionar la capacidad de especificar trayectorias con simples descripciones del movimiento deseado y dejar que el sistema desarrolle los detalles. Siendo posible solo especificar la posición y orientación de destino deseadas del efector final y dejar que el sistema decida la forma exacta de la ruta.
De acuerdo con [29] se puede establecer cuáles son las trayectorias que debe seguir cada articulación del robot a lo largo del tiempo para lograr los objetivos fijados por medio del control cinemático.
De manera general, el control cinemático deberá realizar las siguientes funciones:

CC-1 Interpretar la especificación del movimiento, dada en el programa escrito por el usuario (punto de destino, precisión, tipo de trayectoria deseada, velocidad o tiempo in- vertido, etc.), mediante una trayectoria analítica en el espacio cartesiano.

CC-2 Muestrear la trayectoria cartesiana obteniendo un número finito de puntos de dicha trayectoria. Cada uno de estos puntos vendrá dado por una 6-tupla, típicamente (x,y,z,ϕ,θ,Ψ)

CC-3 Utilizando la transformación homogénea inversa, convertir cada uno de estos puntos en sus correspondientes coordenadas articulares (q_1,q_2,q_3,q_4,q_5,q_6). Debe tenerse en cuenta aquí la posible solución múltiple de la transformación homogénea inversa, así como la posibilidad de ausencia de solución y puntos singulares, de modo que se asegure la continuidad de la trayectoria.

CC-4 Interpolación de los puntos articulares obtenidos, generando para cada variable articular una expresión q_i(t) que pase o se aproxime a ellos de modo que, siendo una trayectoria realizable por los actuadores, se transforme en una trayectoria cartesiana lo más próxima a la especificada.

CC-5 Muestreo de la trayectoria articular para generar referencias al control dinámico.

En casos particulares se pueden omitir algunas de las funciones previamente listadas. Si se tiene una secuencia adecuada de la configuraciones de los ángulos para cada articulación, se pueden omitir las funciones CC-1, CC-2 y CC-3.

Existen 2 principales formas de plantear la planificación de trayectorias:

1. Trayectorias punto a punto: Esta estrategia consiste en desplazar cada articulación desde su configuración inicial hasta la final de forma independiente. El robot puede ejecutar estos movimientos de dos formas: actuando sobre cada eje por separado o moviendo todos los ejes de forma concurrente. Cuando se emplea el movimiento simultáneo, el sistema realiza un precálculo de las trayectorias articulares para mejorar la eficiencia del proceso.
2. Trayectorias continuas: Donde cada articulación sigue un movimiento aparentemente errático con posibles cambios de dirección y velocidad y sin coordinación con el resto de las articulaciones. Sin embargo, el resultado conjunto será que el extremo del robot describirá la trayectoria deseada

En general, no es posible convertir directamente una trayectoria cartesiana descrita de forma continua en una trayectoria articular continua. Esto se debe a que solo podemos calcular una configuración articular específica para una posición cartesiana particular utilizando el modelo cinemático del robot. Por eso, es necesario seleccionar ciertos puntos clave de la trayectoria cartesiana (puntos vía) y convertir esos puntos en configuraciones articulares correspondientes. La selección de la cantidad de puntos vía es importante para asegurar una respuesta en tiempo real del sistema de control del robot y poder disminuir el error entre la trayectoria resultante y la deseada.
Una vez dispuesta esta sucesión de puntos, se requiere generar una función cuyos parámetros o coeficientes se ajusten al imponer las condiciones de contorno: posiciones, velocidades y aceleraciones. Así pues, si se desea pasar por n puntos, se puede considerar un polinomio de grado n + 2 para satisfacer sus condiciones (n puntos de paso y 2 velocidades). Sin embargo, el uso de un polinomio de grado elevado, puede ser computacionalmente exigente y la presencia de potencias elevadas (t^{n+1}), donde t representa el parámetro temporal de la trayectoria, puede generar errores numéricos significativos en los cálculos debido a la amplificación de imprecisiones. Por estos motivos, en lugar de usar una sola función interpoladora que una todos los puntos de la trayectoria, se utilizan conjuntos de funciones más simples, que unen solo unos pocos puntos consecutivos de un intervalo. Las funciones más utilizadas para esta tarea son:

- Interpolación lineal: Une 2 puntos consecutivos, manteniendo una velocidad constante entre dichos puntos, descrita por la ecuación:

  $$\theta(t) = (\theta_f - \theta_i)\frac{t - t_i}{t_f - t_i} + \theta_i$$

  donde:

  - $\theta(t)$: Posición angular en el instante t
  - θ_i, θ_f: Posiciones angular inicial y final del segmento
  - t_i, t_f: Tiempos inicial y final del segmento

- t: Tiempo actual

- Interpolación cúbica: Esta función busca mantener la velocidad continua no solo entre 2 puntos si no entre todos los puntos vía. Al tener cuatro parámetros disponibles, se podrán imponer cuatro condiciones de contorno, dos de posición y dos de velocidad, con la forma:

$$\begin{aligned}\theta(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ \dot{\theta} &= a_1 + 2a_2 t + 3a_3 t^2 \\ \ddot{\theta} &= 2a_2 + 6a_3 t\end{aligned}$$

  y los coeficientes se obtienen mediante:

$$\begin{aligned}a_0 &= \theta_0 \\ a_1 &= \dot{\theta}_0 \\ a_2 &= \frac{3}{t_f^2}(\theta_f - \theta_i) - \frac{2}{t_f}\dot{\theta}_0 - \frac{\dot{\theta}_f}{t_f} \\ a_3 &= -\frac{2}{t_f^3}(\theta_f - \theta_i) + \frac{1}{t_f^2}(\dot{\theta}_f - \dot{\theta}_i)\end{aligned}$$

- Interpolación de quinto grado: Para este tipo de funciones no solo se busca mantener estable la velocidad, si no que también se busca una continuidad en las aceleraciones, contando así con 6 condiciones, 2 para cada variable, posición, velocidad y aceleración. Descrito por el polinomio:

$$\begin{aligned}\theta(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \\ \dot{\theta} &= a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + a_5 t^4 \\ \ddot{\theta} &= 2a_2 + 6a_3 t\end{aligned}$$

  obteniendo los coeficientes mediante:

$$\begin{aligned}a_0 &= \theta_0 \\ a_1 &= \dot{\theta}_0 \\ a_2 &= \frac{\ddot{\theta}_0}{2} \\ a_3 &= \frac{20\theta_f - 20\theta_0 - (8\dot{\theta}_f + 12\dot{\theta}_0)_t f - (3\ddot{\theta}_0 - \ddot{\theta}_f)t_f^2}{2t_f^3} \\ a_4 &= \frac{30\theta_f - 30\theta_0 + (14\dot{\theta}_f + 14\dot{\theta}_0)_t f + (3\ddot{\theta}_0 - 2\ddot{\theta}_f)t_f^2}{2t_f^4} \\ a_5 &= \frac{12\theta_f - 12\theta_0 - (6\dot{\theta}_f + 6\dot{\theta}_0)_t f + (\ddot{\theta}_0 - \ddot{\theta}_f)t_f^2}{2t_f^5}\end{aligned}$$

- Interpolación trapezoidal: Es una mezcla de 2 diferentes funciones de control, en la sección de en medio se emplea una interpolación lineal, mientras que en los extremos se emplea un polinomio de 2do grado. Obtenidas de la ecuaciones:

$$\dot{\theta}_{jk} = \frac{\theta_k - \theta_j}{t_{djk}}$$
$$\ddot{\theta}_k = \text{signo}(\dot{\theta}_{kl} - \dot{\theta}_{jk})|\ddot{\theta}_k|$$
$$t_k = \frac{\dot{\theta}_{kl} - \dot{\theta}_{jk}}{\ddot{\theta}_k}$$
$$t_{jk} = t_{djk} - \frac{1}{2}t_j - \frac{1}{2}t_k$$

Las trayectorias que se calculan en espacio de articulación pueden asegurar que se llegue a los puntos vía y los puntos destino, incluso cuando estos puntos de ruta se hayan especificado mediante tramas cartesianas.

4.4. Seguimiento de Trayectorias

De acuerdo con [32], aunque los controles PID independientes son adecuados para la mayoría de los problemas de regulación de puntos de referencia, hay tareas que requieren un seguimiento efectivo de trayectorias, como la soldadura por plasma, el corte con láser o las operaciones de alta velocidad en presencia de obstáculos.
En estos casos, el uso de esquemas locales obliga a moverse lentamente entre varios puntos intermedios, lo que retrasa la finalización de la tarea. Para mejorar el rendimiento en el seguimiento de trayectorias, los controladores deben considerar el modelo dinámico del manipulador. El problema de control de seguimiento en el espacio articular o de tarea consiste en seguir una trayectoria deseada que varía con el tiempo, $q_d(t)$ y sus derivadas sucesivas $\dot{q}_d$ y $\ddot{q}_d$ que describen las velocidades y aceleraciones deseadas. Para lograr un rendimiento exitoso, se han desarrollado estrategias de control basadas en modelos, para esta tesis se contempla el uso de control de dinámica inversa.

El control de dinámica inversa se enfocada en cancelar términos no lineales y desacoplar las dinámicas de cada enlace. En el espacio articular, el control de dinámica inversa toma la forma de un sistema lineal desacoplado para cada articulación, donde *v* (*v* = $\ddot{q}$) es una entrada de control auxiliar que debe ser diseñada. Opciones típicas para *v* incluyen esquemas con términos proporcionales y derivativos, o también con un componente integral, lo que lleva a diferentes ecuaciones de dinámica del error. Estas ecuaciones son exponencialmente estables con una elección adecuada de las matrices de ganancia con la forma:

$$\tau = \ddot{q}_d + K_v(\dot{q}_d - \dot{q}) + K_p(q_d - q) + K_i \int (q_d - q)dt \tag{4.10}$$

4.5. Modelado de colisiones

Para simplificar el análisis de colisiones en el espacio de trabajo del robot, se sigue la metodología propuesta en [6], donde los obstáculos se modelan como cilindros o esferas con su centro de masa como origen. El volumen de estos objetos define su rango de influencia dentro del campo de acción del robot, mientras que los eslabones del robot se representan como enlaces cilíndricos.
Consideremos un obstáculo esférico con centro O (x_o, y_o, z_o) y su radio r_o junto con un enlace cilíndrico del brazo robótico de radio r_l. La proyección perpendicular r_{ob}, que va desde el centro de la esfera hasta el eje del enlace define el punto F(x_f, y_f, z_f), el cual representa la posición del punto más cercano del robot a un objeto. Esta distancia se puede expresar como:

$$d_{of} = \sqrt{(x_o - x_f)^2 + (y_o - y_f)^2 + (z_o - z_f)^2} \tag{4.11}$$

Una representación simplificada de la evasión de obstáculos se muestra en 4.2.

Para garantizar la seguridad del brazo robótico se establece una distancia de seguridad δ. El proceso para la evasión de obstáculos se puede dividir en 3 casos:

- d_{of} - r $> \delta$ significa que el brazo esta lejos del objeto y a salvo.
- 0 $< d_{of}$ - r $\leq \delta$, el brazo se encuentra cerca del objeto y debe tomar acciones de evasión.
- d_{of} - r = 0, el brazo ya está en contacto con el objeto y debe reaccionar inmediatamente para librarse del objeto.

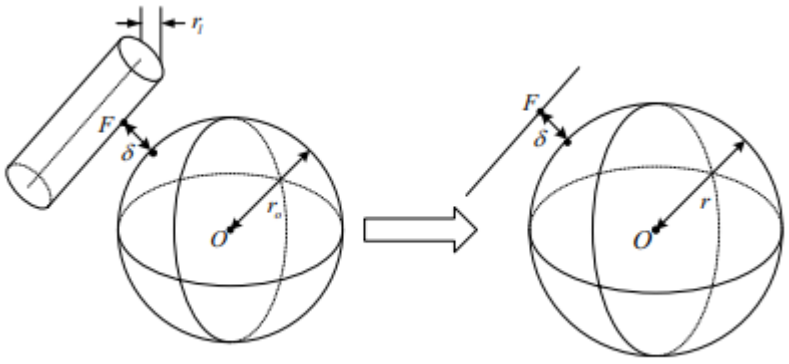


Figura 4.2: Representación simplificada de la evasión de obstáculos.

Fuente: [6]

Con la información obtenida se puede planificar una nueva trayectoria al modificar, las posiciones y/o velocidades articulares, teniendo en consideración las restricciones del sistema.

Capítulo 5

Dinámica

La dinámica se ocupa de la relación entre las fuerzas que actúan sobre un cuerpo y el movimiento que en él se origina. Por tanto, el modelo dinámico de un robot tiene por objetivo conocer la relación entre el movimiento del robot y las fuerzas implicadas en el mismo. Esta relación se obtiene mediante el denominado modelo dinámico, que establece la relación matemática entre:

1. La localización del robot definida por sus variables articulares o por las coordenadas de localización de su extremo, y sus derivadas: velocidad y aceleración.
2. Las fuerzas y pares aplicados en las articulaciones (o en el extremo del robot).
3. Los parámetros dimensionales del robot, como longitudes, masas e inercias de sus eslabones.

Dependiendo de la cantidad de grados de libertad con los que cuente un robot es como aumenta la complejidad del mismo y muchas veces se debe recurrir a métodos iterativos mediante la utilización de un procedimiento numérico. Si bien puede resultar complicado obtener el modelo dinámico de un robot, este es imprescindible para acciones como:

1. Simulación del movimiento del robot.
2. Diseño y evaluación de la estructura mecánica del robot.
3. Dimensionamiento de los actuadores.
4. Diseño y evaluación del control dinámico del robot

La obtención del modelo dinámico de un mecanismo, y en particular de un robot, se basa fundamentalmente en el planteamiento del equilibrio de fuerzas establecido en la segunda ley de Newton, o su equivalente para movimientos de rotación, la denominada ley de Euler:

$$\begin{aligned} \sum F &= \frac{d}{dt}(mv) \\ \sum F &= \frac{d}{dt}(I\omega) = I\dot{\omega} + \omega \times (I\omega) \end{aligned} \tag{5.1}$$

Siendo así para el caso de un péndulo simple como el que se muestra en 8.1 se obtiene la siguiente expresión.

$$\begin{aligned} \tau - MgLcos(q) =& I\frac{d^2q}{dt^2} \rightarrow \\ \tau =& ML^2\ddot{q} + MgLcos(q) \end{aligned} \tag{5.2}$$

De la ecuación (5.2), si se conoce el par motor τ se puede obtener q(t) y sus derivadas $\dot{q}$(t) y $\ddot{q}$(t) con lo que sería posible conocer la evolución de la coordenada articular del robot, su velocidad y aceleración. De forma inversa, si se pretende que q (t) evolucione según una determinada función del tiempo, sustituyendo en (5.2) podría obtenerse el par τ (t) que sería necesario aplicar. Si el robot tuviese que ejercer alguna fuerza en su extremo, ya sea al manipular una carga o realizar un proceso sobre alguna pieza, bastaría con incluir esta condición en la Ecuación (5.2) y proceder del mismo modo.
Al igual que en el caso de la cinemáticas, para la dinámica también cuenta con 2 modelos, modelos dinámicos directo e inverso:

- **Modelo dinámico directo**: expresa la evolución temporal de las coordenadas articulares del robot en función de las fuerzas y pares que intervienen $q(t)$ = f($\tau(t)$)
- **Modelo dinámico inverso:** determina las fuerzas y pares necesarios para conseguir una evolución de las coordenadas articulares determinada τ (t) = g($q(t)$)

Debe tenerse en cuenta que junto con las fuerzas de inercia y gravedad, aparecen fuerzas centrípetas debido a la rotación y fuerzas de Coriolis, ocasionadas por el movimiento relativo existente entre los diversos eslabones, que dependen de la configuración instantánea del manipulador. Las ecuaciones de Euler-Lagrange representan la mejor alternativa de modelado para un robot, debido a las propiedades matemáticas que se deducen de la aplicación de este método. La formulación Lagrangiana se establece en las siguientes ecuaciones:

$$\begin{aligned} \mathcal{L} =& E_c(q,\dot{q}) - E_p(q) \\ \tau =& \frac{d}{dt}\left[\frac{\partial \mathcal{L}(q,\dot{q})}{\partial \dot{q}}\right] - \frac{\partial \mathcal{L}(q,\dot{q})}{\partial q} \end{aligned} \tag{5.3}$$

donde $\mathcal{L}$ es el Lagrangiano que expresa la diferencia entre la energía cinética ($E_c(q,\dot{q})$) y la energía potencial ($E_p(q)$) del robot. $\frac{d}{dt}$ es la derivada temporal de la derivada parcial del Lagrangiano con respecto a la velocidad ($\frac{\partial \mathcal{L}(q,\dot{q})}{\partial \dot{q}}$) y $\frac{\partial \mathcal{L}(q,\dot{q})}{\partial q}$ la derivada parcial del Lagrangiano con respecto a la posición. Por su parte q=[q_1, q_2, ... , q_n]^T es el vector de posiciones articulares, $\dot{q}$=[$\dot{q}_1$, $\dot{q}_2$, ... , $\dot{q}_n$]^T es el vector de velocidades articulares y τ=[τ_1, τ_2, ... , τ_n]^T es el vector de pares aplicados, donde la i-ésimo par τ_i esta relacionado con la i-ésima coordenada generalizada q_i. Teniendo en cuenta que el vector de pares aplicados está construido de la siguiente manera:

$$\tau = \tau_{motor} - \tau_{perturbacion} - \tau_{rozamiento \quad viscoso} - \tau_{rozamientoseco} \tag{5.4}$$

La energía cinética tienen una estructura cuadrática bien definida en función de la velocidad articular de la siguiente forma:

$$E_c(q,\dot{q}) = \frac{1}{2}\dot{q}^T M(q)\dot{q} \tag{5.5}$$

Donde $M(q) \in \mathbf{R}^{nxn}$ es la matriz de inercia del robot y es definida positiva (simétrica), mientras que la energía potencial, no tiene una forma definida pero depende exclusivamente del vector de posición q.
Al sustituir estos valores, las ecuaciones de Euler-Lagrange quedan de la siguiente forma:

$$\frac{\partial \mathcal{L}(q,\dot{q})}{\partial \dot{q}} = \frac{\partial}{\partial \dot{q}}\left[\tfrac{1}{2}\dot{q}^T M(q)\dot{q}\right] - \frac{\partial E_p(q)}{\partial \dot{q}} = M(q)\dot{q} \tag{5.6}$$

$$\frac{d}{dt}\left[\frac{\partial \mathcal{L}(q,\dot{q})}{\partial \dot{q}}\right] = M(q)\ddot{q} + \dot{M}(q)\dot{q} \tag{5.7}$$

$$\frac{\partial \mathcal{L}(q,\dot{q})}{\partial q} = \frac{\partial}{\partial q}\left[\tfrac{1}{2}\dot{q}^T M(q)\dot{q}\right] - \frac{\partial E_p(q)}{\partial q} \tag{5.8}$$

Por lo tanto las ecuaciones de movimiento (5.3) para un robot de n grados de libertad, tienen la siguiente forma:

$$\tau = M(q)\ddot{q} + \dot{M}(q)\dot{q} - \frac{\partial}{\partial q}\left[\frac{1}{2}\dot{q}^T M(q)\dot{q}\right] + \frac{\partial E_p(q)}{\partial q} \tag{5.9}$$

Algunos datos que se deben considerar son:
$C(q,\dot{q})$ es la matriz de fuerza centrípeta y de Coriolis de la forma:

$$C(q,\dot{q})\dot{q} = \dot{M}(q)\dot{q} - \frac{\partial}{\partial q}\left[\frac{1}{2}\dot{q}^T M(q)\dot{q}\right] \tag{5.10}$$

el vector de fuerzas o pares gravitacionales $g(q)$ se obtiene del gradiente de la energía potencial, es decir

$$g(q) = \frac{\partial E_p(q)}{\partial q} \tag{5.11}$$

y q es el vector de posiciones articulares, $\dot{q}$ es el vector de velocidades articulares y $\ddot{q}$ es el vector de aceleraciones articulares.
Teniendo en cuenta las consideraciones anteriores se puede expresar la ecuación (5.9) como:

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) \tag{5.12}$$

Se considera que la ecuación (5.12) solo se aplica a robots que operan en su rango de operación nominal y los eslabones que los componen son rígidos.

5.1. Modelo del péndulo

De las ecuaciones de Euler-Lagrange, el modelo dinámico de un brazo robótico esta dado por:

$$[ml^2]\ddot{q} + b\dot{q} + mlgsen(q) = \tau$$

Siendo la energía cinética K de la siguiente forma:

$$\begin{aligned} K(q,\dot{q}) &= \frac{1}{2}v^t v + \frac{1}{2}I\dot{q}^2 \\ &= \frac{1}{2}[ml^2 + I]\dot{q}^2 \end{aligned}$$

Mientras tanto la energia potencial U adquiere la forma de:

$U(q)$ = $mglh$ = $mgl[1 - cos(q)]$, siendo h una diferencia de posiciones de $l - cos(q)$.

Por lo tanto, el lagrangiano para este caso, esta dado por:

$$\begin{aligned} \mathcal{L}(q,\dot{q}) &= K(q,\dot{q}) - U(q) \\ &= \frac{1}{2}[ml^2 + I]\dot{q}^2 - mgl[1 - cos(q)] \end{aligned}$$

La ecuación de movimiento de Euler-Lagrange para este caso del péndulo, adquiere la siguiente forma (5.3).
Para la sintonización PID de los robots manipuladores, se busca que el control garantice el cumplimiento del objetivo de posición en forma local. Es decir, que para una posición deseada constante q_d, la sintonización asegure que $\lim_{t\to\infty}$ el error de posición $\tilde{q}(t)$ = 0, siempre y cuando el error de posición inicial $\tilde{q}(0)$ y la velocidad $\dot{q}(0)$ sean suficientemente ”pequeños”.

5.2. Método Recursivo de Newton-Euler

Como se demostró con el péndulo la metodología de Euler-Lagrange presenta una adecuada opción para obtener el modelo dinámico de robots manipuladores, sin embargo de acuerdo con [29] el costo computacional de este algoritmo es de orden O(n^4) , haciendo que este algoritmo demasiado demandante para poder obtener el modelo del UR5 (robot de 6 grados de libertad). Por eso se optó por emplear una metodología que nos permita obtener el modelo de una manera mas eficiente. Es el caso de la formulación de Newton-Euler, la cual utiliza un proceso recursivo de propagación de velocidades/aceleraciones hacia adelante y de fuerzas/pares hacia atrás, lo que evita la manipulación de matrices grandes y se implementa como un algoritmo numérico rápido (O(n)), haciendo que este método resulte uno adecuado para el control en tiempo real y la simulación de robots manipuladores. La formulación de Newton-Euler parte

del equilibrio de fuerzas y pares para cada elemento como se muestra en la ecuación:

$$\begin{aligned} \sum F_i &= \frac{d}{dt}(m_i v_i) = m_i \dot{v}_i \\ \sum T_i &= \frac{d}{dt}(I_i \omega_i) = I_i \dot{\omega}_i + \omega_i \times (I_i \omega_i) \end{aligned} \tag{5.13}$$

Donde:

- F_i son las fuerzas ejercidas sobre el eslabón i.
- T_i son los pares ejercidos sobre el eslabón i en torno a su centro de masas.
- m_i es la masa del eslabón i.
- I_i es el tensor de inercia del eslabón i en torno a su centro de masas, expresado en el sistema de referencia $\{Si\}$.
- v_i , $\dot{v}_i$ son la velocidad y aceleración lineal del centro de masas de la articulación i.
- ω_i, $\dot{\omega}_i$ son la velocidad y aceleración angular de la articulación i.

Un adecuado desarrollo de estas ecuaciones conduce a una formulación recursiva en la que se obtienen la posición, velocidad y aceleración del eslabón i referidos a la base del robot a partir de las variables cinemáticas del eslabón $i-1$ y del movimiento relativo de la articulación i. De este modo, partiendo del eslabón 1 se llega al eslabón n. Con estos datos se procede a obtener las fuerzas y pares actuantes sobre el eslabón i referidos a la base del robot a partir de los correspondientes al eslabón $i-1$, recorriéndose de esta forma todos los eslabones desde el eslabón n al eslabón 1.
El algoritmo se basa en operaciones vectoriales (con productos escalares y vectoriales entre magnitudes vectoriales, y productos de matrices con vectores) siendo más eficiente en comparación con las operaciones matriciales asociadas a la formulación Lagrangiana.

Capítulo 6

Control

De acuerdo con [28], el problema general de control en coordenadas articulares de robots manipuladores es el control de trayectorias o control de movimiento, el cual consiste en determinar los pares aplicados a los servomotores que forman las articulaciones, de tal forma que las posiciones asociadas a las coordenadas articulares del robot $q(t)$ sigan con exactitud la posición deseada $q_d(t)$ variante en el tiempo. Mientras el control de posición o regulación es un caso particular de control de robots manipuladores en donde no hay una referencia variable en el tiempo; mas bien, es un punto constante denominado punto deseado o set point.

6.1. Control PID

Anteriormente se mencionó el control PID para un sistema lineal.
Para el caso de un control PID para robots manipuladores tiene la siguiente manera de acuerdo con [18]:

$$\tau = K_p\tilde{q} + K_v\dot{\tilde{q}} + K_i\int_0^t \tilde{q}(\sigma)d\sigma \tag{6.1}$$

donde las matrices K_p, K_v, $K_i \in \mathbf{R}^{nxn}$, son las respectivas ganancias, estas son matrices simétricas y definidas positivas. La mayoría de los robots manipuladores actualmente son controlados por este tipo de control.

