UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA

FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

CARRERA DE ELECTRÓNICA Y AUTOMATIZACIÓN

TRABAJO DE TITULACIÓN

DESARROLLO E IMPLEMENTACIÓN DE UN CONTROLADOR PARA EL SEGUIMIENTO DE TRAYECTORIA DE UN ROBOT MÓVIL CON RUEDAS BASADO EN LINEALIZACIÓN DE REALIMENTACIÓN Y VARIABLES DE ESTADO

MODALIDAD:

TRABAJO DE INTEGRACIÓN CURRICULAR

AUTOR

CARLOS DAMIÁN ASENCIO SUAREZ

PROFESOR TUTOR

ING. JUNIOR FIGUEROA OLMEDO MSC

LA LIBERTAD-ECUADOR

2024

AGRADECIMIENTO

Agradezco a Dios por darme vida y llegar hasta el final de esta carrera universitaria de Electrónica y Automatización la cual empecé hace 5 años y me ha permitido vivir muchas experiencias divertidas en cada semestre haciendo nuevos amigos y aprendiendo cosas nuevas.

Agradezco a mis padres Jacinto Asencio y Cecilia Suarez por haberme salvado la vida hace 23 años y permitirme vivir este momento agradable, sin ustedes nada de esto sería posible, este título es de ustedes.

A mi tutor Junior Figueroa por siempre guiarme y darme buenos consejos para seguir adelante y no rendirme en este proceso de titulación, gracias por siempre tenerme paciencia y compartirme sus conocimientos en cada una de las asignaturas que me dio clase.

Agradezco también a mis amigos Buenaño, Roger y Douglas, que de alguna manera siempre me ayudaban y apoyaban en cualquier actividad, ustedes formaron parte de este corto camino siempre los recordare como mis amigos.

A cada uno de los ingenieros que me dieron clase y siempre enseñaban de buena manera haciendo que los estudiantes se motiven para seguir adelante en la carrera y no se den por vencidos y abandonen sus sueños.

Carlos Damián Asencio Suarez

DEDICATORIA

Dedico este trabajo de titulación a mis padres, Jacinto Asencio y Cecilia Suarez, que me han brindado su apoyo moral para continuar con mis estudios y enseñándome buenos valores que reflejan lo que soy hoy en día, cada logro y meta alcanzada es para ustedes.

A mi abuelita María Laínez que siempre me apoya y me daba buenos consejos, a mis hermanos Katherine, Jesús y Emily que siempre me animan para seguir adelante y no darme por vencido en ningún instante de la vida.

Y me la dedico a mí mismo que sin importar las dificultades que me pongan en frente, jamás me rindo y siempre sonrió, aunque el mundo se me venga abajo siempre encuentro la luz.

Carlos Damián Asencio Suarez

APROBACIÓN DEL TUTOR

En mi calidad de Tutor del trabajo de integración curricular denominado "DESARROLLO E IMPLEMENTACIÓN DE UN CONTROLADOR PARA EL SEGUIMIENTO DE TRAYECTORIA DE UN ROBOT MÓVIL CON RUEDAS BASADO EN LINEALIZACIÓN DE REALIMENTACIÓN Y VARIABLES DE ESTADO ", elaborado por el estudiante Asencio Suarez Carlos Damián, de la carrera de Electrónica y Automatización de la Facultad de Sistemas y Telecomunicaciones de la Universidad Estatal Península de Santa Elena, me permito declarar que luego de haber orientado, estudiado y revisado, lo apruebo en todas sus partes y autorizo al estudiante para que inicien los trámites legales correspondientes

La libertad, 10 de diciembre del 2024

Ing. Junior Figueroa Olmedo, MSc.
DOCENTE TUTOR

TRIBUNAL DE SUSTENTACIÓN

Ph.D. Ronald Humberto Rovira Jurado.
DIRECTOR DE LA CARRERA DE ELECTRÓNICA Y AUTOMATIZACIÓN

Ing. Luis Enrique Chuquimarca Jiménez, Mgt.
DOCENTE GUÍA UIC II

Ing. Junior Figueroa Olmedo, MSc.
DOCENTE TUTOR

Ing. Edisson Coral Salinas, Mgt.
DOCENTE ESPECIALISTA

Ing. Corina Gonzabay De La A, Mgt.
SECRETARIA

DECLARACIÓN

El contenido del presente trabajo de titulación es de mi entera responsabilidad, el patrimonio intelectual del mismo le pertenece a la Universidad Estatal Península de Santa Elena.

Carlos Asencio S

Asencio Suarez Carlos Damián

Autor

RESUMEN

El presente trabajo de titulación se enfoca en el desarrollo e implementación de un controlador para el seguimiento de trayectoria de un robot móvil con ruedas basado en linealización de realimentación y variables de estados.

Para llevara a cabo este proyecto se utilizó el robot móvil mBot Neo, en el cual se desarrollo el algoritmo de control que permite obtener las velocidades lineales y angulares con las que el robot móvil debe girar hacia la derecha o izquierda en un tramo de acuerdo a la trayectoria de referencia.

La creación e implementación de este algoritmo se basó en los fundamentos matemáticos para obtener el modelo cinemático del robot y luego implementarlo a través de un software de alto nivel y especializado como Matlab que permite ajustar y optimizar el controlador a través de la programación.

Se realizaron varias pruebas para comprobar que el controlador funcione correctamente, primero se realizó una interfaz gráfica mediante las herramientas de App Designer que muestran el comportamiento del robot de forma simulada y a través de las métricas se pudieron obtener datos que permiten comparar el desempeño del controlador y determinar en qué trayectorias muestra mejores resultados para finalmente implementarlo de forma física en el mBot Neo en una pista libre de obstáculos.

Palabras claves: controlador, algoritmo de control, robot móvil, seguimiento de trayectoria

ABSTRACT

The present degree work is focused on the development and implementation of a controller for the trajectory tracking of a mobile robot with wheels based on feedback linearization and state variables.

The mBot Neo mobile robot was used to carry out this project, in which the control algorithm was developed to obtain the linear and angular velocities with which the mobile robot must turn to the right or left in a section according to the reference trajectory.

