

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA
SUBDIRECCIÓN DE ESTUDIOS DE POSGRADO



SEGUIMIENTO DE TRAYECTORIA TOLERANTE A FALLAS
PARA UN ROBOT DE TRACCIÓN DIFERENCIAL

POR:

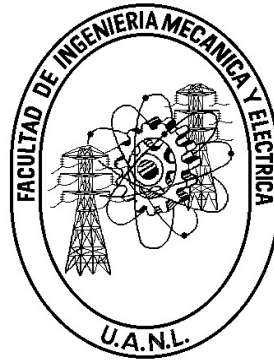
ING. ARTURO RAFAEL CASTRO DANIEL

EN OPCIÓN AL GRADO DE:
MAESTRÍA EN CIENCIAS DE LA INGENIERÍA ELÉCTRICA

SAN NICOLÁS DE LOS GARZA, N.L.

JULIO 2024

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA
SUBDIRECCIÓN DE ESTUDIOS DE POSGRADO



SEGUIMIENTO DE TRAYECTORIA TOLERANTE A FALLAS
PARA UN ROBOT DE TRACCIÓN DIFERENCIAL

POR:

ING. ARTURO RAFAEL CASTRO DANIEL

EN OPCIÓN AL GRADO DE:

MAESTRÍA EN CIENCIAS DE LA INGENIERÍA ELÉCTRICA

SAN NICOLÁS DE LOS GARZA, N.L.

JULIO 2024

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
Facultad de Ingeniería Mecánica y Eléctrica
Posgrado

Los miembros del Comité de Evaluación de Tesis recomendamos que la Tesis "Seguimiento de trayectoria tolerante a fallas para un robot de tracción diferencial", realizada por el estudiante Arturo Rafael Castro Daniel, con número de matrícula 1702129, sea aceptada para su defensa como requisito parcial para obtener el grado de Maestría en Ciencias de la Ingeniería Eléctrica.

El Comité de Evaluación de Tesis

Dr. Efarín Alcorta García
Director

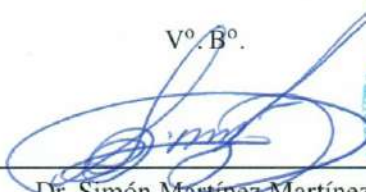
Dr. César Martínez Torres
Co-director

Dr. Miguel Ángel Platas Garza
Revisor

Dr. Juan Ángel Rodríguez Liñán
Revisor

Dr. Ernesto Zambrano Serrano
Revisor

Vº. Bº.


Dr. Simón Martínez Martínez
Subdirector de Estudios de Posgrado



Institución 190001

Programa 557520

Acta N°. 4493

Ciudad Universitaria, a 10 de febrero de 2025.

*Este trabajo está dedicado a mi familia, sobre todo mis
padres, con su apoyo incondicional a lo largo de los años.*

Agradecimientos

A mi director de tesis, Dr. Efraín Alcorta García, por su guía, apoyo y paciencia a lo largo del trabajo.

A mi codirector, Dr César Martínez, su hospitalidad, pláticas y asesoría durante mi estancia en Puebla fueron lo que me ayudó en gran parte a culminar mi trabajo.

A mis padres, Arturo Castro Lastra y Elizabeth del Carmen Daniel Rodríguez, sus enseñanzas y apoyo a lo largo de mi trayectoria estudiantil me han hecho la persona que soy, siempre estaré infinitamente en deuda con ellos.

A mi novia, Megan Atenas Cortez Romero, que me ha visto culminar las últimas dos etapas de mi aprendizaje y siempre estuvo para apoyarme y motivarme durante los días y noches largos de trabajo y revisiones. Mi amor, muchas veces fuiste ese granito de arena que ponía la balanza en mi favor.

Al CONACYT por su gran apoyo al otorgarme la beca y hacer posible el cumplimiento de la maestría.

Resumen

Seguimiento de trayectoria tolerante a fallas para un robot de tracción diferencial

Publicación No. _____

Ing. Arturo Rafael Castro Daniel

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Asesor: Dr. Efraín Alcorta García

Julio 2024

El uso de robots diferenciales en la industria ha aumentado en popularidad debido a su capacidad para realizar múltiples tareas, particularmente en lo que respecta al seguimiento de trayectorias. Como resultado, ha habido una fuerte atención por parte de la comunidad especializada en este campo. Sin embargo, uno de los principales problemas asociados con el uso de robots es la posibilidad de que las fallas afecten significativamente su comportamiento.

A pesar de los diversos esfuerzos realizados en esta dirección, hay poco trabajo reportado en el estudio del efecto de fallas en los actuadores de robots diferenciales. En este trabajo, se presenta un análisis detallado de los efectos de las fallas de los actuadores en el seguimiento de trayectorias de un robot de tracción diferencial llamado TurtleBot3. Para inducir las fallas, se alteró la señal de control del robot en lugar de dañar directamente los actuadores. Específicamente, las fallas se presentaron en forma de una reducción en la velocidad y la aceleración máxima de los desplazamientos de los actuadores.

Se analizó el tamaño de falla que produce desviaciones significativas en la trayectoria del robot y se estableció un criterio para decidir a partir de qué nivel de falla el objetivo de control no se cumple. Los resultados de los experimentos se presentaron tanto en un simulador del TurtleBot3 como en la implementación en un robot real del mismo modelo.

En este estudio también se propone un esquema de control suplementario para disminuir el efecto de las fallas en el seguimiento de trayectoria sin modificar el control nominal o el firmware del robot. Este esquema se ha visto anteriormente en trabajos sobre sistemas lineales y no lineales. La generación del control suplementario dependerá del desempeño nominal del modelo del robot, la identificación de fallas y la sintonización de una ganancia adecuada para el mismo.

En resumen, este trabajo proporciona una comprensión más profunda de los efectos de las fallas de los actuadores en el seguimiento de trayectorias de un robot diferencial y propone un esquema de control suplementario para disminuir el impacto de las fallas sin modificar el control nominal o el firmware del robot. Estos resultados pueden ser útiles para mejorar la eficiencia y la seguridad en el uso de robots en la industria.

Keywords: differential robots, trajectory tracking, actuator faults, control scheme, TurtleBot3, supplementary control, performance analysis, fault identification, gain tuning.

Palabras clave: robots diferenciales, seguimiento de trayectorias, fallas de actuadores, esquema de control, TurtleBot3, control suplementario, análisis de desempeño, identificación de fallas, ajuste de ganancia.

Contenido

Lista de figuras	X
Lista de tablas	XIII
1. Introducción	1
1.1. Preliminares	3
1.2. Hipótesis	5
1.3. Objetivos	5
1.3.1. General	5
1.3.2. Específicos	5
1.4. Metodología	6
1.5. Estado del arte	7
1.6. De Matlab a Python	12
2. Modelo del robot y de las fallas.	14
2.1. Modelo	14
2.1.1. Cálculo del centro de masa	17
2.2. Trayectoria	18
2.3. Inducción de fallas	20
2.4. Detección de fallas	22
2.5. Control Suplementario	23
2.5.1. Diagrama de bloques	23
2.5.2. Efectos en el control nominal	24
3. Implementación	26
3.1. Plataforma de Implementación	26
3.2. Sistema de retroalimentación	28
3.2.1. Sistema de visión para odometría	28

3.2.2.	Implementación y mejora de los sensores para la detección de fallas	29
3.2.3.	Integración y mejoras en el sistema	30
3.3.	Refinamiento del cálculo de ganancias	31
4.	Resultados	32
4.1.	Resultados Simulaciones	32
4.1.1.	Resultados Implementación	34
4.1.1.1.	Consideraciones	34
4.1.1.2.	Falla del 20 % en el motor derecho	35
4.1.1.3.	Falla del 20 % en el motor izquierdo	38
4.1.1.4.	Falla del 40 % en el motor izquierdo	40
4.1.1.5.	Falla del 60 % en el motor izquierdo	43
4.2.	Trabajo a futuro	47
4.3.	Conclusiones	49
A.	Gráficas de resultados	51
A.1.	Apéndice 1: Gráficas de efecto de fallas	51

Lista de figuras

1.1. Diagrama de FTC con Filtros de Kalman Extendidos.	8
1.2. Diagrama de FTC s.	9
1.3. Diagrama de FTC s.	10
2.1. Ubicación del centro de masa.	18
2.2. Trayectoria, velocidades y aceleraciones deseadas.	19
2.3. Diagrama de bloques	20
2.4. Velocidad nominal de la rueda derecha con la trayectoria seleccionada.	21
2.5. Velocidad con falla inducida de la rueda derecha con la trayectoria seleccionada.	22
2.6. Diagrama de bloques con control suplementario.	24
3.1. Diagrama de comunicación entre los nodos de ROS.	27
3.2. Sistema de visión VICON.	29
3.3. Servomotores Dynamixel.	30
3.4. Sistema integrado para el Turtlebot3 WafflePi.	30
4.1. Error con falla.	32
4.2. Error con falla y control suplementario.	33
4.3. Error con el motor derecho al 80 % de eficiencia sin control suplementario	35
4.4. Error con el motor derecho al 80 % de eficiencia con control suplementario	36
4.5. Trayectoria con el motor derecho al 80 % de eficiencia sin control suplementario	37
4.6. Trayectoria con el motor derecho al 80 % de eficiencia con control suplementario	37
4.7. Error con el motor izquierdo al 80 % de eficiencia sin control suplementario	38

4.8. Error con el motor izquierdo al 80 % de eficiencia con control suplementario	39
4.9. Trayectoria con el motor izquierdo al 80 % de eficiencia con control suplementario	39
4.10. Trayectoria con el motor izquierdo al 80 % de eficiencia sin control suplementario	40
4.11. Error con el motor izquierdo al 60 % de eficiencia sin control suplementario	41
4.12. Error con el motor izquierdo al 60 % de eficiencia con control suplementario	41
4.13. Trayectoria con el motor izquierdo al 60 % de eficiencia sin control suplementario	42
4.14. Trayectoria con el motor izquierdo al 60 % de eficiencia con control suplementario	43
4.15. Error con el motor izquierdo al 40 % de eficiencia sin control suplementario	44
4.16. Error con el motor izquierdo al 40 % de eficiencia con control suplementario	45
4.17. Trayectoria con el motor izquierdo al 40 % de eficiencia sin control suplementario	46
4.18. Trayectoria con el motor izquierdo al 40 % de eficiencia con control suplementario	46
4.19. Algoritmo para control de seguimiento de trayectoria en tiempo real.	48
A.1. Error Nominal	51
A.2. Error con falla en el motor derecho a los 10 segundos, con pérdida del 5.0 % de potencia.	52
A.3. Error con falla en el motor derecho a los 10 segundos, con pérdida del 10.0 % de potencia.	52
A.4. Error con falla en el motor derecho a los 10 segundos, con pérdida del 20.0 % de potencia.	53
A.5. Error con falla en el motor derecho a los 10 segundos, con pérdida del 30.0 % de potencia.	53
A.6. Error con falla en el motor derecho a los 10 segundos, con pérdida del 40.0 % de potencia.	54

A.7. Error con falla en el motor derecho a los 10 segundos, con pérdida del 50.0 % de potencia.	54
A.8. Error con falla en el motor derecho a los 10 segundos, con pérdida del 60.0 % de potencia.	55

Lista de tablas

- 3.1. Parámetros del Turtlebot3 26
- 4.1. Resultados de error en mm 33
- 4.2. Resultados de error Máximo en mm 47
- 4.3. Resultados de error Final en mm 47

Capítulo 1

Introducción

Los robots móviles son un equipo común en aplicaciones de manufactura, almacenaje, transporte, vigilancia, limpieza y reconocimiento. Estos se desplazan a través de las instalaciones siguiendo una trayectoria fija enmarcada por líneas guía, cinta magnética y puntos RFID de referencia entre otros. La asistencia de visión computacional para maniobrar es de los nuevos avances que se presentan. El traslado y almacenaje de productos se ha usado desde los inicios de la producción en masa, el grado de automatización de estas actividades ha crecido junto con las tecnologías de robótica, visión, computación y la integración de esas en el contexto de la Industria 4.0. Los nuevos conocimientos en la rama de control de robots crean, con lo anterior, un ambiente con una creciente complejidad de interacción entre las distintas disciplinas necesarias para lograr que un robot móvil consiga seguir una trayectoria. Se han reportado diversos algoritmos utilizados para el control de robots diferenciales, ver por ejemplo [3], [8]. [9]. Debido a esta situación, hay una gran cantidad de algoritmos para el control de seguimiento de estos sistemas, ver por ejemplo [5] y las referencias de este trabajo.

En esquemas de control tolerante a fallas (FTC) como los de generador de residuos adaptable, desigualdades de matrices lineales y esquema de redundancia, son diseñados y utilizados según un solo tipo de falla de actuador. En la implementación se pueden presentar distintos tipos de fallas de actuador. Esto ha llevado integración de distintos controles, los cuales se activan según la falla presentada, lo que hace que la sintonización y activación pueda volverse complicada, además de ser propenso a presentar fallas de rendimiento en cuanto a procesamiento. Por las razones mencionadas anteriormente, se trabaja en el desarrollo de un esquema de FTC capaz de rechazar el efecto de la mayor cantidad de fallas, si no es que de todas. Esto debe

permitir a un sistema acercarse a la respuesta deseada en la presencia de fallas dentro de un rango de error permisible.

Existen muchos reportes en la literatura sobre esquemas de control tolerante a fallas para robots diferenciales. En [16] un enfoque de tolerancia a fallas en actuadores es diseñado e implementado basado en modelos-múltiples, lo cual requiere de un esfuerzo adicional para la implementación. Utilizando teoría de información en [1] se propone un esquema de diagnóstico y tolerancia a fallas el cual también es implementado. El esquema utiliza un modelo estimado y otro asimétrico además de predicciones. Un esquema de tolerancia a fallas utilizado modos deslizantes es propuesto en [15].

Para lograr el mayor entendimiento acerca de la deflexión de fallas, se debe de conocer el efecto de estas sobre el sistema. Con el desarrollo de la robótica y sensores, cada vez se posee más información respecto a las variables de interés en un sistema para poder detectar e identificar fallas. Algunas de las herramientas para ello son los sensores de velocidad y posición en los servos que mueven las ruedas, sistemas de visión ya sean montados en el robot o de vigilancia externa y sensores de aceleración internos. Estos avances han permitido la creación de lazos de control adicionales a los sistemas de control de un robot. Una de las propiedades que da valor agregado a los lazos de control es la característica de tolerancia a fallas por parte de estos. Partiendo de que todo lazo cerrado de control cuenta con una cierta capacidad para tolerar fallas, [6].