La acción integral del controlador PID (6.1) introduce una variable de estado adicional denotada por ξ y cuya derivada temporal es $\xi = \tilde{q}$.
La ley de control se pude expresar en las siguientes 2 ecuaciones:

$$\begin{aligned} \tau &= K_p\tilde{q} + K_v\dot{\tilde{q}} + K_i\xi \\ \xi &= \tilde{q} \end{aligned} \tag{6.2}$$

La ecuación de lazo cerrado se obtiene sustituyendo τ de (6.3) en (5.12) quedando:

$$\begin{aligned} M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) &= K_p\tilde{q} + K_v\dot{\tilde{q}} + K_i\xi \\ \xi &= \tilde{q} \end{aligned} \tag{6.3}$$

esta se puede expresar en el vector de estados $[\xi^t \tilde{q}^T \dot{\tilde{q}}^T]^T$ como:

$$\frac{d}{dt}\begin{bmatrix}\xi\\ \tilde{q}\\ \dot{\tilde{q}}\end{bmatrix} = \begin{bmatrix}\tilde{q}\\ \dot{\tilde{q}}\\ \ddot{q}_d - M(q)^{-1}\left[K_p\dot{q} + K_v\dot{\tilde{q}} + K_i\xi - C(q,\dot{q})\dot{q} - g(q)\right]\end{bmatrix} \tag{6.4}$$

6.2. Control predictivo basado en modelos

De acuerdo con [33] la mayoría de los sistemas no lineales derivados de principios físicos se describen mediante modelos en tiempo continuo en forma de ecuaciones diferenciales no lineales:

$$\frac{dx}{dt} = f(x,u) \tag{6.5}$$

Para esta clase de sistemas, la ley de control con las mejores propiedades en lazo cerrado se obtiene resolviendo el siguiente problema de control óptimo en horizonte infinito con restricciones. El costo asociado se define como:

$$v_\infty(x, u(\cdot)) = \int_0^\infty l(x(t)), u(t))dt \tag{6.6}$$

donde $x(t)$ y $u(t)$ deben satisfacer $\dot{x} = f(x,u)$. El problema de control optimo $P_\infty(x)$ esta definido por:

$$\text{mín}\, V_\infty(x, u(\cdot))$$

sujeto a:

$$\dot{x} = f(x,u) \quad x(0) = x_0 \tag{6.7}$$

$$(x(t), u(t)) \in F \quad \text{para todo } t \in \mathbb{R} \geq 0 \tag{6.8}$$

Dado que el MPC puede inferir la secuencia de control para todos los estados, se puede emplear esta formulación para calcular dichos estados en el horizonte infinito. Sin embargo, no se recomienda este enfoque, siendo más práctico el utilizar Control Predictivo Basado en Modelos (MPC) para calcular en línea solo la secuencia de control óptima para el estado actual (x).

Inicialmente, se considera el problema de regulación, donde el estado objetivo es el origen. Si x es el estado actual e *i* es el instante actual, el problema de control óptimo consiste en minimizar un costo definido en el intervalo de tiempo desde *i* hasta *N + i*

El problema de control óptimo $P_N(x,i)$, evaluado en el evento (x,i), se define como minimizar el costo total:

$$\sum_{k=i}^{i+N-1} L(x(k), u(k)) + V_f(x(N+i)) \tag{6.9}$$

Si se asume que *L* es una función continua y *L* (0,0) = 0 el control optimo y la secuencia de estados se obtienen de resolver $P_N(x,i)$, debido a que las funciones *L* y V_f son invariantes en el tiempo, se puede expresar el problema de control óptimo como una minimización de:

$$\sum_{k=0}^{N-1} L(x(k), u(k)) + V_f(x(N)) \tag{6.10}$$

Delimitando el caso para un sistema no lineal, el sistema a ser controlado está dado por la ecuación:

$$x^+ = f(x, u) \tag{6.11}$$

Donde se asume que f es dos veces continuamente diferenciable y el sistema esta sujeto a estados y limites de control:

$$x \in \mathbf{x}, u \in \mathbf{u}$$

donde **x** esta acotada y la entrada de control ***u*** está físicamente delimitada (troque máximo, voltaje máximo), cada conjunto contiene al origen. La función de costos esta definida por:

$$V_N(x, u) = \sum_{i=0}^{N-1} l(x(i), u(i)) + V_f(x(N)) \tag{6.12}$$

El problema $P_N(x)$ busca encontrar la secuencia de control u que minimice el costo total $V_N(x, u)$ mientras cumple las restricciones dinámicas, de estado, de control y terminales. Este enfoque asegura que el sistema evolucione de manera eficiente desde el estado inicial x hacia la región terminal $\mathbb{X}_f$, garantizando un comportamiento estable y óptimo en un horizonte finito. En esta formulación, para cada instante i, el estado $x(i)$ se define como:

$$x(i) = \phi(i; x, u)$$

donde ϕ (i;x,u) es la solución a la ecuación $x^+ = f(x, u)$ en el instante i, si el estado inicial es x en el tiempo cero y la secuencia de control aplicada es u.
De acuerdo con [34], dado un sistema descrito por la ecuación (6.11) donde $f : \mathbb{X} \times \mathbb{U} \to \mathbb{X}$ representa un mapeo no lineal conocido que determina la evolución del estado x^+ a partir del estado actual x y la acción de control u, es posible construir trayectorias de estado mediante iteración sucesiva del modelo. Partiendo del estado actual $x(n)$, para cualquier secuencia de control $u(0), ..., u(N-1)$ con un horizonte de predicción $N \geq 2$, se puede iterar el sistema para construir la trayectoria x_u definida por:

$$x_u(0) = x(n), \quad x_u(k+1) = f(x_u(k), u(k)), \quad k = 0, ..., N-1 \tag{6.13}$$

De esta manera, obtenemos predicciones $x_u(k)$ para los estados del sistema $x(n+k)$ en el tiempo t_{n+k} en el futuro.
Se usa el control optimo para determinar $u(0), ..., u(N-1)$ tal que x_u sea tan cercano posible a x_*=0. Para ello, medimos la distancia entre $x_u(k)$ y $x_* = 0$ mediante una función *l*($x_u(k)$,$u(k)$). Aquí no solo permitimos penalizar la desviación del estado con respecto a la referencia, sino también, si se desea, la distancia de los valores de control $u(k)$ a un control de referencia u_*, que se selecciona como u_* = 0. Una opción común y popular para este propósito es la función cuadrática:

$$l(x_u(k), u(k)) = ||x_u(k)||^2 + \lambda ||u(k)||^2,$$

donde $|| \cdot ||$ denota la norma Euclidiana y $\lambda \geq 0$ es un parámetro de ponderación para el control, que también podría elegirse como 0 si no se desea ninguna penalización del

control. El problema de control óptimo ahora se lee

$$\text{minimizar} \quad J(x(n), u(\cdot)) := \sum_{k-0}^{N-1} l(x_u(k), u(k))$$

con respecto a todas las secuencias de control admisibles $u(0), ..., u(N-1)$ con x_u generadas por (6.13).
Suponiendo que el problema de control óptimo tiene una solución dada por la secuencia de control minimizadora $u^\star(0), ..., u^\star(N-1)$, para obtener el valor de retroalimentación deseado $\mu(x(n))$ahora establecemos $\mu(x(n))$:= $u(0)$, es decir, aplicamos el primer elemento de la secuencia de control óptima.
En los siguientes instantes de tiempo t_{n+1}, t_{n+2}, ... repetimos el procedimiento con las nuevas mediciones $x(n+1)$, $x(n+2)$, ... para obtener los valores de retroalimentación $\mu(x(n+1))$, $\mu(x(n+2)), ...$ En otras palabras, obtenemos la ley de retroalimentación μ mediante una optimización iterativa en línea sobre las predicciones generadas por nuestro modelo. Por lo tanto, el horizonte de predicción es móvil, y este horizonte móvil es la segunda característica clave del control predictivo del modelo.

En cualquier caso, requerimos que si estamos en el equilibrio x_* y usamos el valor de control u_* para permanecer en el equilibrio, entonces el costo debería ser 0. Sin embargo, fuera del equilibrio, el costo debería ser positivo.

$$l(x_*, u_*) = 0 \quad \text{y} \quad l(x, u) > 0 \quad \text{para todo} \quad x \in X, u \in U \text{ con } x \neq x_* \tag{6.14}$$

Si el sistema esta definido en el espacio Euclidiano *X* = $\mathbb{R}^d$ y *U* = $\mathbb{R}^m$, entonces se puede asumir sin perdida de generalidad x_*= 0 y u_* = 0, de no ser el caso se puede reemplazar $f(x, u)$ por $f(x + x_*, u + u_*)$ - x_*, lo cual corresponde a una simple transformación lineal de coordenadas en *X* y *U*. En este caso, una opción popular para cumplir la condición (6.14) es la función cuadrática.

$$l(x, u) = ||x||^2 + \lambda||u||^2$$

donde $||\cdot||$ denota la norma euclidiana habitual, y $\lambda \geq 0$ es un parámetro que pondera el costo asociado al control u. Para extender este concepto a espacios métricos generales, donde los conjuntos de estados *X* y de control *U* cuentan con métricas d_x y d_u respectivamente, la función de costo análoga se define como:

$$l(x, u) = d_x(x, x_*)^2 + \lambda d_u(u, u_*)^2 \tag{6.15}$$

Dada la función de costo $l(x, u)$ y un horizonte de predicción $N \leq 2$, podemos formular el esquema básico de un control predictivo basado en modelo no lineal (NMPC). El problema de control óptimo (OCPN) implica minimizar la función de costo total sobre un conjunto de secuencias de control admisible U^N. Se define U^N (x_0) $\subseteq$ *U* ^{N} como el conjunto de secuencias de control de longitud N que dependen del estado inicial x_0.
Por el momento, simplemente establecemos U^N (x_0): *U*^{N} para todo $x_0 \in$ *X*,es decir, que no se imponen restricciones adicionales a las del estado inicial.
Con base en esto se establece un algoritmo NMPC con referencia variable en el tiempo x^{ref} para cada instante de muestreo t_n, $n = 0, 1, 2...$:

1. Medir el estado del sistema $x(n) \in X$
2. Establecer x_0 como $x(n)$, resolver el problema de control optimo:

$$\begin{aligned} \text{minimizar} \quad & J_N(n, x_0, u(\cdot)) := \sum_{k=0}^{K-1} l(n+k, x_u(k, x_0), u(k)) \\ \text{con respecto a} \quad & u(\cdot) \in \mathbb{U}^N(x_0), \quad \text{sujeto a} \\ & x_u(0, x_0) = x_0, \quad x_u(k+1, x_0) = f(x_u(k, x_0), u(k)) \end{aligned}$$

   y denotamos la secuencia de control óptima obtenida por $u^\star$ (·) $\in \mathbb{U}^N(x_0)$.

3. Definir el valor de retroalimentación NMPC $\mu_N(n, x(n))$:= $u^\star$ (0) $\in U$ y utilizar este valor de control en el próximo período de muestreo.

Una de las principales razones del éxito del MPC es su habilidad para tomar en cuenta restricciones explicitas, aceptando tanto restricciones de control, como de estados. Para ello, se introduce un conjunto de restricciones de estado no vacío $\mathbb{X} \subseteq X$ y para cada $x \in \mathbb{X}$ se introduce un conjunto de restricciones de control no vacío $U(x) \subseteq U$. Por supuesto, U también puede elegirse independientemente de x. La idea tras la introducción de estos conjuntos es que queremos que las trayectorias se encuentren en $\mathbb{X}$ y los valores de control correspondientes en $U(x)$.
Considerando el sistema (6.11) y los conjuntos de restricciones de estado y control $\mathbb{X} \subseteq X$ y $U(x) \subseteq U$.

1. El estado x $\in \mathbb{X}$ se denominan estados admisibles y los valores de control $u \in \mathbb{U}$(x) se denominan valores de control admisibles para x.
2. Para $N \in \mathbb{N}$ y un valor inicial $x_0 \in \mathbb{X}$ se llama secuencia de control u $\in U^N$ y la trayectoria correspondiente x_u (k, x_0) admisible para x_0 hasta el tiempo N, si

$$u(k) \in \mathbb{U}(x_u(k, x_0)) \quad \text{y} \quad x_u(k+1, x_0) \in \mathbb{X}$$

   Se cumple para todo $k = 0, ..., N-1$. Se denota el conjunto de secuencias de control admisibles para x_0 hasta el tiempo N como U^N (x_0).
3. Una secuencia de control u $\in U^\infty$ y la correspondiente trayectoria $\mathrm{x}_u(k, x_0)$ se denominan admisibles para x_0 si son admisibles para x_0 en cada instante $N \in \mathbb{N}$. Se denota el conjunto de secuencias de control admisibles para x_0 como $U^\infty(x_0)$.
4. Una ley de retroalimentación (posiblemente variable en el tiempo) $\mu : \mathbb{N}_0 \times \mathrm{X} \to U$ se considera admisible si $\mu(n, x) \in \mathbb{N}^1(x)$ se cumple para todo $x \in \mathbb{X}$ y todo $n \in \mathbb{N}_0$.

La función de costo que se utilizará en nuestra optimización debe penalizar la distancia de un estado arbitrario $x \in X$ a x_*. Además, a menudo se desea penalizar el control u $\in U$. Esto puede ser útil por razones computacionales, ya que los problemas de control óptimo pueden ser más fáciles de resolver si la variable de control está penalizada. Por otro lado, penalizar u también puede ser deseable para fines de modelado, por ejemplo, porque queremos evitar el uso de valores de control $u \in U$ correspondientes a energías altas y costosas.

6.3. Teorema de Steiner o teorema de ejes paralelos

Debido al alto costo computacional y los beneficios limitados de controlar la orientación del efector final, se propuso una simplificación del robot UR5 mediante la fijación de las articulaciones correspondientes a los eslabones 4, 5 y 6. Estos eslabones se consideran rígidamente unidos entre sí y con el eslabón 3, formando un nuevo eslabón compuesto. Como resultado, la masa y el centro de masa de este eslabón compuesto se calculan como la combinación de las propiedades físicas de los eslabones 3, 4, 5 y 6, lo que implica que el centro de masa individual de cada eslabón se reemplaza por un centro de masa conjunto. Esta aproximación reduce la complejidad del modelo dinámico y facilita el control del robot al disminuir el número de grados de libertad. De igual manera que [35] se planteo emplear el teorema de Steiner para hacer el traslado de los momentos de Inercia al nuevo centro de masa compuesto.
El Teorema de los Ejes Paralelos o Teorema de Steiner establece que el momento de inercia I de un cuerpo rígido de masa m respecto a cualquier eje paralelo a un eje que pasa por su centro de masa (CM) está dado por:

$$I = I_{cm} + md^2 \tag{6.16}$$

donde:

- I_{cm} es el momento de inercia del cuerpo respecto a un eje paralelo que pasa por su centro de masa.
- m es la masa total del cuerpo.
- d es la distancia perpendicular entre el eje que pasa por el CM y el eje paralelo de interés.

El momento de inercia depende del eje de rotación. Al cambiar la configuración de nuestro sistema (eslabones independientes a eslabones fijos), tanto los ejes de rotación como el centro de masa varían. Dado que los momentos de inercia de cada eslabón están referidos a sus propios centros de masa, no se pueden sumar directamente para obtener la inercia del conjunto. Steiner permite homogenizar la referencia al mismo centro de masa compuesto.

Capítulo 7

Control en Tiempo real

El presente trabajo propone un esquema de control predictivo basado en modelo (MPC) para un robot de tres grados de libertad (3 GDL), con el objetivo de evadir un obstáculos dinámicos , el robot RR, que comparte el mismo plano de trabajo durante una tarea de pick and place.

La implementación inicial del controlador se llevó a cabo en un entorno de simulación, donde se evaluó su capacidad de predicción y su eficacia en la evasión de colisiones bajo diferentes condiciones. Sin embargo, las simulaciones no consideran de manera directa las limitaciones temporales, retardos de comunicación, discretización del muestreo y otros efectos que aparecen al ejecutar el control sobre un sistema físico. Por esta razón, se plantea como siguiente etapa la implementación en tiempo real del controlador predictivo, la cual constituye un paso intermedio esencial antes de su aplicación sobre un sistema robótico real.

El control en tiempo real es aquel en el que el controlador recibe, procesa y responde a la información del sistema dentro de límites de tiempo muy estrictos, garantizando que las acciones de control ocurran con la rapidez necesaria para mantener el comportamiento deseado del sistema físico. De acuerdo con [36], el tiempo de muestreo y la duración del ciclo de control deben diseñarse conjuntamente en función de la dinámica del sistema, ya que ambos parámetros determinan la capacidad del controlador para responder adecuadamente en tiempo real.

7.1. Software para Optimización Dinámica (ACADOS)

Frente a los problemas de optimización que se presentan al intentar controlar un sistema con varios estados, existe una herramienta que ayuda a realizar este proceso conocida como ACADOS. De acuerdo con [37] ACADOS es un paquete de software modular y eficiente para la resolución de programas no lineales (PLN) con una estructura de problema de control óptimo (PCO). Estos problemas deben resolverse repetidamente en un MPC y estimación de horizonte móvil (MHE). Su eficiencia computacional y modula-

ridad hacen de ACADOS la opción ideal para aplicaciones en tiempo real. Está diseñado para aplicaciones de alto rendimiento y computación embebida, y se ha utilizado con éxito en una amplia gama de aplicaciones.
ACADOS está escrito en C, pero los problemas de control se pueden formular fácilmente utilizando el marco simbólico CasADi a través de las interfaces de alto nivel de ACADOS con los lenguajes de programación Python, MATLAB y Octave.
ACADOS se basa en un flujo de trabajo coordinado, comienza con la formulación del problema mediante CasADi a través de Python. Internamente, emplea el método de disparo múltiple para discretización temporal y utiliza solucionadores SQP (Square Cuadratic Problem) especializados que aprovechan la estructura de los problemas de control óptimo. Para garantizar eficiencia, integra métodos numéricos avanzados para resolver ODE/DAE con cálculo de sensibilidades, mientras que su arquitectura modular y implementación en C permiten optimizar el rendimiento para aplicaciones en tiempo real y sistemas embebidos, haciendo posible su uso en aplicaciones de control de alta frecuencia.

ACADOS se utiliza en una amplia gama de aplicaciones. Algunos ejemplos incluyen; Robótica: NMPC en tiempo real para cuadricópteros, locomoción con patas y plataformas robóticas ágiles. Vehículos autónomos: Se utiliza en proyectos para sistemas de asistencia a la conducción. Sistemas energéticos: Control basado en optimización para micro-redes y aerogeneradores. Biomecánica: Control óptimo en biomecánica mediante bibliotecas como bioptim. Aeroespacial: Aplicaciones en optimización y control de trayectorias para drones y aeronaves de alas móviles.
Algunas de las limitaciones con las que cuenta ACADOS son:

1. No permite definir restricciones no lineales diferentes para cada etapa directamente, se requiere usar parámetros que varían por etapa y realizarlo en la interfaz de C.

2. Puede llegar a presentar problemas de convergencia: Si el problema es difícil de resolver, el solver puede alcanzar el número máximo de iteraciones.

3. Técnicas paras mejorar la convergencia solo están disponibles en las ultimas versiones de ACADOS.

Debido a esto a pesar de que ACADOS es una herramienta útil para el control MPC o MHE se requiere de ciertas consideraciones para poder emplearlo adecuadamente.

7.2. CasADI

Debido a que ACADOS requiere del lenguaje CasADI para formular los problema de control, se presentaran las características de este lenguaje.
Con base en [38] CasADi es una herramienta de código abierto para optimización no lineal y diferenciación automática. Su principal función es facilitar la formulación y resolución de problemas de optimización, especialmente aquellos con estructura de control óptimo. CasADi utiliza un enfoque basado en expresiones simbólicas para construir

modelos matemáticos y calcular derivadas de manera exacta y eficiente mediante AD. Sus características principales son:

- Diferenciación automática : Calcula derivadas exactas (hasta precisión de máquina) mediante la descomposición de funciones en operaciones elementales y la aplicación de la regla de la cadena.
- Representación simbólica flexible: Permite definir variables y funciones simbólicas (escalares, vectoriales o matriciales) utilizando tipos de datos como SX (para expresiones escalares) o MX (para operaciones matriciales más generales).
- Integración con solvers numéricos: CasADi no es un solver en sí mismo, sino un ”lenguaje de modelado”que genera las derivadas y estructuras necesarias para conectarse con solvers numérico.
- Multiplataforma y multi-lenguaje: Está disponible para Python, MATLAB/Octave y C++, lo que facilita su uso en entornos diversos.

A pesar de estas características CasADI tiene algunos inconvenientes, CasADi no es una herramienta de diferenciación automática convencional que pueda usarse con código existente sin modificaciones, es decir, si se cuenta con un modelo escrito en C++, Python o MATLAB/Octave, se debe reestructurarlo completamente usando la sintaxis simbólica de CasADi, esto implica:

- Aprender un nuevo lenguaje de modelado simbólico (tipo SX y MX).
- Familiarizarse con conceptos de grafos de expresiones y manipulación simbólica.
- No es un sistema de álgebra computacional (CAS) completo, por lo que sus capacidades simbólicas son limitadas en comparación con herramientas como Mathematica o Maple.

Por estas razones tanto CasADi como ACADOS son consideradas herramientas potentes para el control óptimo y la optimización no lineal. Sin embargo, su uso viene acompañado de una curva de aprendizaje significativa y ciertas limitaciones en flexibilidad, desempeño numérico y facilidad de despliegue.

Para ilustrar la integración de ACADOS y CasADi en la implementación del control MPC, se desarrolló un diagrama de flujo que describe el proceso de generación de control predictivo en tiempo real empleando estas herramientas.

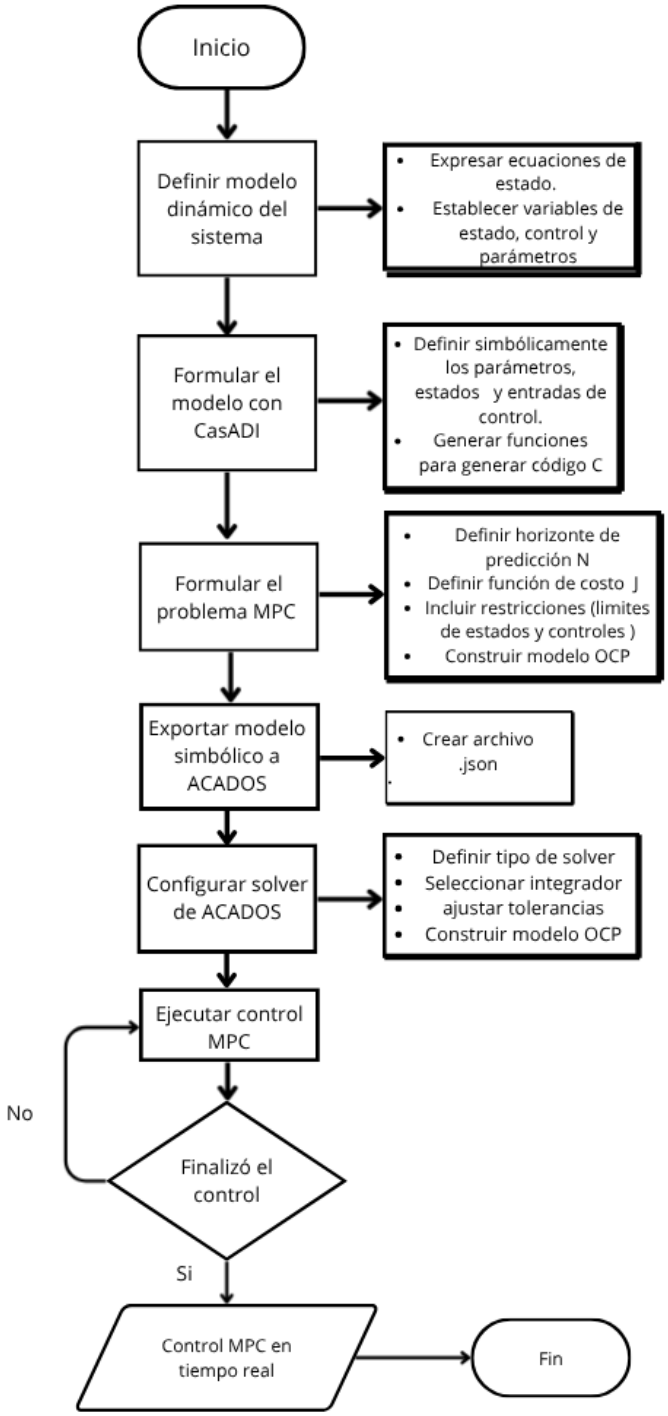


Figura 7.1: Diagrama de flujo de ACADOS y CasADI

Capítulo 8

Resultados

8.1. Especificaciones del Hardware

Para realizar las simulaciones de esta tesis se cuenta con un equipo de computo con las siguientes características:

- Un procesador Intel(R) Core(TM) i7 de 13va generación.
- Un disco duro de estado solido de un Tera.
- Una tarjeta grafica NVIDIA GeForce RTX 4070 de 8 GB.
- 2 tarjetas RAM de 16 GB, es decir 32 GB de RAM.
- Sistema operativo Windows 11 con base de 64 byts.