The creation and implementation of this algorithm was based on mathematical fundamentals to obtain the kinematic model of the robot and then implement it through a high-level and specialized software such as Matlab that allows adjusting and optimizing the controller through programming.

Several tests were performed to verify that the controller works correctly, first a graphical interface was made using App Designer tools that show the behavior of the robot in a simulated way and through the metrics it was possible to obtain data that allow to compare the performance of the controller and determine in which trajectories it shows better results to finally implement it physically in the mBot Neo in a track free of obstacles.

Keywords: controller, control algorithm, mobile robot, trajectory tracking.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPITULO I

1.1 ANTECEDENTES

La robótica ha tenido un papel muy importante a lo largo de la historia de la humanidad y ha evolucionado rápidamente en los últimos años con la finalidad de crear unidades que facilitan el trabajo en diversas áreas. Desde las antiguas civilizaciones ya se comenzaban a predecir seres vivos mecánicos que eran capaces de accionarse a través de poleas y bombas hidráulicas, Sin embargo, la idea de los robots como tal comenzó a tener más fuerza en las civilizaciones árabes haciendo un énfasis en esto mecanismos para la comodidad humana y es así como empieza el sorprendente desarrollo evolucionario de la robótica.

Con la llegada de nuevos proyectos y el avance de la tecnología, entre los años 1966 y 1972 se implementó el primer robot móvil llamado Shakey el cual era una plataforma móvil independiente que estaba controlada por visión mediante una cámara y habilitada con un detector táctil [1]. Desde ese instante la investigación y el diseño de robots móviles con diversas características ha ido creciendo a grandes pasos hasta la actualidad, los podemos utilizar para diferentes aplicaciones como exploración minera, reconocimiento de terreno, misiones de búsqueda y rescate, automatización de procesos, etc.

A diferencia de los robots estacionarios o fijos, los robots móviles cambian su posición constantemente y a su orientación a lo largo de los ejes de coordenadas básicas, por esta razón deben estar en constante reconocimiento del entorno que los rodea para adaptarse y realizar movimientos naturales evitando obstáculos que se presentan, para que un sistema robótico móvil pueda moverse en su entorno deben tener sensores que le permitan monitorear continuamente el entorno en que se encuentran [1].

En [2], la Empresa Locus Robotics ofrece soluciones de robot móviles para la logística y el cumplimiento de pedidos en almacenes. Sus robots como el modelo Locus Origin es inteligente puede moverse y seguir trayectorias para el transporte de materiales y tareas de recolección. Este robot está diseñado específicamente para el cumplimiento colaborativo de gran volumen, mejora la productividad de cumplimiento al eliminar el tiempo de caminata improductivo y garantiza la precisión de los pedidos. Además, Locus Origin se puede configurar fácilmente para usar una amplia gama de contenedores y estanterías de varios

niveles, admite perfectamente la intercalación dinámica de tareas lo que permite completar trabajos de selección, almacenamiento y reabastecimiento al mismo tiempo.

En [3], la Empresa Fetch Robotics ofrece soluciones de automatización de almacenes y logística utilizando robots móviles autónomos, como el modelo Fetch Mobile Manipulator, puede moverse de forma autónoma siguiendo trayectorias predefinidas para realizar tareas con la recolección de productos y el transporte en entorno industriales, Así existen varias aplicaciones que puede realizar este robot y si se lo une con otro pueden mejorar su trabajo de una forma más eficiente , como por ejemplo Fetch es capaz de recoger artículos de los estantes, mientras que los robots de Freight transportan los artículos seleccionados rápidamente por un almacén.

En [4], la empresa Clearpath Robotics se especializa en el desarrollo de robots móviles autónomos para aplicaciones industriales y de investigación. Su robot como el modelo HusKy UGV, está diseñado para el seguimiento de trayectorias precisas en diferentes terrenos y entornos como por ejemplo en la agricultura ya que cuenta con brazo robótico que es utilizado para evitar plagas y reproducción de insectos siendo un gran apoyo para el agricultor en diversos ámbitos de la gestión de la viña. Husky UGV también es utilizado para realizar investigaciones, debido a que está construido robustamente puede seguir trayectorias precisas en superficies heladas como el polo tomando muestras a pingüinos y además cuenta con una cámara para saber cuál es la dirección que se está tomando a cada momento.

En [5], la empresa Aethon se especializa en soluciones de robótica móvil para aplicaciones comerciales y de atención médica. Sus robots como el modelo TUG puede seguir trayectorias para realizar entregas en entornos hospitalarios, por ejemplo, entrega medicamentos de forma segura a través del hospital y directamente a las unidades de enfermería. Además, proporciona entrega automatizada y rentable de comidas a los pisos de los pacientes y devuelve las bandejas sucias al servicio de alimentos.

La robótica móvil tiene ilimitadas aplicaciones y continúa evolucionando en todo el mundo, cada día aparecen nuevos métodos de control y otros que ya se actualizan y mejoran los existentes. Los robots móviles pueden seguir una trayectoria optima basándose en la orientación y posicionamiento con la finalidad de que se generen cambios de velocidad para poder encontrar la trayectoria más corta y así lograr llegar a su destino final. Siempre que

haya imaginación para los nuevos desarrolladores la evolución continua de los robots móviles estará garantizada y seguirá avanzando hasta alcanzar una autonomía total.

1.2 DESCRIPCIÓN DEL PROYECTO

Este proyecto tiene como propuesta, desarrollar un algoritmo de control basado en linealización de realimentación y el uso de variables de estado para tener un rendimiento óptimo y una alta precisión; estás técnicas serán empleadas en conjunto para implementar el controlador que permitirá realizar el seguimiento de trayectoria de un robot móvil con ruedas con configuración diferencial.

Esta propuesta se va a realizar en varias etapas, comenzando por la determinación de la cinemática del robot móvil ya que se requiere de un modelo matemático que muestre el comportamiento cinemático del robot móvil, en este modelo se incluirán las variables de estado como la posición, la orientación y la velocidad angular de las ruedas. Luego se hará la linealización del modelo matemático, que se basa en aproximar el sistema no lineal a un sistema lineal en un determinado punto de referencia para facilitar el diseño del controlador.