En este trabajo se presenta primeramente una forma de incluir fallas en actuador de un Turtlebot3 sin necesidad de modificar la estructura de hardware del robot diferencial. Esta implementación de la falla toma ventaja del conocimiento del modelo nominal, para alterar la señal de control. Una falla en el actuador, como pérdida de eficiencia del suministro de energía a uno de los motores es implementada y probada experimentalmente. Una vez que se tiene la falla implementada para uno de los motores se procedió a mostrar el efecto de diferentes magnitudes de falla en una trayectoria predefinida. La trayectoria incluye una línea recta así como giros a ambas direcciones. Se propone un intervalo alrededor de la variable de posición para considerar el comportamiento aceptable. Entonces se determina los tamaños de falla que pueden ser tolerados por el control nominal. Se utilizó esta aproximación debido a que una falla de actuador como pérdida de eficiencia en el suministro de energía, deterioro de la rueda o algún objeto que bloquee el giro del motor pueden

ser representadas a manera de pérdida de velocidad de giro. Una vez que se tiene la falla implementada para uno de los motores se procedió a mostrar el efecto de diferentes magnitudes de falla en una trayectoria predefinida. La trayectoria incluye una línea recta, así como giros a ambas direcciones. Se propone un intervalo alrededor de la variable de posición para considerar el comportamiento aceptable. Entonces se determina los tamaños de falla que pueden ser tolerados por un controlador. Esto se realiza con el objetivo de generar una herramienta para delimitar los alcances de diversos controladores aplicados al seguimiento de trayectoria de un robot móvil seleccionado, bajo condiciones de falla. En este estudio se seleccionó el Turtlebot3 con un controlador robusto obtenido de [5]. Seguido se propone un esquema de FTC con un lazo cerrado adicional para minimizar el efecto de las fallas el cual toma en cuenta la diferencia entre un desempeño nominal modelado y el error de posición retroalimentado. Este se probó en un simulador y se llevó a la implementación. En la simulación se utilizaron las mediciones de posición dadas por el mismo, las cuales no presentan un error de medición por la naturaleza de su obtención. En la implementación se integró un sistema de visión de alta precisión que presentó retos adicionales los cuales son descritos en el Capítulo 3

En la siguiente sección se revisan algunos preliminares generales sobre fallas y su tolerancia en lazos de control. En la sección 3 se discute el modelo del Turtlebot3 y sus parámetros además de presentar la manera de incluir las fallas. En la sección 4 se discute el tamaño de fallas tolerables mediante el control nominal y el intervalo de la magnitud de las fallas que produce trayectorias fuera del intervalo considerado como aceptable. En la sección 5 se presentan las conclusiones.

1.1. Preliminares

En el diseño y análisis de sistemas robóticos, es fundamental tener una definición clara y operativa de lo que constituye una falla. Siguiendo la terminología estándar aceptada en la comunidad de diagnóstico de fallas, definimos una falla como un cambio en el valor de un parámetro que se desvía más allá de los límites de tolerancia predefinidos. Esta definición es ampliamente utilizada en la literatura pertinente, como se puede ver en [14], por ejemplo.

Para establecer un criterio que permita determinar si un sistema puede tolerar o no una falla específica, presentamos la siguiente definición.

Considérese un sistema dinámico en lazo cerrado que es estable y cumple un objetivo de control específico. Supongamos que una falla $f_i(t)$ afecta al sistema en el tiempo $t = t_f$. Si la desviación de la salida del sistema, después del tiempo en que se presenta la falla, se mantiene dentro de un intervalo alrededor de la salida nominal:

$$y_n(t) - y_m \leq y(t) \leq y_n + y_M, \quad (1.1)$$

donde $y_n(t)$ representa la salida nominal del sistema, y_m y y_M representan las cotas de los valores inferiores y superiores respectivamente, que definen el intervalo de comportamiento aceptable del sistema. Si $y(t)$ se encuentra dentro de este intervalo, entonces decimos que el sistema tolera la falla.

Esta definición permite distinguir entre fallas que son toleradas por el sistema de control en lazo cerrado (si la salida del sistema permanece dentro del intervalo aceptable) y las que no lo son (si la salida del sistema se sale del intervalo aceptable). Es importante destacar que una falla no tolerada no implica necesariamente que la salida del sistema diverge o que el sistema se vuelve inestable. Esto nos proporciona una base para discutir métodos para incrementar la tolerancia a fallas de un sistema. En el contexto de los sistemas robóticos, consideramos especialmente los sistemas dinámicos de la forma Port-Hamiltoniana (PHC):

$$\dot{x} = J(x)\nabla H(x) + G(x)u, \quad (1.2)$$

donde $x \in \mathbb{R}^n$ es el vector de estado, $u \in \mathbb{R}^m$ es la entrada de control, $J(x)$ es una matriz antisimétrica, $G(x)$ es una matriz de entrada, y $H(x)$ es una función Hamiltoniana.

Las ecuaciones de movimiento de los sistemas PHC se pueden escribir de forma compacta como:

$$\dot{x} = -\nabla \cdot \mathbb{F}(x) + G(x)u, \quad (1.3)$$

donde $\mathbb{F}(x)$ es el vector de flujo de energía. La entrada de control u se puede obtener como:

$$u = -G(x)^+\mathbb{K}(x) + G(x)^+\nabla \cdot \mathbb{F}_{des}(x), \quad (1.4)$$

donde $G(x)^+$ es la pseudo-inversa de $G(x)$, $\mathbb{K}(x)$ es una matriz de retroalimentación, y $\mathbb{F}_{des}(x)$ es el vector de flujo de energía deseado.

Al elegir apropiadamente las matrices de retroalimentación $\mathbb{K}(x)$, es posible estabilizar el sistema alrededor de un punto de equilibrio deseado, mientras se satisfacen ciertos objetivos de control. En este sentido, es importante destacar que el diseño del controlador debe llevarse a cabo teniendo en cuenta la tolerancia a las fallas del sistema.

El uso de controladores basados en la formulación Port-Hamiltoniana puede permitir la consideración explícita de las energías en el sistema, lo que puede ser beneficioso en términos de tolerancia a fallas. En particular, puede ser posible diseñar controladores que gestionen de manera efectiva la energía en el sistema en presencia de fallas, evitando que las fallas conduzcan a situaciones indeseables, como un aumento incontrolado de la energía.

Además, la formulación Port-Hamiltoniana también puede proporcionar un marco útil para el diseño de observadores de fallas, los cuales pueden ser utilizados para detectar y diagnosticar fallas. Estos observadores de fallas pueden basarse en la estructura energética del sistema y pueden ser capaces de detectar fallas basándose en sus efectos sobre esta estructura.

1.2. Hipótesis

Es posible atenuar el error de seguimiento generado por fallas de actuador de un robot diferencial mediante un segundo lazo de control que retroalimente una salida nominal generada por un modelo sin fallas del robot en cuestión.

1.3. Objetivos

1.3.1. General

- Desarrollar e implementar un esquema de control robusto y tolerante a fallas para el seguimiento preciso de trayectorias en un robot de tracción diferencial, específicamente el Turtlebot 3.

1.3.2. Específicos

1. Investigar y seleccionar un esquema de control adecuado para el seguimiento de trayectorias que pueda ser adaptado para ser tolerante a fallas (FTC).

2. Diseñar y simular este esquema de control FTC en un entorno de simulación virtual utilizando el modelo de un robot de tracción diferencial.
3. Implementar el esquema de control FTC propuesto en una plataforma de desarrollo basada en ROS y Gazebo.
4. Transferir y poner a prueba el esquema de control FTC en un robot real (Turtlebot 3), utilizando un sistema de retroalimentación externo para verificar el seguimiento preciso de trayectorias.

1.4. Metodología

La metodología propuesta para alcanzar los objetivos planteados consta de los siguientes pasos:

1. Realizar una revisión bibliográfica rigurosa acerca del efecto de las fallas en los actuadores sobre el seguimiento de trayectorias, además de explorar las aproximaciones actuales para mitigar y tolerar diferentes tipos de falla.
2. Seleccionar un algoritmo de control apropiado para el seguimiento de trayectorias de un robot diferencial y programarlo utilizando el lenguaje de programación Python.
3. Crear la arquitectura de control en el entorno de simulación basado en ROS-Gazebo para poder modelar y probar el algoritmo de control.
4. Implementar el algoritmo de control seleccionado en el entorno de ROS-Gazebo y analizar su funcionamiento.
5. Desarrollar e implementar un algoritmo que pueda simular fallas en los actuadores, tanto en el entorno de Python como en el simulador ROS-Gazebo.
6. Realizar un estudio de los efectos de las fallas en el desempeño del algoritmo de control y establecer los límites de error permisibles (y_m, y_M) para que el sistema aún se considere funcional.
7. Investigar y seleccionar algoritmos de control tolerantes a fallas que se enfoquen en la tolerancia a fallas en cascada, es decir, que puedan manejar fallas múltiples y consecutivas.

8. Implementar el esquema de control tolerante a fallas seleccionado y probar su funcionamiento en el entorno de simulación.
9. Evaluar la efectividad y robustez del esquema tolerante a fallas en el simulador y realizar ajustes si es necesario.
10. Desarrollar un sistema de retroalimentación de posición para poder implementar el controlador en un ambiente real. Este sistema debe ser compatible con el entorno de ROS y utilizar un sistema de visión con suficiente exactitud.
11. Implementar el esquema de control tolerante a fallas en el Turtlebot 3 en un entorno real y evaluar su desempeño.
12. Analizar los resultados obtenidos, elaborar las conclusiones y discutir las posibilidades para trabajos futuros en este ámbito.

Este protocolo se llevará a cabo de manera secuencial, siguiendo los estándares más rigurosos para garantizar la calidad y la validez de los resultados obtenidos, con la finalidad de cumplir a cabalidad los objetivos planteados en este estudio.

1.5. Estado del arte

Los robots de tracción diferencial, también conocidos como robots diferenciales, forman una subclase de robots móviles que utilizan dos ruedas motrices independientes para moverse en el espacio. Cada rueda está conectada a un motor independiente, lo que le otorga al robot una gran flexibilidad para controlar la velocidad y dirección de cada rueda de forma individual. Este diseño aporta simplicidad mecánica, flexibilidad en la movilidad y habilidad para maniobrar en terrenos irregulares y sortear obstáculos con gran eficiencia. Además, en términos de costos y peso, la tracción diferencial ha demostrado ser una configuración de ruedas y motores altamente eficiente.

Estas cualidades han llevado a la amplia adopción de robots diferenciales en diversas áreas de aplicación, incluyendo agricultura de precisión, inspección de tuberías, minería y exploración espacial. En la industria automotriz, se usan para transporte de partes, soldadura y pintura de carrocerías de vehículos. Dentro del ámbito de la investigación y la educación en ciencia, tecnología, ingeniería y matemáticas (STEM),

los robots diferenciales han demostrado ser plataformas robustas para el desarrollo, la prueba de algoritmos de control y la exploración de nuevas técnicas de inteligencia artificial.

No obstante, aún quedan desafíos significativos que deben abordarse en el uso de estos robots. Uno de los principales es el manejo de fallas en los actuadores, las cuales pueden derivar en comportamientos impredecibles del robot. Para abordar este problema, se han desarrollado diversas arquitecturas de control tolerantes a fallas (FTC, por sus siglas en inglés) que buscan aumentar la robustez de los robots móviles en presencia de fallos.

El control de este tipo de robots ha sido objeto de numerosos estudios, los cuales han propuesto diversas arquitecturas de control tolerantes a fallas. Estas estructuras han demostrado ser una técnica efectiva para aumentar la robustez de los robots diferenciales ante la eventualidad de fallos en los componentes.

En los últimos años, han surgido diversos enfoques para mejorar la tolerancia a fallas en robots de tracción diferencial. Estos enfoques incluyen desde el uso de algoritmos sofisticados de control y detección de fallos hasta la implementación de redundancias en el hardware y software del robot.

Zhang et. al 2017 [17] propuso una arquitectura FTC para localización de robots móviles que combina métodos de localización relativa y absoluta utilizando filtros de Kalman extendidos (EKF). El sistema puede detectar e identificar fallas comparando las salidas redundantes de los procesos de fusión.



Fig. 1.1: Diagrama de FTC con Filtros de Kalman Extendidos.

Por otro lado, Zhao et. al [18] diseñó una arquitectura FTC que identifica fallos en la percepción, planificación y control del robot de forma independiente. En su etapa de detección de fallos, se emplean filtros de Kalman extendidos para combinar las salidas de los sensores y generar una estimación de la posición del robot [18].

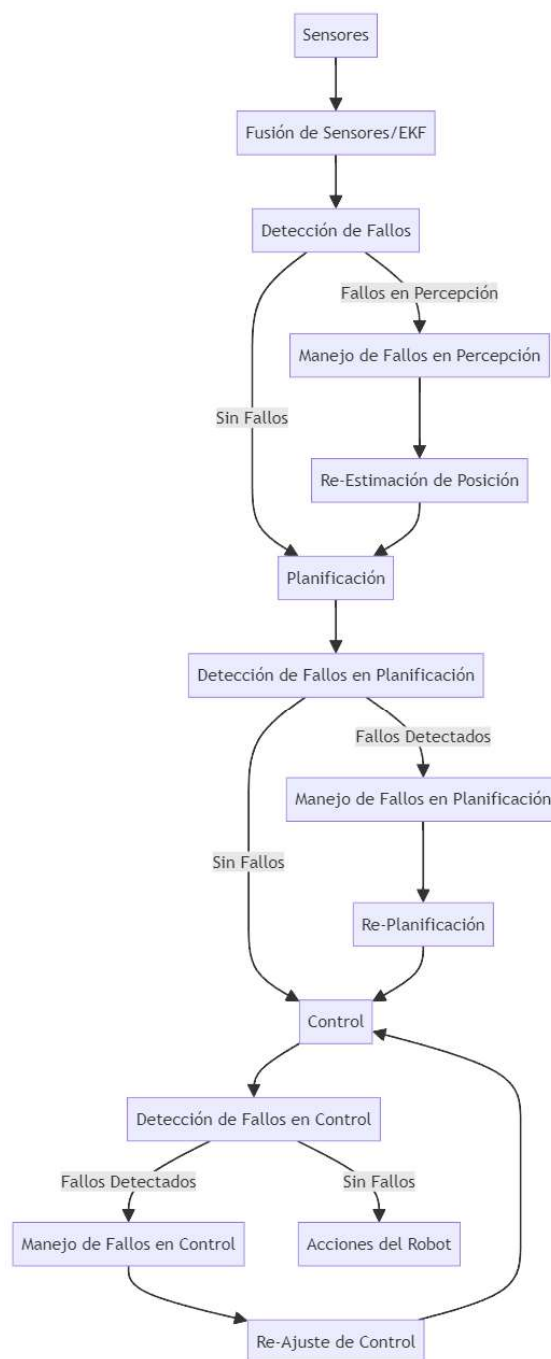


Fig. 1.2: Diagrama de FTC s.

Sun, Yajing y Wei [13] presentaron un enfoque de control tolerante a fallos para un robot móvil no holonómico basado en una técnica de ocultación de fallos y un controlador de modo deslizante. Los resultados de las pruebas realizadas en escenarios simulados de fallos mostraron una mejora significativa en el rendimiento del robot.

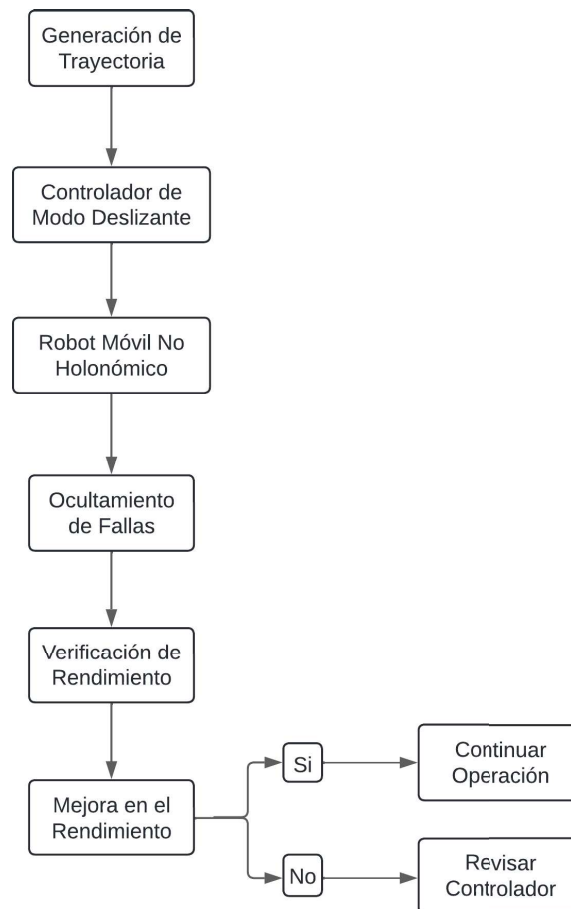


Fig. 1.3: Diagrama de FTC s.