El hardware seleccionado para este proyecto constituye una plataforma computacional robusta y adecuada para ejecutar simulaciones complejas como evasión de obstáculos debido al alto volumen de memoria RAM que permite manejar múltiples operaciones simultáneamente.

8.2. Especificaciones de software

Para levar a cabo esta tesis se empleó el software siguiente:

- **Sistema operativo**: Windows 11 (64 bits)
- **Entorno de programación**: Python 3.12
- **Bibliotecas principales**:
    - NumPy y SciPy para los cálculos de los sistemas.
    - Matplotlib para la visualización de los resultados.
    - Time para medir los tiempos de ejecución.

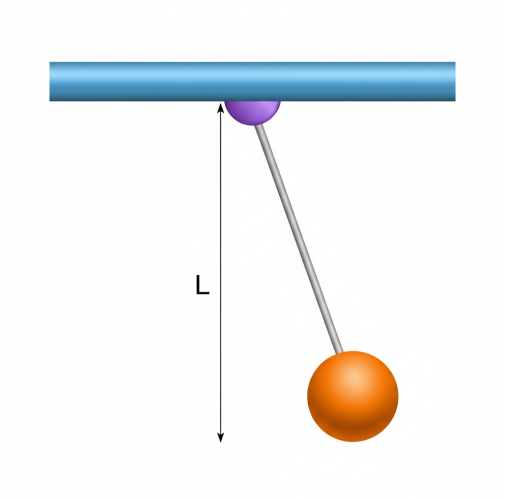


Figura 8.1: Representación de un péndulo simple

- **Entorno de modelado**: SolidWorks
- **Documentación**: LaTeX para la redacción de la tesis

La combinación del hardware de alto rendimiento (i7-13va, RTX 4070, 32GB RAM) con un stack de software especializado (Python, SolidWorks y Adams View) constituye una plataforma integral para la investigación en robótica. Este ecosistema permitió ejecutar simulaciones complejas de evasión de colisiones en tiempo real.

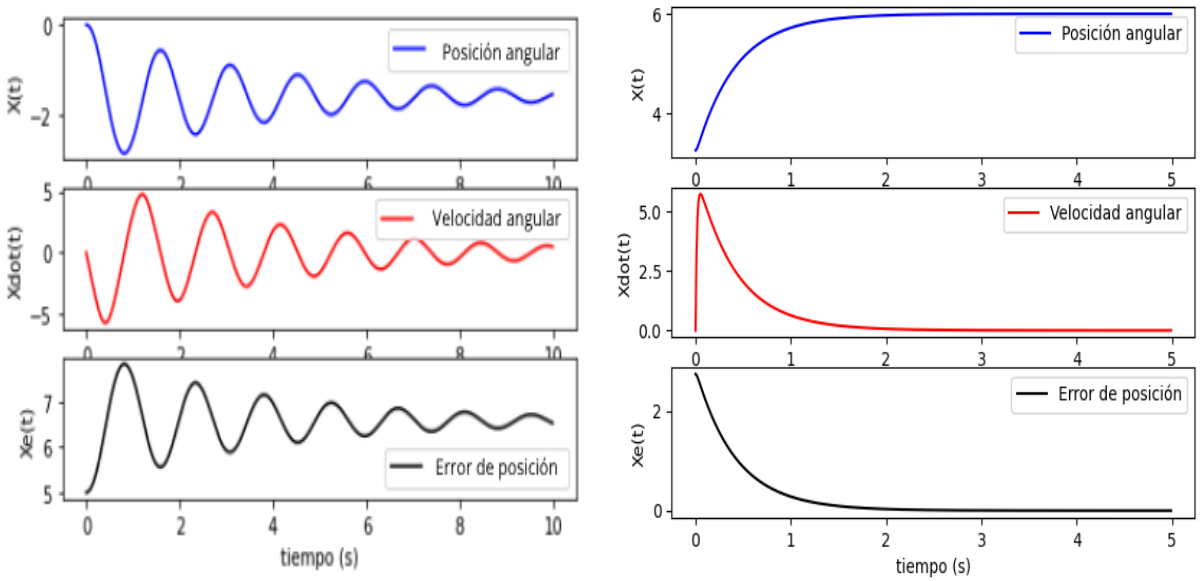
8.3. Control PID de un péndulo

En este apartado se implementará el control PID a un péndulo como un acercamiento a la programación requerida para programar el Control PID del robot RR y la obtención del modelo dinámico del robot RR. Como parte de la metodología que se esta siguiendo basa en ir de lo simple a lo complejo, es decir de un caso aislado a uno más general. A continuación se presentan las gráficas obtenidas de la aplicación de un control PID a un péndulo simple. Como el que se muestra en la Figura 8.1:

Con las ganancias Proporcional (Kp), Integral (Ki) y Derivativa (Kv) :
kp = 30, kv = 16 y ki = 5 . En las siguientes figuras las posición se representa en la gráfica de color azul, la velocidad en la gráfica de color verde y el error de posición en la gráfica de color negro.

8.4. Prototipado Virtual

En esta sección se abordara la creación de la plataforma virtual en el programa de Adams View.



(a) Péndulo sin control

(b) Péndulo con control PID

Figura 8.2: Comparación de un péndulo sin control contra uno con control PID

8.4.1. Familiarización con Adams View

Se realizo un péndulo básico como se muestra en la figura 8.3, el cual solo se sometió a los efectos de la gravedad sin fricción y se midió el ángulo de desplazamiento del péndulo con respecto a su punto de equilibrio. Como se observa en la gráfica, el movimiento describe una función sinodal. Para esta primera experiencia se conoció cómo generar cuerpos y situarlos dentro del espacio de trabajo, como medir ciertas características del modelo y los efectos que tienen estos cuerpos ante fuerzas como la gravedad.

Para seguir con esta línea, se construyó un brazo robótico simple de 3 grados de libertad con un gripper para simular la tarea de pick and place , como se muestra en la figura 8.4. En esta ocasión, se trabajó con elementos más complejos, así como el empleo de fuerzas de contacto entre los objetos y las mesas o el gripper para poder simular realistamente la tarea antes mencionada y un control de posición básico al aplicar restricciones con respecto al tiempo para la posición de las diferentes articulaciones actuadas del robot.

Una vez realizado este modelo se hizo evidente que este programa no es apropiado para realizar el modelado de estructuras complejas. Sin embargo, ofrece una gran cantidad de información, así como la capacidad de aplicar funciones para el control de los eslabones, incluso cuenta con una pestaña especializada en preparar el ambiente para poder realizar una virtualización.

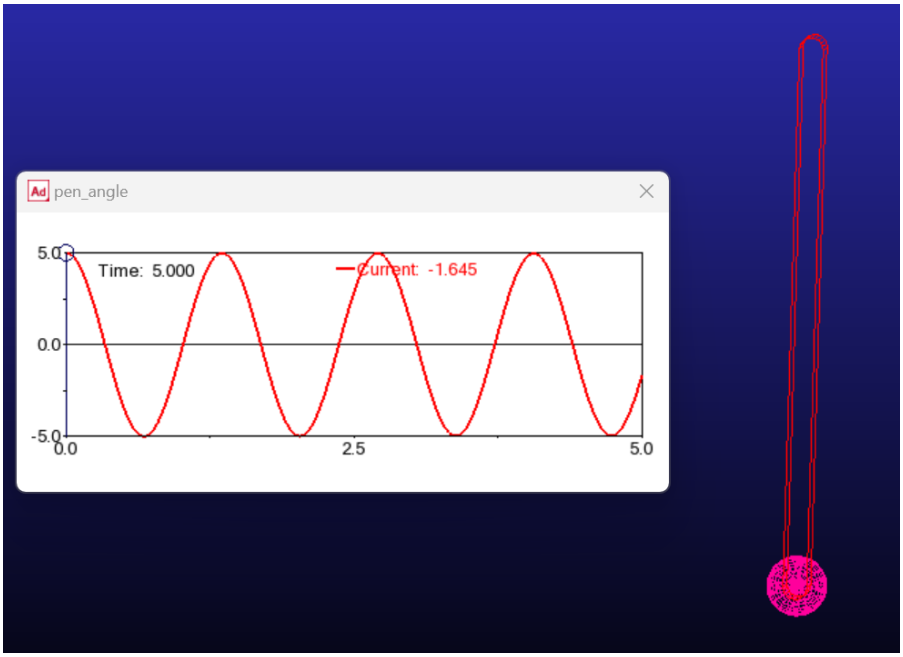


Figura 8.3: Péndulo en Adams View

8.4.2. Modelado 3D

Debido a que realizar el modelado de un robot en el entorno que ofrece Adams View resulta una tarea que consume mayor tiempo que realizarlo en un software especializado para este tipo de tareas, el modelado 3D de tanto el robot de 2 grados de libertad como el cobot se llevaron a cabo en el programa SolidWorks. Puesto que los cobots de Universal Robot cuentan con una gran cantidad de información accesible, se optó por emplear un robot UR5, se requirió diseñar cada uno de los componentes del cobot así como realizar el ensamble de los mismo para poder tener un modelo 3D funcional, en lugar de usar el CAD disponible en la página de Universal Robots, el cual corresponde a un sólido rígido sin articulaciones.
Una vez obtenido el modelo del robot, se buscó un gripper y se seleccionó un RG6 de On Robot; se descargaron los componentes y se realizó el ensamble del mismo. Después, se generó una pieza para acoplar el gripper al ensamble del robot. Para el robot RR se propuso un robot genérico. Se integraron dos bandas transportadoras para completar este proceso y se generó el ensamble de todos los componentes.
Una vez realizado el ensamble se identificó que existía un problema debido a que las dimensiones del cobot son relativamente pequeñas en comparación con las dimensiones de un robot industrial tradicional, por lo que se tuvo que agregar una mesa sobre la cual se posicionó el cobot. Se seleccionó una mesa de uso comercial (Universal Robots Station for UR5, with Workspace), sobre la cual se colocó el cobot.

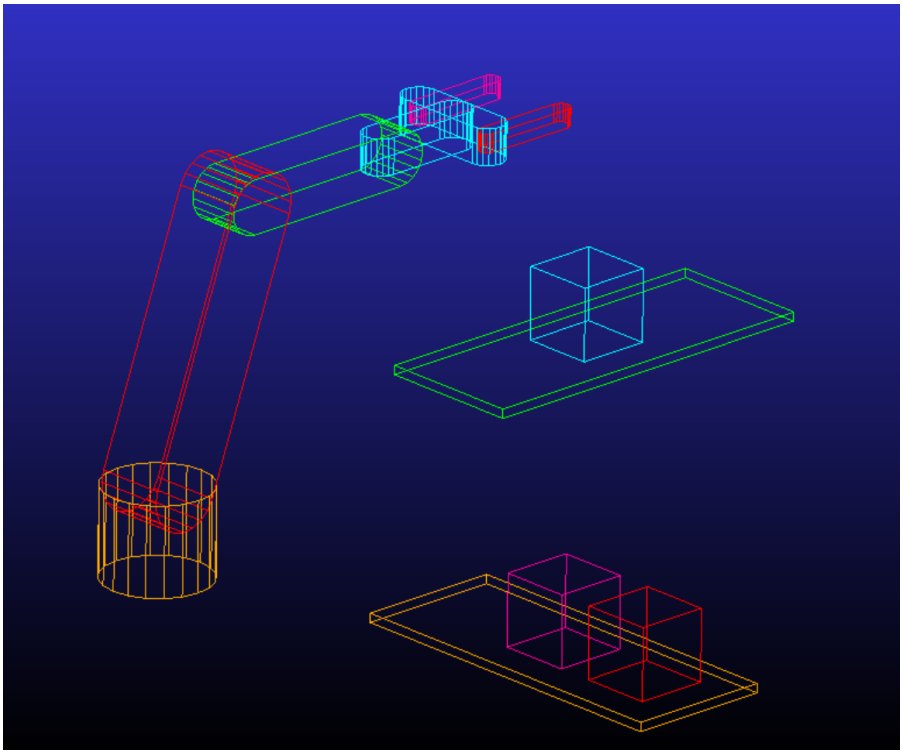


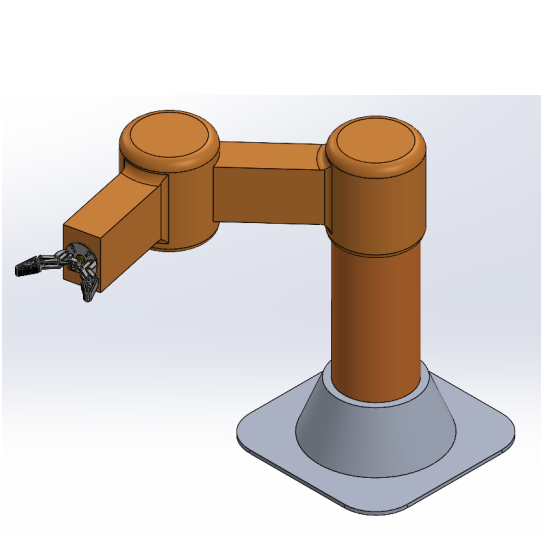
Figura 8.4: Brazo robotico en Adams View

8.4.3. Prototipado Virtual en Adams View

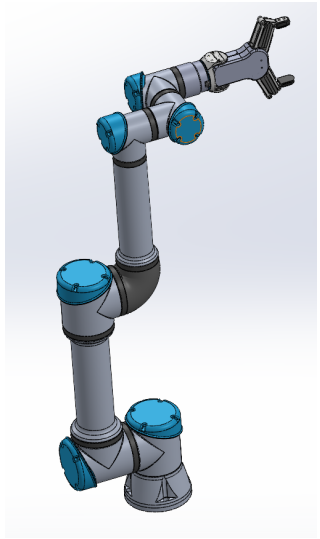
Una vez se contó con el ensamble completo, se empleó el complemento de SolidWorks motion para realizar un estudio de movimiento, el cual determina la ubicación espacial de los componentes y permite generar archivos .adm, desde los cuales se genera una plataforma digital en el entorno del programa. Al igual que para el ensamble final, se generaron modelos de los robots UR5 y RR en el ambiente de Adams View, los mismos que servirán para realizar la virtualización de los robots sin obstáculos, como se muestra en la figura 8.6.

Debido a que Adams no acepta más que las relaciones básicas de posición, como pueden ser coincidencia, distancia o concentricidad, se tiene que revisar que los modelos exportados a esta plataforma trabajen de la misma manera como lo hacen en SolidWorks y de no ser así realizar cambios en el modelo de Solid Work, volver a realizar un análisis de movimiento y exportar el ensamble corregido a Adams View. Una vez que se repitió este proceso la cantidad de veces necesarias para lograr que el ensamble final funcione correctamente se obtuvo el resultado mostrado en la figura 8.7.

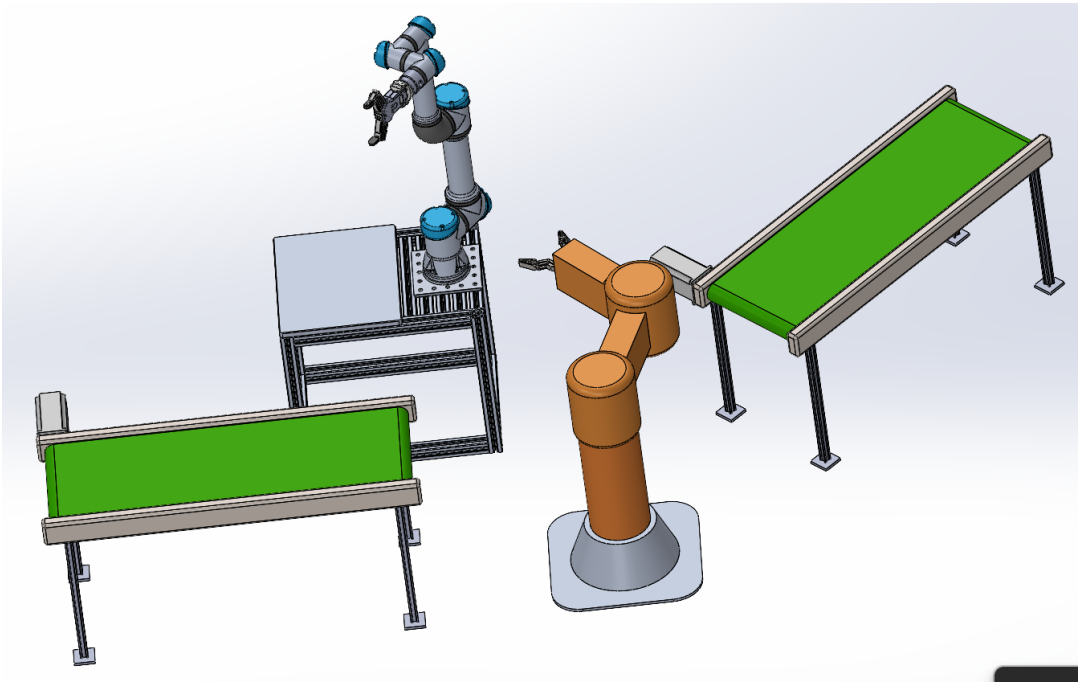
Cabe destacar que a pesar de que se contaba previamente con los modelos virtuales (en Adams View) de los componentes que forman el ensamble final, no se puede generar un ensamble final en Adams View como en SolidWorks, se requiere convertir el modelo, que se dese integrar al ensamble, en un archivo .cmd y fusionarlo con el ensamble actual, en un nuevo modelo. Es necesario verificar que las cotas que asocian el robot



(a) Robot RR



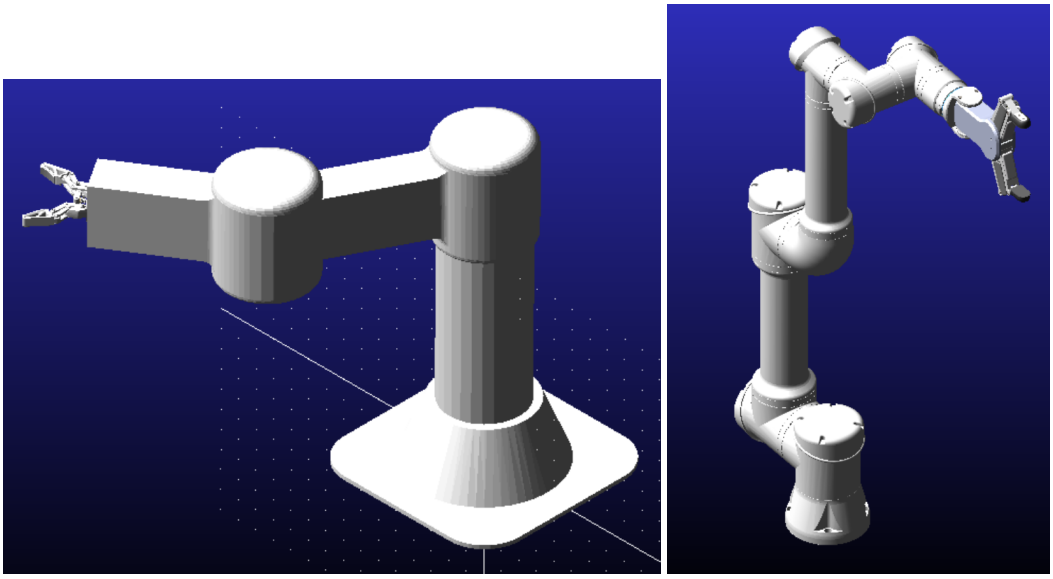
(b) UR5



(c) Ensamble final

Figura 8.5: Modelos 3D

con su espacio de trabajo permanezcan activas y consistentes después de la integración, asegurando la validez de las simulaciones en el modelo de ensamble generado. A pesar de los inconvenientes previamente mencionados, se logró integrar una plataforma que permite simular un trabajo cooperativo de 2 robots.



(a) Robot RR (b) UR5

Figura 8.6: Modelos en Adams View

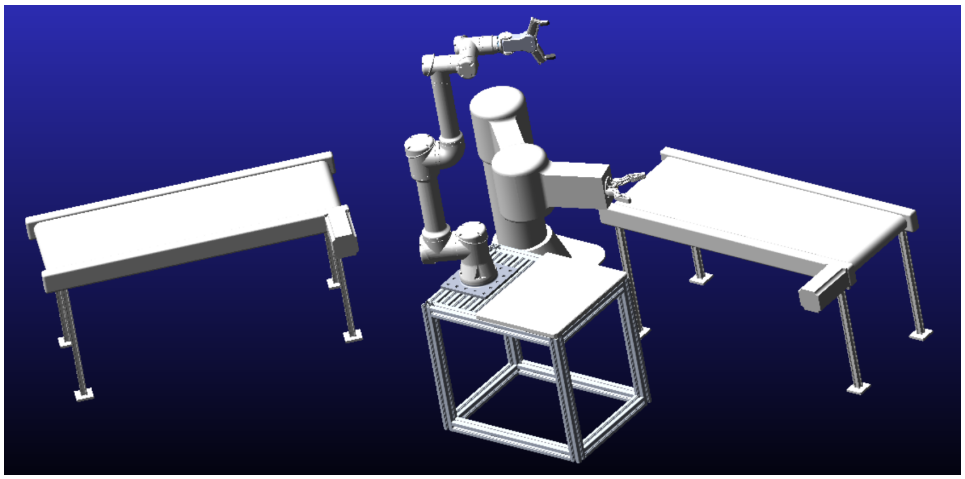


Figura 8.7: Ensamble final en Adams View

8.5. Aplicación práctica

8.5.1. Cinemática directa

En esta sección se revisarán los datos obtenidos de forma analítica para la obtención del modelo cinemático directo e inverso, así como la obtención del modelo dinámico para el robot de 2 grados de libertad.
Lo primero que se debe hacer es asignar los sistemas de referencia tanto a la base del robot como a las articulaciones siguiendo las especificaciones de la convención de

Denavit-Hartenberg, como se muestra en la figura 8.8.
Una vez ubicados los sistemas de referencia, se puede realizar la tabla de parámetros.
De la tabla de D-H se obtienen las siguientes relaciones de matrices para la cinemática

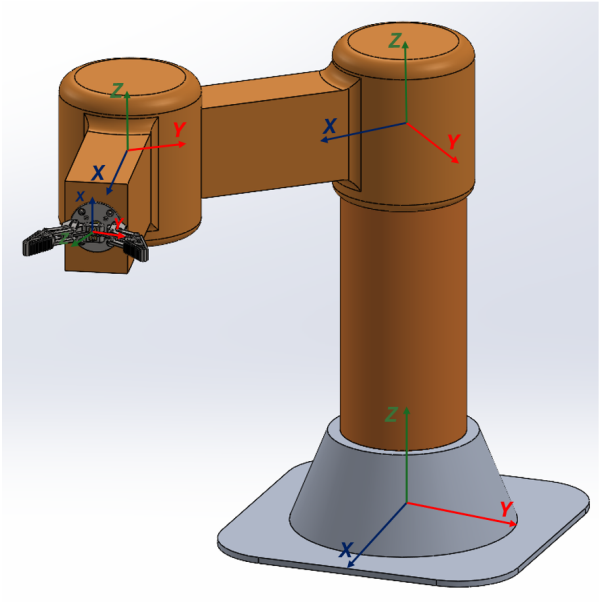


Figura 8.8: Sistemas de referencia

Link, i	θ_i	d_i	a_i	α_i
1	50	57.7	46.33	0
2	32	0	46.31	0

Tabla 8.1: Tabla D-H

directa.

$$
\begin{aligned}
{}^0A_1 &= \begin{bmatrix} cos(\theta_1) & -sen(\theta_1) & 0 & 0 \\ sen(\theta_1) & cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
{}^0A_1 &= \begin{bmatrix} cos(\theta_1) & -sen(\theta_1) & 0 & a_1 * C_1 \\ sen(\theta_1) & cos(\theta_1) & 0 & a_1 * S_1 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
{}^1A_2 &= \begin{bmatrix} cos(\theta_1) & -sen(\theta_1) & 0 & 0 \\ sen(\theta_1) & cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
{}^1A_2 &= \begin{bmatrix} cos(\theta_1) & -sen(\theta_1) & 0 & a_2 * C_2 \\ sen(\theta_1) & cos(\theta_1) & 0 & a_2 * S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
{}^0A_2 &= {}^0A_1 \times {}^1A_2 = \begin{bmatrix} C_{12} & -S_{12} & 0 & a_1 * C_1 + a_2 * C_{12} \\ S_{12} & C_{12} & 0 & a_1 * S_1 + a_2 * S_{12} \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
\tag{8.1}
$$

Teniendo en cuenta que C_1 es coseno de θ_1, S_1 es seno de θ_1, C_2 es coseno de θ_2, S_2 es seno de θ_2, C_{12} es coseno de $\theta_1 + \theta_2$ y S_{12} es seno de $\theta_1 + \theta_2$.
También se realizó un programa en Python para resolver este problema de forma numérica. Obteniendo como resultados:
La matriz de transformación homogénea:

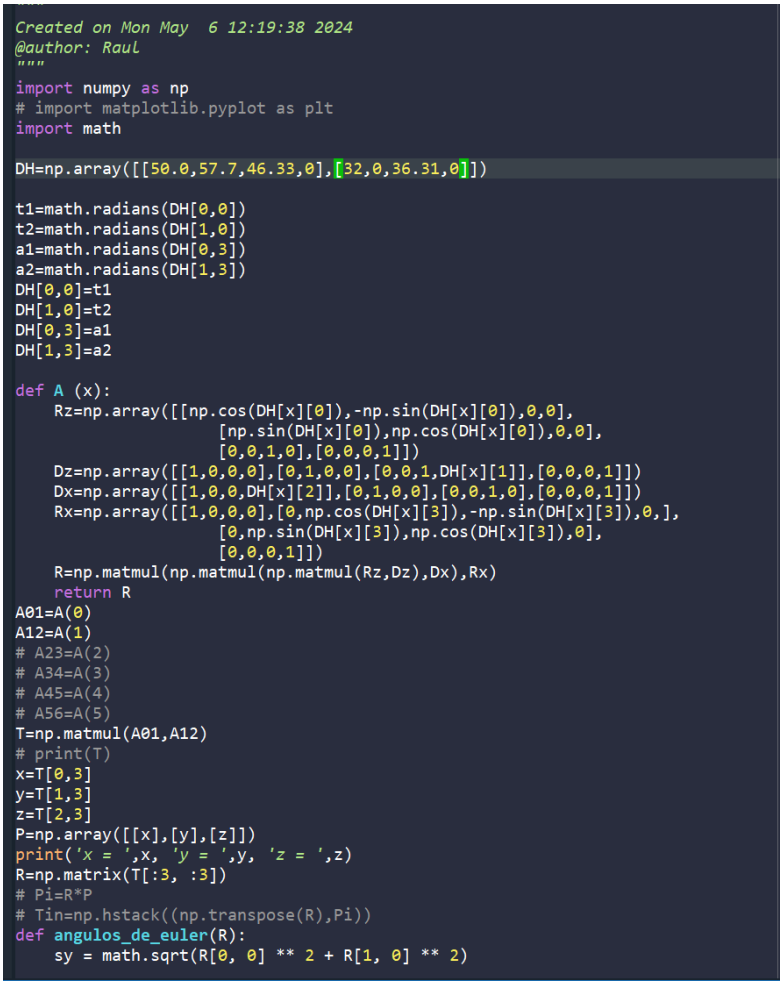
$$
\begin{bmatrix} 0.1391731 & -0.99026807 & 0 & 34.83372525 \\ 0.99026807 & 0.1391731 & 0 & 71.44747263 \\ 0 & 0 & 1 & 57.7 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

La posición del efector final:

- x = 34.83
- y = 71.44
- z = 57.7

y su orientación:

- yaw = 82.0
- pitch = 82.0



```
Created on Mon May  6 12:19:38 2024
@author: Raul
"""
import numpy as np
# import matplotlib.pyplot as plt
import math

DH=np.array([[50.0,57.7,46.33,0],[32,0,36.31,0]])

t1=math.radians(DH[0,0])
t2=math.radians(DH[1,0])
a1=math.radians(DH[0,3])
a2=math.radians(DH[1,3])
DH[0,0]=t1
DH[1,0]=t2
DH[0,3]=a1
DH[1,3]=a2

def A (x):
    Rz=np.array([[np.cos(DH[x][0]),-np.sin(DH[x][0]),0,0],
                 [np.sin(DH[x][0]),np.cos(DH[x][0]),0,0],
                 [0,0,1,0],[0,0,0,1]])
    Dz=np.array([[1,0,0,0],[0,1,0,0],[0,0,1,DH[x][1]],[0,0,0,1]])
    Dx=np.array([[1,0,0,DH[x][2]],[0,1,0,0],[0,0,1,0],[0,0,0,1]])
    Rx=np.array([[1,0,0,0],[0,np.cos(DH[x][3]),-np.sin(DH[x][3]),0,],
                 [0,np.sin(DH[x][3]),np.cos(DH[x][3]),0],
                 [0,0,0,1]])
    R=np.matmul(np.matmul(np.matmul(Rz,Dz),Dx),Rx)
    return R
A01=A(0)
A12=A(1)
# A23=A(2)
# A34=A(3)
# A45=A(4)
# A56=A(5)
T=np.matmul(A01,A12)
# print(T)
x=T[0,3]
y=T[1,3]
z=T[2,3]
P=np.array([[x],[y],[z]])
print('x = ',x, 'y = ',y, 'z = ',z)
R=np.matrix(T[:3, :3])
# Pi=R*P
# Tin=np.hstack((np.transpose(R),Pi))
def angulos_de_euler(R):
    sy = math.sqrt(R[0, 0] ** 2 + R[1, 0] ** 2)
```

Figura 8.9: Fragmento de programa en Phyton para cinemática de robot RR

- roll = 0.0

siendo ’yaw’ el componente de x, ’pitch’ el componente de y, ’roll’ el componente de z del efector final con respecto al sistema de la base.

8.5.2. Cinemática inversa

Para la resolución del problema cinemático inverso se empleó la ley de cosenos para realizar el análisis geométrico de los ángulos a los que se encuentran las articulaciones del robot, como se muestra en la figura 8.10. Esto se puede realizar debido a que el movimiento de las 2 articulaciones se puede describir en un solo plano (xy) y las relaciones de éstas se pueden hallar por medio de la descomposición del problema en varios triángulos.
Para la resolución de este problema se llevaron a cabo los siguientes pasos:

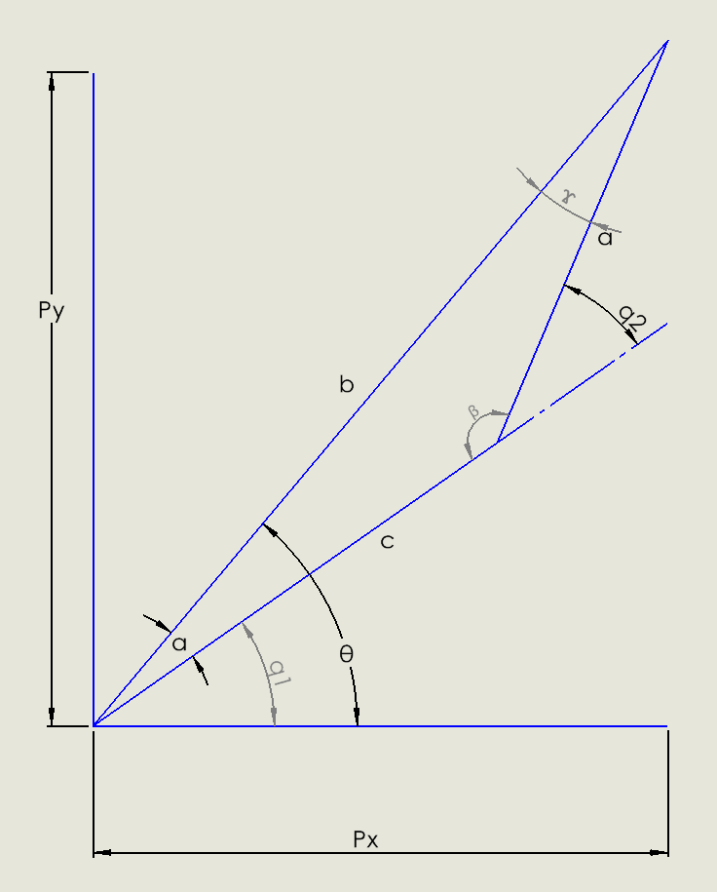


Figura 8.10: Aplicación de ley de coseno

1. Aplicar la ley de cosenos; para este problema solo se busca obtener α y β que serán los valores que se despejen de las siguientes ecuaciones.
   a^2 = b^2 + c^2 - $2bc * cos(\alpha)$ y b^2 = a^2 + c^2 - $2ac * cos(\beta)$
2. Se asignan los valores conocidos a las incógnitas de la ecuación para poder despejar la variable deseada, quedando: c = l_1 a = l_2 y b = $\sqrt{P_x^2 + P_y^2}$
3. Se despejan las incógnitas deseadas α y β, quedando de la siguiente forma:
   α = $cos^{-1}\left(\frac{b^2+C^2-a^2}{2bc}\right)$ β =$cos^{-1}\left(\frac{a^2+C^2-b^2}{2ac}\right)$
4. Se establecen las ecuaciones para identificar el valor de los ángulos que faltan por conocer para la resolución del problema.
   θ = $tan^{-1}\left(\frac{P_y}{P_x}\right)$, q_2 = 180 - β y q_1 = θ - α
5. Se sustituyen los valores conocidos de l_1, l_2, P_x y P_y y se resuelve el problema.

De este método se obtienen los valores q_1 = 50 y q_2 = 32.
En el programa que se muestra en la figura 8.9, cuenta con el algoritmo para la solución

de la cinemática directa por medio de Denavit-Hartenberg, así como la solución de la cinemática inversa por método geométrico.

Cinemática inversa por matriz de transformación homogénea

De acuerdo con la ecuación 4.9 es posible encontrar los valores de las articulaciones por medio de las relaciones geométricas que se generan. Para poder realizar este procedimiento se debe conocer que la matrices de transformación homogénea son un tipo especial debido a la ortonormalidad de la matriz de rotación, facilitando el proceso de hallar la matriz inversa, la cual se puede obtener de la siguiente forma:

$$T = \begin{bmatrix} R & -R^T P \\ 0 & 1 \end{bmatrix} \tag{8.2}$$

Tomando en cuenta lo anterior se procede a resolver el problema, primero se buscan las matrices ${}^{i-1}A_i$ para todas las articulaciones y la matriz de trasformación homogénea **T** de estas siendo:

$$\begin{gathered} {}^0A_1 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1c_1 \\ s_1 & c_1 & 0 & a_1s_1 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^1A_2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ {}^0A_2 = \begin{bmatrix} c_12 & -s_12 & 0 & a_1c_1 + a_2C_12 \\ s_12 & c_12 & 0 & a_1s_1 + a_2s_12 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{gathered} \tag{8.3}$$

Una vez obtenidas las matrices anteriores se deben obtener las matrices inversas de cada articulación (${}^{i-1}A_i$) obteniendo:

$${}^0A_1^{-1} = \begin{bmatrix} c_1 & s_1 & 0 & -a_1 \\ -s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & -d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^1A_2^{-1} = \begin{bmatrix} c_2 & s_2 & 0 & -a_2 \\ -s_2 & c_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.4}$$

Con los datos anteriores se puede construir las ecuaciones para poder encontrar los ángulos de las articulaciones, como se muestra a continuación:

$$\mathbf{T} = {}^{0}A_{1}^{1}A_{2}$$

$${}^{0}A_{1}^{-1}\mathbf{T} = {}^{1}A_{2}$$

$$= \begin{bmatrix} c_2 & s_2 & 0 & -a_2 \\ -s_2 & c_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_x & a_y & p_y \\ n_z & o_x & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_1n_x + s_1n_y & c_1o_x + s_1o_y & c_1a_x + s_1a_y & c_1p_x + s_1p_y - a_1 \\ -s_1n_x + c_1n_y & -s_1o_x + c_10_y & -s_1a_x + c_1a_y & -s_1p_y + c_1p_y \\ n_z & o_x & a_z & p_z - d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De estas igualdades se obtuvieron las siguientes ecuaciones:

$$\begin{aligned} a_2s_2 =& c_1p_y - s_1p_x \Rightarrow a_2(-o_xc_1 - o_ys_1) = c_1p_y - s_1p_xs_1(p_x - a_2o_y) \\ =& c_1(p_y + a_2o_x) \Rightarrow \tan(q_1) = \frac{p_y + a_2o_x}{p_x - a_2o_y} \\ q_1 =& \tan^{-1}\left(\frac{p_y + a_2o_x}{p_x - a_2o_y}\right) = 50^\circ \end{aligned}$$

Una vez obtenido un ángulo, este se emplea en la búsqueda del segundo grado siendo:

$$\begin{aligned} c_2 =& n_xc_1 + nys_1 \\ q_2 =& c^{-1}\left(n_xc(50) + n_ys(50)\right) = 32^\circ \end{aligned}$$

Cabe recalcar que los valores de la matriz **T** son conocidos y provienen de la relación de **T** = ${}^{0}T_2$. De esta forma se pueden conocer los ángulos de las articulaciones por medio de matrices de transformación homogénea.

8.5.3. Modelo dinámico

En este apartado se abordará la obtención del modelo dinámico del robot RR por medio de la formulación de Lagrange. Considerando el modelo del robot como se muestra en la figura 8.11 se obtienen las siguientes ecuaciones:
Coordenadas y velocidades de los centros de masa:
Masa del eslabón 1:

$$\begin{cases} x_1 = & d_1C_1 \\ y_1 = & d_1S_1 \end{cases} \Rightarrow \begin{cases} \dot{x_1} = & -d_1S_1\dot{q_1} \\ \dot{y_1} = & d_1C_2\dot{q_1} \end{cases} \Rightarrow v_1^2 = \dot{x_1}^2 + \dot{y_1}^2 = d_1^2\dot{q_1}^2$$

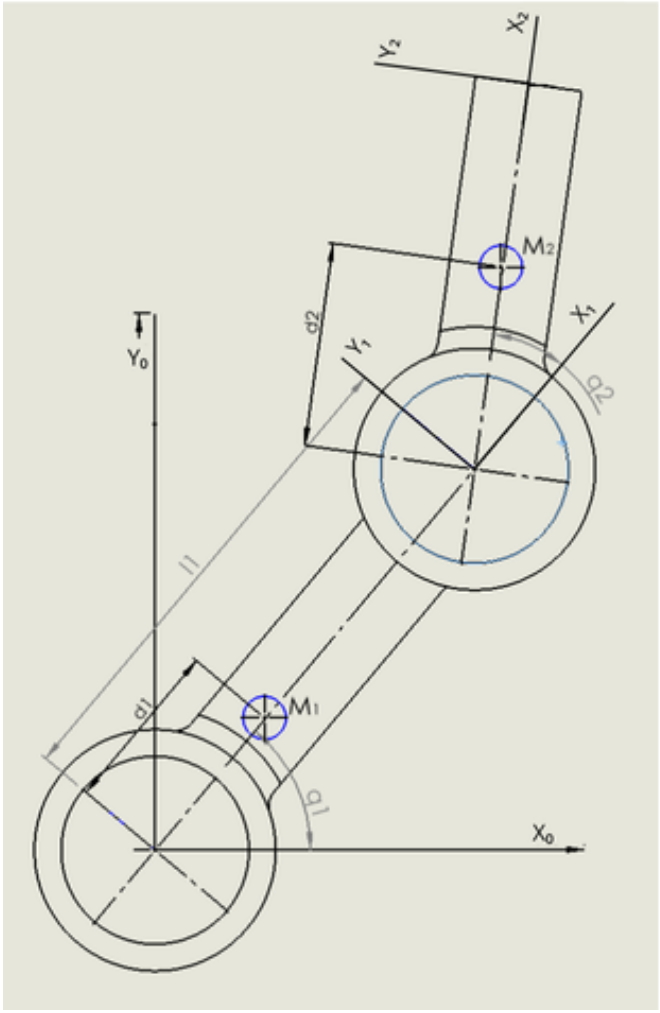


Figura 8.11: Robot de 2 grados de libertad

Masa del eslabón 2:

$$\left\{\begin{array}{rl} x_2 = & l_1C_1 + d_2C_12 \\ y_2 = & l_1S_1 + d_2S_12 \end{array}\right. \Rightarrow \left\{\begin{array}{rl} \dot{x_2} = & -(l_1S_1 + d_2S_{12})\dot{q_1} - d_2S_{12}\dot{q_2} \\ \dot{y_2} = & (l_1C_1 + d_2C_{12})\dot{q_1} + d_2C_{12}\dot{q_2} \end{array}\right. \Rightarrow$$

$$v_2^2 = \dot{x_2}^2 + \dot{y_2}^2 = (l_1^2 + d_2^2 + 2l_1d_2C_2)\dot{q_1}^2 + d_2^2\dot{q_2}^2 + 2d_2(l_1C_2 + d_2)\dot{q_1}\dot{q_2}$$

Energía Cinética:

$$E_C = \frac{1}{2}(m_1v_1^2 + m_2v_2^2) = \frac{1}{2}[m_1d_1^2 + m_2(l_1^2 + m_2(l_1^2 + d_2^2 + 2l_1d_2C_2))]\dot{q_1}^2 + \frac{1}{2}(m_2d_2^2)\dot{d_2}^2 + \frac{1}{2}[m_2 2d_2(l_1C_2 + d_2)]\dot{q_1}\dot{q_2}$$

Energía Potencial:

$$E_p = g(m_1h_1 + m_2h_2) = g(m_1y_1 + m_2y_2) = g(m_1d_1S_1 + m_2d_2S_{12})$$

Lagrangiano:

$$L = E_c - E_p = \frac{1}{2}[m_1 d_1^2 + m_2(l_1^2 + m_2(l_1^2 + d_2^2 + 2l_1 d_2 C_2))]\dot{q_1}^2$$
$$+ \frac{1}{2}(m_2 d_2^2)\dot{d_2}^2 + \frac{1}{2}[m_2 2 d_2(l_1 C_2 + d_2)]\dot{q_1}\dot{q_2} - g(m_1 d_1 S_1 + m_2 d_2 S_{12})$$

Derivadas con respecto de $\dot{q_i}$ y con respecto al tiempo:

$$\frac{\partial L}{\partial \dot{q_1}} = [m_1 d_1^2 + m_2(l_1^2 + d_2^2 + 2l_1 d_2 C_2)]\dot{q_1} + [m_2 d_2(l_1 C_2 + d_2)]\dot{q_2}$$
$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q_1}} = [m_1 d_1^2 + m_2(l_1^2 + d_2^2 + 2l_1 d_2 C_2)]\ddot{q_1} + [-m_2(2l_1 d_2 S_2 \dot{q_2})]\dot{q_1} + [m_2 d_2(l_1 C_2 + d_2)]\ddot{q_2}$$
$$+ [-m_2 d_2(l_1 S_2 \dot{q_2})]\dot{q_2}$$
$$= [m_1 d_1^2 + m_2(l_1^2 + d_2^2 + 2l_1 d_2 C_2)]\ddot{q_1} + [m_2 d_2(l1 C_2 + d_2)]\ddot{q_2} - (2m_2 l_1 d_2 S_2)\dot{q_1}\dot{q_2} - (m_2 d_2 l_1 S_2)\dot{q_2}^2$$
$$\frac{\partial L}{\partial \dot{q_2}} = (m_2 d_2^2)\dot{q_2} + [m_2 d_2(l1 C_2 + d_2)]\dot{q_1}$$
$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q_2}} = (m_2 d_2^2)\ddot{q_2} + [m_2 d_2(l1 C_2 + d_2)]\ddot{q_1} + [-m_2 d_2(l_1 S_2 \dot{q_2})]\dot{q_1}$$
$$= (m_2 d_2^2)\ddot{q_2} + [m_2 d_2(l1 C_2 + d_2)]\ddot{q_1} - (m_2 d_2 l_1 S_2)\dot{q_1}\dot{q_2}$$

Derivadas respecto a q_i:

$$\frac{\partial L}{\partial q_1} = -g[(m_1 d_1 + m_2 l_1)C_1 + m_2 d_2 C_{12}]$$
$$\frac{\partial L}{\partial q_2} = -(m_2 l_1 d_2 S_2)\dot{q_1}^2 - [m_2 d_2 l_1 S_2]\dot{q_1}\dot{q_2} - g m_2 d_2 C_{12}$$

Sustituyendo valores en la ecuación (5.3):

$$\tau_i = \frac{d}{dt}\frac{\partial L}{\partial \dot{q_i}} - \frac{\partial L}{\partial q_1} \Rightarrow$$
$$\begin{cases} \tau_1 = [m_1 d_1^2 + m_2(l_1^2 + d_2^2 + 2l_1 d_2 C_2)]\ddot{q_1} + [m_2 d_2(l1 C_2 + d_2)]\ddot{q_2} - (2m_2 l_1 d_2 S_2)\dot{q_1}\dot{q_2} \\ \qquad -(m_2 d_2 l_1 S_2)\dot{q_2}^2 + g[(m_1 d_1 + m_2 l_1)C_1 + m_2 d_2 C_{12}] \\ \tau_2 = [m_2 d_2(l1 C_2 + d_2)]\ddot{q_1} + (m_2 d_2^2)\ddot{q_2} - (m_2 d_2 l_1 S_2)\dot{q_1}\dot{q_2} + (m_2 l_1 d_2 S_2)\dot{q_1}^2 \\ \qquad +(m_2 d_2 l_1 S_2)\dot{q_1}\dot{q_2} + g m_2 d_2 C_{12} \end{cases}$$

Éstas expresiones pueden escribirse en forma matricial como:

$$\begin{bmatrix}\tau_1 \\ \tau_2\end{bmatrix} = \begin{bmatrix} m_1 d_1^2 + m_2(l_1^2 + d_2^2 + 2l_1 d_2 C_2) & m_2 d_2(l_1 C_2 + d_2) \\ m_2 d_2(l_1 C_2 + d_2) & m_2 d_2^2 \end{bmatrix}\begin{bmatrix}\ddot{q_1} \\ \ddot{q_2}\end{bmatrix}$$
$$+ \begin{bmatrix} -2m_2 l_1 d_2 S_2 \dot{q_2} & -m_2 d_2 l_1 S_2 \dot{q_2} \\ m_2 d_2 l_1 S_2 \dot{q_1} & 0 \end{bmatrix}\begin{bmatrix}\dot{q_1} \\ \dot{q_2}\end{bmatrix} \qquad (8.5)$$
$$+ \begin{bmatrix} g[(m_1 d_1 + m_2 l_1)C_1 + m_2 d_2 C_{12}] \\ g m_2 d_2 C_{12} \end{bmatrix}$$

De esta manera se puede obtener el modelo dinámico del robot RR con la configuración que se muestra en la Figura 8.11.

8.6. Planificación de trayectorias

La planificación de trayectorias describe el camino o ruta que el robot debe seguir para poder realizar la tarea encomendada. En esta tesis se tiene planeado que el robot de 2 grados de libertad interrumpa la trayectoria planeada del cobot con el fin de que este último tenga que evitar colisionar con el robot RR se planteó una serie de puntos como se muestra en la figura 8.12 en donde los puntos rojos describen los puntos vía del robot RR y los puntos azules son los puntos vía del cobot, siendo que los robots tienen la posibilidad de colisionar al rededor del punto (80,31,0) como se muestra en la figura 8.13.

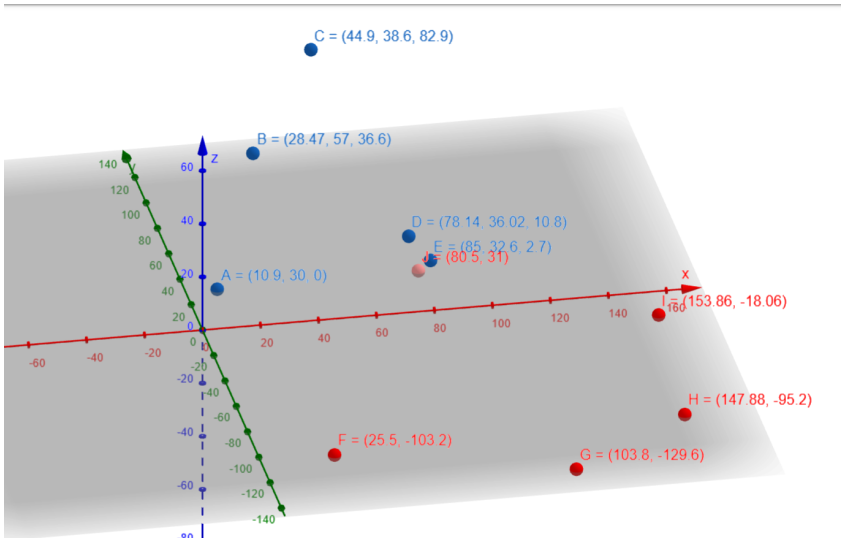


Figura 8.12: Puntos via de ambos robots

Para asegurarse que los puntos vía que se seleccionaron estén dentro del rango de operación para cada uno de ellos, se planeó el movimiento en la plataforma digital diseñada en SolidWorks, donde se comprobó que no existe ninguna singularidad en los puntos seleccionados, siendo este uno de los principales problemas que se presentan al momento de definir una trayectoria en el espacio cartesiano.

Una vez que se establecieron los puntos vía, se requiere emplear la cinemática inversa de cada robot para obtener las posiciones angulares de cada articulación en estos puntos. Además se realizó una interpolación cúbica para generar una trayectoria suave entre los puntos, garantizando continuidad en la posición y la velocidad, lo que resulta en un movimiento más natural y menos exigente para la mecánica del robot. Obteniendo resultados como los que se presentan en las figuras 8.14 y 8.15.

Como se puede ver en la figura 8.15, la gráfica obtenida de las posiciones cartesianas

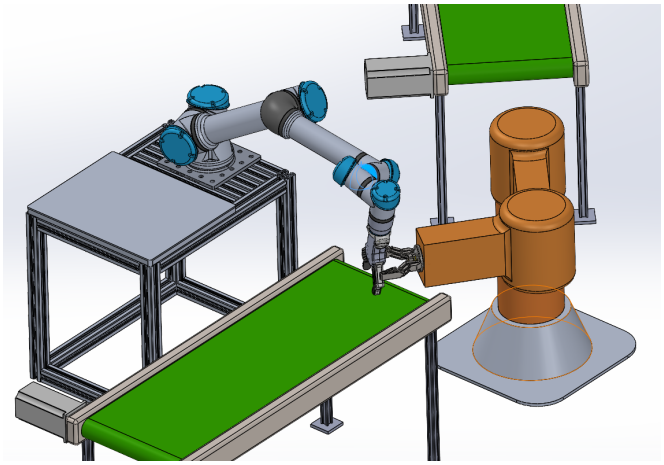


Figura 8.13: Punto de colisión

con respecto de x, y dibuja una trayectoria similar a un medio círculo, movimiento que muchas veces se emplea para este tipo de robots, debido al amplio conocimiento de las funciones que describen este tipo de movimiento.