Posteriormente se efectuará el diseño del controlador basado en realimentación, utilizando técnicas de control moderno, en el cual se tomarán las variables de estado para estabilizar y controlar los movimientos del robot móvil generando señales de control sobre las velocidades de las ruedas, lo cual permitirá que el robot siga la trayectoria deseada. Para realizar el seguimiento de trayectoria, se considerarán los requisitos de rendimiento como la velocidad de respuesta, la precisión y la robustez del controlador. El controlador deberá tener información en tiempo real sobre la orientación y posición de los estados actuales en que se encuentra el robot, por esta razón se usará sensores como encoders en las ruedas que permitirán conocer estos parámetros en todo momento. Una vez diseñado el controlador, los algoritmos serán traducidos al lenguaje de programación mediante un software especializado que sea capaz de verificar inicialmente mediante simulación la validez del controlador en el proceso del seguimiento de varias trayectorias.

La implementación en simulación será de gran ayuda para realizar varias pruebas que permitirán evaluar y ajustar algunos parámetros para conseguir un buen rendimiento del controlador en diferentes escenarios de seguimiento de trayectorias en las que se incluirán curvas, giros y cambios de velocidad con el fin de validar la precisión y la estabilidad del

controlador. Es posible que el controlador requiera ajuste de parámetros para adaptarse a las características específicas del robot móvil con el fin de optimizar su desempeño. Luego de haber verificado el rendimiento del controlador mediante simulación, se procederá a su implementación sobre el robot móvil y en un entorno físico real.

La implementación se llevará a cabo utilizando un software especializado y una estructura robótica comercial que incorporará los sensores y actuadores necesarios para la implementación del controlador. El software se encargará de procesar los datos de los sensores y calcular la acción del control necesaria para ajustar la velocidad y dirección del robot móvil. Los actuadores como los motores se encargarán de aplicar la acción de control al robot para que se mueva de forma adecuada siguiendo la trayectoria trazada. La implementación puede ser compleja y requiere una integración cuidadosa de los componentes de hardware y software. En la figura 1 se muestra un esquema general de la interacción de las variables del controlador que se aplicará en proceso del seguimiento de la trayectoria.

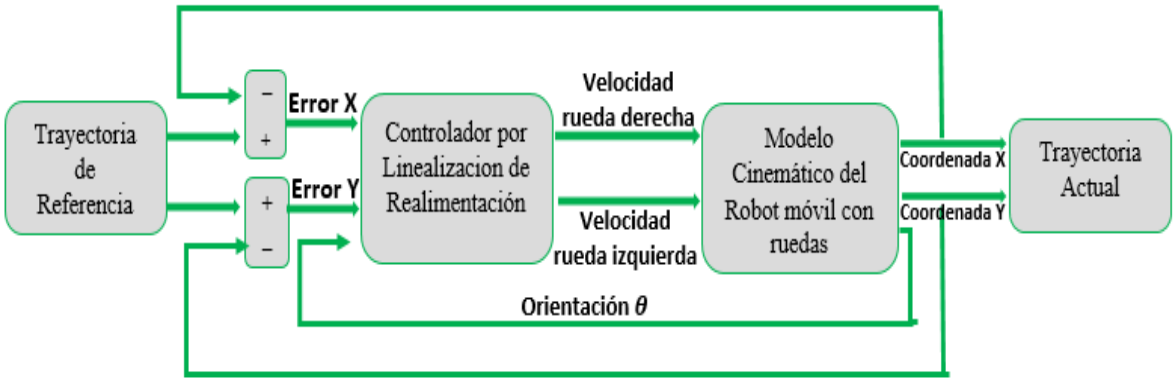

Figura 1*Diagrama de bloque del controlador para el seguimiento de trayectoria [Fuente: Autor]*

Este diagrama de bloques representa el sistema de control del robot móvil con ruedas, primero se ingresa la trayectoria que está representada por un eje X y Y, las dos coordenadas y un conjunto de puntos forman la trayectoria, estos conjuntos de puntos de una trayectoria conocida se los va ingresando al controlador para luego obtener las velocidades de las dos ruedas del robot, dependiendo de las velocidades se van generando todas las coordenadas que conforman la trayectoria de referencia. Siempre que el robot se esté moviendo se deben estar leyendo las posiciones X y Y reales, también la orientación ϑ de cuanto ha girado el robot con respecto al eje X todo esto va estar realimentado y siempre se van actualizando los

nuevos valores de velocidades de las ruedas para poder alcanzar el último punto de la trayectoria.

1.3 OBJETIVOS DEL PROYECTO

1.3.1 OBJETIVO GENERAL

Desarrollar e implementar un controlador mediante un software y algoritmos para el seguimiento de trayectoria de un robot móvil con ruedas basado en linealización de realimentación y variables de estado.

1.3.2 OBJETIVOS ESPECÍFICOS

- Analizar diversas investigaciones relacionados con controladores aplicados a robots móviles con ruedas en procesos de seguimiento de trayectorias con la finalidad de tener una referencia matemática ya comprobada y obtener posteriormente el modelo cinemático del robot e implementar el controlador propuesto.
- Desarrollar un controlador basado en variables de estado y linealización que permita el seguimiento de trayectorias con precisión y estabilidad considerando posibles limitaciones como la velocidad o la aceleración del robot móvil.
- Utilizar un software especializado que permita programar los algoritmos de control necesarios para poder simular en primera instancia el comportamiento del robot en entornos virtuales y corregir posibles errores en el código de programación.
- Ajustar y optimizar el controlador modificando diversos parámetros y líneas de programación para poder mejorar el desempeño y robustez en diferentes condiciones de operación y tipos de trayectoria.
- Evaluar el desempeño del robot móvil en tiempo real y poder corregir los errores de seguimiento y estabilidad del sistema en un determinado punto de la trayectoria logrando así un correcto funcionamiento.