En 'Control Tolerante a Fallas de Actuadores en Robots Móviles' de Parisa Yazdjerdi y Nader Meskin [16], se ofrece un controlador tolerante a fallos diseñado para abordar fallos de actuadores en robots móviles de tracción diferencial. El controlador utiliza el filtro de Kalman extendido para determinar la efectividad del actuador. Una de las contribuciones destacadas es la capacidad de este método para detectar, aislar e identificar fallos de actuadores, mientras que muchos métodos anteriores solo podían detectar o aislar.

En paralelo, analizamos 'Control de seguimiento tolerante a fallos con retroalimentación adaptativa para robots móviles bajo pérdida parcial de efectividad del actuador' de Yeong-Hwa Chang, Chun-I Wu y Cheng-Yuan Yang [7], este

documento presenta una estrategia de control tolerante a fallos con retroalimentación, que se centra en garantizar la operación del robot incluso si algunos actuadores no funcionan completamente. El esquema de control aquí utiliza técnicas de retroceso combinadas con filtros de paso bajo, garantizando estabilidad a través del análisis de estabilidad de Lyapunov.

Además de los enfoques mencionados anteriormente, se han propuesto otros métodos para mejorar la robustez de los robots de tracción diferencial. Ayyıldız, Mustafa y Tilki, Umut [2] propusieron un controlador tolerante a fallas para robots móviles de tracción diferencial utilizando el esquema de modos deslizantes. El controlador está diseñado para manejar una falla de actuador de pérdida de eficacia (LOE), que puede ser modelada como una perturbación de entrada emparejada. El esquema de controlador propuesto se validó experimentalmente en un robot móvil.

Otro acercamiento comúnmente explorado es el uso de control por modos deslizantes (SMC), que ha demostrado ser efectivo en la mitigación de fallos en robots móviles. Sin embargo, el SMC por sí solo no es suficiente para hacer frente a errores de censo o fallas totales del actuador, lo que limita su aplicabilidad en entornos complejos y dinámicos. Para abordar esta limitación, Stancu et al. (2016) [12] propusieron agregar una capa de diagnóstico al sistema que permita detectar, aislar y excluir los sensores que fallan de la estimación del estado del robot.

Lo que no se hace en ninguno de los dos enfoques es un segundo bucle de retroalimentación que compare la salida con un comportamiento conocido del robot, como se ve en algunos otros documentos. Este bucle de retroalimentación puede proporcionar una capa adicional de confirmación sobre el rendimiento del sistema y la reducción de errores. Tal bucle ofrece un mecanismo correctivo para asegurar que el comportamiento del robot se alinee más estrechamente con los resultados esperados. El artículo titulado "Estimación de fallos de actuador y control tolerante a fallos en tres robots móviles 2WD físicamente conectados" de Daniel A. Pereira y Ayad Al-Dujaili [10] es otra contribución destacada en el ámbito del control tolerante a fallos en robots móviles. Este documento aborda la estimación y compensación de fallos de actuadores para tres robots de dos ruedas conectados físicamente. A diferencia de los dos documentos mencionados anteriormente, este documento introduce una redundancia física al conectar robots. Si surge un fallo de actuador en un robot, los demás pueden compensarlo, asegurando que el grupo continúe su trayectoria.

En términos de mecanismos de retroalimentación, este documento utiliza un

controlador cinemático y un controlador dinámico, ampliados con estimaciones de fallos y estados del sistema. Se emplea un observador no lineal para la estimación de fallos. Este enfoque, especialmente el uso de un observador no lineal para la estimación de fallos, introduce otra capa de retroalimentación, comparando indirectamente la salida con un comportamiento conocido del robot. Ayuda a lograr una estimación de estado más precisa, que puede ser esencial para mantener el rendimiento del robot en presencia de fallos.

Los controles tolerantes a fallos de actuadores siguen siendo esenciales para la operación confiable y eficiente de los robots móviles, especialmente en escenarios donde las condiciones ambientales pueden ser impredecibles o severas. Cada documento presenta un enfoque único para manejar estos problemas, y aunque cada uno tiene sus fortalezas, la introducción de un segundo bucle de retroalimentación para comparar el comportamiento del robot con un resultado esperado conocido podría mejorar aún más las capacidades tolerantes a fallos. Esto es de especial interés ya que, a medida que los robots se integran cada vez más en nuestra vida diaria y en aplicaciones críticas, la necesidad de sistemas robustos y tolerantes a fallas seguirá aumentando.

1.6. De Matlab a Python

El desarrollo de este estudio se basó en un trabajo previo que utilizó MATLAB como el software para realizar las simulaciones.

La migración del código de control robusto de MATLAB a Python implicó superar varios desafíos técnicos y conceptuales como la naturaleza del código centrada en la simulación y ecuaciones dinámicas.

Una de las ventajas de MATLAB es que ofrece funciones y herramientas integradas específicamente diseñadas para cálculos matemáticos y simulaciones, como las operaciones matriciales, que son intuitivas y eficientes. Python, aunque posee potentes bibliotecas como NumPy y SciPy, puede requerir más líneas de código para realizar operaciones equivalentes, pero con la ventaja de ser un software que no necesita licencia.

Fue necesario familiarizarse con las bibliotecas de Python equivalentes y adaptar el código para usar estas bibliotecas de manera eficiente. Por ejemplo, utilizar NumPy para operaciones matriciales y SciPy para soluciones numéricas avanzadas.

MATLAB tiene capacidades de visualización y debugging altamente integradas y fáciles de usar. Python, por otro lado, ofrece bibliotecas como Matplotlib para la visualización, para el debugging se pueden utilizar variables temporales o ejecuciones pausadas en IDEs como Visual Studio. Otro problema que se presentó fue la indexación, en MATLAB está basada en 1, mientras que en Python está basada en 0 por lo que todas las matrices y vectores tenían una dimensión extra no deseada. La forma en que MATLAB y Python manejan la precisión numérica puede variar, especialmente en cálculos complejos y operaciones con matrices. Esto llevó a muchos errores de cálculo cuando no se le indicaba explícitamente a Python que tipo de dato se debería usar. Estos errores numéricos fueron remarcablemente notorios en la implementación de integradores numéricos, como el método de Runge-Kutta utilizado en el código inicial.

Las interfaces de software en MATLAB eran prácticamente inexistentes debido a que el código estaba pensado para ejecutarse dentro del mismo entorno de MATLAB. Para el software de Python se debió realizar una investigación de las interfaces necesitaras para poderse comunicar el en ambiente de ROS. Esto se realizó mediante nodos, requiriendo la adición de librerías adicionales como rospy para lograr la comunicación.

La ejecución en tiempo real y la temporización se comporta diferente en Python en comparación con MATLAB, afectando la precisión y la respuesta del sistema de control, lo cual se pudo visualizar el primer día de pruebas de campo. Se requirió hacer un cambio en la definición del tiempo real para bajar de 1000Hz, en los que se estaba programado originalmente, a 20 Hz para poder estar sincronizado con el sistema de visión externo y el controlador del robot.

Capítulo 2

Modelo del robot y de las fallas.

2.1. Modelo

El modelo del robot es una descripción matemática que nos permite entender su comportamiento dinámico y controlarlo adecuadamente. En este caso, se utilizan las ecuaciones de Euler-Lagrange para obtener el modelo dinámico a partir del modelo cinemático del robot. Este enfoque nos permite considerar errores de posicionamiento al seguir una trayectoria, pero no considera fallas en los actuadores [4]. Sin embargo, existe otro enfoque que encapsula los sistemas de Euler-Lagrange, los sistemas Hamiltonianos.

En particular, se tiene interés en la estructura Hamiltoniana controlada por puerto (PHC), cuyo comportamiento está descrito en las ecuaciones (2.1).

$$\begin{aligned} \begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} &= \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix} \begin{bmatrix} \partial_q H^T \\ \partial_p H^T \end{bmatrix} + \begin{bmatrix} 0 \\ G(q) \end{bmatrix} u \\ y &= G^T(q) \partial_p H^T \end{aligned} \quad (2.1)$$

En la PHC, q son las coordenadas generalizadas y p los impulsos generalizados. Ambos están descritas en (2.2).

$$q = \begin{bmatrix} q_{Bx} \\ q_{By} \\ \theta \end{bmatrix}, p = \begin{bmatrix} p_{Bx} \\ p_{By} \\ p_\theta \end{bmatrix} \quad (2.2)$$

La entrada u representa la fuerza lineal y el torque aplicado al robot como se muestra en (2.3).

$$u = \begin{bmatrix} u_f \\ u_\theta \end{bmatrix} \quad (2.3)$$

La matriz de entrada $G(q)$ está definida en (2.4).

$$G(q) = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \quad (2.4)$$

Las restricciones en el movimiento están dadas por la matriz S , estas son de tipo no-holonómicas y nos dicen que el robot debe girar para ubicarse en el punto deseado.

$$S(q) = \begin{bmatrix} \cos \theta & -d_{AB} \sin \theta \\ \sin \theta & -d_{AB} \cos \theta \\ 0 & 1 \end{bmatrix} \quad (2.5)$$

La función de energía del robot está dada por $H = \frac{1}{2}p^T M^{-1}p$, donde la matriz de inercia es $M = \text{diag}(m, m, I_{cm})$ con I_{cm} siendo la inercia del centro de masa y m la masa total del robot. Con este modelo, es posible desarrollar un control adecuado para el robot y lograr el seguimiento de trayectorias y tareas específicas. Cabe destacar que este modelo es una aproximación del comportamiento real del robot, por lo que es importante validar el control y ajustar el modelo si es necesario. La dinámica del robot móvil en el espacio de estados con restricciones de dirección de movimiento es:

$$\begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} 0 & S(q) \\ -S^T(q) & \bar{C}(\bar{p}) - \bar{D} \end{bmatrix} \begin{bmatrix} \partial_q \bar{H}^T \\ \partial_{\bar{p}} \bar{H}^T \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} u \quad (2.6)$$

$$y = \partial_{\bar{p}} \bar{H}^T$$

Con una función de energía $\bar{H} = \frac{1}{2}\bar{p}^T \bar{M}^{-1}\bar{p}$ y una matriz de inercia $\bar{M} = \text{diag}(M, I_{cm})$ El centro de masa no se encuentra en el eje de las ruedas, la distancia entre estos dos se denomina $d_a b$ y la matriz de fuerzas centrífugas y de coriolis evaluada en las trayectorias deseadas $(q_d, \dot{q}_d)$ está definida en (2.7).

$$\bar{C}(\bar{p}) = \begin{bmatrix} 0 & \frac{m d_{ab}}{m d_{AB}^2 + I_{cm}} \bar{p}_\theta \\ -\frac{m d_{ab}}{m d_{AB}^2 + I_{cm}} \bar{p}_\theta & 0 \end{bmatrix} \quad (2.7)$$

El controlador robusto para este sistema es el que se muestra en (2.8).

$$\begin{aligned}
u = & -\bar{C}\bar{M}^{-1}\bar{p} + \bar{M}\dot{S}^{-1}\dot{q}_d + \bar{M}\dot{S}^{-1}\ddot{q}_d \\
& -(\bar{M}^{-1} + K_d)S^TK_p(q - q_d) \\
& -(K_d + K_p\bar{M}^{-1})(\bar{p} - \bar{p}_d) + K_i\dot{z} + (I_T + K_d)K_i z
\end{aligned} \tag{2.8}$$

donde $\dot{z}$ está definido por (2.9).

$$\dot{z} = -(\bar{M}^{-1} + I_T)S^TK_p(q - q_d) - (\bar{p} - \bar{p}_d) \tag{2.9}$$

Las ganancias utilizadas fueron:

1. $K_p = 1.2 * I$
2. $K_d = 0.5 * I$
3. $K_i = 4 * I$

Las inercias fueron determinadas por medio de (2.10).

$$I_w^b = \begin{bmatrix} I_{xx}^b & 0 & 0 \\ 0 & I_{yy}^b & 0 \\ 0 & 0 & I_{zz}^b \end{bmatrix} = \begin{bmatrix} \frac{m_w}{12}(3r^2 + g^2) & 0 & 0 \\ 0 & \frac{m_w r^2}{12} & 0 \\ 0 & 0 & \frac{m_w}{12}(3r^2 + g^2) \end{bmatrix} \tag{2.10}$$

$$I_{zz}^b = \frac{m_w}{12}(3r^2 + g^2) \tag{2.11}$$

$$I_B = 0.5 * m_b * b^2 \tag{2.12}$$

$$m_T = m_B + 2m_w \tag{2.13}$$

$$I_T = I_B + m_B d^2 + 2m_w b^2 + 2I_{zz}^b \tag{2.14}$$

2.1.1. Cálculo del centro de masa

El cálculo del centro de masa para un objeto irregular como el robot diferencial Turtlebot 3 Waffle Pie requiere un análisis detallado de la distribución de peso en su estructura. Afortunadamente, debido a la filosofía de hardware abierto de Turtlebot3, se dispone de información detallada de los ensambles en la web. Sin embargo, estos ensambles no especifican el tipo de material de cada pieza, lo que resulta necesario para determinar su peso específico.

Para superar este desafío, se realizó un trabajo de asignación manual del material a cada componente de la estructura del robot. Notablemente, el modelo del robot no incluye elementos esenciales como las baterías o las tarjetas controladoras. Para abordar esto, se decidió distribuir el peso de estos componentes de manera equitativa en los soportes disponibles para alojarlos. De tal forma que el objeto como tal no se agregó al modelo 3D, solo se ejercieron fuerzas relacionadas al peso del componente. Esto ahorró el tiempo de modelarlos.

Con la ayuda de un software de diseño mecánico, se pudieron obtener las coordenadas del centro de masa del robot. Utilizando el centro de la base del robot como referencia, y considerando la dirección positiva de x hacia el frente y la dirección positiva de z hacia arriba, se encontró que el centro de masa está en $x = 0.02$ m y $z = 0.05$ m.

Aquí, parece haber una discrepancia en las ecuaciones proporcionadas y los valores reportados para las coordenadas del centro de masa. La ecuación proporcionada para calcular la coordenada x del centro de masa es:

$$x = \frac{h_1 L_2 + h_2 L_1 + h_3 L_2 + h_4 L_1}{2(h_1 + h_2 + h_3 + h_4)} \quad (2.15)$$

Y al reemplazar los valores medidos, se obtiene:

$$x = \frac{(0.0405)(0.1125) + (0.0395)(0.1125) + (0.0405)(0.1125) + (0.0395)(0.1125)}{2(0.0405 + 0.0395 + 0.0405 + 0.0395)} = 0.1125m \quad (2.16)$$

Esto indicaría que el centro de masa del Turtlebot3 Waffle Pi se encuentra a una distancia de 0.1125 m desde el borde posterior del robot, lo cual trasladado a nuestro punto de referencia que es el centro geométrico del robot sería -0.02m en x. Esta variación hacia el frente del centro de masa se debe a la concentración de peso de los motores y de la tarjeta de control. En cuanto al Eje z, la estimación geométrica del centro de masa sería de 0.075m desde la base lo que también difiere de los

0.05m calculados debido a la concentración de masa de los mismos componentes que afectaron el eje x. El eje y se considera simétrico por lo que se puede utilizar el centro geométrico como centro de masa a 0.1625m de los bordes laterales del robot.