Para realizar la interpolación cúbica de las posiciones angulares, es necesario complementarlas con las velocidades articulares en cada punto vía. Las velocidades articulares se obtienen a partir de la relación que tienen las velocidades cartesianas deseadas con la inversa del Jacobiano, como se expresa en la ecuación 8.6. En esta tesis, aunque el robot RR y el cobot UR5 poseen Jacobianos cuadrados y, por lo tanto, un Jacobiano inverso definido, en la implementación se optó por el uso de la pseudo-inversa. Esta decisión garantiza estabilidad numérica ante configuraciones cercanas a singularidades. Para un cobot de 6 grados como el UR5, se obtiene un Jacobiano y su inversa como se muestra a continuación:

$$\text{Jacobiano} = \begin{bmatrix} -7.619 \times 10^{-3} & 3.534 \times 10^{-17} & -2.120 \times 10^{-19} & -9.003 \times 10^{-18} & -8.230 \times 10^{-2} & -2.082 \times 10^{-17} \\ 1.092 \times 10^{-1} & 5.847 \times 10^{-1} & 2.167 \times 10^{-1} & -1.230 \times 10^{-1} & 2.758 \times 10^{-18} & -3.442 \times 10^{-18} \\ 0 & 7.619 \times 10^{-3} & 2.201 \times 10^{-1} & 2.399 \times 10^{-2} & -5.378 \times 10^{-19} & 5.962 \times 10^{-18} \\ 0 & 1.000 & 1.000 & 1.000 & -2.241 \times 10^{-17} & 8.364 \times 10^{-17} \\ 0 & -6.123 \times 10^{-17} & -6.123 \times 10^{-17} & -6.123 \times 10^{-17} & -5.000 \times 10^{-1} & 8.660 \times 10^{-1} \\ 1.000 & 6.123 \times 10^{-17} & 6.123 \times 10^{-17} & 6.123 \times 10^{-17} & 8.660 \times 10^{-1} & 5.000 \times 10^{-1} \end{bmatrix}$$

$$\text{Jacobiano inverso} = \begin{bmatrix} 1.571 \times 10^{1} & -4.243 \times 10^{-15} & 2.289 \times 10^{-15} & 1.928 \times 10^{-15} & -6.465 \times 10^{-1} & 1.120 \times 10^{0} \\ -2.329 \times 10^{0} & 1.358 \times 10^{0} & -2.353 \times 10^{0} & 2.236 \times 10^{-1} & 9.585 \times 10^{-2} & -1.660 \times 10^{-1} \\ -1.945 \times 10^{-1} & 1.134 \times 10^{-1} & 4.902 \times 10^{0} & -1.037 \times 10^{-1} & 8.003 \times 10^{-3} & -1.386 \times 10^{-2} \\ 2.524 \times 10^{0} & -1.472 \times 10^{0} & -2.549 \times 10^{0} & 8.801 \times 10^{-1} & -1.039 \times 10^{-1} & 1.799 \times 10^{-1} \\ -1.360 \times 10^{1} & 3.829 \times 10^{-15} & -2.280 \times 10^{-15} & -1.679 \times 10^{-15} & 5.985 \times 10^{-2} & -1.037 \times 10^{-1} \\ -7.855 \times 10^{0} & 1.759 \times 10^{-15} & 1.279 \times 10^{-15} & -1.043 \times 10^{-15} & 1.189 \times 10^{0} & -5.985 \times 10^{-2} \end{bmatrix}$$

$$\text{Comprobación} = \begin{bmatrix} 1.000 & 1.280 \times 10^{-16} & 1.618 \times 10^{-15} & 2.472 \times 10^{-15} & -9.992 \times 10^{-16} & -1.110 \times 10^{-16} \\ -2.044 \times 10^{-15} & 1.000 & -3.886 \times 10^{-15} & 3.955 \times 10^{-16} & 1.638 \times 10^{-15} & 5.690 \times 10^{-16} \\ 2.166 \times 10^{-16} & 2.033 \times 10^{-15} & 1.000 & 2.193 \times 10^{-15} & -4.007 \times 10^{-16} & 1.396 \times 10^{-16} \\ 2.262 \times 10^{-15} & -2.717 \times 10^{-15} & 2.109 \times 10^{-15} & 1.000 & -1.610 \times 10^{-15} & -8.049 \times 10^{-16} \\ -3.400 \times 10^{-15} & 5.183 \times 10^{-17} & -1.358 \times 10^{-15} & -2.092 \times 10^{-15} & 1.000 & -1.886 \times 10^{-16} \\ -1.860 \times 10^{-15} & -3.589 \times 10^{-16} & -4.552 \times 10^{-16} & -1.235 \times 10^{-15} & 1.110 \times 10^{-16} & 1.000 \end{bmatrix}$$

Se presentó el Jacobiano y su inversa de forma numérica con fines ilustrativos.
Para obtener las velocidades angulares se requiere satisfacer la siguiente ecuación:

$$\dot{q} = J^{-1} v \tag{8.6}$$

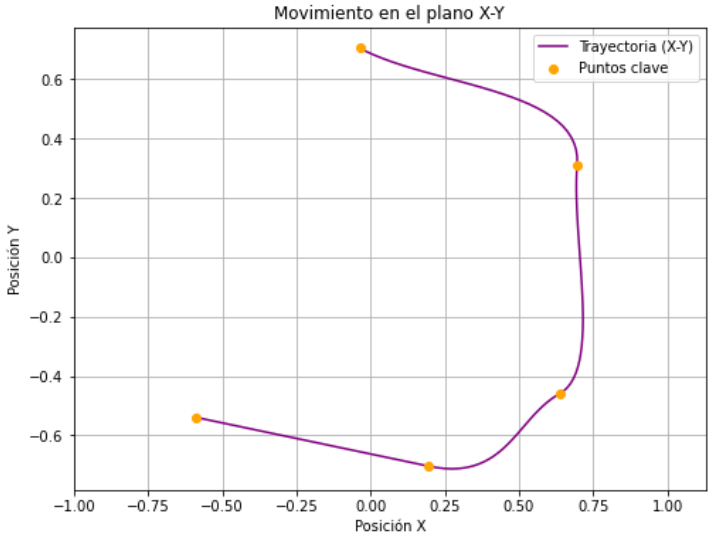
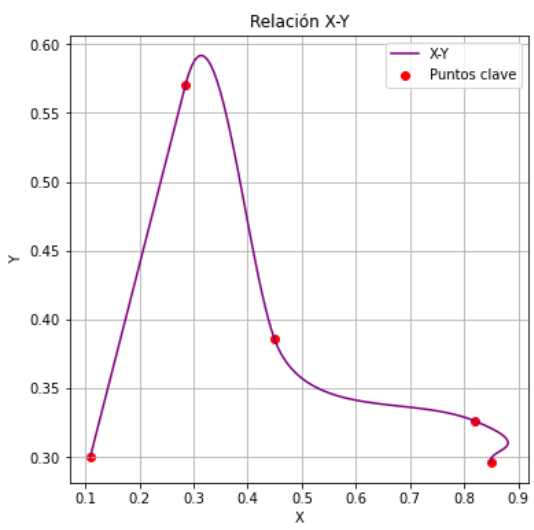
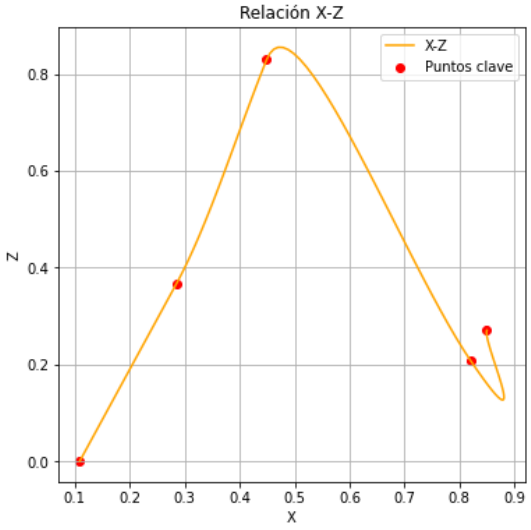


Figura 8.14: Trayectoria del Robot RR

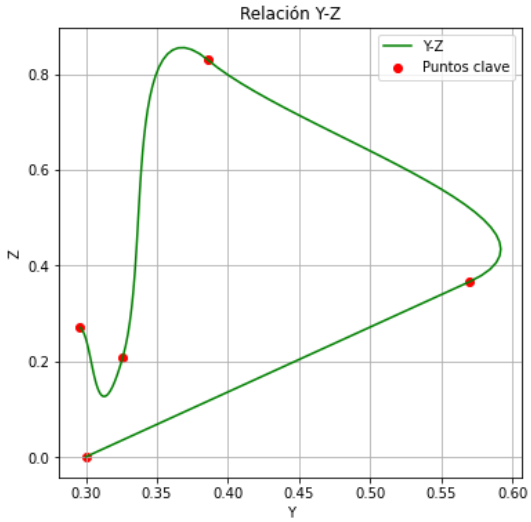
Donde J^{-1} es la matriz inversa que se obtuvo anteriormente por las velocidades deseadas que se pueden obtener de la interpolación cúbica que se realizó para las trayectorias cartesianas.
Una vez obtenidos todos los datos antes mencionados se puede realizar un algoritmo que nos permita controlar el movimiento del robot.



(a) Vista X-Y de trayectoria



(b) Vista X-Z de trayectoria



(c) Vista Y-Z de trayectoria

Figura 8.15: Trayectoria robot UR5

8.7. Control PID para seguimiento de trayectorias

Para realizar un control de seguimiento de trayectorias para el robot RR, se partió de un control de posición PID, el cual consiste en llevar cada uno de los eslabones que componen la cadena cinemática del robot de un valor o posición inicial a una posición deseada, procurando minimizar los errores de posición y velocidad y evitando el sobretiro, como se muestra en la figura 8.16 en donde ambos eslabones parten del reposo (0,pi/4) y se busca llevar el primer eslabón a pi/2 y el segundo a pi radianes.
Una vez obtenidos los resultados deseados para el control de posición, se cambió el con-

trol de posición por uno de seguimiento de trayectorias. Considerando la trayectoria que se generó para el robot en la sección anterior, se aplica una interpolación cúbica. Los resultados obtenidos de esta implementación se muestran en la figura 8.17. A este control se le aplicó una métrica de desempeño conocida como Integral del Error Absoluto (IAE) por su siglas en inglés, con valores de 49.7846° para la primera articulación y 58.6638° para la segunda.

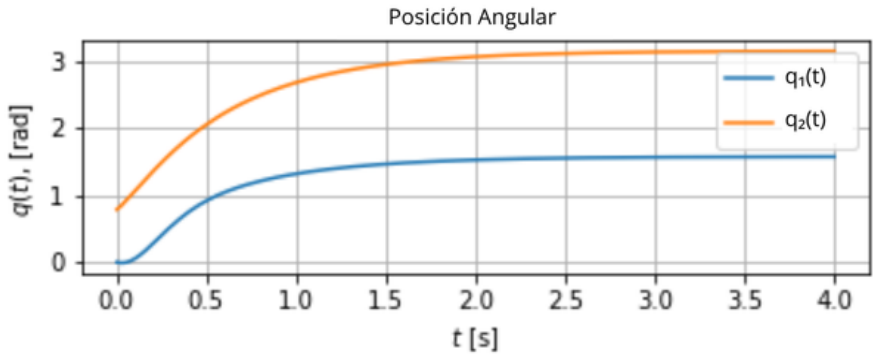


Figura 8.16: Resultados control posición

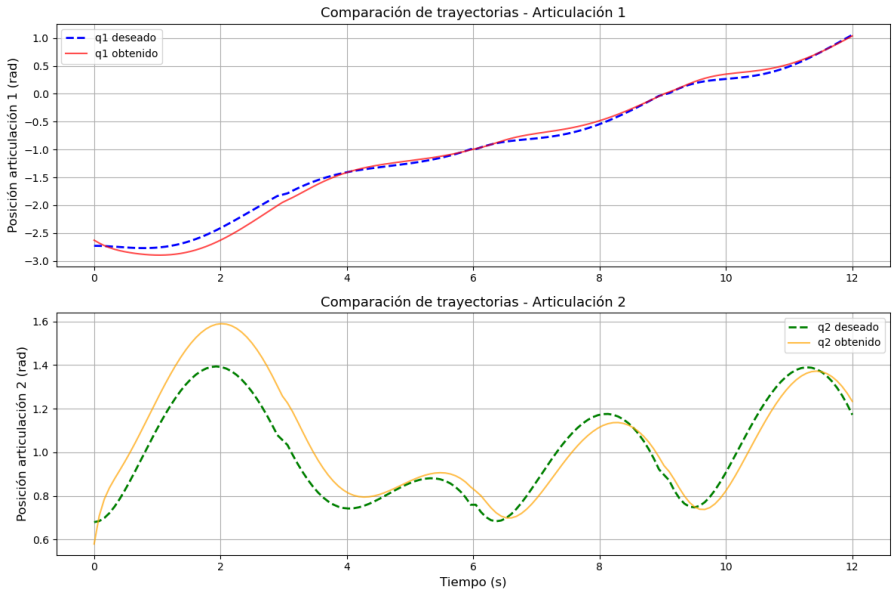


Figura 8.17: Control de Trayectorias

Para establecer el modelo dinámico del robot UR5, se partió de la tabla de parámetros de Denavit-Hartenberg, como se muestra en la figura 8.18, la cinemática directa e inversa, y el Jacobiano geométrico. La matriz de inercia se determinó por medio de un método iterativo basado en [15] :

i	a_i	α_i	d_i	θ_i
0	0	0	-	-
1	0	$\frac{\pi}{2}$	0.08916	θ_1
2	0.425	0	0	θ_2
3	0.39225	0	0	θ_3
4	0	$\frac{\pi}{2}$	0.10915	θ_4
5	0	$-\frac{\pi}{2}$	0.09456	θ_5
6	-	-	0.0823	θ_6

Figura 8.18: Tabla de parámetros de Denavit-Hartenberg para el cobot UR5

$$M(q) = \sum_{i=1}^{n} (m_i J_{v_i}^T J_{v_i} + J_{\omega_i}^T R_i I_i R_i^T J_{\omega_i}) \tag{8.7}$$

Se verificó que esta cumple con la propiedad de simetría y es definida positiva. De igual forma se emplea la fórmula propuesta en [32] para la obtención de la matriz de Coriolis:

$$[C(q,\dot{q})]_{ij} = \sum \Gamma_{ijk}(\mathbf{q})\dot{q}_k \tag{8.8}$$

$$\text{donde} \quad \Gamma_{ijk}(\mathbf{q}) = \frac{1}{2}\left(\frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ij}}{\partial q_j} - \frac{\partial M_{ij}}{\partial q_i}\right)\dot{q}_k \tag{8.9}$$

De esta forma se ha obtenido la matriz de inercia y la matriz de Coriolis como se muestra en la figura 8.19 la cual no se puede visualizar de otra manera en esta tesis debido a la extensión de cada uno de los términos.

```
"""
M(q) = Sum{i}{n}(miJvi^TJvi +Jomi^TRiIiRi^TJomi)
"""
M1 = m1*Jv1.transpose()*Jv1 +Jom1.transpose()*R1*I1*(R1.transpose())*Jom1
M2 = m2*Jv2.transpose()*Jv2 +Jom2.transpose()*R2*I2*(R2.transpose())*Jom2
M3 = m3*Jv3.transpose()*Jv3 +Jom3.transpose()*R3*I3*(R3.transpose())*Jom3
M4 = m4*Jv4.transpose()*Jv4 +Jom4.transpose()*R4*I4*(R4.transpose())*Jom4
M5 = m5*Jv5.transpose()*Jv5 +Jom5.transpose()*R5*I5*(R5.transpose())*Jom5
M6 = m6*Jv6.transpose()*Jv6 +Jom6.transpose()*R6*I6*(R6.transpose())*Jom6
M = M1+M2+M3+M4+M5+M6
```

Figura 8.19: Matriz de inercia de UR5

También se obtuvo la matriz de pares gravitacionales. Obteniendo asi el modelo dinámico en forma simbólica del robot UR5 mediante programación en Python.

8.8. Control MPC

El control PID permite un seguimiento de trayectorias adecuado en ambos robots en condiciones normales. Sin embargo, en el escenario propuesto donde el robot RR debe interrumpir la trayectoria del UR5, se evidencia una limitación del control PID: su naturaleza reactiva que le impide responder ante obstáculos no considerados en la planificación inicial. La naturaleza reactiva del control PID, el cual actúa únicamente sobre el error presente limitando su capacidad de evasión. Para superar esta limitación, se propone la implementación de Control Predictivo Basado en Modelos (MPC) en el UR5, permitiendo la evasión proactiva de obstáculos mediante la optimización en línea de trayectorias futuras, manteniendo al mismo tiempo el desempeño en el seguimiento de la trayectoria deseada.

Para comprender mejor el funcionamiento del MPC se implementó el MPC para el control de seguimiento de trayectorias en un sistema masa-resorte-amortiguador con modelo:

$$\dot{x}_1 = x_2 \tag{8.10}$$

$$\dot{x}_2 = (u - cx_2 - kx_1)/m \tag{8.11}$$

Se busca minimizar la función de costos, la cual para este caso tiene la forma:

$$J = \sum_{i=0}^{N-1} [(x_{i,1} - x_{ref,i})^2 + \lambda u_i^2] \tag{8.12}$$

siendo $x_{i,1}$ la posición predicha del sistema en el paso i , $x_{ref,i}$ la posición deseada en el paso i y $\lambda\ u_i^2$ penalización de control, como referencia deseada para este modelo se le ingreso una función senoidal.

Para la minimización de la función de costos se empleó la biblioteca scipy.optimize de Python que nos permite emplear la función minimize, la cual cuenta con distintos algoritmos de minimización. Se eligió el método L-BFGS-B (Limited-memory Broyden-Fletcher-Goldfarb-Shannon with Box constraints), el cual es un algoritmo de optimización numérica (que pertenece a la familia de métodos cuasi-Newtonianos), diseñado para suma de errores cuadráticos, como los que se presentan en el MPC. Una vez aplicado este control obtenemos una gráfica como la de la figura 8.20, como referencia deseada para este modelo se le ingreso una función senoidal.

Siguiendo este mismo principio se trabaja un control NMPC para el robot RR, para poder demostrar la ventaja que ofrece este método de control con respecto del PID. Mientras que para el control del UR5 se tiene planeado proporcionar una información de la posición del robot RR antes de generar el control dinámico en el esquema de planificación de trayectorias, quedando un diagrama como el de la figura 8.21

Se tiene contemplado que el robot UR5 conozca a priori la posición del robot RR, permitiendo así poder evitar su presencia en la trayectoria previamente establecida para el UR5.

Debido a la no linealidad de los robots manipuladores y la función de costo, así como

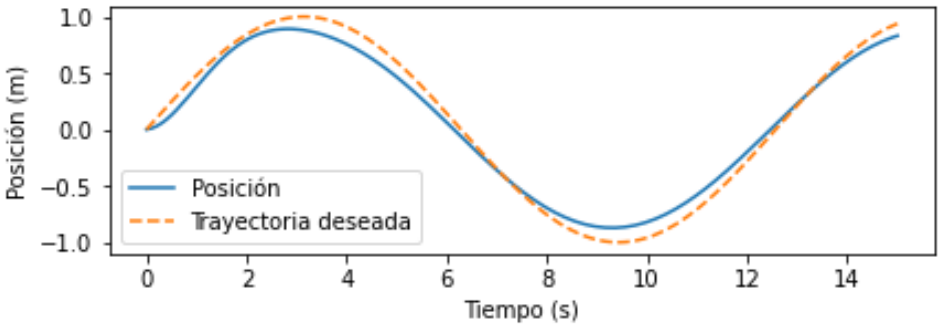


Figura 8.20: control de trayectoria por MPC

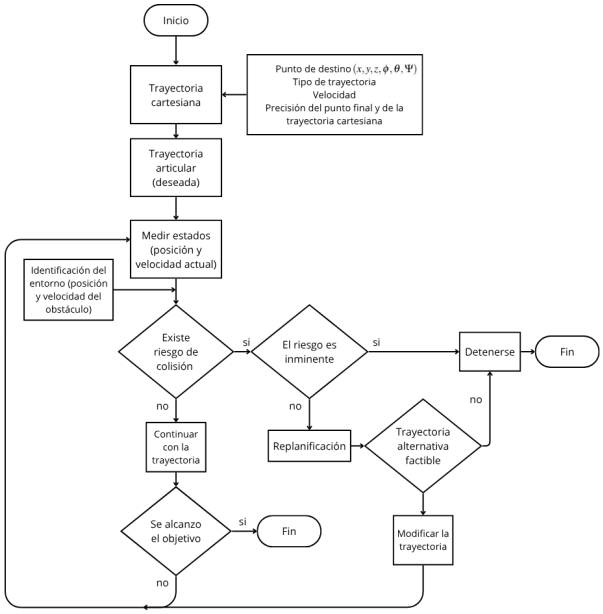


Figura 8.21: Esquema para evasión de Obstáculos

el optimizador están planteados para trabajar con sistemas no lineales, el control empleado es un control predictivo basado en modelos no lineales (NMPC).
Siguiendo la línea de acción que se planteó al inicio de esta tesis de ir escalando la complejidad del sistema hasta llegar al objetivo deseado de controlar el robot UR5 por medio de un control NMPC para la evasión de obstáculos, se generó un control NMPC para el seguimiento de trayectoria del robot RR.
Con base en la ecuación 8.5 se obtiene el modelo del robot en su representación en espacio de estados de la siguiente forma:

$$\begin{aligned} \dot{q} &= \quad \dot{q} \\ \ddot{q} &= (M(q))^{-1}[\tau - (C(q,\dot{q}) + b)\dot{q} - G(q)] \end{aligned} \tag{8.13}$$

Siendo $\dot{q}$ y $\ddot{q}$ las velocidades y aceleraciones del sistema, respectivamente, τ son los torques, M(q) es la matriz de inercia, C(q,$\dot{q}$) es la matriz de Coriolis, G(q) es el vector de

gravedad y b la fricción viscosa.
Una vez obtenido el modelo matemático se requiere muestrearlo para poder predecir a intervalos de tiempo futuro a partir del estado actual del sistema. Para esto se utiliza el método de Runge-Kutta de orden 4 como en [39], debido a su precisión y facilidad de implementación numérica, y porque nos permite aproximar la solución de ecuaciones diferenciales ordinarias (EDOs) en un intervalo de tiempo Δt para realizar la discretización del sistema sin que éste pierda su carácter no lineal. Se establecen el horizonte de predicción y el horizonte de control. El primero consiste en la cantidad de pasos hacia el futuro en donde se revisa el error de posición, y el segundo es el número de pasos en que se optimizará la entrada de control. Tomando en cuenta que el horizonte de control siempre debe de ser de igual o menor tamaño que nuestro horizonte predictivo, este paso es sumamente importante, ya que dependiendo de qué tan grande sea el horizonte de predicción será la precisión del control. Si el horizonte de predicción es muy pequeño, la precisión del control disminuye, pero si el horizonte es muy grande, el costo computacional se eleva.
Se establecen restricciones del desempeño del sistema a la vez que se mantiene el objetivo de control deseado. Con este propósito se limitó el torque máximo y mínimo que se emplea a lo largo del proceso de optimización. Para llevar a cabo dicho proceso se empleó el método Sequential Least Squares Programming SLSQP, el cual nos permite integrar restricciones no lineales. Se busca una función de costo adecuada que permita minimizar el error futuro y obtener una respuesta que se empleará como entrada de retroalimentación para el modelo matemático previamente seleccionado. De [40] se seleccionó la siguiente ecuación 8.14 como base para generar una función de costo que nos permita realizar el proceso de optimización.

$$J_N = \sum_{i=0}^{N-1} ||x_{k+i} - x_{ref}||_Q^2 + \sum_{i=0}^{M-1} ||U_{k+i}||_R^2 + ||x_{k+N} - x_{ref}||_P^2 \tag{8.14}$$

Siendo el primer término el error de posición futuro, el segundo termino refiriéndose al control futuro y el tercer termino el costo terminal, J_N la función de costo, x_{k+i} es la posición futura, x_{ref} es la posición deseada, N es el horizonte de predicción, M es el horizonte de control, U_{k+i} es el paso de control futuro y Q, R y P son las matrices de ponderación. Para obtener los valores de las matrices Q y R, se aplica la regla de Bryson 8.15, P se obtiene resolviendo la ecuación de Riccati 8.16. Como punto de partida se empleó la función de costo sin el ultimo término, pero debido a que el desempeño no era el esperado, se tuvo que agregar este término.