1.4 JUSTIFICACIÓN

En estos días es muy común hablar de robots con alguna forma de movimiento autónomo, debido al alto interés y crecimiento que ha despertado esta área de la robótica. El movimiento autónomo se ha logrado mediante la implementación de técnicas de control auxiliadas por software y hardware moderno. Una forma de lograr la autonomía en el robot es implementarle rieles o ruedas dependiendo de la tarea a realizar. El diseño del control para los robots móviles con ruedas dependerá de su capacidad de desplazarse en una determinada dirección [6] .

En la actualidad, la robótica móvil ha emergido como un campo tecnológico altamente desarrollado y prometedor para resolver problemas complejos en diversas áreas. Este campo de estudio se enfoca en el desarrollo y la implementación de robots que sean capaces de moverse de forma autónoma en entornos dinámicos y cambiantes. Las aplicaciones de la robótica móvil son muy amplias y abarcan desde el control y la programación hasta la inteligencia artificial y la instrumentación.

Los vehículos con rueda son la solución más sencilla para conseguir la movilidad en terrenos medianamente duros y libre de obstáculos alcanzando velocidades relativamente altas. Por otra parte, dotar de sistemas de estabilidad para adaptarse a la configuración de un terreno no es una tarea barata [7], el control en robots móviles impulsa la innovación y el avance de la robótica en general, abriendo nuevas posibilidades de aplicación y superando desafíos técnicos.

En el ámbito de control se busca constantemente mejorar los algoritmos utilizados para guiar y monitorear el movimiento de los robots móviles. Estos algoritmos deben ser precisos y eficientes para garantizar una navegación fluida y segura, evitando colisiones y optimizando el desplazamiento en entornos muy complejos. La robótica móvil ha logrado grandes avances significativos en el desarrollo de soluciones tecnológicas innovadoras, pero aún enfrenta desafíos y limitaciones. Si bien es cierto, los algoritmos de control pueden presentar imprecisiones y dificultades en situaciones complejas lo que puede afectar el rendimiento y la eficiencia de los robots móviles.

Los controladores de seguimiento de trayectoria para robots móviles con ruedas son herramientas esenciales para optimizar el rendimiento y la precisión de este tipo de robots. Implementar este tipo de controladores permite al robot moverse de manera suave y precisa

a lo largo de una trayectoria determinada, ofreciendo numerosos beneficios y ventajas como la confiabilidad de que el robot móvil con rueda seguirá la ruta deseada ya que los algoritmos y técnicas que se usarán en el controlador permitirán al robot móvil mantener la posición y orientación exacta, evitando desviaciones no deseadas especialmente en aplicaciones que requieran alta precisión como en los entornos estrechos.

Al utilizar un controlador en el seguimiento de trayectoria se maximiza la eficiencia de la ejecución de tareas. Al minimizar las desviaciones y los errores, se reducen los tiempos de corrección y se optimizan los recursos disponibles. para que el robot móvil con ruedas puede seguir una trayectoria predefinida de manera autónoma lo que permite a los operadores enfocarse en otras actividades. Un controlador bien diseñado puede integrarse y comunicarse fácilmente con otros sistemas.

En términos generales, diseñar un controlador para los robots móviles con ruedas no forma parte del contenido académico en alguna de las asignaturas de la carrera de Electrónica y Automatización de la UPSE, los estudiantes si pueden analizar, comprender e implementar estos tipos de robots en alguna aplicación específica en base a los conocimientos adquiridos previamente en asignaturas como control continuo, control discreto, robótica, microcontroladores, entre otras. La propuesta de este proyecto de titulación puede ser de gran aporte para las futuras generaciones de estudiantes de esta carrera ya que les permitirá aprender conceptos de control y programación de una manera práctica y tangible. Además, los docentes pueden fomentar proyectos de investigación en los que el estudiante trabaje con robots móviles y desarrollen algoritmos de control avanzados para abordar desafíos específicos como la navegación autónoma, el seguimiento de trayectorias o la interacción con el entorno, permitiendo al estudiante explorar y aplicar conceptos teóricos en situaciones reales.

1.5 ALCANCE DEL PROYECTO

El enfoque de este proyecto se centrará en el empleo de un robot móvil con ruedas con configuración diferencial, excluyendo otros modelos de robots móviles terrestres. Se aplicarán técnicas de linealización de realimentación y variables de estado considerando la dinámica y las perturbaciones que dificultan el poder lograr un seguimiento preciso y

confiable de una trayectoria predefinida; este modelo de regulación será fundamental para el diseño del controlador y la simulación.

Como parte del proceso del desarrollo del proyecto se determinará el modelo matemático del robot móvil con ruedas considerando únicamente la cinemática de este, puesto que dicho modelo será suficiente para poder implementar el controlador. La precisión del modelo que se va a utilizar puede afectar la exactitud del controlador y existe la posibilidad de que ciertas variables de la dinámica del robot puedan afectar el buen desempeño del robot cuando este se movilice con una velocidad alta.

El controlador diseñado será inicialmente evaluado en un entorno de simulación utilizando un software especializado, durante la fase de simulación se visualizarán diferentes parámetros y condiciones que se presentan en el seguimiento de trayectorias, como variaciones en la velocidad angular o lineal de las ruedas, cambios bruscos de dirección, obstáculos imprevistos, entre otros. Esto permitirá visualizar el rendimiento del controlador en diversos escenarios, identificando posibles áreas de mejora y optimización, al utilizar un entorno virtual se evitarán los riesgos y costos vinculados con las pruebas de un robot móvil real.

Una vez que se haya realizado la optimización y ajustes necesarios en el algoritmo de programación se procederá a la implementación del controlador sobre el robot móvil real. Sin embargo, es importante mencionar que esta etapa se abordará luego de haber alcanzado un nivel satisfactorio en el rendimiento y confiabilidad de la simulación. El buen funcionamiento del robot móvil en el seguimiento de la trayectoria requerirá de una programación y configuración de software y hardware adecuados que permitan que el controlador ejecute las acciones previstas.