La diferencia entre estos dos puntos es de 0.0471m lo cual representa una cantidad significativa al momento de utilizar el centro de masa para el cálculo de las ecuaciones Hamiltonianas descritas en REF

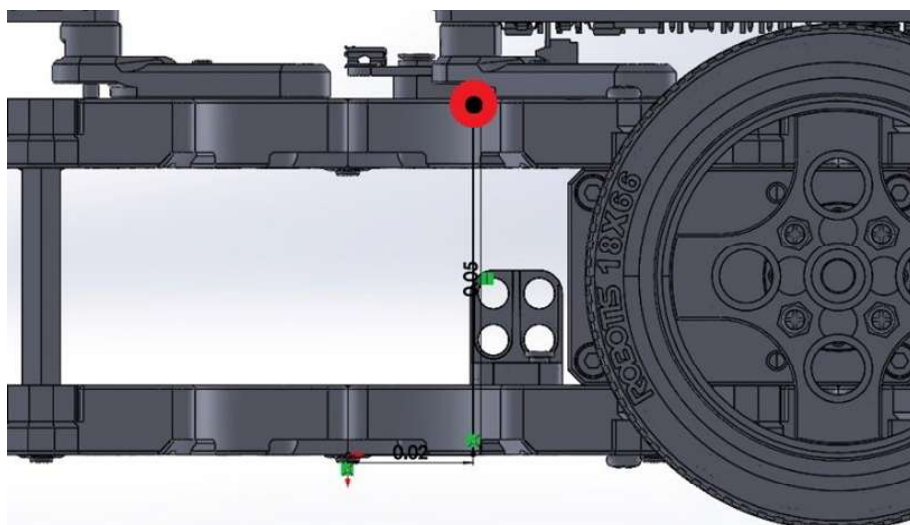


Fig. 2.1: Ubicación del centro de masa.

La localización precisa del centro de masa proporciona información valiosa para el control y la estabilidad del Turtlebot 3 Waffle Pie durante su operación. Este es un aspecto crítico del diseño y análisis de robots, ya que el centro de masa afecta a la dinámica del movimiento del robot, su estabilidad y su capacidad para realizar ciertas tareas.

2.2. Trayectoria

La trayectoria deseada se genera utilizando una distribución uniforme de puntos continuos. Esta distribución se genera en python, el programa puede ser modificado fácilmente para agregar líneas rectas o curvas en la trayectoria. El algoritmo que genera la trayectoria además utiliza la separación entre los puntos para dar la velocidad y aceleración necesarias durante el seguimiento, esta velocidad y aceleración sirven de referencia para generar el comportamiento nominal además de los estados deseados mencionados en(2.2) que estará siguiendo el centro de masa.

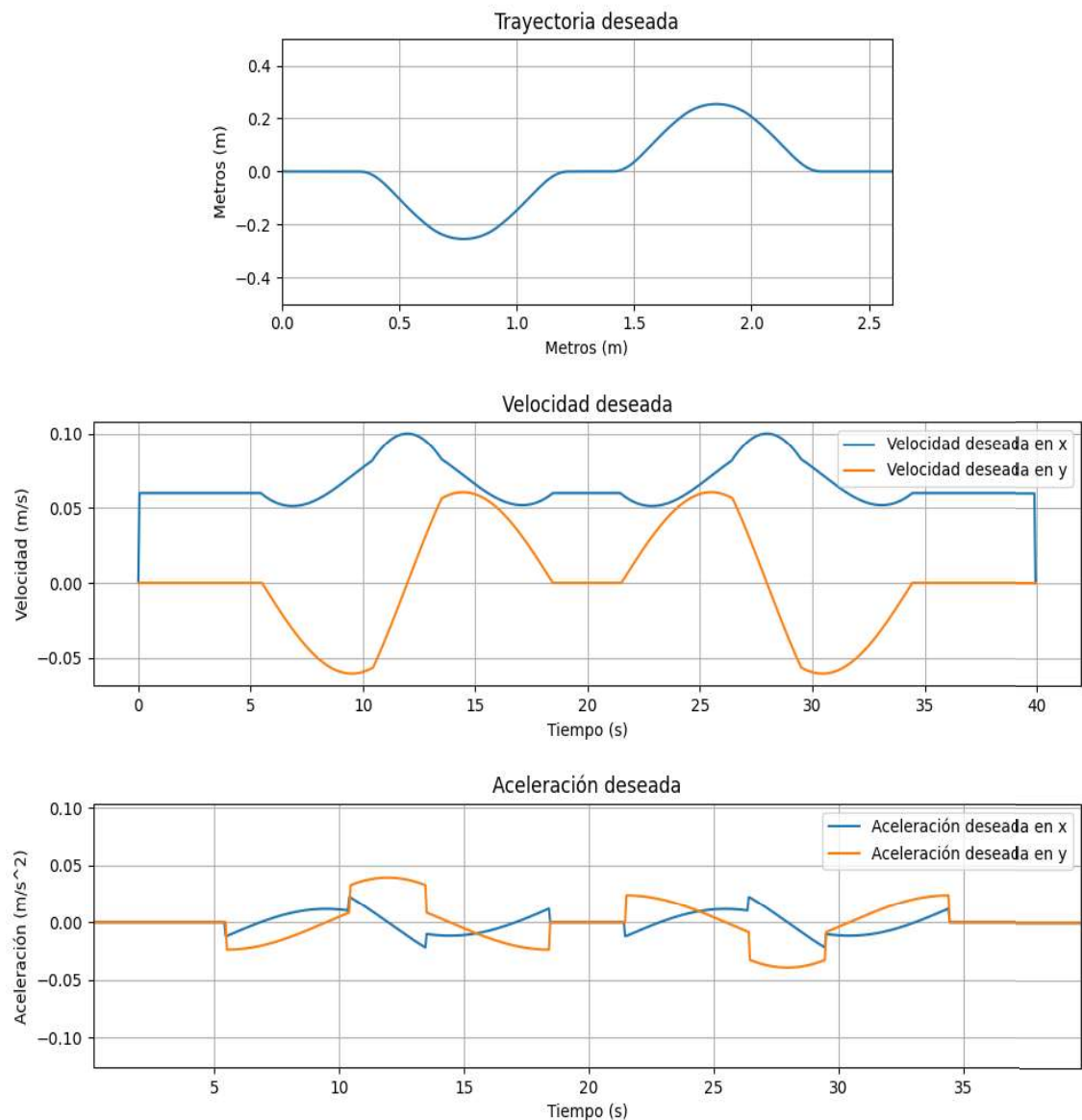


Fig. 2.2: Trayectoria, velocidades y aceleraciones deseadas.

Cabe remarcar que esta etapa se realiza previa a al seguimiento de trayectoria real, pero en trabajos futuros se podría utilizar la misma lógica para calcular una trayectoria en tiempo real y alimentarla al modelo del robot para determinar un comportamiento nominal reconfigurable.

2.3. Inducción de fallas

La falla de actuador fue inducida en la respuesta de los actuadores a las entradas del control, limitando la capacidad de los mismos. Al ser una falla multiplicativa, se limita la velocidad máxima de giro de las ruedas, las cuales son componentes de las velocidades angular y lineal del robot diferencial, como se muestra en el diagrama de la figura 2.3.

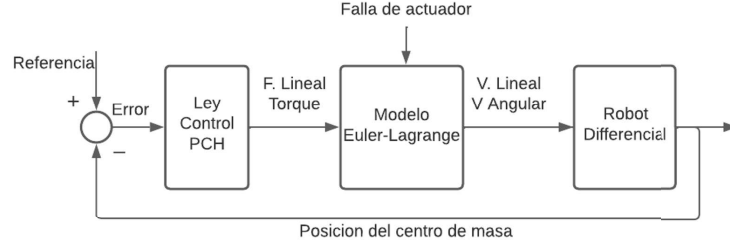


Fig. 2.3: Diagrama de bloques

Se probarán fallas de pérdida de potencia en los motores. Esto significa una disminución de la velocidad real con la solicitada por el control. Estas pérdidas serán de una magnitud de un 5 % hasta un 50 %, a partir de un tiempo determinado después del inicio del seguimiento de trayectoria, es decir, la falla no será parte de las condiciones iniciales de operación.

Para inducir la falla de actuador, es necesario tomar el cambio de posición en x y y del sistema ($\dot{q}_{Bx}, \dot{q}_{By}$) y convertirlas a sus componentes de rueda izquierda y rueda derecha. Primero hay que transformar $\dot{q}_{Bx}, \dot{q}_{By}$ a sus equivalentes de velocidad lineal y angular (V_B, ω_B) como se muestra en (2.18). Posteriormente se transforman V_B, ω_B en las velocidades independientes de cada rueda. Estas transformaciones serán utilizadas constantemente ya que tanto el modelo del robot como el sistema de visión para retroalimentación nos brindan los cambios de posición del sistema en $\dot{q}_{Bx}, \dot{q}_{By}$

$$\begin{bmatrix} V_B \\ 0 \\ \omega_B \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \dot{q}_{Bx} \\ \dot{q}_{By} \\ \dot{\theta} \end{bmatrix} \quad (2.17)$$

$$\begin{bmatrix} \dot{\varphi}_R \\ \dot{\varphi}_L \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2b} & \frac{-1}{2b} \end{bmatrix}^{-1} \begin{bmatrix} V_B \\ \omega_B \end{bmatrix} \quad (2.18)$$

En la ecuación de transformación de velocidades lineal y angular a las velocidades de cada rueda(2.18), $2b$ es la distancia entre las dos ruedas y $\dot{\varphi}_{R,L}$ velocidad de desplazamiento de cada rueda. Estas velocidades independientes de las ruedas, se multiplican por un factor de falla mayor a 0 y menor que 1. Estos factores están definidos de la siguiente forma.

- F_{md} : falla motor derecho
- F_{mi} : falla motor izquierdo

Las velocidades con la falla inducida se transforman a las velocidades lineal y angular utilizando (2.19), donde V_{fB} y ω_{fB} son las velocidades lineal y angular con falla inducida.

$$\begin{bmatrix} V_{fB} \\ \omega_{fB} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2b} & \frac{-1}{2b} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_R F_{md} \\ \dot{\varphi}_L F_{mi} \end{bmatrix} \quad (2.19)$$

Para la siguiente simulación de ejemplo se implementará una falla del 40% en el actuador derecho, la cual limitará su velocidad de desplazamiento. Este efecto de la falla sobre el actuador puede verse en la comparativa de la velocidad nominal y la de falla inducida en las figuras (2.3) y (2.3) respectivamente.

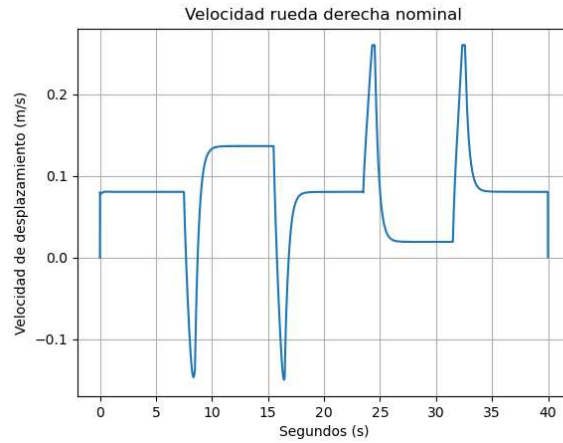


Fig. 2.4: Velocidad nominal de la rueda derecha con la trayectoria seleccionada.

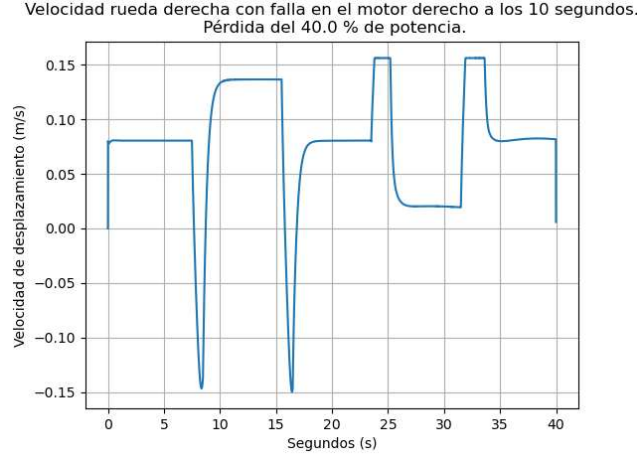


Fig. 2.5: Velocidad con falla inducida de la rueda derecha con la trayectoria seleccionada.

Se puede apreciar que en desempeño nominal, la rueda llega a una velocidad máxima de $2.6m/s$ mientras que con la falla inducida del 40 % esta llega a un máximo de $1.56m/s$. Los efectos de estas limitaciones se verán en la sección de resultados.

Para lograr estas transformaciones en el robot se utilizará un nodo en python que hace uso de distintas librerías para operaciones matriciales. Es importante tomar en cuenta el tiempo de ejecución del sistema, por ello las operaciones son realizadas en la computadora de mando del sistema y no la Raspberry que está en el robot. La Raspberry solo se encarga de recibir los comandos de velocidad linear y angular y transmitirlo a los actuadores del Turtlebot.

2.4. Detección de fallas

La detección de fallas es una parte integral del mantenimiento y operación de un robot diferencial. Este procedimiento implica la monitorización continua de la velocidad nominal (la velocidad esperada o establecida) y la comparación con la velocidad registrada por los encoders del robot. Los encoders son dispositivos electrónicos que convierten el movimiento o la posición de la rueda en una señal digital. En el caso de un robot diferencial, se suele tener un encoder para cada rueda.

La ecuación (2.18) proporciona las velocidades individuales esperadas de cada rueda ($\dot{\varphi}_{Lnom}$ y $\dot{\varphi}_{Rnom}$ para la rueda izquierda y la rueda derecha, respectivamente) bajo

condiciones normales de operación. Estas velocidades son calculadas a partir de los comandos de control enviados al robot y dependen de la estructura y la configuración específicas del sistema de control del robot.

Se define entonces una medida de fallo ($Failure_L$ y $Failure_R$ para la rueda izquierda y derecha, respectivamente) como la relación entre la velocidad registrada por los encoders ($\dot{\varphi}_{Lencoder}$ y $\dot{\varphi}_{Rencoder}$) y la velocidad nominal. Esta relación se calcula por separado para cada rueda. Si las condiciones de funcionamiento son normales y no hay fallos, esta relación debería ser cercana a uno. Sin embargo, si hay un fallo en el sistema, como un motor atascado, un encoder defectuoso o un deslizamiento de las ruedas, la velocidad registrada por los encoders se desviará de la velocidad nominal, y la medida de fallo aumentará o disminuirá significativamente.

La detección de fallos se realiza entonces analizando las medidas de fallo. Si la medida de fallo es significativamente mayor o menor que uno, se puede inferir que ha ocurrido un fallo en el sistema.