$$Q = diag(\frac{1}{\Delta x_i^2}) \quad \text{y} \quad R = diag(\frac{1}{\Delta u_j^2}) \tag{8.15}$$

$$P = A^T PA - A^T PB(R + B^T PB)^{-1} B^T PA + Q \quad \text{con} \quad A = \left.\frac{\partial f}{\partial x}\right|_{(x^*,u^*)}, \; B = \left.\frac{\partial f}{\partial u}\right|_{(x^*,u^*)} \tag{8.16}$$

Para mejorar la eficiencia del proceso de optimización, se implementó la función de pérdida de Huber. Esta función penaliza los errores grandes de manera lineal, evitando

así penalizaciones excesivas, mientras que los errores pequeños se penalizan de forma cuadrática. La expresión matemática de la función es la siguiente:

$$\rho(e_{pre}) = \begin{cases} \frac{1}{2}e_{pre}^2, & \text{para } |e_{pre}| \leq k, \\ k|e_{pre}| - \frac{1}{2}k^2, & \text{para } |e_{pre}| > k. \end{cases} \tag{8.17}$$

Al integrar la función de perdida de Huber, la función de costo queda de la forma:

$$J_N = \sum_{i=0}^{N-1} \rho Q(x_{k+i} - x_{ref}) + \sum_{i=0}^{M-1} ||U_{k+i}||_R^2 + ||x_{k+N} - x_{ref}||_P^2 \tag{8.18}$$

$$\text{donde} \quad \rho Q(e) = \sum_{j=1}^{n} \rho Q_{jj}, k(e_j) \quad \rho Q_{jj}, k(e_j) = \begin{cases} \frac{1}{2}Q_{jj}e_j^2, & \text{para } |e_j| \leq k \\ Q_{jj}\left(k|e_j| - \frac{1}{2}k^2\right), & \text{para } |e_j| > k \end{cases} \tag{8.19}$$

Habiendo definido los eslabones principales que conforman el NMPC, se establecen las condiciones iniciales de la posición y la velocidad de cada sistema como 0, sin pérdida de generalidad; es decir, se supone a los sistemas partiendo del reposo. Se integran todos los componentes, modelo, método de discretización, horizonte de predicción, función de costo y los valores iniciales en código de programación para generar el componente predictivo y obtener una secuencia de control óptima.
Se aplica la primera entrada de la secuencia de control obtenida por el proceso de optimización como entrada de alimentación para el modelo y se repite el proceso de medición del estado actual del sistema, integración y optimización para cada instante de tiempo (horizonte desplazado).
Una vez aplicada la metodología de control, es necesario sintonizar las ganancias de las matrices Q, R y P de forma heurística para mejorar el desempeño del sistema de control, obteniendo los resultados siguientes.
Se observa cómo el seguimiento de trayectoria de cada uno de los eslabones es casi idéntico, obteniendo un indice IAE para la articulación uno de 1.5409° y de 0.4392° para la segunda articulación. Teniendo un desempeño mejor con respecto al control PID de 32 veces para la articulación 1 y más de 100 veces para la articulación 2.
De igual manera, se presentan los errores de posición y torques aplicados a cada una de las articulaciones a lo largo del tiempo, en donde se puede observar que los errores de posición se mantienen en casi en 0 para la segunda articulación y valores pequeños con variaciones más considerables en los instantes de tiempo al momento de hacer un cambio de referencia. Mientras que los torques se mantienen controlados en un margen menor a los 5 Nm.
Una vez obtenido un adecuado control para el seguimiento de trayectorias del robot RR, se procedió a trabajar con la evasión de obstáculos para el mismo. Para poder incluir el obstáculo dentro del control predictivo es necesario modelar el obstáculo para poder predecir su comportamiento. Para ello se definió una trayectoria con velocidad constante y puntos de inicio y final, con repeticiones cíclicas de ida y vuelta.
Se incorpora una distancia de seguridad para que nos permita tomar acciones respecto

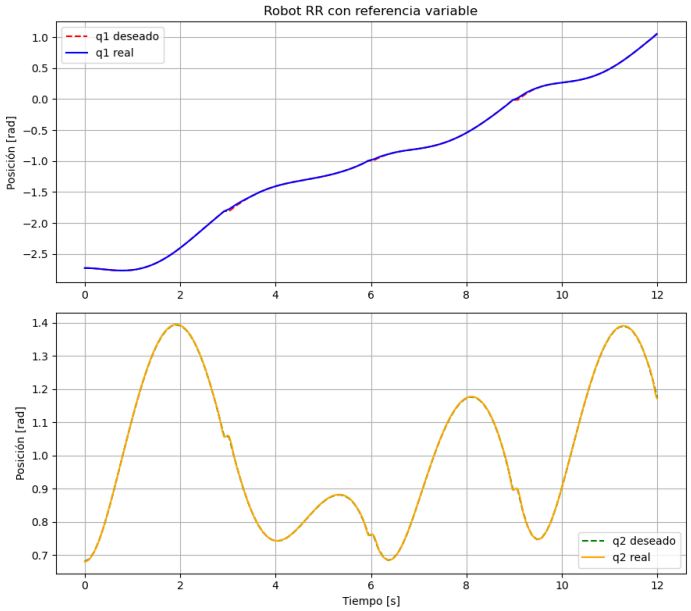


Figura 8.22: Control de seguimiento del robot RR

a la proximidad del robot con respecto al obstáculo. Con base en [41] se modifica la función de costo para penalizar la cercanía del efector final con el obstáculo y se ejecuta el programa para comprobar su desempeño. Con los datos obtenidos del programa se decide si es necesario modificar las ganancias P, Q o R, las condiciones de activación o las ganancias de penalización, para obtener el control deseado.
Al aplicar una penalización basada en la distancia y la posición del obstáculo con respecto al robot, se obtuvo un resultado como el siguiente:
Se puede apreciar que el robot busca esquivar el obstáculo sin detenerse, pero al momento de realizar esto, se pierde precisión del seguimiento de la trayectoria planificada, ya que este programa busca modificar la trayectoria para evitar que el robot colisione con el obstáculo.
En la figura 8.25 se muestran las posiciones y velocidades articulares en cada instante de tiempo. En la gráficas de posición se puede ver una diferencia entre la posición deseada y la real al instante que el robot se acerca al obstáculo; sin embargo, las velocidades no son las mismas que las esperadas, ya que al estar activas las ganancias de penalización para evasión éstas intentan modificar la posición pero al mismo tiempo la ganancia P que controla el error de posición y velocidad entra en acción y prioriza el seguimiento de posición. Debido a que el enfoque que buscamos es tanto el seguir la trayectoria que se planificó como la evasión del obstáculo, se plantea una función de costo que altere las velocidades articulares (acelerar o frenar al robot), lo cual nos permite llevar a cabo la evasión sin modificar el seguimiento de posición de la tarea en cuestión.
Como se muestra en la figura 8.26 se realiza adecuadamente el control de seguimiento de trayectorias del robot RR teniendo un índice de Integral del Valor Absoluto del Error

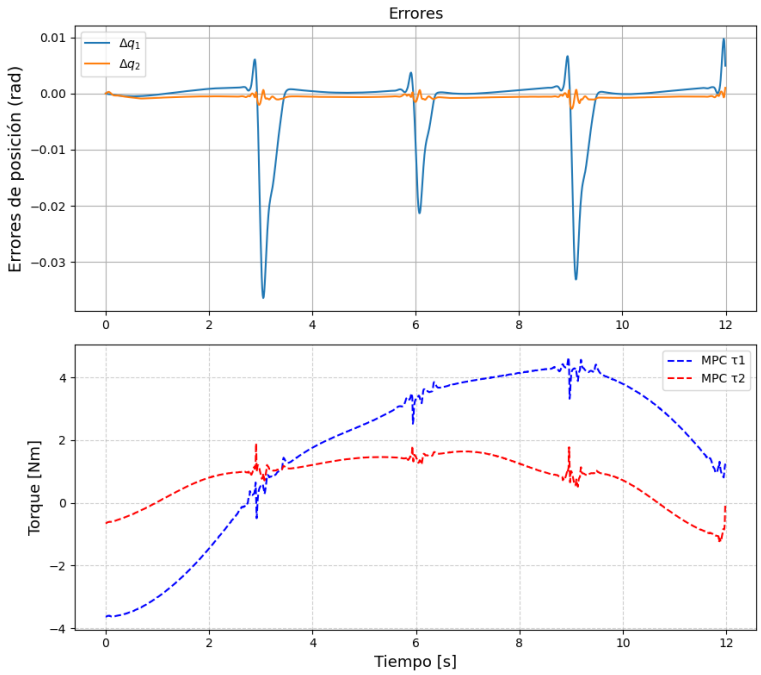


Figura 8.23: Errores de posiciones y torques del robot RR

de 0.0003 rad para el primer eslabón y 0.0001 rad para el segundo; de igual manera, la evasión de obstáculos se realizó adecuadamente al tener una distancia mínima de 14.9 cm. En la gráfica 8.27 se presentan las posiciones y velocidades angulares obtenidas. En ella se observa que, mientras las posiciones coinciden prácticamente con las deseadas, las velocidades reales varían respecto a las estimadas en la planificación de la trayectoria. Esto demuestra que, al penalizar las velocidades en el control, es posible modificarlas para realizar la evasión sin alterar la trayectoria seguida.

Una vez lograda la evasión de obstáculos para el robot RR, se planteó realizar el control NMPC para la evasión de obstáculos en el robot UR5. En comparación con el modelo matemático del robot RR, para este robot no se uso la formulación de Euler-Lagrange, si no por el método de Newton-Euler, siendo su representación en espacios de estados de la forma general como la ecuación 8.14. Siguiendo la metodología anteriormente descrita, se establece el método de integración, se seleccionan los horizontes de predicción y de control, las restricciones, la función de costo y las ganancias Q y R. La ganancia P se estableció como 2 veces Q, para disminuir la carga computacional.

Al contar con un código para realizar el control NMPC de seguimiento de trayectorias se realizaron pruebas para sintonizar las ganancias Q, R pero debido a la complejidad del problema de optimización el tiempo requerido para poder realizar la simulación resultaba alto, ya que el tiempo de optimización para cada paso es de entre 80 y 150 segundos, con un horizonte de predicción N = 3 y un paso de tiempo de 0.04, ocasionando que se generen 300 instancias de tiempo a ejecutar, obteniendo un tiempo aproximado de simulación de 9 horas y 35 minutos como mínimo.

Se realizó una simulación de 2 segundos para poder identificar el comportamiento del

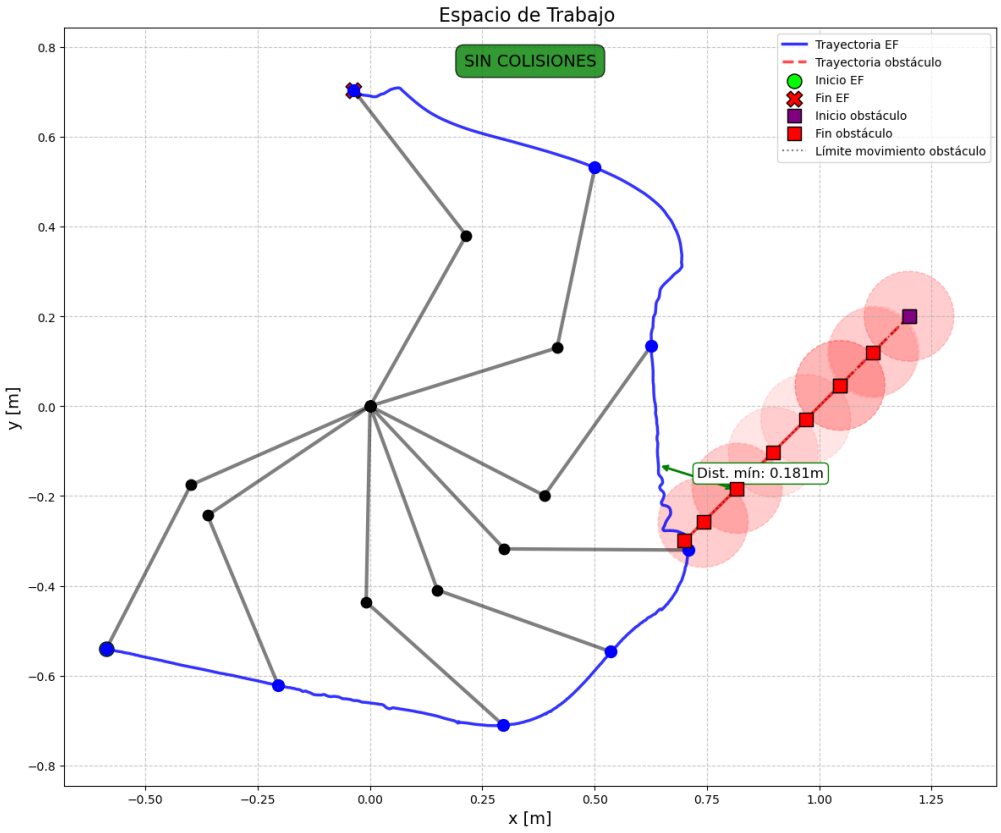
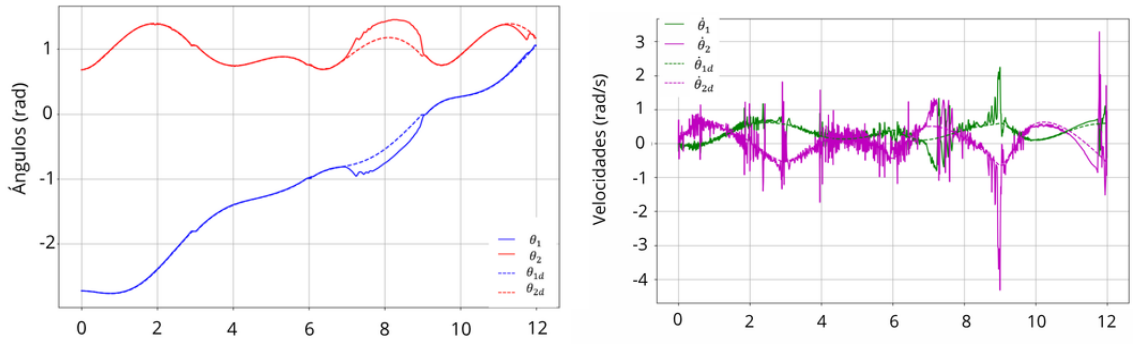


Figura 8.24: Evasión de obstáculos por penalización de posición



(a) Posiciones articulares

(b) Velocidades articulares

Figura 8.25: Posición y velocidad con penalización de posición

control. De dicha simulación se identifico que solo las articulaciones 5 y 6 tenían un error aceptable de al rededor de 1 grado, mientras que las articulaciones 2 y 3 presentaban un error de posición muy grande, de hasta 52 grados.
Considerando este comportamiento y buscando mejorar los tiempos requeridos para la simulación, se propuso controlar solo la posición del robot, dejando fijos los eslabones

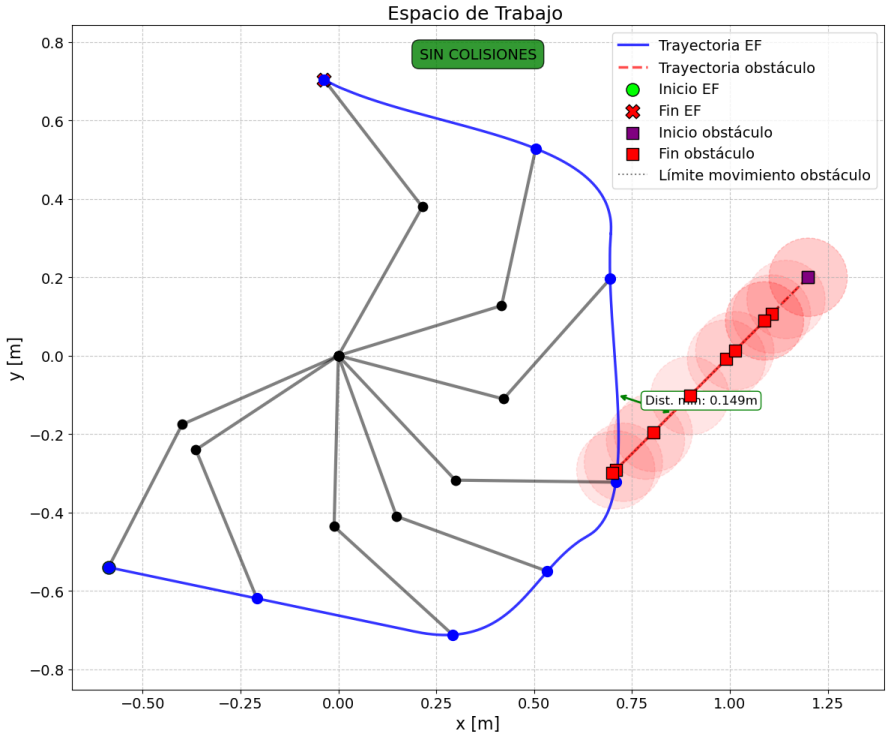
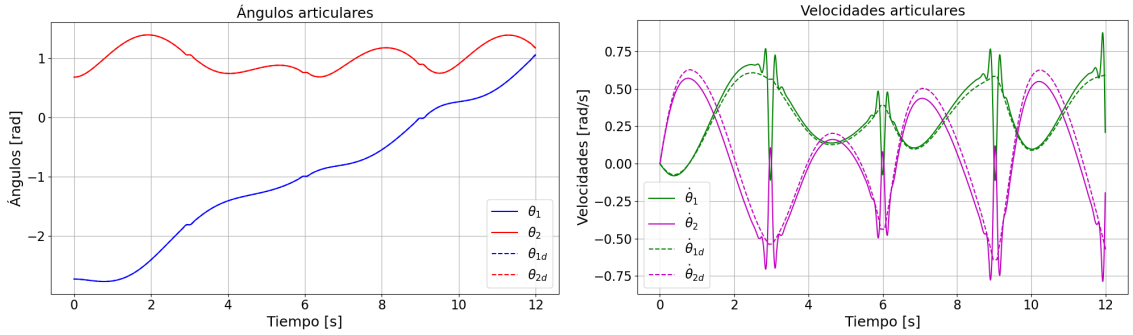


Figura 8.26: Evasión de obstáculos por penalización de velocidad



(a) Posiciones articulares (b) Velocidades articulares

Figura 8.27: Posición y velocidad con penalización de velocidad

4, 5 y 6 correspondientes a la orientación del robot, ya que estos no aportan una mejora significativa en la tarea de tomar y poner que se tiene planeada. Para esto se aplicó el teorema de Steiner, el cual nos permite conocer la masa, centro de masa y momento de inercia del eslabón compuesto por los 4 eslabones, del eslabón 3 al 6, de la siguiente manera: Se calcula el centro de masa del conjunto compuesto (cm_c) por medio de la ecuación:

$$cm_c = \frac{\sum_{i=3}^{6} m_i r_i}{\sum_{i=3}^{6} m_i}$$

donde m_i es la masa del eslabón i y r_i es la posición de su CM respecto al sistema de referencia. Se calcula el momento de inercia total con respecto al centro de masa compuesto por la ecuación:

$$I_{com} = \sum_{i=3}^{6} (I_{i,cm} + m_i d_i^2)$$

donde $I_{i,cm}$ es el momento de inercia del eslabón i respecto a su propio CM y d_i es la distancia perpendicular entre el CM del eslabón i y el CM compuesto.
Al realizar este cambio se redujo el tiempo de optimización a 12 segundos promedio, lo cual nos permite reducir el paso de tiempo de 0.04 a 0.01 generando 1200 instancias de tiempo, con un promedio de simulación de 4 horas, pero con mejor precisión para el control de seguimiento de trayectorias. Se busca generar la evasión de obstáculos del robot UR5 en donde el robot RR se considerará el obstáculo en cuestión.
Al tener en cuenta el tiempo de simulación que se estimó para un robot de 6 grados de libertad y considerando que para generar el NMPC es necesario modelar y optimizar tanto el robot UR5 con 3 grados de libertad y el robot RR, se propuso una técnica de control híbrida PID-NMPC. En este esquema de control, el seguimiento de trayectorias se llevará a cabo por un controlador PID, y una vez que la distancia al obstáculo sea igual o menor a la distancia de seguridad establecida, se cambiará al esquema de control NMPC para llevar a cabo la evasión de obstáculos hasta que el cobot UR5 se encuentre a una distancia mayor o igual a 1.2 veces la distancia segura volver, entonces, al control PID. De este control se obtienen los siguientes gráficos de seguimiento de trayectorias 8.28.
El error máximo mostrado en 8.29 es de 0.025 rad; es decir, un error máximo de 4.5 grados. De igual manera se aplico la métrica de desempeño anteriormente establecida, obteniendo un IAE en grados de 2.4944 para la primer articulación, 3.4147 para la segunda y 0.8180 para la tercera.
Por otra parte, el diagrama de torques 8.30 muestra que mientras el control PID se encuentra activo, los torques obtenidos son los que requiere el sistema para poder llegar a la posición deseada; sin embargo, en cuanto entra el control NMPC trata de disminuir los torques hasta que se encuentren dentro de los limites establecidos y entra en acción una función de transición, la cual permite hacer un cambio más suave entre el torque demandado por el control PID y los torques con los que trabaja el MPC, esto con la finalidad de disminuir el error de posición al realizar un cambio brusco de los torques.
A continuación se muestra la representación grafica de ambos robots en un espacio de trabajo 8.31, donde el riesgo de colisión se presenta cerca de las coordenadas (0.85,0.3).
De igual forma se grafico la distancia del efector final del robot al obstáculo 8.32, para comprobar que no existe colisión en ningún instante de tiempo.
Para poder comprobar que la función de costo afecta principalmente a las velocidades, se generaron las siguientes gráficas 8.33, en donde se puede apreciar que la velocidad de la primera articulación es la más afectada debido a que es la que más influye sobre el desplazamiento en el plano x-y del robot debido a que el robot RR solo tiene efecto en este plano. Con las gráficas obtenidas podemos visualizar que se lleva a cabo adecuadamente la evasión del obstáculo sin dejar de realizar el seguimiento de trayectorias.

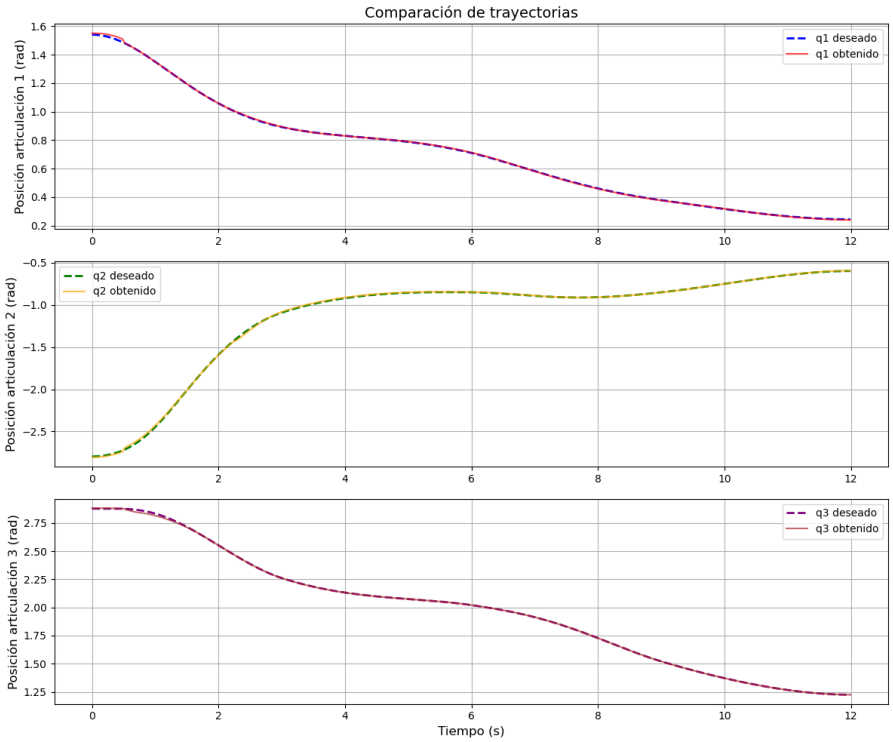


Figura 8.28: Posiciones articulares del robot UR5

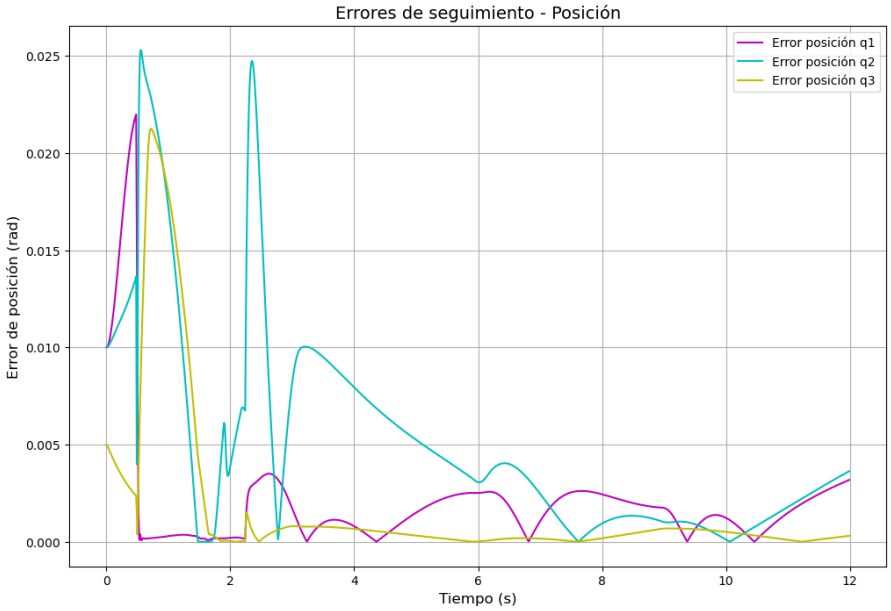


Figura 8.29: Errores de posición del robot UR5

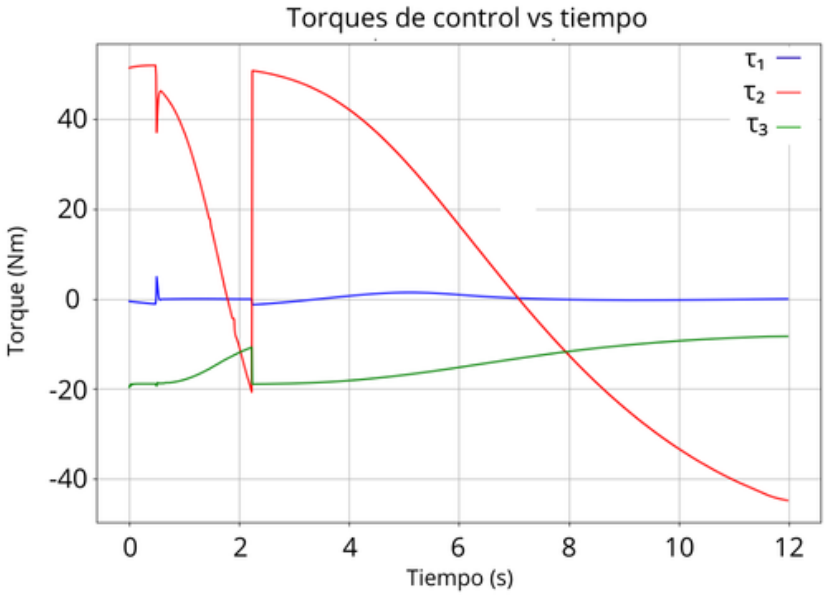


Figura 8.30: Diagrama de Torques con NMPC entre el segundo 0.3 y 2.1

Como parte del enfoque de virtualización, se utilizaron los datos de evasión de obstáculos generados por el control NMPC para calcular las posiciones articulares de cada robot mediante cinemática inversa. Estas trayectorias resultantes se integraron en los modelos virtuales, permitiendo una visualización más clara del comportamiento del sistema.