El rendimiento del controlador puede verse afectado por las condiciones del entorno como cambios en la iluminación y superficies irregulares, lo que podría generar incertidumbre y afectar el desempeño del controlador en tiempo real. Además, la capacidad del procesamiento del hardware aplicado en el robot móvil puede limitar la velocidad y eficiencia provocando retrasos en la respuesta del controlador y afectar el seguimiento de la trayectoria.

El entorno de prueba para verificar la generación de trayectorias tiene que ser un entorno plano sin obstáculos y en lo posible sin elementos pequeños que se puedan adherir a las

llantas, debido a que estos elementos posteriormente pueden afectar a la movilización del robot produciendo fricción o deslizamiento. Además, no se emplearán velocidades tan elevadas en la movilización del robot ya que el tiempo de respuesta de los encoders incorporados en el robot móvil a veces no son lo suficientemente precisos para hacer la detección en tiempo real de las velocidades de las ruedas y esto podría afectar a la lógica de implementación de los algoritmos y la generación de trayectoria podría tener errores elevados.

En el proceso del seguimiento de trayectorias también se debe considerar las dimensiones del robot móvil con ruedas, ya para que ciertas curvas muy cerradas no se generarían de manera adecuada los movimientos que permitan seguir ese tramo de la trayectoria. Por último, se debe indicar que podrían aparecer perturbaciones debido a diversos factores propios del entorno y en donde el algoritmo implementado deberá solventar estas señales perjudiciales modificando ciertos parámetros siempre y cuando tengan una limitada acción sobre el proceso de seguimiento de trayectoria puesto que para otras situaciones el algoritmo fallar podría.

1.6 METODOLOGÍA

Para llevar a cabo el desarrollo de este proyecto, se debe comenzar realizando varios tipos de investigaciones relacionadas con el seguimiento de trayectorias de un robot móvil con ruedas y sobre los tipos de algoritmos específicos que se pueden aplicar a los controladores, Además, es fundamental buscar información relacionada con los softwares en los que se puede realizar la simulación y programación del controlador para realizar varias pruebas. Esto permitirá tener los fundamentos necesarios para el diseño y desarrollo del controlador basado en la linealización de realimentación y variables de estado. Se deben realizar varias fases en este proyecto las cuales se mencionarán a continuación para garantizar un enfoque sólido.

En las investigaciones comparativas se realizará una comparación de los diferentes métodos de control utilizados en los robots móviles con ruedas para el seguimiento de trayectorias, como la linealización de realimentación y variables de estado con otro control como la proporcional integral derivativo (PID) o con el control basado en redes neuronales en términos de estabilidad y eficiencia.

En las investigaciones diagnosticas se identificarán las variables que afectan al algoritmo en el momento que el robot móvil con ruedas siga una determinada trayectoria desde un punto inicial hasta un punto final, es decir en el análisis de situaciones como el cambio de entorno, variación de obstáculos y cambios en el tipo de algoritmos. En la figura 2 se pueden visualizar las diferentes fases del proyecto, A continuación, se describirán cada una de ellas

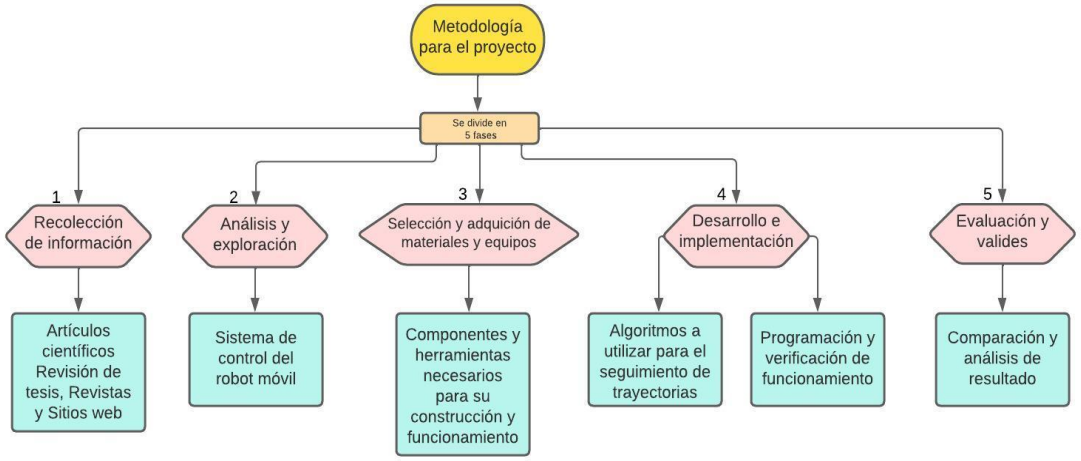

Figura 2. Fases de la metodología del proyecto [Fuente: Autor]

Fase 1. En la recolección de información se procederá con la revisión bibliográfica para el diseño del controlador mediante investigaciones en artículos científicos, revistas, tesis y sitios web relacionados con el seguimiento de trayectoria en los robots móviles con ruedas y las diferentes funciones que realizan.

Fase 2. En el análisis y exploración se procederá con el modelo matemático para comprender el rendimiento del robot móvil en términos de velocidad, distancia recorrida, aceleración, etc. Además, en este apartado se incluirá la investigación de tecnologías emergentes en el campo de la robótica móvil y nuevas estrategias de control con el fin de mejorar el diseño, el control y las capacidades del robot móvil.

Fase 3. En la selección y adquisición de materiales y equipos se procederá a identificar los dispositivos con los que se va a trabajar, sensores, actuadores, baterías, etc. Estos materiales serán seleccionados teniendo en cuenta las especificaciones técnicas para el proyecto como la resistencia, peso, durabilidad y compatibilidad con otros componentes para garantizar un funcionamiento adecuado del robot móvil con ruedas.

Fase 4. En el desarrollo e implementación se procederá con el diseño del controlador basado en realimentación, en el cual se tomarán las variables de estado para estabilizar y controlar los movimientos del robot móvil generando señales de control sobre las velocidades de las ruedas, lo cual permitirá que el robot siga la trayectoria deseada. Una vez diseñado el controlador, los algoritmos serán traducidos al lenguaje de programación mediante un software especializado que sea capaz de verificar el proceso del seguimiento de varias trayectorias.