La ecuación que define las medidas de fallo es:

$$\begin{bmatrix} F_L \\ F_R \end{bmatrix} = \begin{bmatrix} \dot{\varphi}_L \\ \dot{\varphi}_R \end{bmatrix} \begin{bmatrix} \frac{1}{\dot{\varphi}_{Lnom}} & 0 \\ 0 & \frac{1}{\dot{\varphi}_{Rnom}} \end{bmatrix} \quad (2.20)$$

Donde:

- F_L y F_R son las medidas de fallo de las ruedas izquierda y derecha, respectivamente.
- $\dot{\varphi}_L$ y $\dot{\varphi}_R$ son las velocidades registradas por los encoders de las ruedas izquierda y derecha, respectivamente.
- $\dot{\varphi}_{Lnom}$ y $\dot{\varphi}_{Rnom}$ son las velocidades nominales de las ruedas izquierda y derecha, respectivamente.

2.5. Control Suplementario

Una vez detectadas las fallas, es esencial responder de manera adecuada para garantizar el funcionamiento continuo del robot y prevenir daños adicionales. Una estrategia para manejar las fallas detectadas es implementar un control suplementario.

2.5.1. Diagrama de bloques

El diagrama de bloques final del sistema de control, incluido el nuevo lazo de retroalimentación suplementario, se presenta en la Figura 2.6. La salida nominal

de este sistema se calcula antes de la simulación, llevando a cabo una ejecución en la que no se programan fallas.

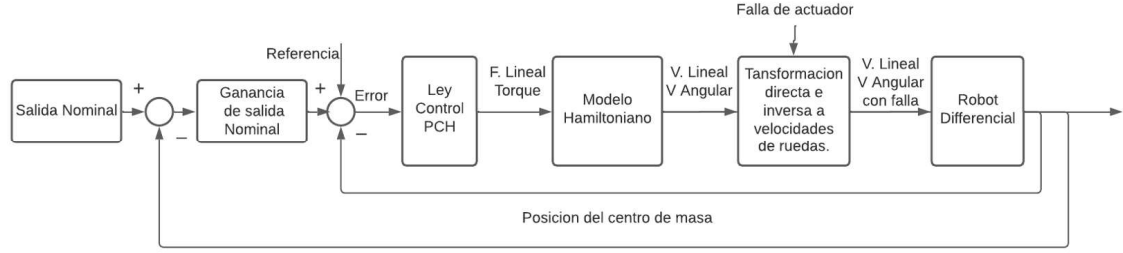


Fig. 2.6: Diagrama de bloques con control suplementario.

2.5.2. Efectos en el control nominal

El controlador suplementario modifica externamente el control nominal agregando un nuevo término, $K_{sup}q_{nom}$, a las ecuaciones del controlador nominal, tal como se muestra en las Ecuaciones (2.21) y (2.22). Este nuevo término es la salida nominal del sistema multiplicada por una ganancia del control suplementario, K_{sup} . Esta ganancia se sintonizó manualmente para encontrar un valor que atenuara el error máximo sin sacrificar el error final.

El control suplementario se puede expresar de la siguiente manera:

$$u_{sup} = -\bar{C}\bar{M}^{-1}\bar{p} + \bar{M}\dot{S}^{-1}\dot{q}_d + \bar{M}\dot{S}^{-1}\ddot{q}_d - (\bar{M}^{-1} + K_d)S^T K_p (q - q_d - K_{sup}(q_{nom} - q_{real})) \quad (2.21)$$

$$-(K_d + K_p\bar{M}^{-1})(\bar{p} - \bar{p}_d) + K_i\dot{z} + (I_T + K_d)K_i z$$

Donde $\dot{z}_{sup}$ se define por la Ecuación (2.22),

$$\dot{z} = -(\bar{M}^{-1} + I_T)S^T K_p (q - q_d - K_{sup}(q_{nom} - q_{real})) - (\bar{p} - \bar{p}_d) \quad (2.22)$$

Las ganancias utilizadas fueron:

1. $K_p = 1.2 * I$
2. $K_d = 0.5 * I$

$$3. K_i = 4 * I$$

$$4. K_{sup} = 1.5 * I$$

En estas ecuaciones, I representa la matriz identidad y $\bar{M}$, $\bar{p}$, y $\bar{C}$ representan la matriz de inercia, el vector de fuerzas generalizadas y la matriz de amortiguamiento del robot, respectivamente. S representa la matriz de selección, q , q_d , $\dot{q}_d$, y $\ddot{q}_d$ representan la posición actual, la posición deseada, la velocidad deseada y la aceleración deseada, respectivamente.

Capítulo 3

Implementación

3.1. Plataforma de Implementación

Un ejemplo representativo de un robot de tracción diferencial es el Turtlebot3. Se eligió esta plataforma debido a la disponibilidad y su naturaleza de origen educativa. Este modelo se encuentra disponible para imprimir y armar de manera gratuita. El hardware que utiliza también es de código abierto de manera que se puede aprovechar para darle la configuración deseada a libertad. Sus especificaciones son proporcionadas por [11] y se pueden apreciar en la Tabla 3.1.

Tabla 3.1: Parámetros del Turtlebot3

Parámetro	Valor
Velocidad de traslación máxima	0.26 m/s
Velocidad de rotación máxima	1.82 rad/s
Torque	3.7 N.m @14.8V,2.7A
Distancia $2b$	0.287 m
Distancia d_{AB}	0.0265 m
Radio de las ruedas	0.033 m
Masa del cuerpo	1.64 kg
Masa de las ruedas	0.08 kg

Para la implementación del sistema de simulación y control del robot, se empleó Ubuntu 20.04, un sistema operativo basado en Linux. Debido a las limitaciones en el rendimiento que pueden presentar las máquinas virtuales, se optó por realizar

una instalación autónoma de Ubuntu mediante la partición del disco duro. En este entorno, se desarrollaron los algoritmos de control y la estructura requerida para integrar estos algoritmos en la simulación y la implementación real del sistema robótico.

El desarrollo se llevó a cabo íntegramente utilizando el Robot Operating System (ROS), una plataforma que facilita la programación modular mediante la comunicación de nodos, cada uno de los cuales tiene una función específica. Los nodos intercambian información entre ellos mediante el envío de mensajes. Además, el simulador de ROS ya incluye un modelo del robot que se utilizó en este proyecto, lo que agilizó considerablemente el proceso de desarrollo. Este modelo puede recibir y procesar instrucciones de velocidad lineal y angular (V y ω , respectivamente), que son las mismas que se utilizarán para controlar el robot en el mundo real.

Para instalar ROS en Linux, se requiere trabajar en la consola. Aunque el proceso de instalación puede generar una gran cantidad de texto, no se incluyen capturas de pantalla de la instalación en este documento. En lugar de ello, se documenta la versión de la distribución de ROS utilizada (ROS NOETIC). Catkin es el sistema de compilación y gestión de paquetes utilizado en ROS. Dentro de este entorno, se creó el modelo del robot y se programaron sus funciones. Catkin proporciona un entorno de programación modular y permite alternar entre C++ y Python según sea necesario. ROS Gazebo es una herramienta que proporciona una interfaz entre ROS y el simulador de robótica de código abierto Gazebo. Esta interfaz facilita la creación de un entorno de simulación tridimensional realista, lo que es esencial para el desarrollo y prueba de algoritmos de control, planificación y percepción.

A continuación se presenta una representación gráfica de la comunicación entre cada nodo.

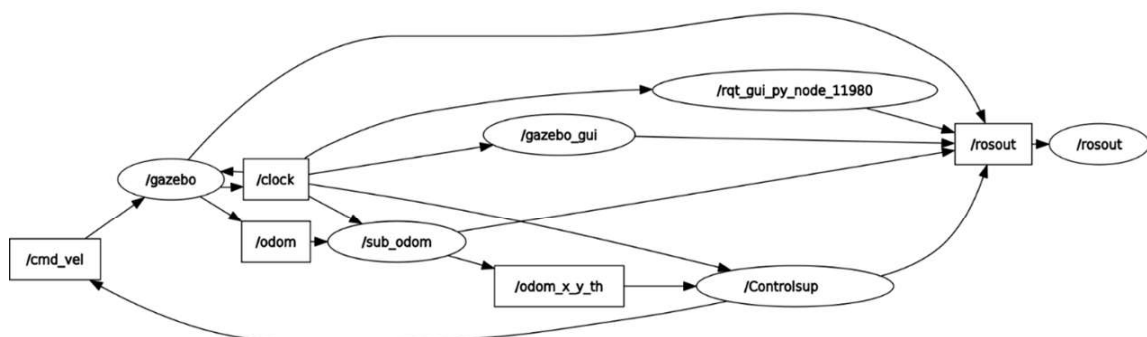


Fig. 3.1: Diagrama de comunicación entre los nodos de ROS.

Los nodos que se describen a continuación están presentes en este gráfico:

- `/cmd_vel`: Este nodo es responsable de recibir las instrucciones de velocidad del controlador y transmitir las al modelo del robot para su ejecución.
- `/gazebo`: Este nodo es la representación del simulador Gazebo, que recibe las instrucciones del nodo `/cmd_vel` y las aplica al modelo del robot.
- `/clock`: Este nodo proporciona información sobre el tiempo, esencial para sincronizar los eventos en el sistema.
- `/odom`: Este nodo proporciona información de odometría, es decir, estima la posición del robot basándose en el movimiento de sus ruedas.
- `/sub_odom`: Este nodo es el suscriptor de la información proporcionada por `/odom`.
- `/odom_x_y_th`: Este nodo recibe la información de odometría y la procesa para su uso en los cálculos de control.
- `/gazebo_gui`: Este nodo representa la interfaz gráfica de usuario de Gazebo, que proporciona una visualización en tiempo real del comportamiento del robot.
- `/rqt_gui_py_node_11980`: Este nodo es responsable de la interfaz gráfica de usuario de ROS, que permite monitorear y controlar los nodos en tiempo real.
- `/rosout`: Este nodo proporciona un registro de los mensajes de diagnóstico generados por los demás nodos.

3.2. Sistema de retroalimentación

3.2.1. Sistema de visión para odometría

La odometría es el proceso mediante el cual se estima la posición del robot basándose en el movimiento de sus ruedas. Este proceso se realiza siguiendo los métodos presentados en el libro "Modern Robotics" de Frank C. Park y Kevin M. Lynch. Para optimizar la precisión de este proceso, se utilizó un sistema de visión y captura de movimiento VICON. Este sistema emplea cámaras de alta resolución y marcadores reflectantes para registrar la posición y orientación de objetos y personas en tiempo

real. Debido a su alta precisión y versatilidad, VICON es ampliamente utilizado en campos como la animación, el cine, la medicina y la investigación biomecánica.

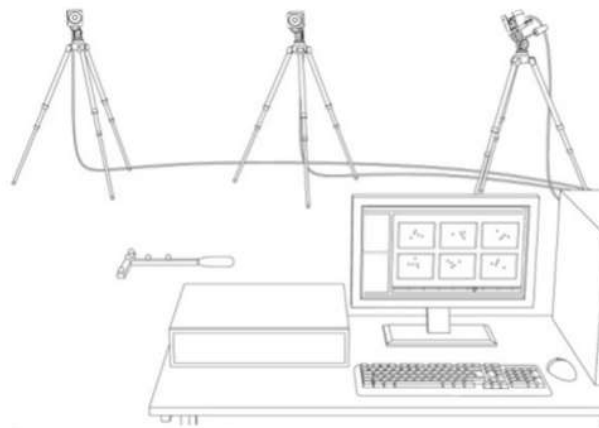


Fig. 3.2: Sistema de visión VICON.

El sistema VICON proporciona datos de posición y orientación del robot en tiempo real, lo que mejora la precisión de la odometría y, en última instancia, la eficacia del sistema de control.

3.2.2. Implementación y mejora de los sensores para la detección de fallas

Los servomotores Dynamixel de la serie X desempeñan un papel crucial en el sistema robótico desarrollado en este proyecto. Estos motores cuentan con un codificador absoluto sin contacto que proporciona mediciones precisas de la posición, velocidad y torque del motor. Este tipo de información es esencial para el funcionamiento del sistema y también puede ayudar a detectar y manejar las fallas de manera eficaz.



Fig. 3.3: Servomotores Dynamixel.

3.2.3. Integración y mejoras en el sistema

La integración exitosa de los sistemas necesarios para controlar el robot depende de la combinación de varias tecnologías. Primero, tenemos nuestro hardware principal, el TurtleBot Waffle Pi3, que proporciona una base móvil robusta y versátil para nuestra implementación. Este se complementa con un sistema de visión VICON de alta precisión, que rastrea la posición y orientación del robot en tiempo real y proporciona esta información a nuestro algoritmo de control.

Nuestra elección de Ubuntu y ROS como plataforma de software facilita la integración de estos componentes y nos permite controlar el TurtleBot, ejecutar nuestros algoritmos de control, y recibir y procesar los datos de VICON de una manera eficiente y organizada.

Por otro lado, hemos realizado diversas mejoras para optimizar la integración y rendimiento de nuestro sistema. Por ejemplo, hemos desarrollado una serie de nodos ROS personalizados para manejar la comunicación con el sistema VICON y para ejecutar nuestros algoritmos de control. También hemos implementado un sistema de supervisión y diagnóstico para monitorear el estado de los componentes del sistema y detectar posibles fallas.



Fig. 3.4: Sistema integrado para el Turtlebot3 WafflePi.

3.3. Refinamiento del cálculo de ganancias

En las primeras etapas de este proyecto, trabajamos con simulaciones que asumían una frecuencia de muestreo de 1000 Hz, y calculamos las ganancias de nuestro controlador PID en base a esta frecuencia. Sin embargo, en el proceso de implementación del sistema, nos dimos cuenta de que trabajar con una frecuencia de muestreo tan alta en un sistema real puede ser innecesario y complicado.

Por lo tanto, decidimos bajar la frecuencia de muestreo a 30 Hz. Para ajustar las ganancias de nuestro controlador PID a esta nueva frecuencia, empleamos el método de Ziegler-Nichols. Este es un método ampliamente utilizado para afinar las ganancias de los controladores PID, basado en la respuesta en lazo abierto del sistema.

Aplicando las formulas de transformación para las ganancias con la nueva frecuencia de muestreo, obtenemos:

$$\begin{aligned} Kp' &= 1.2 \times \frac{0.0333}{0.001} = 39.96 \\ Ki' &= \frac{0.5}{0.0333 \times 0.0333} = 450.45 \\ Kd' &= 4 \times 0.0333 = 0.1332 \end{aligned}$$

De este modo, las nuevas ganancias del controlador para una frecuencia de muestreo de 30 Hz serían $Kp' = 39.96$, $Ki' = 450.45$, y $Kd' = 0.1332$. Este cambio nos permitió implementar el controlador de manera más eficiente y precisa en nuestro sistema real, optimizando el rendimiento general y la capacidad de respuesta del robot.

Capítulo 4

Resultados

4.1. Resultados Simulaciones

En las simulaciones se tomaron en cuenta las restricciones físicas del robot para el movimiento, tanto geométricas como de potencia. La falla fue inducida por medio de software como se describe en capítulos anteriores. La trayectoria utilizada también es la mencionada anteriormente. A continuación se muestra la gráfica de error con una pérdida de potencia del 40 por ciento en el segundo 10 de la simulación.