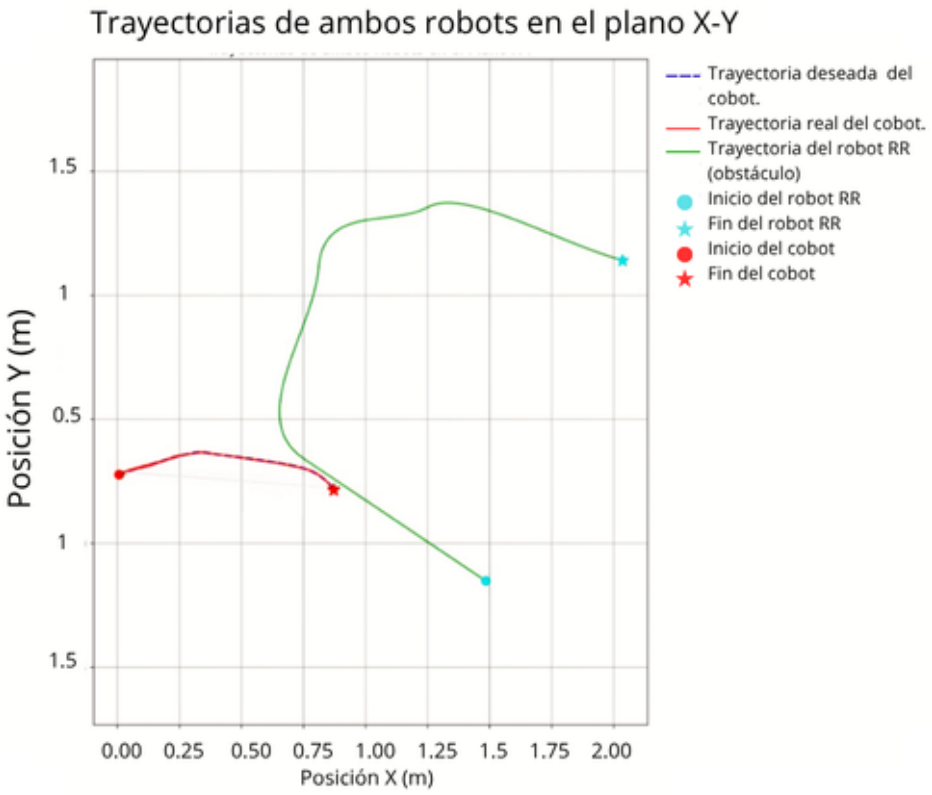


Figura 8.31: Representación del espacio de trabajo de ambos robots

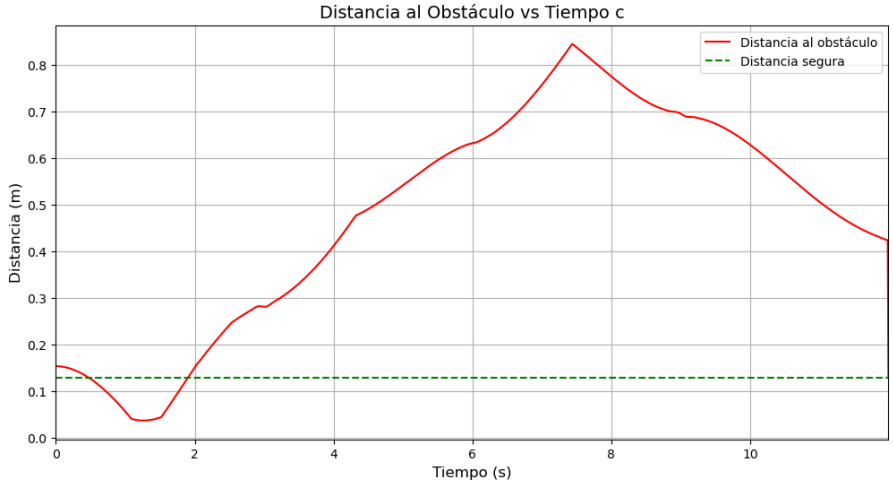


Figura 8.32: Distancia mínima al obstáculo

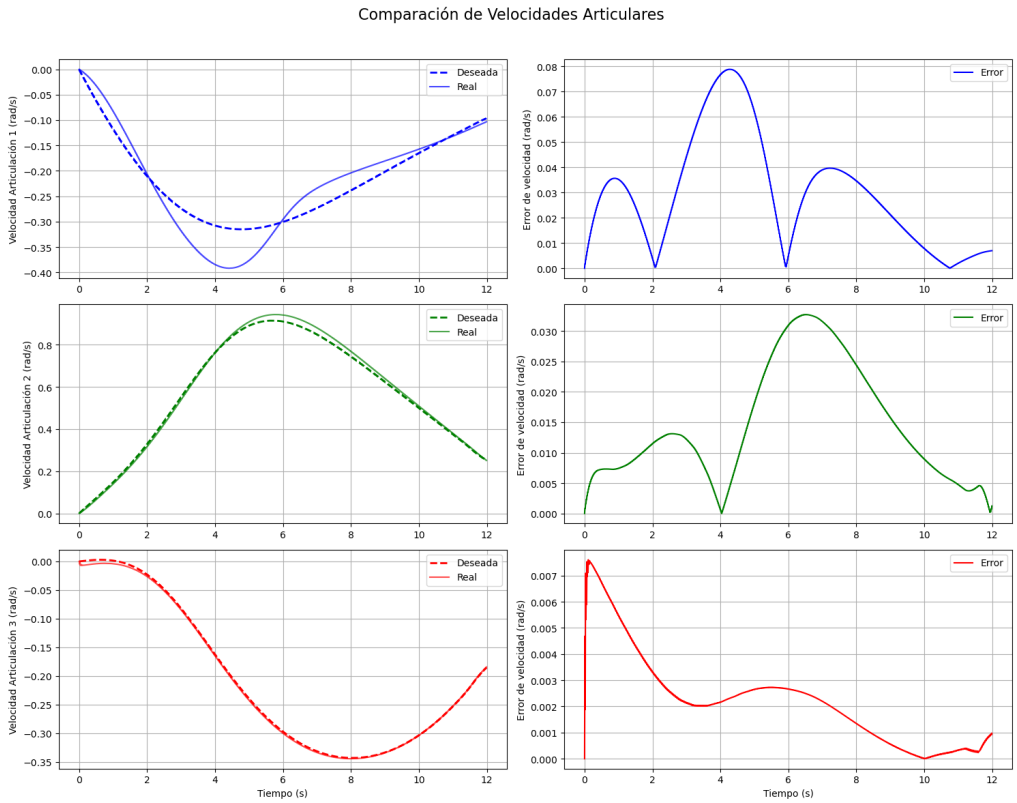


Figura 8.33: Velocidades articulares

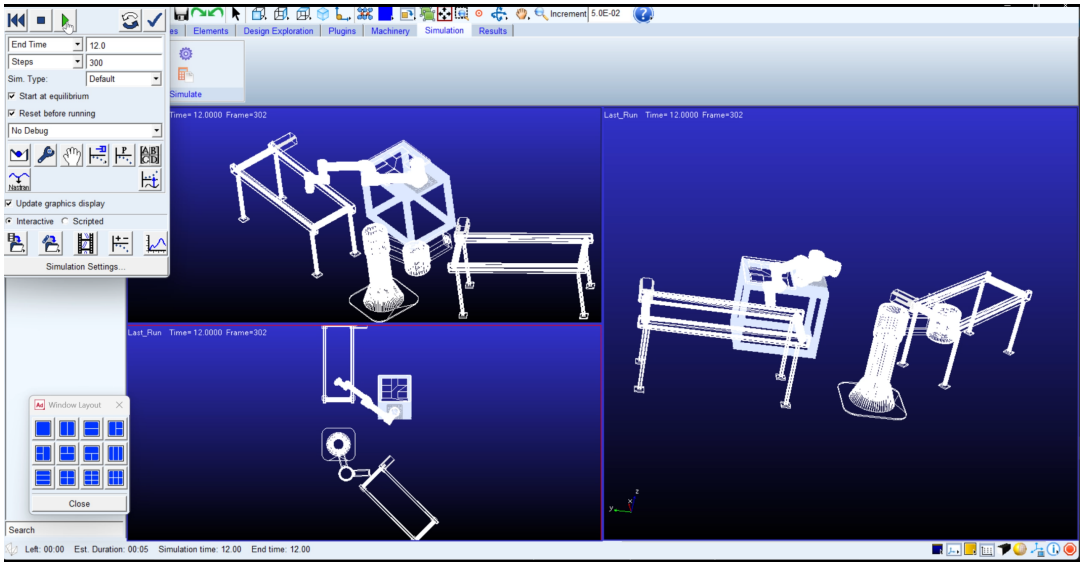


Figura 8.34: Simulación de evasión en Adams view

8.9. Software y Hardware

Para la implementación física y pruebas en tiempo real, se utilizó una Raspberry Pi 4 Model B como unidad de control principal, con las siguientes especificaciones técnicas:

- **Arquitectura**: Computadora de placa única (Single-Board Computer)
- **Procesador**: ARM Cortex-A72 de 64 bits
- **Frecuencia de operación**: 1.5 GHz (configurable hasta 1.8 GHz)
- **Memoria RAM**: 8 GB LPDDR4
- **Almacenamiento**: Tarjeta microSD de 64 GB con velocidad de escritura de 10 MB/s
- **Nota de rendimiento**: La capacidad de procesamiento limita la velocidad de cálculo en comparación con sistemas de gama media/alta, pero resulta suficiente para aplicaciones de control básico en robótica.

Esta tarjeta cuenta con un sistema operativo Raspberry Pi OS, la versión de bullseye con un sistema de 32 bits.

8.10. Implementación de ACADOS

Buscando realizar un control en tiempo real y conociendo que la parte del programa donde se consume la mayor cantidad de recursos y tiempo es la optimización, se propuso incorporar una metodología conocida como ACADOS, la cual es un paquete de software modular y eficiente para la resolución de programas no lineales (PNL) con una estructura de problema de control óptimo.

Al investigar sobre este software se identificó que fue planeado para sistemas operativos basados en Linux, como Ubuntu o Raspbian (sistema de la Raspberry Pi). Por ello, se integró ACADOS en la tarjeta Raspberry Pi. Dado que el lenguaje CasADi, empleado por ACADOS, no era familiar con anterioridad, se realizó una fase de familiarización utilizando la biblioteca `acados_template`, compatible con Python. Como parte de este proceso, se ejecutaron ejemplos precargados, entre los que se incluye el control de un péndulo con referencia sinusoidal, cuyo resultado se muestra en la figura 8.35. De igual manera, se revisó el programa de control de posición de un péndulo invertido por medio de un control de mínimos cuadrados, partiendo de la posición invertida. Conociendo estos ejemplos e identificando algunas de las características necesarias para describir el modelo de los robots que se deseaba controlar, se logró generar un control NMPC para el robot RR, el cual cuenta con un horizonte de predicción $N = 20$ e índices IAE de $2.03°$ para la primera articulación y $4.06°$ para la segunda. De este código se obtuvieron las graficas donde se compara la posición angular deseada contra la obtenida, los errores de posición, así como los torques obtenidos. Como se muestra en 8.37, el control fue realizado en la Raspberry Pi, lo que implicaría tiempos de cálculo más lentos en comparación con una computadora de gama media/alta. A pesar de esta limitación, el tiempo total de ejecución fue de 7.2203 segundos, resultando notablemente inferior a los 62.5148 segundos que consume una computadora de gama alta con un horizonte de predicción 10 veces menor, manteniendo una precisión casi idéntica. Además, los tiempos de optimización por instante se mantuvieron entre 3 ms y 5 ms, lo que refuerza la

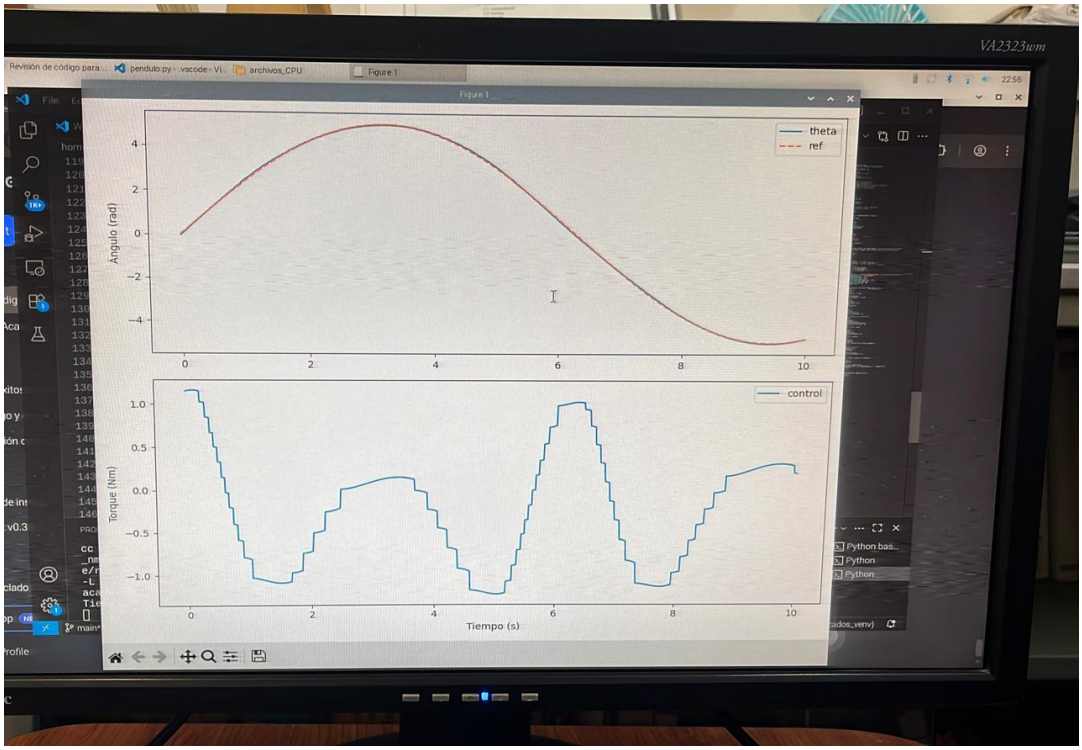
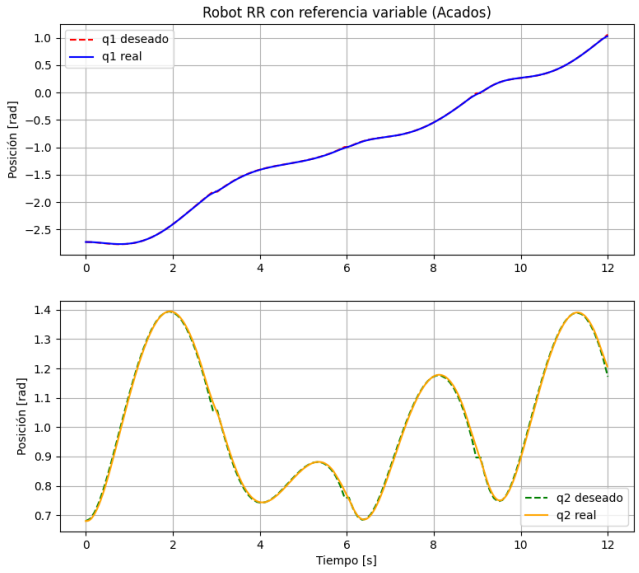


Figura 8.35: Control de péndulo con ACADOS

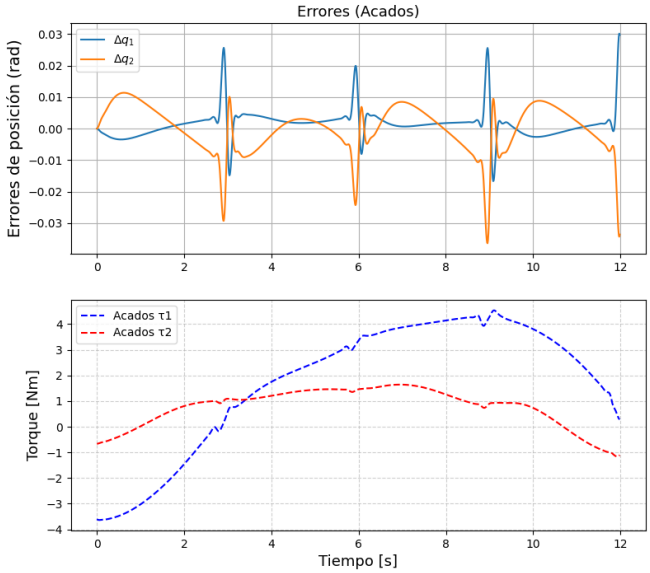
viabilidad de la implementación en este sistema embebido.

Para contextualizar estos resultados en el ámbito de sistemas embebidos para control robótico, resulta adecuado comparar el desempeño de la Raspberry Pi con plataformas típicamente empleadas en robótica. De acuerdo con [42] a Raspberry Pi presenta un rendimiento intermedio entre un microcontrolador de gama baja como el Atmel ARM Cortex-M3 ,incapaz de ejecutar problemas con 6 o más estados por limitaciones de memoria, y plataformas más especializadas como el PYNQ FPGA, que ofrece menores tiempos de ejecución y uso de memoria, tomando como condición el uso de ACADOS para la resolución de problemas. Tomando como ejemplo la implementación de NMPC para un modelo de hovercraft con 9 estados, la Raspberry Pi registró un tiempo de ejecución por iteración de 3.33 ms con qpOASES, mientras que el PYNQ FPGA logró 2.77 ms en las mismas condiciones. Aunque la Raspberry Pi no iguala el rendimiento del PYNQ, su capacidad para resolver problemas de alta complejida y su balance entre costo, accesibilidad y potencia la convierten en una opción viable y competitiva para implementaciones de control robótico en tiempo real.

Si bien las características limitadas de la Raspberry Pi y la necesidad de emplear CasADI para integrar ACADOS a los programas realizados en lenguaje Python, son condiciones que limitan la capacidad de aplicar adecuadamente ACADOS a sistemas complejos, este acercamiento a ACADOS nos permite identificar las fortalezas del software y los requerimientos del mismo para poder aplicarlos a trabajos futuros.



(a) Posiciones articulares obtenidas en ACADOS



(b) Errores de posición y torques

Figura 8.36: Gráficas de ACADOS

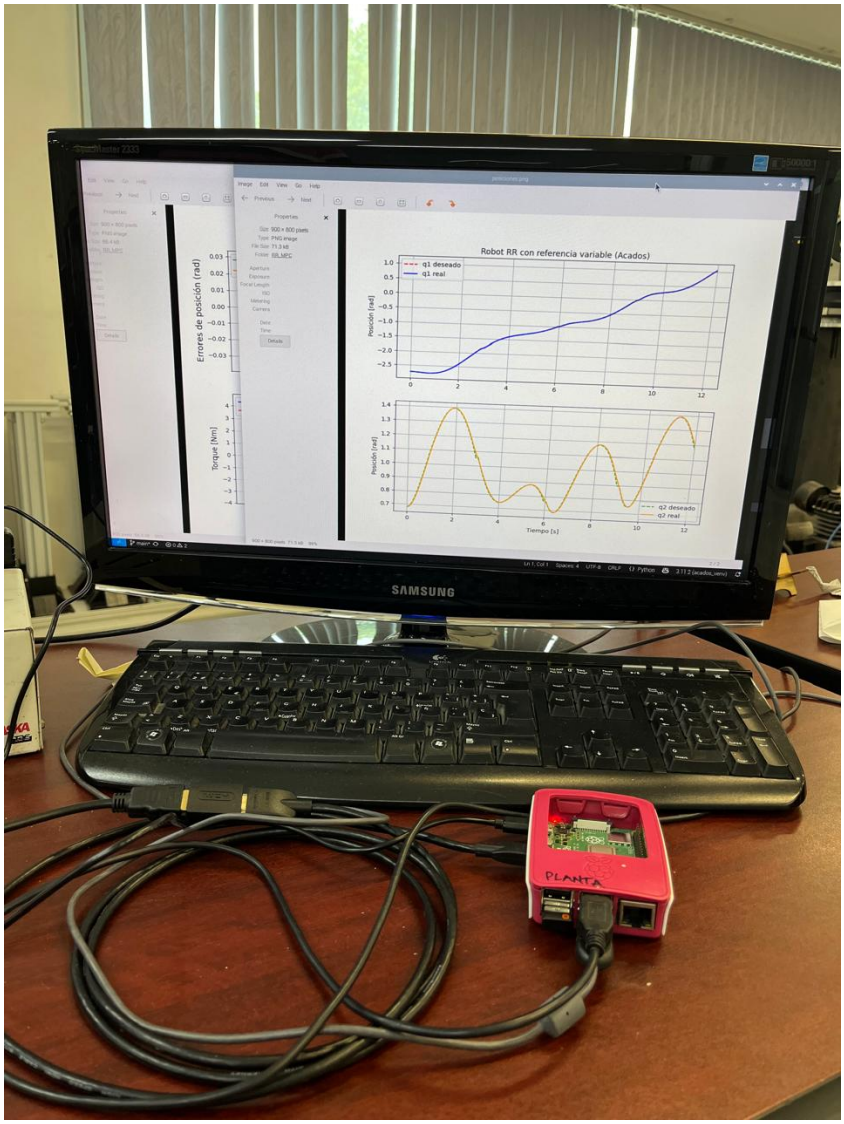


Figura 8.37: Velocidades articulares

Capítulo 9

Conclusiones

La presente tesis logró el diseño e implementación de una plataforma de simulación virtual en Python que integra un cobot de 6 GDL y un robot RR, demostrando la efectividad de un esquema de control predictivo no lineal (NMPC) para evitar colisiones con obstáculos dinámicos mientras se completa una tarea de pick-and-place.

El sistema implementado empleó un método de detección de obstáculos basado en la emulación de sensores de proximidad, capaz de determinar la distancia al obstáculo, su velocidad relativa y dirección de aproximación. Esta información permitió desarrollar una estrategia de control que incorpora penalizaciones en la función de costo para modificar selectivamente las velocidades del robot, logrando la evasión de obstáculos mientras se mantiene la trayectoria de referencia. La función de costo seleccionada considera los errores de posición y velocidad, mientras que el proceso de optimización se lleva a cabo mediante el optimizador SLSQP de la biblioteca `minimize`, elegido por su capacidad para manejar restricciones de desigualdad de forma eficiente.

Entre las ventajas demostradas del MPC frente al control PID convencional se encuentra su capacidad para limitar explícitamente los torques de actuación, así como su habilidad para evadir obstáculos no considerados en la planificación inicial, gracias a su naturaleza predictiva y la incorporación de restricciones de estado. La formulación mediante modelos de Euler-Lagrange y Newton-Euler resultó fundamental para capturar la dinámica no lineal inherente a los robots manipuladores, justificando el uso de un esquema NMPC que evita las pérdidas de precisión asociadas a la linealización.