Fase 5. En la evaluación y validez se procederá inicialmente evaluando el controlador diseñado en un entorno de simulación virtual y luego de forma real para visualizar los diferentes parámetros y condiciones que se presentan en el seguimiento de trayectorias, como variaciones en la velocidad, cambios bruscos de dirección, entre otros. En función de los resultados obtenidos se puede realizar ajustes o mejoras en el controlador para optimizar su desempeño, con el fin de lograr un seguimiento de trayectorias eficiente y preciso en el robot móvil con ruedas.

CAPITULO II

2.1 MARCO CONTEXTUAL

Este proyecto se enfoca en el campo de la robótica móvil, un campo que ha generado un gran impacto e interés en diversas áreas industriales. Los robots móviles, especialmente los de ruedas son los más utilizados debido a su eficiencia y versatilidad pueden desplazarse sobre diversas superficies o terrenos, esto les otorga una ventaja significativa en maniobrabilidad, además cuentan con una gran cantidad de aplicaciones disponibles como la logística, la agricultura, la seguridad y la asistencia médica.

La amplia utilidad y versatilidad de los robots móviles con ruedas han captado la atención de individuos que los ven como una herramienta invaluable para mejorar su la calidad de vida y de la sociedad en general. Con el fin de aprovechar estas capacidades de los robots móviles un creciente número de científicos y catedráticos se encuentran inmersos en diversos proyectos de investigación como el seguimiento de trayectoria donde la adquisición de datos y el procesamiento de información son una parte fundamental, ya que la recopilación de datos sobre el entorno y puntos de referencia combinados con algoritmos de cálculos matemáticos permiten a los robots móviles planificar trayectorias de forma segura y eficientes.

El desarrollo e implementación de un controlador para el seguimiento de trayectoria de un robot móvil con ruedas puede enfrentar ciertas limitaciones de recursos por lo que no podría capturar todas las características y comportamientos complejos del robot móvil en situaciones reales, como el robot es pequeño debido a sus dimensiones puede tener limitaciones en cuanto a las curvaturas máximas o mínimas con las que podría seguir el robot. Además, el desempeño del controlador puede verse afectado por las limitaciones inherentes de los sensores y actuadores, por ejemplo, la resolución de los sensores o la precisión de los motores pueden imponer ciertas restricciones en la capacidad del controlador para seguir trayectorias con alta precisión y velocidad.

Una vez terminado este proyecto los estudiantes de la carrera de Electrónica y Automatización tendrán acceso a una documentación detallada sobre el algoritmo matemático que se ha desarrollado, con el propósito de que puedan realizar modificaciones pertinentes y generar proyectos relacionados con el seguimiento de trayectorias para robots móviles con ruedas. A su vez se podrían beneficiar los docentes de esta carrera ya que tendrán

a disposición esta información para emplearla en asignaturas relacionadas con la robótica. Los algoritmos desarrollados permitirán enriquecer las actividades de enseñanza en el campo de la robótica abriendo nuevas oportunidades de colaboración y exploración. Además, los resultados obtenidos podrán ser presentado en capacitaciones o conferencias con el fin de contribuir al avance de la investigación en robótica móvil.

2.2 MARCO CONCEPTUAL

2.2.1 Definición y clasificación de los robots móviles

Se denomina robot móvil a un sistema electromagnético capaza de moverse de forma autónoma sin estar restringido a un único punto físico. Posee varios sensores que permiten monitorear constantemente su posición relativa a su punto de origen y a su punto de destino final. Habitualmente, el robot móvil es controlado en un sistema de retroalimentación, y sus desplazamiento son logrados mediante mecanismos de locomoción como las ruedas, patas, orugas, entre otros [8].

El creciente interés por los robots móviles ha llevado al desarrollo de diversas estructuras, esto refleja la adaptabilidad de los robots móviles a diferentes entornos y desafíos específicos, tal como se muestra en la figura 3se pueden clasificar de diversos tipos según el medio en que se desplazan como terrestres, aéreos o acuáticos dependiendo de la superficie en la que se van a implementar

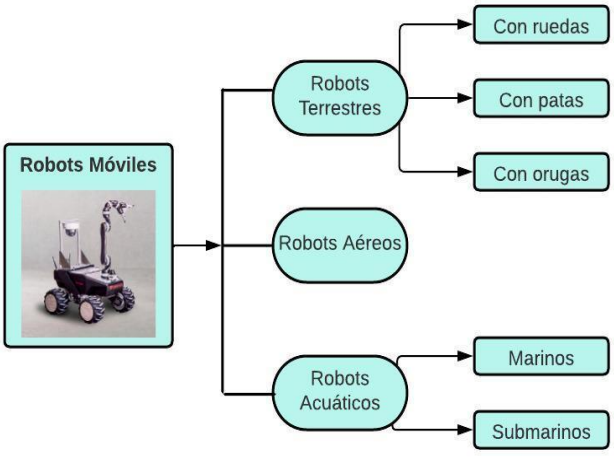

Figura 3. Clasificación de los robots móviles [Fuente: Autor]]

2.2.1.1 Robots terrestres

Los robots móviles terrestres son conocidos como vehículos terrestres no tripulados, sus formas comunes de locomoción son las ruedas, orugas y las patas como se muestra en la figura 4 y debido a estos mecanismos se los puede clasificar como robots móviles con ruedas, robots móviles con patas y robots móviles con orugas [9].

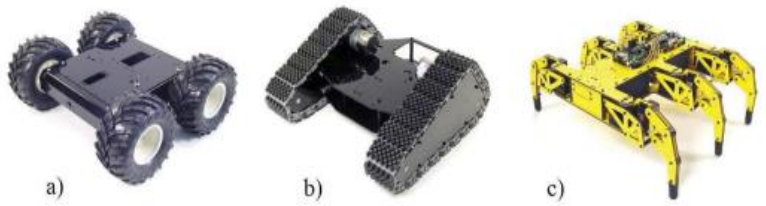

Figura 4. Robots móviles con sistemas de locomoción: a) ruedas, b) orugas, c) con patas [10]

2.2.1.2 Robots con ruedas

En este tipo de robots resalta su eficacia energética en superficies planas y sólidas, no generan desgastes en el terreno donde se desplazan y necesitan de menos componentes en su fabricación, lo que permite que su construcción sea más sencilla [8]. Estos robots se emplean en superficies lisas para realizar trabajos como transporte de materiales o mercancías, las ruedas instaladas en estos robots son maniobrables y eficientes para las distancias recorridas, en la figura 5 podemos ver dos ejemplos de estas ruedas, esto les permite moverse con facilidad y cumplir con las tareas asignadas.