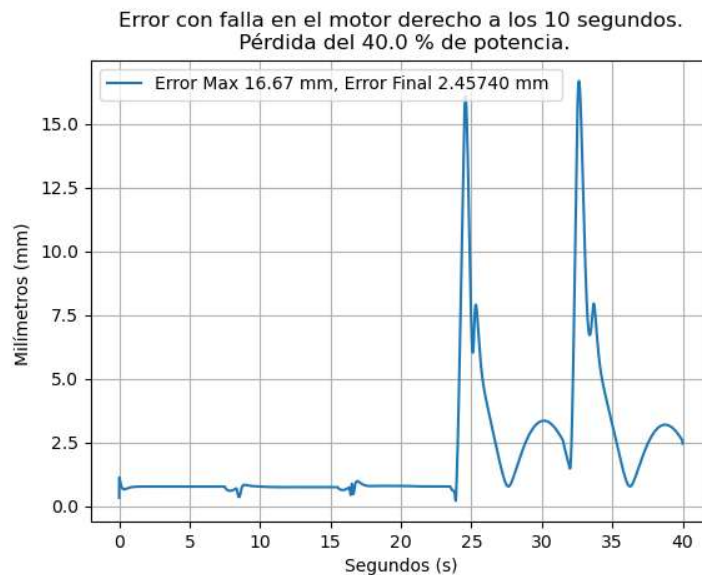


Fig. 4.1: Error con falla.

Se aprecia un error constante debido a la diferencia del centro de masa con el centro

del eje del robot. Además de dos picos de error que suceden durante las vueltas. La Reducción del error con en control suplementario es notoria, en este caso se implementó una falla del 40 % de pérdida de velocidad. La comparación se puede apreciar en las figuras (4.1) y (4.2).

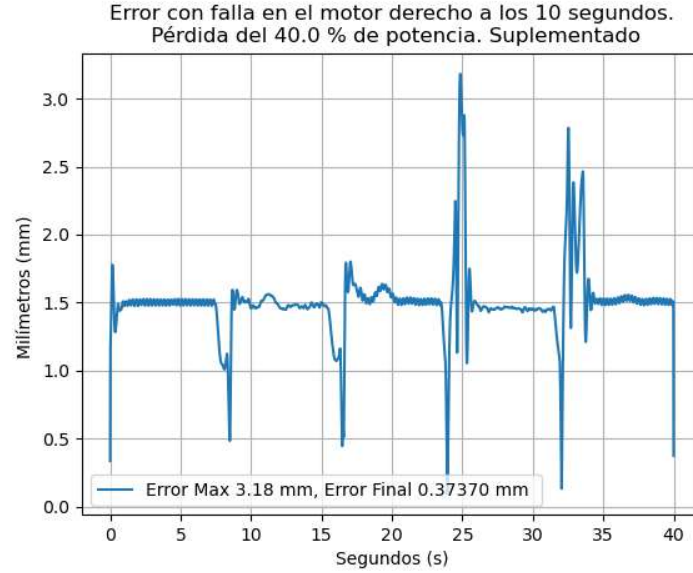


Fig. 4.2: Error con falla y control suplementario.

Se ejecutaron las simulaciones con las condiciones de falla propuestas anteriormente. Se obtuvo el error máximo y el final en todos los escenarios. A continuación se muestra la tabla 4.3 con los resultados de error máximo y final en milímetros.

Tabla 4.1: Resultados de error en mm
Magnitud de falla Error Max Error Final

0 %	1.13	0.356
5 %	2.15	0.3568
10 %	3.72	0.442
20 %	7.48	0.929
30 %	11.86	1.636
40 %	16.67	2.457

Considerando una banda de $\pm 6mm$ alrededor de la trayectoria nominal (referencia), entonces podemos decir que fallas hasta de 15 % pueden ser toleradas por el control

nominal (2.8). Por lo tanto fallas mayores a 15% no son toleradas por el control robusto utilizado. Si se desea mantener un correcto seguimiento de la trayectoria de referencia, será necesario implementar un control tolerante a fallas, es decir, que sea capaz de detectar, aislar y reconfigurar el controlador de acuerdo a la falla presentada. Cabe tomar en cuenta que en las simulaciones se consideró una superficie lisa y no se simuló deslizamientos de ruedas.

4.1.1. Resultados Implementación

4.1.1.1. Consideraciones

Para los resultados se mostrarán gráficos del error en milímetros (mm) a lo largo del tiempo en segundos (s), este representa la desviación de una trayectoria deseada. Además se muestran los picos de error señalando el momento en el que el error fue máximo. La falla en todos los experimentos fue inducida a los 12 segundos, esta se indica con una línea punteada roja en todas las gráficas. Otra restricción importante es el tiempo, el seguimiento de trayectoria dura lo mismo en todos los casos, por lo que este método está pensado para apegarse lo más posible a la trayectoria en un tiempo finito.

Para todos los casos existe un periodo de estabilización después de que el control suplementario entra en funcionamiento, sin embargo, hay picos de error menores persistentes, estos se deben a irregularidades en el terreno y deslizamiento sobre el mismo. El área en la que se realizaron las pruebas está pensada para el seguimiento de drones, por lo que el piso no estaba preparado para robots móviles.

En todas las gráficas se indica el error final, esto es la distancia absoluta del destino ideal del robot. En la descripción se indica la potencia máxima del actuador, esto es el recíproco de la magnitud de la falla. Existe un error en estado estacionario debido a la colocación de los puntos de seguimiento, esto se hizo de forma manual y da lugar a error humano.

En futuros trabajos se puede plantear el recalibrar el seguimiento del centro de masa al inicio de cada prueba para eliminarlo. Se pensó en corregir la trayectoria de manera numérica, pero esto sería editar los datos y se optó por presentarlos tal y como fueron adquiridos. Por lo tanto este método asume que el error es constante a lo largo de la trayectoria. Un enfoque más completo podría involucrar considerar efectos dinámicos, ajustes de control de retroalimentación y modelado del sistema.

4.1.1.2. Falla del 20 % en el motor derecho

Los resultados de inducir una falla del 20 % fueron los siguientes:

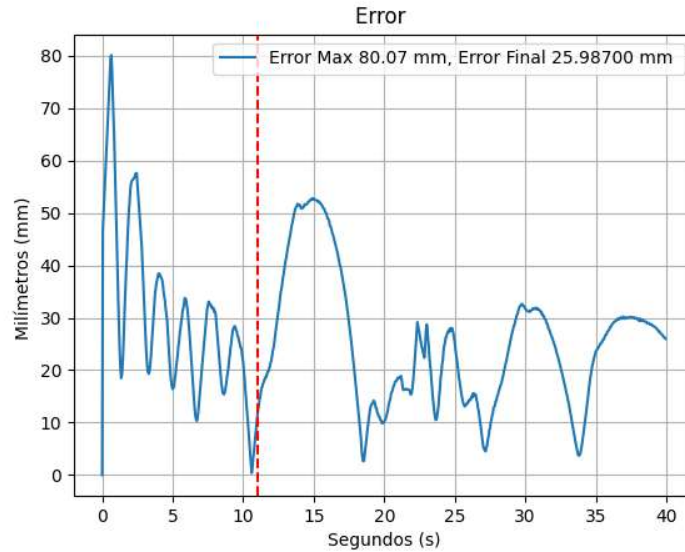


Fig. 4.3: Error con el motor derecho al 80 % de eficiencia sin control suplementario

Se puede apreciar un pico de error al inicio que se debe al posicionamiento del robot y un intento inicial de corrección debido al control básico. Al momento de inducir la falla(12s) , se dispara el error. Este va disminuyendo con el tiempo pero permanece en el rango de 5 mm a 27 mm.

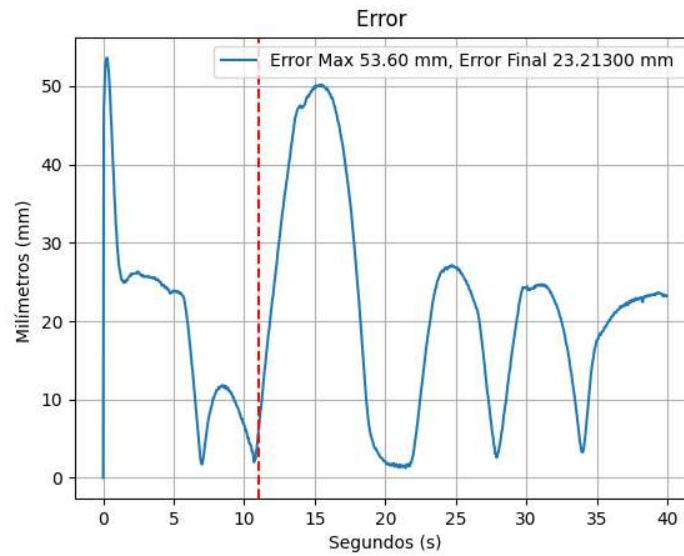


Fig. 4.4: Error con el motor derecho al 80 % de eficiencia con control suplementario

Con el control suplementario, iniciando desde las mismas condiciones, se atenúa tanto el error inicial como el posterior a la falla, además de suavizar la trayectoria y reducir el error final.

Sobre las trayectorias se puede concluir que con el control suplementario, esta se apegas más a la deseada, suavizando el seguimiento y llegando a un punto más cercano al final. Esta falla fue relativamente pequeña y tolerable aun con el control básico, pero el suplementario logró mejorar el comportamiento.

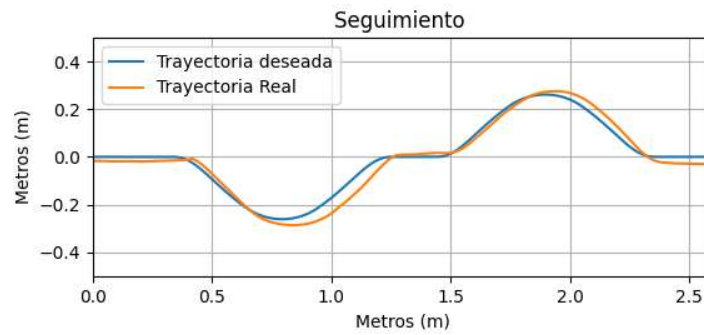


Fig. 4.5: Trayectoria con el motor derecho al 80% de eficiencia sin control suplementario

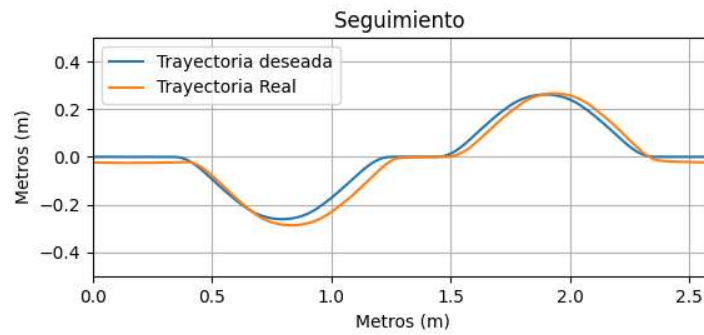


Fig. 4.6: Trayectoria con el motor derecho al 80% de eficiencia con control suplementario

4.1.1.3. Falla del 20 % en el motor izquierdo

Similar a los resultados presentados para la falla en el actuador derecho, el seguimiento con la falla en el actuador izquierdo fue ligeramente mejorada con respecto al error inicial, final y máximo. Esto se repitió para pruebas con fallas en el rango del 10 % al 30 % por lo que las pruebas solo se indujo la falla en un actuador por cuestiones de tiempo. De igual forma, se dejan las gráficas de error y trayectoria

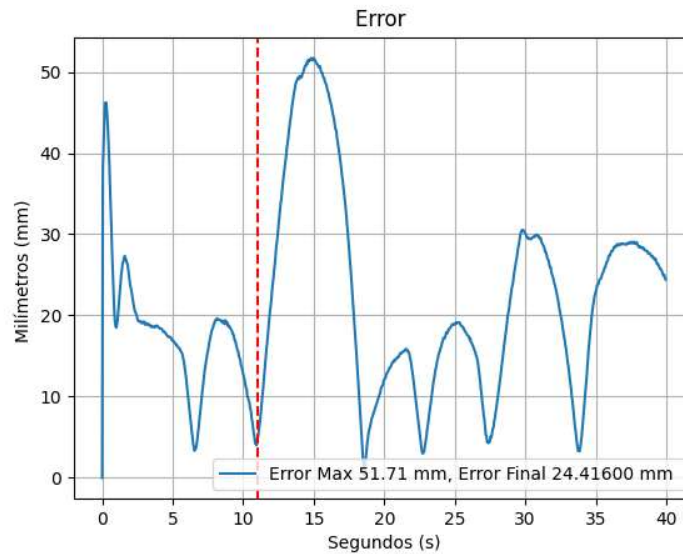


Fig. 4.7: Error con el motor izquierdo al 80 % de eficiencia sin control suplementario

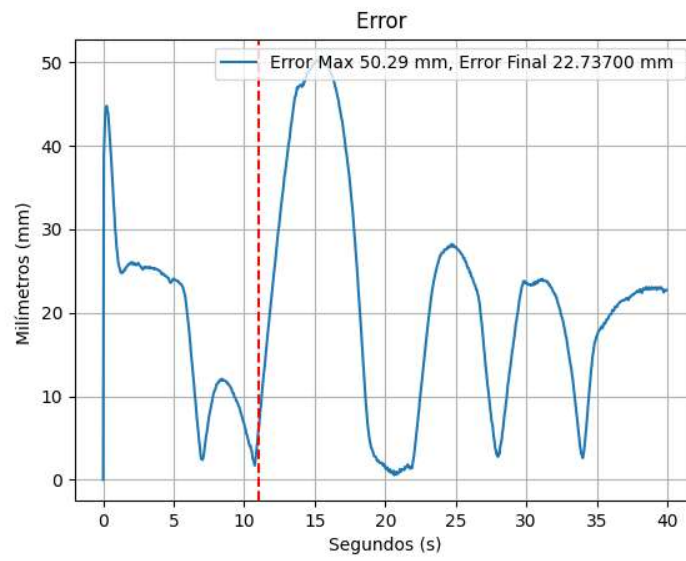


Fig. 4.8: Error con el motor izquierdo al 80 % de eficiencia con control suplementario

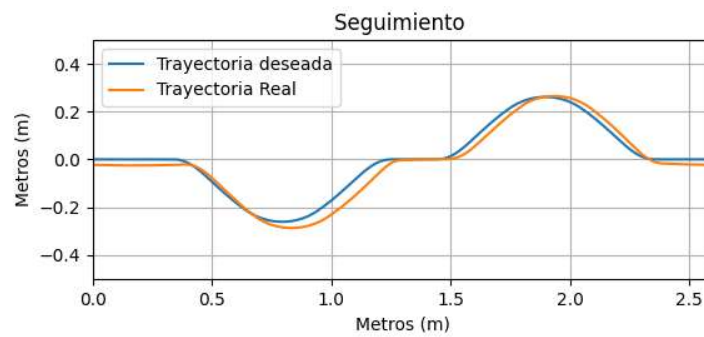


Fig. 4.9: Trayectoria con el motor izquierdo al 80 % de eficiencia con control suplementario

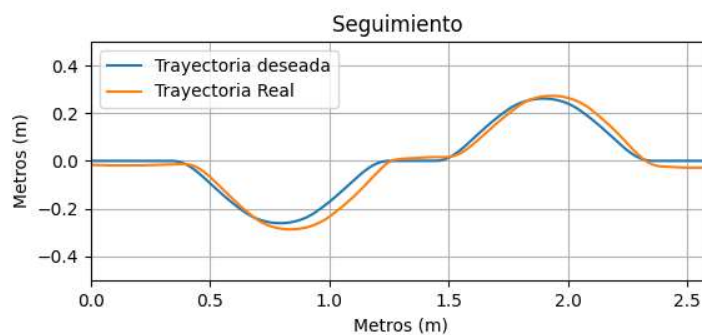


Fig. 4.10: Trayectoria con el motor izquierdo al 80 % de eficiencia sin control suplementario

4.1.1.4. Falla del 40 % en el motor izquierdo

Esta es una falla con un mayor impacto en el seguimiento, se puede observar que los picos de error posteriores al primero siguen alcanzando un valor importante utilizando solo el control base, oscilando entre los 2mm y 39mm.