Para abordar la complejidad computacional del NMPC, se implementó un esquema de control híbrido que combina estrategias de PID y MPC, reduciendo significativamente la carga computacional sin comprometer la precisión del control.

La arquitectura modular del software desarrollado permite adaptar el sistema a diferentes configuraciones de obstáculos con modificaciones mínimas, facilitando su extensión futura hacia implementaciones en plataformas físicas. Finalmente, la implementación exitosa del control en tiempo real en una Raspberry Pi demuestra la viabilidad de ejecutar control predictivo en sistemas embebidos de bajo costo, con tiempos de optimización consistentemente inferiores a 5 ms, cumpliendo con los requisitos de control en tiempo real que típicamente requieren latencias menores a 10 ms para aplicaciones robóticas de precisión.

La selección de Python como entorno de desarrollo resultó estratégica debido a su prac-

ticidad, versatilidad funcional, amplio ecosistema de bibliotecas y robusto soporte comunitario, características que en conjunto permitieron implementar una solución computacionalmente eficiente y metodológicamente sólida.

9.1. Contribuciones Originales

Las contribuciones originales de esta tesis incluyen:

- El diseño e implementación de un entorno virtual de simulación computacional en Python para la prueba y validación de algoritmos de control (MPC) en escenarios de robótica colaborativa multi-robot.
- El desarrollo y análisis comparativo de un controlador MPC frente a un controlador PID para la tarea de evasión de colisiones, demostrando las ventajas del esquema predictivo en un entorno dinámico simulado.
- El desarrollo de una estrategia de control híbrido que reduce la carga computacional del MPC no lineal, manteniendo su nivel de precisión.
- La implementación en tiempo real del algoritmo MPC en una Raspberry Pi, demostrando su potencial para ser desplegado en sistemas embebidos de bajo costo.

9.2. Trabajo a futuro

El trabajo a futuro de este proyecto contempla las siguientes líneas:

- Incorporar incertidumbre en los modelos, para verificar un comportamiento más realista.
- Implementar una co-simulacion con un sistema CAE para validar las respuestas obtenidas antes de llevarlas al entorno físico.
- Implementar sensores LIDAR para la detección de obstáculos, caracterizando experimentalmente los tiempos de adquisición y procesamiento de datos.
- Implementar el sistema de control en una plataforma robótica física para validar los resultados de la simulación.
- Usar esquemas de aprendizaje por refuerzo para optimizar los parámetros del MPC.

Apéndice A

Inicio de Co-Simulación

A.1. Estándar FMI

Este capítulo examina la co-simulación como alternativa para la integración de modelos virtuales, en donde se requiere poder compartir uno o más modelos de diferentes sistemas o software e integrarlos en uno. El estándar FMI, Interfaz de Maqueta Funcional o Functional Mock-up Interface por sus siglas en inglés, es el resultado del proyecto MODELISAR, el cual fue una colaboración de 29 socios europeos con el propósito de desarrollar maquetas funcionales, las cuales son métodos, estándares y herramientas para respaldar el diseño colaborativo, la simulación y las pruebas de sistemas y software integrado.
Los programas con el estándar FMI pueden generar un FMU (functional Mock-up unit; unidad de maqueta funcional), el cual se puede importar a un entorno cerrado para poder usarlo. Para poder abrir los archivos FMU se requiere cambiar la extensión .fmu por .zip, la cual puede contener diversas carpetas y documentos como archivos:

1. Archivo .xml, denominado ”modelDescription.xml”, debido a que proporciona información acerca del modelo como: nombre, fecha y hora de creación, la herramienta donde se generó yel tipo de variables.

2. Una carpeta denominada *”binaries”*, que contiene el código compilado (generalmente en lenguaje C), con las funciones necesarias para la simulación del modelo.

3. archivos ”.dll”. archivos donde se guardan recursos ejecutables del programa.

4. archivos ”.h”, son archivos donde se almacenan definiciones de las variable, constantes y funciones del sistema.

5. archivos ”.log”, registro de errores de la ejecución del programa.

6. archivos ”.json”, son similares a los archivos .xlm, pero solo admiten datos numéricos y de texto.

7. archivos ”.lib.ªlmacenan información de los objetos y datos de la exportación de otros programas.

Actualmente existen 3 versiones de FMI, la FM 1., FM 2.0 y FM 3.0, la más actual. No todos los programas que cuentan con el estándar, son de la misma versión en [43], se puede ver una lista de los programas que cuentan con este estándar y cuál es la versión del mismo.

Co-simulación en Adams

Una vez que se tiene el prototipado virtual se busca establecer la co-simulacion del mismo con el sistema de control. De acuerdo con [44], Adams cuenta con 3 formas de co-simulación.

1. co-simulación (Modo Discreto): En este método, Adams resuelve las ecuaciones del sistema mecánico, mientras que la aplicación de control resuelve las ecuaciones del sistema de control. Ambos sistemas intercambian entradas y salidas en intervalos de tiempo predefinidos, conocidos como intervalos de comunicación.

2. Evaluación de Funciones (Modo Continuo): Aquí, la aplicación de control resuelve tanto las ecuaciones del sistema mecánico como las del sistema de control.

3. Importación del Sistema de Control: Adams resuelve las ecuaciones combinadas del sistema mecánico y del sistema de control importando una Biblioteca de Sistema Externo (ESL).

Debido a que se tiene planeado realizar el control en lenguaje Python, se empleará el tercer método para poder realizar la co-simulación. Para poder llevar a cabo este proceso es necesario convertir el archivo de control que se desee integrar a la co-simulación como un archivo FMU (Unidades de Maqueta Funcional), que siga el estándar FMI para su correcto funcionamiento.
La Interfaz Funcional de Maqueta (FMI) es un estándar abierto diseñado para apoyar el intercambio de modelos de simulación y su integración en diferentes herramientas de simulación. Adams soporta los estándares FMI 1.0 y 2.0 tanto para co-simulación como para intercambio de modelos. En co-simulación, Adams puede actuar como maestro o esclavo, mientras que en intercambio de modelos, Adams opera en modo maestro.

- Modo Maestro FMI: Permite que Adams importe modelos externos (Unidades de Maqueta Funcional, FMUs) compatibles con los estándares FMI para fines de simulación, lo cual se tiene planeado realizar en esta tesis.

- Modo Esclavo de co-simulación FMI: Permite exportar modelos de Adams como FMUs, que luego pueden ser utilizados en otro software que soporten el modo maestro de co-simulación FMI.

Estructura del Archivo FMU

Un archivo FMU es un archivo zip que contiene:

- Un archivo XML de descripción del modelo

- Un archivo binario para el modelo
- Archivos de recursos y código fuente opcionales

El archivo XML proporciona metadatos como definiciones de entrada/salida y banderas de capacidad.
Para poder realizar la conversión de un archivo de Python a un formato FMU, se requiere el uso de una biblioteca que nos permita brindarle las características adecuadas a nuestro programa. Al momento de generar el archivo FMU, se debe tener en cuenta que existen scripts de Python que no se pueden convertir directamente a un archivo FMU, sin embargo se pueden generar archivos tipo .txt o entornos .yml que permitan empaquetar estos datos dentro del archivo FMU para su correcto funcionamiento.

A.2. Unidad de Maqueta Funcional (FMU)

Una Unidad de Maqueta Funcional o FMU (Functional Mock-up Unit) es un formato estandarizado utilizado en la simulación de sistemas dinámicos para integrar y compartir modelos entre diferentes herramientas y entornos de simulación. Las FMU son parte de un estándar llamado Functional Mock-up Interface (FMI), que fue desarrollado para facilitar la interoperabilidad entre diversas plataformas de simulación, promoviendo la reutilización de modelos y reduciendo la dependencia de herramientas específicas.
Una FMU es esencialmente un paquete que contiene todo lo necesario para simular un modelo en un entorno externo. Las principales características incluyen:

1. Es un paquete comprimido con la extensión .fmu que contiene:
   - Código binario o fuente del modelo.
   - Archivos de configuración y metadatos en formato XML.
   - Arcivos adicionales necesarios para el modelo.
2. Estandarización: Los FMU siguen las especificaciones del estándar FMI, lo que asegura su compatibilidad con herramientas que soporten este estándar.
3. Interfaz definida: Los parámetros, entradas, salidas y estados del modelo están definidas dentro del archivo.
4. Autonomía: permitiendo que sea utilizado de manera independiente del software en el que fue creado.

Este tipo de archivos se pueden crear para intercambio de modelos o co-simulación. En el intercambio de modelos el FMU describe las ecuaciones diferenciales del modelo pero la resolución numérica se realiza en el software destino. Mientras que los FMU de co-simulación incluyen un solucionador que resuelve las ecuaciones del modelo de forma independiente.
Debido a sus características este tipo de modelos se puede emplear en diversos entornos

de simulación sin necesidad de realizar cambios, emplear los mismos modelos para diferentes propósitos, eliminar la dependencia de softwares propietarios y adaptar el FMU a las necesidades del proyecto.

A.3. Creación del FMU

Para poder generar el archivo FMU es necesario realizar la creación de un archivo Python. En dicho archivo se debe especificar el modelo con el que se desee trabajar; éste debe contar con las entradas que requiere el modelo para funcionar, las señales de salida que el modelo va a proporcionar, los parámetros y las variables del modelo, así como especificar qué tipo de variables son y qué tipo de cambio pueden presentarse en cada uno de éstos; como se muestra en la figura A.1. Debido a que Python no cuenta como lenguaje nativo para la generación de archivos FMU, se precisa del uso de bibliotecas como Pyfmi, fmpy o Pythonfmu, las cuales permiten transformar el lenguaje Python a uno compatible con el estándar FMI.

```
        self.register_variable(Real("kv1", causality=Fmi2Causality.parameter, variability="fixed", start=self.kv1))
        self.register_variable(Real("kv2", causality=Fmi2Causality.parameter, variability="fixed", start=self.kv2))
        self.register_variable(Real("ki1", causality=Fmi2Causality.parameter, variability="fixed", start=self.ki1))
        self.register_variable(Real("ki2", causality=Fmi2Causality.parameter, variability="fixed", start=self.ki2))

        # Register input variables
        self.register_variable(Real("q1d", causality=Fmi2Causality.input, start=self.q1d))
        self.register_variable(Real("q2d", causality=Fmi2Causality.input, start=self.q2d))
        self.register_variable(Real("q1dotd", causality=Fmi2Causality.input, start=0.0))
        self.register_variable(Real("q2dotd", causality=Fmi2Causality.input, start=0.0))

    def robotRR(self, x, t):
        q1 = x[0]
        q2 = x[1]
        q = np.array([[q1], [q2]])

        q1dot = x[2]
        q2dot = x[3]
        qdot = np.array([[q1dot], [q2dot]])

        xe = self.qd - q
        xedot = self.qdotd - qdot

        m11 = self.m2 * self.l2 ** 2 + 2 * self.m2 * self.l1 * self.l2 * np.cos(q2) + self.m2 * self.l1 ** 2
        m12 = self.m2 * self.l2 ** 2 + self.m2 * self.l1 * self.l2 * np.cos(q2)
        m21 = self.m2 * self.l2 ** 2 + self.m2 * self.l1 * self.l2 * np.cos(q2)
        m22 = self.m2 * self.l2 ** 2
        M = np.array([[m11, m12], [m21, m22]])
```

Figura A.1: Modelo en archivo fuente

Unas vez realizado el archivo fuente, se debe generar un archivo .xml el cual se emplea para describir el comportamiento que se espera obtener del archivo binario, éste funciona como un intérprete para poder acoplar el código fuente a diferentes programas destino. Es en este código, donde se debe especificar el tipo de FMU que se busca generar, teniendo la opción de ser para intercambio de modelo o para co-simulación. De ser el caso de co-simulación se debe recordar que el código fuente debe de contar con un solucionador para resolver las ecuaciones diferenciales presentes en el modelo.
El tercer aspecto importante es la inclusión de recursos adicionales que puedan complementar el modelo, así como imágenes, tablas de datos o bibliotecas externas. Para esta tesis fue necesario generar un archivo .yaml, el cual sirve para indicar aquellos

recursos adicionales que no son intrínsecos del modelo, pero que se requieren para su funcionamiento, como la biblioteca numpy o scipy que se emplean en el modelo para la creación de arreglos y la resolución de las ecuaciones diferenciales, respectivamente. Debido a que las bibliotecas mencionadas no son nativas de Python, es necesario indicar que se deben tomar en cuenta al momento de empaquetar los archivos.
Al contar con estos tres archivos principales se puede proceder a la creación del archivo .fmu. Para esto es necesario generar un código que nos permita empaquetar cada uno de los archivos con las características deseadas, es decir, que el archivo .py se considere el codigo fuente, el .xml lo reconozca como el documento descriptivo o intérprete y el archivo. yaml sean los recurso adicionales para el correcto funcionamiento del modelo.

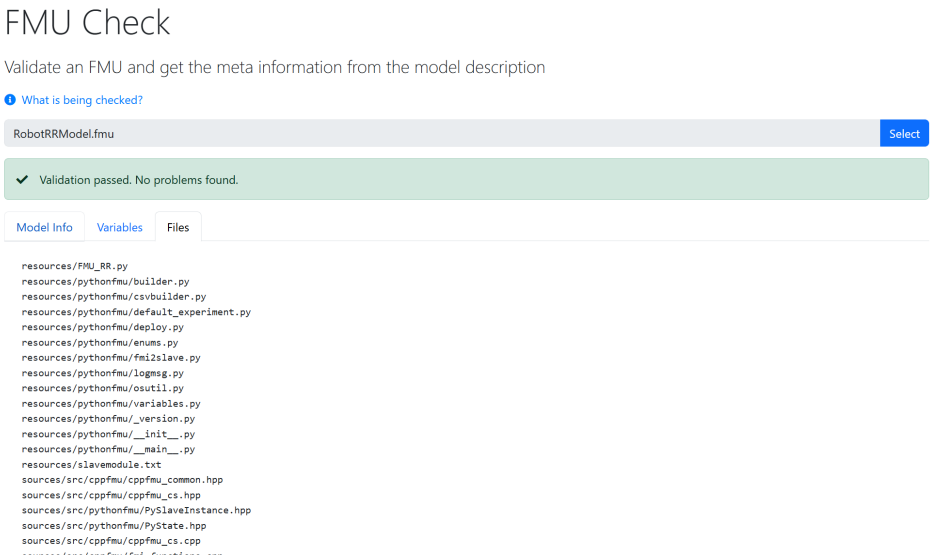


Figura A.2: Verificador de Modelica

Una vez que se ha generado el archivo FMU se requiere corroborar que éste funciona adecuadamente. Para hacer esto existen varias opciones:

- Ejecutar el modelo en el programa destino.
- Generar un código verificador basado en FMIpy, para revisar la recepción de entradas y salidas.
- Emplear el verificador de la asociación Modelica (organización que desarrolla y promueve estándares abiertos para el modelado y simulación de sistemas), para asegurar el correcto uso del estándar FMI

El verificador de Modelica nos permite revisar si el archivo .fmu funciona adecuadamente. Verifica la información del modelo, las variables del mismo y los archivos que se generaron para la creación del FMU, como se muestra en la figura A.2.

Una vez que el archivo .fmu funciona adecuadamente, se realiza su integración con el sistema objetivo, en este caso ADAMS View. Para esto, en necesario indicar en ADAMS el control de las entradas del sistema como, se muestra en la figura A.3, provenientes del sistema de control (FMU) y las salidas que se éspera obtener de este para que el FMU pueda realizar el control.
Una vez que las entradas del sistema están relacionadas con las salidas del modelo y viceversa, ADAMS view genera un archivo del tipo DLL_ADAMS_Master, el cual integra tanto el FMU (modelo) y el sistema de control del propio ADAMS. El propio sistema de ADAMS view reconoce este archivo como un archivo para co-simulación.

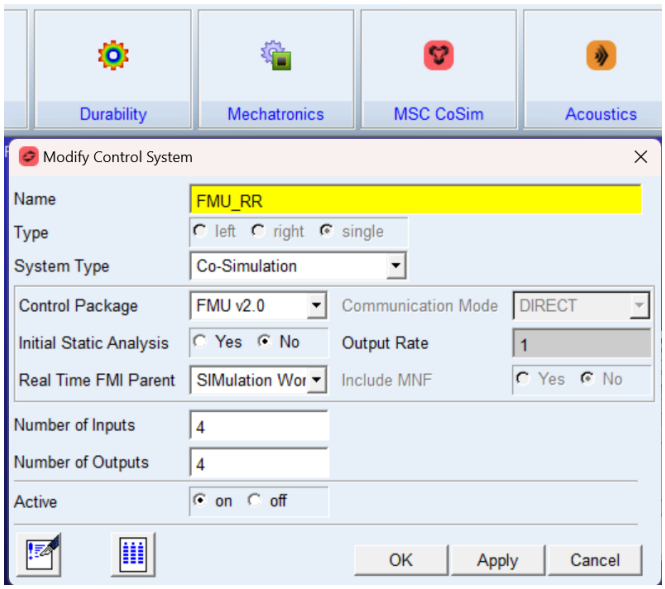


Figura A.3: Interfaz de control mecánico de ADAMS

El archivo DLL_ADAMS_Master que se generó para ADAMS nos permite llevar el robot RR de la posición inicial a una posición deseada.Para poder controlar al robot para que realice adecuadamente la tarea de tomar y colocar que se le asigna, es necesario generar una trayectoria que describa el movimiento que éste debe de realizar.

Bibliografía

[1] N. Kozamernik, J. Zaletelj, A. Košir, F. Šuligoj, and D. Bračun, "Visual quality and safety monitoring system for human-robot cooperation," *International Journal of Advanced Manufacturing Technology*, vol. 128, pp. 685–701, Sept. 2023.

[2] M. Dastbaz, "Industry 4.0 (i4.0): The hype, the reality, and the challenges ahead," in *Industry 4.0 and engineering for a sustainable future*, pp. 1–11, Cham: Springer, 2019.

[3] M. L. Monja, "Cobots: The revolution of collaborative robots," sep 2023. Innovar o Morir.

[4] G. T. Torres, "Cobots-humanos: innovación codo a codo," dec 2021. Urany.

[5] J. F. Castillo, J. J. Ortiz, M. F. D. Velázquez, and D. F. Saavedra, "Cobots in industry 4.0: Safe and efficient interaction," in *IntechOpen - Open Science Open Minds*, ..., aug 2021.

[6] H. Li, D. Gong, and J. Yu, "An obstacles avoidance method for serial manipulator based on reinforcement learning and artificial potential field," *International Journal of Intelligent Robotics and Applications*, vol. 5, pp. 186–202, jun 2021.

[7] A. Y. Afaghani and Y. Aiyama, "On-line collision detection of n-robot industrial manipulators using advanced collision map," in *2015 International Conference on Advanced Robotics (ICAR)*, (Istanbul, Turkey), pp. 422–427, IEEE, jul 2015.

[8] A. Y. Afaghani and Y. Aiyama, "Advanced-collision-map-based on-line collision and deadlock avoidance between two robot manipulators with ptp commands," in *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, (New Taipei, Taiwan), pp. 1244–1251, IEEE, aug 2014.

[9] H. Alagi *et al.*, "Evaluation of on-robot capacitive proximity sensors with collision experiments for human-robot collaboration," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Kyoto, Japan), pp. 6716–6723, IEEE, oct 2022.

[10] Z. Yuan, L. Ma, X. Liu, and W. Zhang, "Research on collision avoidance method of intelligent ship navigation based on reinforcement learning," in *2022 41st Chinese Control Conference (CCC)*, (Hefei, China), pp. 3220–3224, IEEE, jul 2022.

[11] C.-P. Tsai, C.-T. Chuang, M.-C. Lu, W.-Y. Wang, S.-F. Su, and S.-L. Chang, "Machine-vision based obstacle avoidance system for robot system," in *2013 International Conference on System Science and Engineering (ICSSE)*, (Budapest, Hungary), pp. 273–277, IEEE, jul 2013.

[12] P. E. Orukpe, "Model predictive control fundamentals," *Nigerian Journal of Technology (NIJOTECH)*, vol. 31, pp. 139–148, jul 2012.

[13] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, pp. 335–348, may 1989.

[14] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in identification and control* (A. Garulli and A. Tesi, eds.), vol. 245 of *Lecture Notes in Control and Information Sciences*, pp. 207–226, London: Springer London, 1999.

[15] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Madison, Wisconsin: Nob Hill Publishing, 2 ed., 2017.

[16] J. B. Rawlings and K. R. Muske, "The stability of constrained receding horizon control," *IEEE Transactions on Automatic Control*, vol. 38, pp. 1512–1516, oct 1993.

[17] S. C. Guerrero, "Ventajas de la co-simulación y símil eléctrico en simuladores virtuales." Trabajo Fin de Grado en Ingeniería Aeroespacial, Universidad de Sevilla, 2023. Depósito de investigación Universidad de Sevilla.

[18] R. Kelly, *Control de movimiento de robots manipuladores*. Madrid, México: Pearson Educación: Prentice Hall, 2003.

[19] S. Kelly and S. Kelly, "What is python?," in *Python, PyGame and Raspberry Pi Game Development*, pp. 3–5, ..., 2017.

[20] "Adams student edition." Accedido en 2024.

[21] "Adams python interface help (for adams 2018)." Accedido en 2024.

[22] A. Tika, N. Gafur, V. Yfantis, and N. Bajcinca, "Optimal scheduling and model predictive control for trajectory planning of cooperative robot manipulators," in *IFAC-PapersOnLine*, no. 2, pp. 9080–9086, 2020.

[23] H. Yan *et al.*, "Hierarchical mpc-based safe shared control for forward collision avoidance in collaborative driving," *Control Engineering Practice*, vol. 165, p. 106581, Dec. 2025.

[24] N. Eslami and R. Amiadifard, "Moving target tracking and obstacle avoidance for a mobile robot using mpc," in *2019 27th Iranian Conference on Electrical Engineering (ICEE)*, (Yazd, Iran), pp. 1163–1169, IEEE, apr 2019.

[25] H. M. Aafia *et al.*, "A comparison between linear, nonlinear, and adaptive mpc controllers for abb irb120 robot in collaborative assembly," in *2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, (Cairo, Egypt), pp. 495–500, IEEE, may 2022.

[26] A. Palma, A. Reyes, J. Rohten, N. Risso, D. Quezada, and V. Esparza, "Mpc-based traction control for electric vehicles," in *2022 IEEE International Conference on Automation/XXV Congress of the Chilean Association of Automatic Control (ICA-ACCA)*, (Curicó, Chile), pp. 1–5, IEEE, oct 2022.

[27] G. Mannel, C. Hoffmann, and P. Rostalski, "A robust model predictive control approach to intelligent respiratory support," in *2018 IEEE Conference on Control Technology and Applications (CCTA)*, (Copenhagen), pp. 12–17, IEEE, aug 2018.

[28] F. R. Cortés, *Robótica: control de Robots manipuladores*. México: Alfaomega, 2011.

[29] A. Barrientos, *Fundamentos de robótica*. Aravaca, Madrid: McGraw-Hill Interamericana de España, 2 ed., 2017.

[30] A. C. Reddy, "Difference between denavit-hartenberg (d-h) classical and modified conventions for forward kinematic of robots with case study," 2014.

[31] J. J. Craig, *Introduction to robotics: mechanics and control*. Upper Saddle River, N.J: Pearson/Prentice Hall, 3 ed., 2005.

[32] B. Siciliano, *Springer Handbook of Robotics*. Springer Handbooks, Cham: Springer, 2 ed., 2016.

[33] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control: Theory and Algorithms*. Communications and Control Engineering, London: Springer, 2011.

[34] P. M. Kebria, S. Al-wais, H. Abdi, and S. Nahavandi, "Kinematic and dynamic modelling of ur5 manipulator," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, (Budapest, Hungary), pp. 4229–4234, IEEE, oct 2016.

[35] A. Rosyid and B. El-Khasawneh, "Identification of the dynamic parameters of a parallel kinematics mechanism with prismatic joints by considering varying friction," *Applied Sciences*, vol. 10, p. 4820, July 2020.

[36] Y. Oh, T. Lee, S. Ryoo, K. C. Koh, S. Han, and Y. S. Lee, "Reinforcement learning to achieve real-time control of a quadruple inverted pendulum," *International Journal of Control, Automation and Systems*, vol. 23, pp. 2797–2806, Sept. 2025.

[37] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, 2021.

[38] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[39] S. Echeandia and P. M. Wensing, "Numerical methods to compute the coriolis matrix and christoffel symbols for rigid-body systems," *Journal of Computational and Nonlinear Dynamics*, vol. 16, p. 091004, sep 2021.

[40] B. Kouvaritakis and M. Cannon, *Model Predictive Control*. Advanced Textbooks in Control and Signal Processing, Cham: Springer International Publishing, 2016.

[41] Y. Li, G. Li, and K. Peng, "Research on obstacle avoidance trajectory planning for autonomous vehicles on structured roads," *World Electric Vehicle Journal (WEVJ)*, vol. 15, p. 168, Apr. 2024.

[42] S. Adhau, S. Patil, D. Ingole, and D. Sonawane, "Implementation and analysis of nonlinear model predictive controller on embedded systems for real-time applications," in *2019 18th European Control Conference (ECC)*, (Naples, Italy), pp. 3359–3364, IEEE, June 2019.

[43] "Tools - functional mock-up interface." Consultado en 2024.

[44] "Adams 2021.1 controls user guide," 2024.