Figura 5. Ejemplos de Robots móviles con ruedas [9]

2.2.1.3 Robots con patas

La locomoción de los robots móviles de tipo patas encuentra su inspiración en los sistemas biológicos presentes en la naturaleza. Entre las ventajas que este tipo de robots presentan se encuentra la capacidad de lograr un movimiento adaptable en terrenos sin preparación previa. En la figura 6 se muestran ejemplos de este tipo de robots, en el cual las patas le proporcionan gran maniobrabilidad y pueden alcanzar grandes velocidades. El diseño del control para este sistema de robots puede ser un poco complejo debido al número de patas ya sea que aumenten o disminuyan [9]

Figura 6. Ejemplos de Robots móviles con patas [11]

2.2.1.4 Robots con oruga

Los robots que emplean orugas resultan más beneficiosos que los que se apoyan en ruedas. Las orugas proporcionan una tracción superior en terrenos y minimizan el deslizamiento. En la figura 7 se ilustra la configuración de los sistemas de desplazamiento con orugas, donde se emplean cadenas de tracción capaces de girar de manera independiente, de modo que cualquier cambio de dirección se consigue a través de la variación de velocidades relativas de la cadena [9].

Figura 7. Ejemplos de Robots móviles con orugas [12]

2.2.1.5 Robots acuáticos

Son conocidos también como vehículos acuáticos no tripulados (UWV, Unmanned Water Vehicles), solo hay pocos robots acuáticos construidos con éxito como se muestra en la figura 8. Uno de los principales desafíos al diseñar y crear un robot acuático es hacer que todos los componentes electrónicos sean impermeables para evitar los cortocircuitos [9]. Este tipo de robots acuáticos están diseñados para funcionar en el agua y algunos de ellos trabajan a grandes profundidades como podemos observar en la figura 8

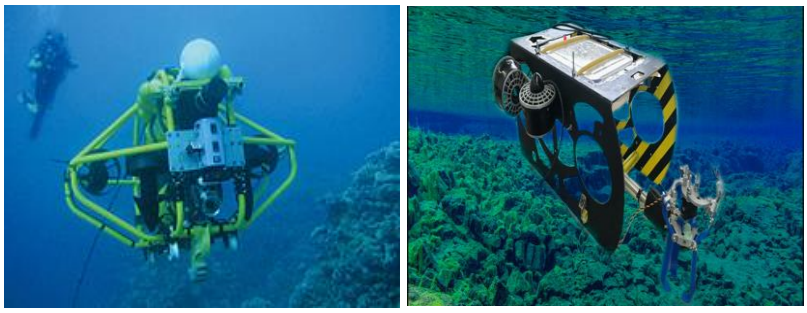

Figura 8. Ejemplos de robots acuáticos [13]

2.2.1.6 Robots aéreos

Son llamados también vehículos aéreos no tripulados (UAV, Unmamed Aerial Vehicles) y están conformados por sistemas móviles que vuelan en un determinado espacio aéreo tal como se visualiza en la figura 9 (drones, cohetes, aviones, helicópteros). Los robots de tipo aéreos están experimentando un notable avance gracias a las ventajas que superan a otros tipos de robots móviles. Por ejemplo no se ven afectados por obstáculos presentes en el terreno que podrían generar inmovilización [9].

Figura 9. Ejemplos de robots aéreos [14]

2.2.2 Robots móviles con ruedas

Los robots móviles equipados con ruedas encuentran un extenso uso en diversas aplicaciones dentro del ámbito de la robótica autónoma. Las ruedas constituyen los elementos esenciales que brindan la capacidad de desplazamiento a estos robots, siendo el método de movilización más prevalente. Los robots de ruedas presentan diseños simples, un menor consumo energético y una velocidad de movimiento superior en comparación con otros mecanismos de locomoción. A pesar de que pueden enfrentar dificultades en terrenos accidentados o irregulares, los robots móviles con ruedas son adecuados para una amplia gama de entornos en aplicaciones prácticas. Se emplean en casi todas las aplicaciones de robots de servicio en el interior de edificios y también en exteriores cuando el terreno no es demasiado abrupto [9].

En lo que respecta a los componentes de un robot móvil con ruedas (RMR), se incluye un diseño cinemático y un conjunto de actuadores que confieren movimiento a la configuración cinemática. Existen diversas estructuras cinemáticas para los RMR, cuya elección esta principalmente influenciada por la aplicación específica a la cual se destinara el RMR [8].

2.2.2.1 Sistemas de locomoción

Los robots móviles adoptan diversas modalidades de movimiento a través de ruedas, las cuales les otorgan atributos y particularidades variables en términos de eficiencia energética, tamaño, capacidad de carga y maniobrabilidad. Existen múltiples sistemas de locomoción utilizados en los robots móviles, y a continuación se describirán los más frecuentes [10].

Ackerman

El modelo Ackerman o tipo coche es usado en vehículos con cuatro ruedas convencionales como se muestra en la figura 10, dos de ellas son usadas para la tracción y se sitúan en la parte trasera, en paralelo al chasis principal del vehículo. Las ruedas delanteras se encargan del direccionamiento y cada una posee dos ángulos de giro distintos. Sin embargo, estas características de doble ángulo de giro pueden generar desafíos en el control del vehículo En muchos casos se unen los ángulos de direccionamiento en uno solo, sin embargo, ambas trayectorias no serían paralelas y por lo tanto las ruedas se deslizarían. Su mayor problema es la limitación de maniobrabilidad [10].