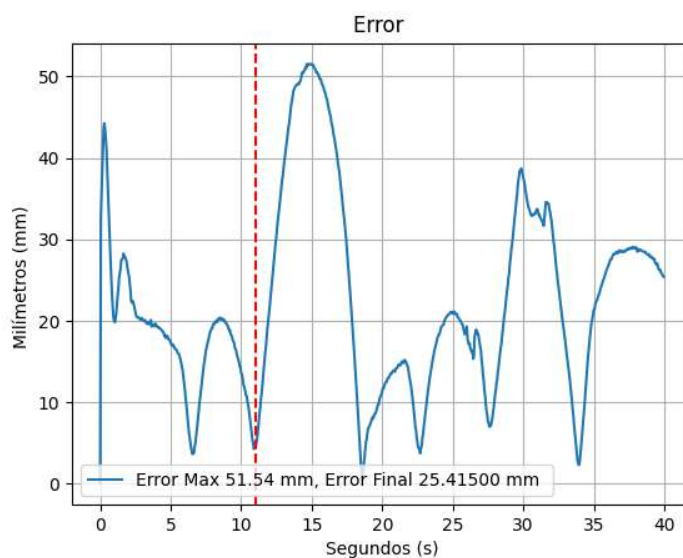


Fig. 4.11: Error con el motor izquierdo al 60 % de eficiencia sin control suplementario

Al utilizar el control suplementario, no solo se reduce el error inicial, máximo y final, sino también los picos de error subsecuentes que ahora se encuentran entre 2mm y 31mm, además se suaviza la curva del error.

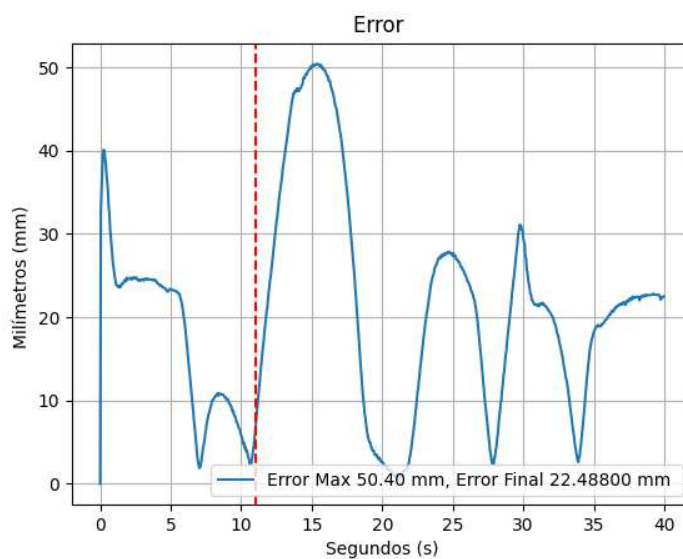


Fig. 4.12: Error con el motor izquierdo al 60 % de eficiencia con control suplementario

De las trayectorias podemos concluir que con el control suplementario la trayectoria

real se mantiene con un error menor que la trayectoria con el control básico. La restricción de tiempo de 40 segundos se hace más notoria puesto que el controlador sacrifica un porcentaje de error por llegar en un tiempo finito al lugar deseado.

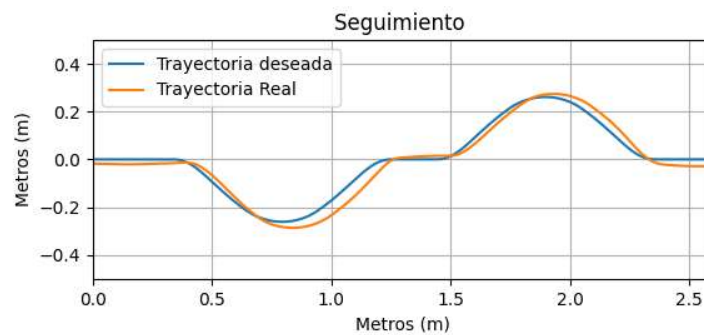


Fig. 4.13: Trayectoria con el motor izquierdo al 60% de eficiencia sin control suplementario

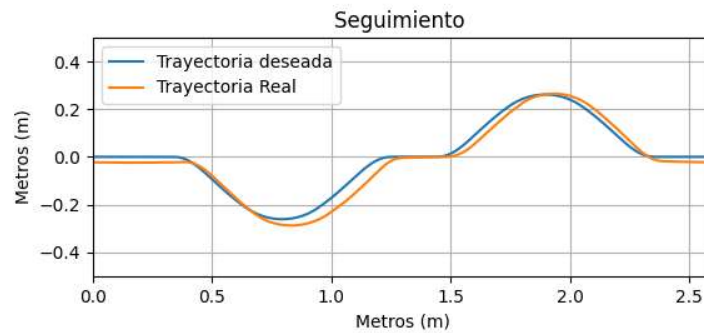


Fig. 4.14: Trayectoria con el motor izquierdo al 60% de eficiencia con control suplementario

4.1.1.5. Falla del 60% en el motor izquierdo

Esta es una falla crítica en la cual se observa por primera vez un comportamiento comparativo interesante entre ambos tipos de control. Si bien el error con el control básico presenta un error máximo menor que el error con el control suplementario. El error final se reduce importantemente.

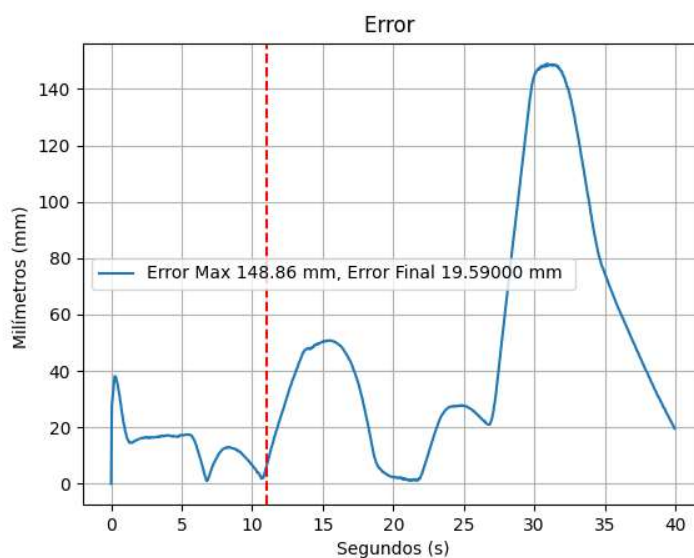


Fig. 4.15: Error con el motor izquierdo al 40 % de eficiencia sin control suplementario

El error máximo con el control suplementario se crea por la retroalimentación de la trayectoria conocida, lo que le da más peso a el lugar de finalización deseado, por lo que el controlador sacrifica el seguimiento de trayectoria del segundo 27 en adelante para acercarse lo más posible al punto final de la trayectoria. Reduciendo el error final a más de la mitad que sin usar el control suplementario.

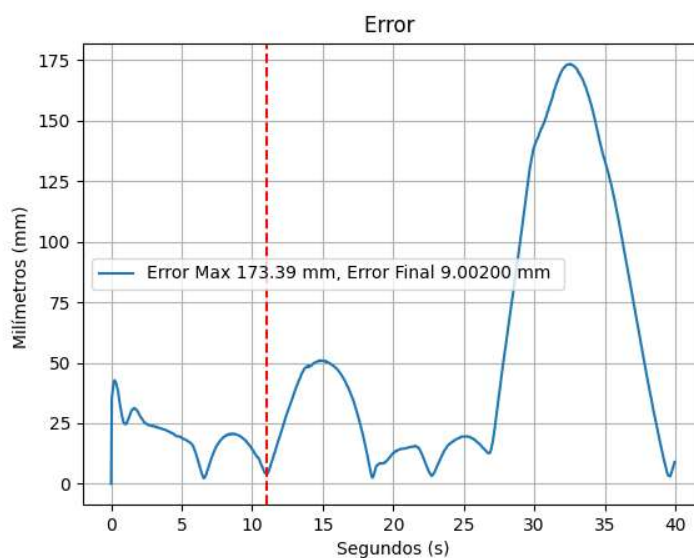


Fig. 4.16: Error con el motor izquierdo al 40 % de eficiencia con control suplementario

Con esta falla las trayectorias muestran de una forma notoria los efectos de cada controlador. El control básico ignora el final conocido de la trayectoria para apegarse a esta lo más posible, pero el control suplementario va directo al punto final para poder cumplir con llegar en los 40s que se dan para el experimento.

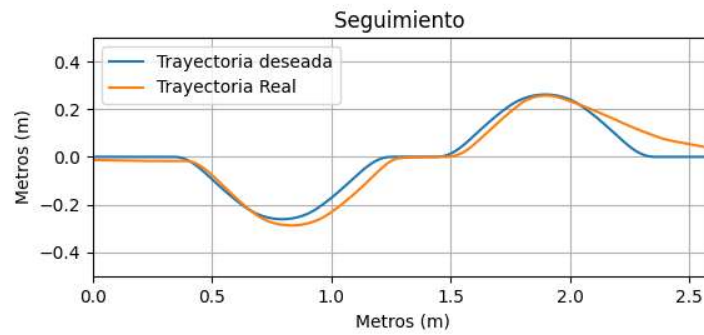


Fig. 4.17: Trayectoria con el motor izquierdo al 40% de eficiencia sin control suplementario

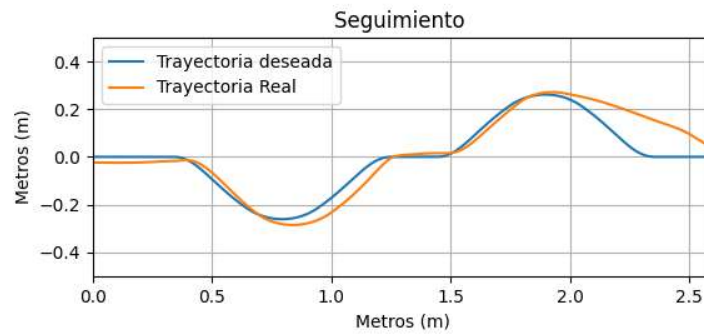


Fig. 4.18: Trayectoria con el motor izquierdo al 40% de eficiencia con control suplementario

A continuación se muestra la tabla 4.3 con los resultados de error máximo con el control suplementario y sin el.

Tabla 4.2: Resultados de error Máximo en mm

Magnitud de falla	Error Max Control Base	Error Max Control Sup
-------------------	------------------------	-----------------------

20 %	51.71	50.29
40 %	51.54	50.40
60 %	148.86	173.39

Como se comentó anteriormente, el uso del control suplementario disminuye ligeramente el error máximo durante fallas menores al 40 %, pero lo incrementa en fallas mayores. Esto es debido a la restricción de tiempo y el conocimiento del punto final deseado.

A continuación se muestra la tabla 4.3 con los resultados de error máximo con el control suplementario y sin el.

Tabla 4.3: Resultados de error Final en mm

Magnitud de falla	Error Final Control Base	Error Final Control Sup
-------------------	--------------------------	-------------------------

20 %	25.98	23.21
40 %	25.41	22.48
60 %	19.6	9

Aquí podemos ver el el error final con el control suplementario es cada vez menor comparado con el error final con el control base. Esto se hace más evidente en fallas grandes donde el controlador se dirige directamente al punto final deseado.

4.2. Trabajo a futuro

La plataforma desarrollada en este estudio permite salto cualitativo en la investigación y desarrollo de sistemas robóticos autónomos, específicamente para la navegación y control de robots móviles como el Turtlebot. Esta infraestructura facilita la ejecución de experimentos en tiempo real y sienta la base para la exploración de nuevos algoritmos y técnicas de control. A continuación se listan las componentes del sistema que pueden contribuir a futuros trabajos:

- Generador de trayectoria
- Simulador de comportamiento ideal

- Controlador en Python para el Turtlebot
- Interfaz inalámbrica con el Turtlebot
- Interfaz con el sistema de visión para retroalimentación
- Plataforma para guardado y procesamiento de datos

El Turtlebot ya cuenta con un sistema de propio para navegación autónoma por lo que se plantea el siguiente algoritmo para siguientes estudios.

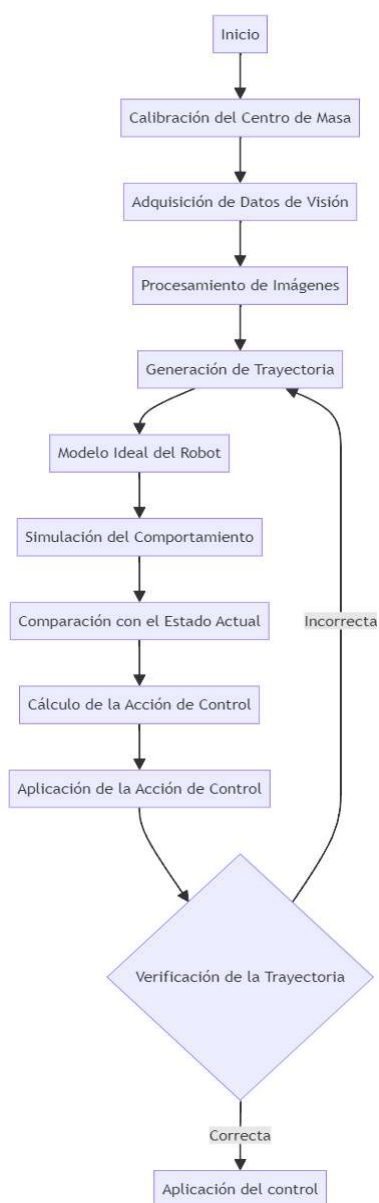


Fig. 4.19: Algoritmo para control de seguimiento de trayectoria en tiempo real.

4.3. Conclusiones

La transición de un enfoque de control robusto, teóricamente fundado, a una aplicación práctica en un contexto real, fue un reto técnico y logístico lleno de aprendizaje, adaptación y superación de desafíos emergentes. El proceso de migrar el código de MATLAB a Python, lejos de ser una mera traducción de lenguajes de programación, exigió una comprensión profunda de las capacidades y limitaciones propias de Python, especialmente en lo que respecta a la manipulación de matrices y la precisión numérica crítica para el control en tiempo real de robots.

La integración dentro de una arquitectura de nodos para ROS y su implementación en un entorno simulado en Gazebo permitieron una aproximación al comportamiento del sistema en un entorno controlado, destacando la importancia de una simulación para probar conceptualmente el funcionamiento del controlador. Sin embargo, la posterior implementación en un TurtleBot físico mostró la gran diferencia entre un simulador y todas las perturbaciones externas del mundo real, esto requirió de flexibilidad, adaptación y hasta improvisación para acoplar el software del robot al entorno de pruebas.

El desafío logístico de realizar pruebas de campo en un laboratorio con limitaciones de tiempo y recursos, remarcó la importancia de la capacidad para realizar ajustes rápidos y eficaces ante imprevistos, como la adaptación a un terreno no ideal para el robot y la configuración de una red LAN propia para los experimentos. Estas circunstancias enfatizaron la necesidad de un enfoque multifacético para la resolución de problemas y aprendizaje rápido sobre nuevas tecnologías como los sistemas de visión VICON.

En lo referente al controlador, el uso de un segundo lazo de retroalimentación el cual indique la diferencia del comportamiento actual con uno nominal es fundamental destacar cómo esta estrategia eleva la capacidad del controlador para gestionar las incertidumbres y fallos potenciales, asegurando así que el robot mantenga su curso y destino final según la trayectoria deseada. Este segundo lazo actúa como una memoria, comparando constantemente el comportamiento actual del robot con un comportamiento nominal o ideal. Esta comparación permite al sistema identificar y corregir desviaciones, ajustando las acciones del robot en tiempo real.

En lo que respecta al control suplementario, la necesidad de definir pesos para maniobras de giro y desplazamiento en línea recta refleja una comprensión detallada de la dinámica de movimiento del robot. Dando un peso mayor a los giros, este

enfoque subraya la importancia crítica de la orientación para la navegación precisa del robot. Los giros, siendo inherentemente más complejos y susceptibles a errores debido a factores como el deslizamiento o la inercia, requieren una atención especial para asegurar que el robot pueda ajustar su trayectoria de manera eficaz y mantener la alineación con la ruta planeada. Esta propuesta de añadir pesos al control suplementario se basa en la premisa de que no todos los movimientos deben ser tratados de igual manera por el sistema de control. Al diferenciar entre giros y desplazamientos en línea recta, y asignar una importancia variable a cada uno, el controlador se vuelve más sensible a las exigencias específicas de la tarea de navegación. Esta propuesta se puede describir partiendo de la definición de control suplementario (4.1)

$$u_{sup} = \begin{bmatrix} u_{f_{sup}} \\ u_{\theta_{sup}} \end{bmatrix} \quad (4.1)$$

Se propone añadir un peso para la componente lineal y angular, definido en (4.2)

$$W_{u_{sup}} = \begin{bmatrix} W_{f_{sup}} \\ W_{\theta_{sup}} \end{bmatrix} \quad (4.2)$$

Donde $1 \geq W_{f_{sup}}, W_{\theta_{sup}} > 0$ y $W_{\theta_{sup}} > W_{f_{sup}}$

Para terminar con una nueva entrada de control suplementario propuesta en (4.3)

$$u_{sup}W = \begin{bmatrix} u_{f_{sup}} \\ u_{\theta_{sup}} \end{bmatrix} * W_{u_{sup}} \quad (4.3)$$

Esta propuesta, abonada al algoritmo de control en tiempo real comentado en la sección de trabajo a futuro, dejan una dirección en la cual se pudiera continuar este trabajo.

Junto con este estudio y algunos meses de mi vida, dejo a disponibilidad de público general el software y recomendaciones utilizadas en mi trabajo en la siguiente dirección.

- <https://github.com/RafaelCastro-FIME/ARCD-MCIE-ROS-DIFF-ROB>

Espero de la manera mas honesta que cualquier lector que esté interesado en utilizar mi trabajo como base tenga la libertad de contactarme a mi correo:

- rafael.castrodaniel@gmail.com

Sin nada más que agregar, doy por terminado este documento.

Apéndice A

Gráficas de resultados

A.1. Apéndice 1: Gráficas de efecto de fallas

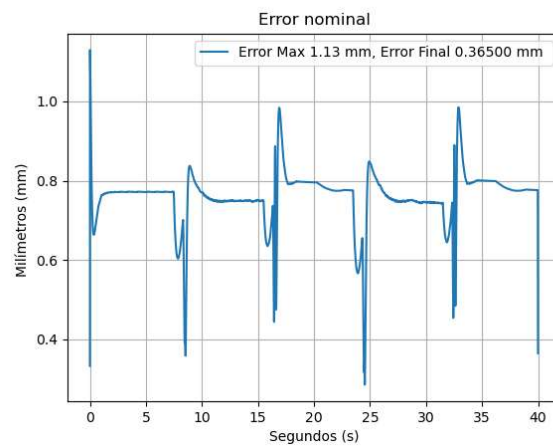


Fig. A.1: Error Nominal

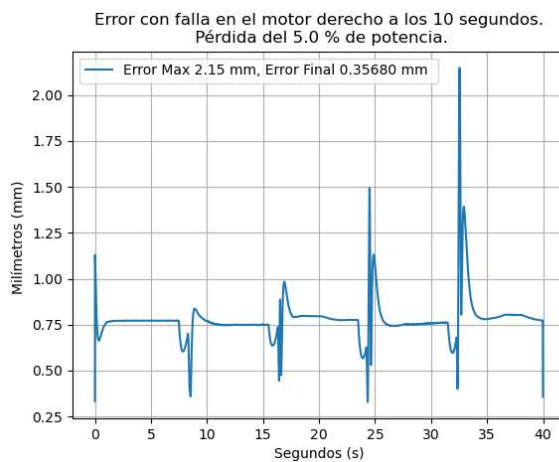


Fig. A.2: Error con falla en el motor derecho a los 10 segundos, con pérdida del 5.0 % de potencia.

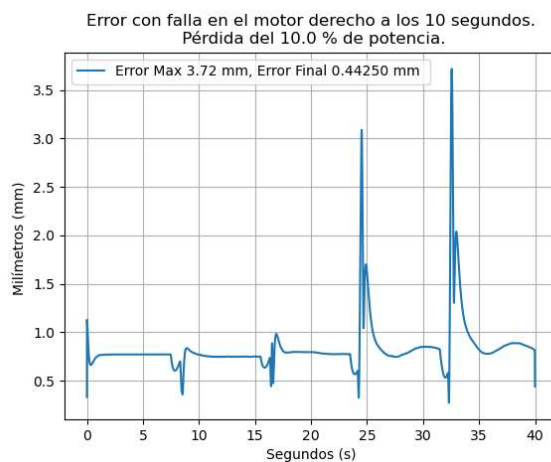


Fig. A.3: Error con falla en el motor derecho a los 10 segundos, con pérdida del 10.0 % de potencia.

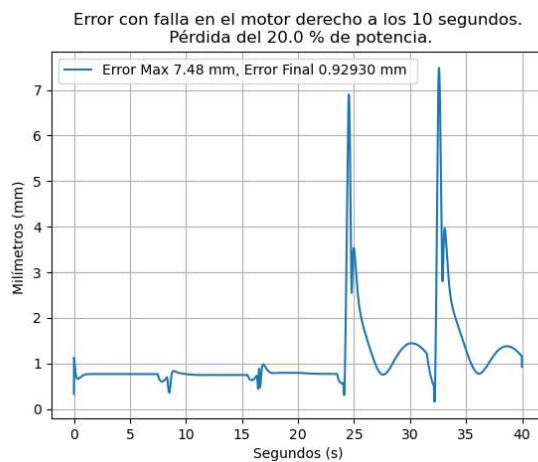


Fig. A.4: Error con falla en el motor derecho a los 10 segundos, con pérdida del 20.0 % de potencia.

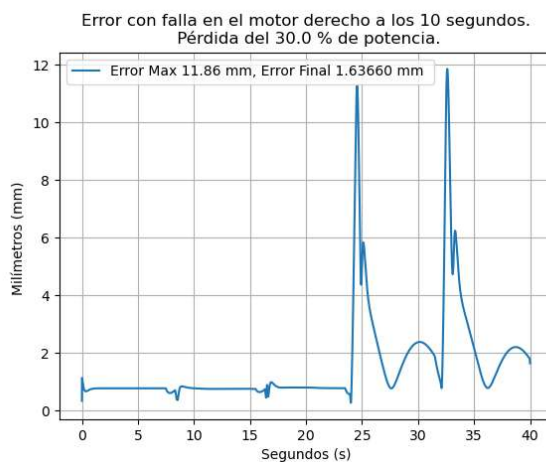


Fig. A.5: Error con falla en el motor derecho a los 10 segundos, con pérdida del 30.0 % de potencia.

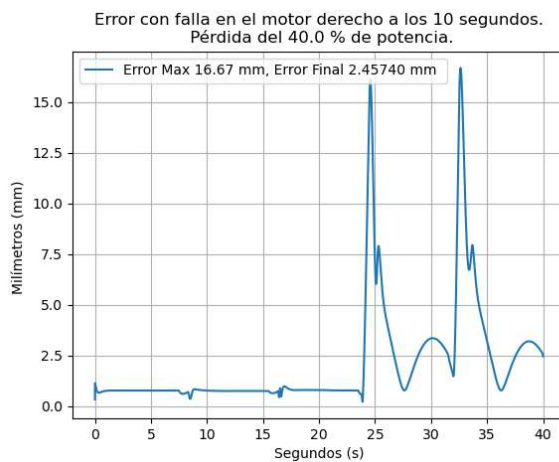


Fig. A.6: Error con falla en el motor derecho a los 10 segundos, con pérdida del 40.0 % de potencia.

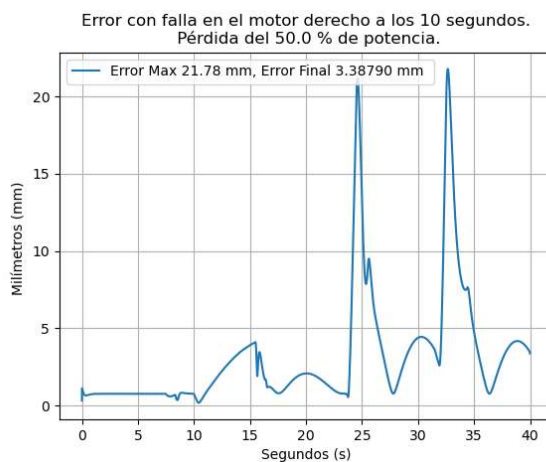


Fig. A.7: Error con falla en el motor derecho a los 10 segundos, con pérdida del 50.0 % de potencia.

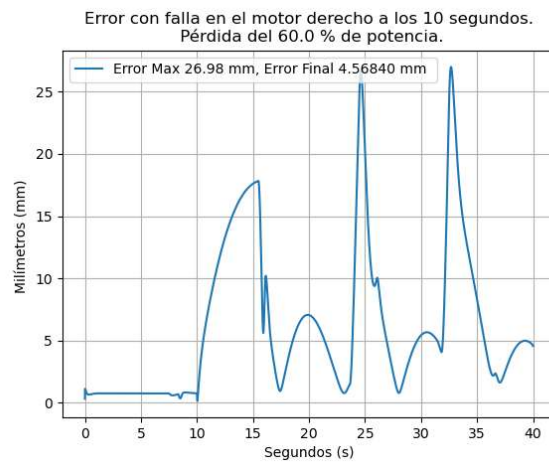


Fig. A.8: Error con falla en el motor derecho a los 10 segundos, con pérdida del 60.0% de potencia.

Bibliografía

- [1] B. Abci1, M. E. B. E. Najjar, V. Cocquempot1, and G. Dherbomez1. An informational approach for sensor and actuator fault diagnosis for autonomous mobile robots. *Journal of Intelligent & Robotic Systems*, 2019.
- [2] Mustafa Ayyıldız and Umut Tilki. Fault tolerant control of wheeled mobile robots using adaptive sliding mode controller. In *2021 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, pages 1–6, 2021.
- [3] F. R. Freire Carrera. Control de las ruedas delanteras de robots móviles. *Ingenius, Revista de Ciencia y Tecnología*, 3:40–45, 2008.
- [4] J. L. González Martínez. Relación entre el control con modelo interno y el enfoque cascada para la tolerancia a fallas. Technical report, Universidad Autónoma de Nuevo León, Facultad de Ingeniería Mecánica y Eléctrica., 2021.
- [5] E. J. Guajardo Benavides, E. Alcorta Garcia, and M. A. Alcorta Garcia. Port-hamiltonian control of a differential robot. In *Moreno, H.A., Carrera, I.G., Ramírez-Mendoza, R.A., Baca, J., Banfield, I.A. (eds) Advances in Automation and Robotics Research. LACAR 2021. Lecture Notes in Networks and Systems, vol 347. Springer*, 2021.
- [6] I. E. Leal Leal and E. Alcorta Garcia. Increased fault tolerance through generalized pi control. In *Memorias del Congreso Nacional de Control Automático*, 2020.
- [7] Lin Liu, Gang Feng, Lihua Xie, Guoquan Lai, and Wenshuai Wei. Adaptive distributed fault-tolerant formation control for multi-robot systems under partial loss of actuator effectiveness. *IEEE Transactions on Control Systems Technology*, 27(3):1032–1044, 2018.

- [8] Ch. Myint and N. N. Win. Position and velocity control for two-wheel differential drive mobile robot. *International Journal of Science, Engineering and Technology Research (IJSETR)*, 5(9):2849–2855, 2016.
- [9] S. Nurmaini, K. Dewi, and B. Tutuko. Differential-drive mobile robot control design based-on linear feedback control law. In *IOP Conf. Series: Materials Science and Engineering*, pages 1–7, 2017.
- [10] Daniel Augusto Pereira, Ayad Al-Dujaili, Maan El Badaoui El Najjar, Vincent Cocquempot, and Yajie Ma. Actuator fault estimation and fault tolerant control in three physically-linked 2wd mobile robots. *IFAC-PapersOnLine*, 51:709–716, 01 2018.
- [11] Robotis. Robotis-e.
- [12] Alexandru Stancu, Eduard Codres, and Vicenc Jpuig. A fault hiding approach for the sliding mode fault-tolerant control of a non-holonomic mobile robot. In *2016 3rd Conference on Control and Fault-Tolerant Systems (SysTol) Barcelona, Spain, Sept. 7-9, 2016*. IEEE, 2016.
- [13] Yajing Sun, Weijie Wei, Xingxing Wang, et al. Cpg-based gait planning for hexapod robot with feedback. *International Journal of Advanced Robotic Systems*, 17(3):1–13, 2020.
- [14] C. Verde, S. Gentil, and R. Morales Menéndez. *Monitoreo y diagnóstico automatico de fallas en sistemas dinámicos*. Trillas, 2013.
- [15] P. Yazdjerdi and N. Meskin. Fault tolerant control of differential drive mobile robots using sliding mode controller. In *5th International Conference on Control, Instrumentation, and Automation (ICCIA)*, pages 270–274, 2017.
- [16] P. Yazdjerdi and N. Meskin. Design and real-time implementation of actuator fault-tolerant control for differential-drive mobile robots based on multiple-model approach. *Proc IMechE Part I: J Systems and Control Engineering*, 232(6):652–661., 2018.
- [17] Y. Zhang, J. Jiang, and P. Shi. Fault-tolerant control systems: A comparative study between active and passive approaches. *Annual Reviews in Control*, 44:18–31, 2017.

- [18] Zuoquan Zhao, Jiadong Wang, Jiawei Cao, Wenchao Gao, and Qinyuan Ren. A fault-tolerant architecture for mobile robot localization. In *2019 IEEE 15th International Conference on Control and Automation (ICCA) Edinburgh, Scotland, July 16-19, 2019*. IEEE, 2019.