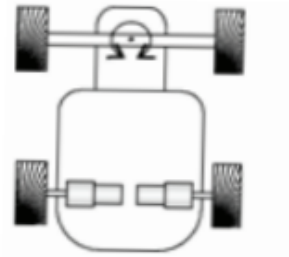

Figura 10. Sistema de locomoción ackerman [15]

Tracción omnidireccional

Este sistema de tracción se basa en la utilización de tres ruedas directrices y motores. Esta configuración posee tres grados de libertad tal como se muestra en la figura 11, por lo que puede realizar cualquier movimiento y posicionarse en cualquier orientación o posición y no presenta limitaciones cinemáticas [10].

Figura 11. Sistema de locomoción tracción omnidireccional [15]

Tracción diferencial

Este tipo de direccionamiento viene dado por la diferencia de velocidades de dos ruedas laterales, estas son montadas en un único eje cada una independientemente pulsada y controlada tal como se muestra en la figura 12, ambas proporcionando tracción y direccionamiento. Es un sistema muy útil si consideramos su capacidad de movimiento, ya que presenta la posibilidad de cambiar de orientación sin movimiento de traslación. Adicionalmente, se pueden incluir una o más ruedas de soporte. Esta configuración es la más común en robots de dimensiones reducidas [10].

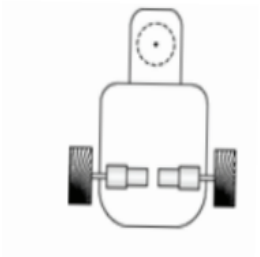

Figura 12. Sistema de locomoción tracción diferencial [15]

Triciclo clásico

Como se observa en la figura 13, este sistema se basa en una rueda delantera que sirve para la tracción y para el direccionamiento. El eje trasero, con dos ruedas laterales, es pasivo y sus ruedas se muevan libremente. Su capacidad de maniobra supera la del sistema Ackermann, dado que solo cuenta con una rueda para el direccionamiento. Sin embargo esta configuración puede conllevar problemas de estabilidad en terrenos complicados [10].

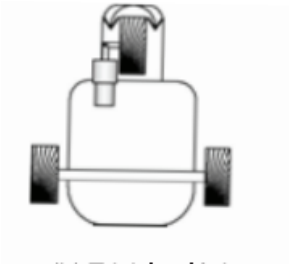

Figura 13. Sistema de locomoción tracción clásico [15]

Skid-steer

En este enfoque se incorporan múltiples ruedas en cada lado del vehículo, como se ilustra en la figura 14. Estas ruedas operan de manera simultánea, y el movimiento resulta de la combinación de las velocidades de la ruedas izquierdas y derechas. El uso de pistas de deslizamiento se emplea como método de locomoción en vehículos de tipo oruga, en los cuales tanto la propulsión como la dirección se logran mediante estas pistas de deslizamiento. Las pistas actúan de forma análoga a ruedas de gran diámetro. La locomoción a través de pistas de deslizamiento resulta eficaz en superficies irregulares, donde muestra un desempeño sólido. En este contexto, la capacidad de propulsión se ve menos restringida por el deslizamiento y se reduce la incidencia de desgastes [10].

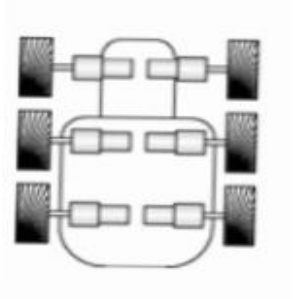

Figura 14. Sistema de locomoción tracción skid-steer [15]

Síncrona

Consiste en la actuación simultanea de todas las ruedas, que giran en forma síncrona tal como se muestra en la figura 15. La transmisión se logra mediante coronas de engranajes o con correas concéntricas. Cada rueda puede ser impulsada y dirigida individualmente, la configuración más común consta de tres ruedas directrices dispuestas en los vértices del triángulo equilátero, a menudo bajo una plataforma cilíndrica [10].

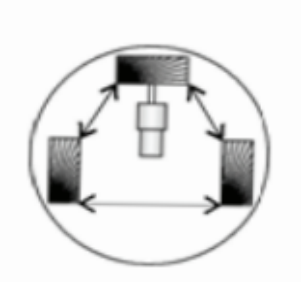

Figura 15. Sistema de locomoción síncrona [15]

2.2.2.2 Tipos de ruedas

Para diseñar un robot móvil lo primero que se debe tener en cuenta es el tipo de rueda que se va a utilizar para el diseño. Según el tipo de rueda se presentarán las diferentes características en el movimiento que desarrolla el robot, el cual puede ser o no un desplazamiento omnidireccional. En robots móviles las ruedas pueden estar conectadas a un actuador, como un motor eléctrico para impulsar su movimiento, o pueden no tener un actuador propio para su accionamiento, en el primer escenario se les denomina ruedas activas y en el segundo se las conoce como ruedas pasivas [9].

El diseño de ruedas únicas se podría clasificar en dos tipos de ruedas que serían las ruedas convencionales o estándar y las ruedas especiales omnidireccionales. Estos dos tipos de

ruedas únicas presentan una subclasificación de acuerdo a diversas características particulares y diseño tal como se presentarán a continuación.

Ruedas convencionales o estándar

Este tipo de ruedas pueden visualizarse o entenderse como neumáticos convencionales. Estas ruedas se distinguen en ruedas fijas, ruedas orientables centradas y ruedas orientables descentradas [9].

- **Ruedas Fijas:** pueden ser utilizadas como ruedas activas o pasivas, pero en ambos casos están montadas en posiciones fijas del vehículo como se muestra en la figura 16. Su eje de rotación tiene una dirección fija con respecto al sistema de coordenadas de la plataforma. Giran exclusivamente alrededor del eje de la rueda con una velocidad y orientación de giros constantes con respecto al chasis. Esto implica que carecen de mecanismos de articulación para cambiar su dirección [9].

Figura 16. Robot móvil con ruedas fijas [9]

- **Ruedas Orientables Centradas:** tienen un motor de accionamiento para su rotación y pueden orientarse con un ángulo alrededor de un eje perpendicular a su eje de rotación mediante otro motor. Giran alrededor del eje de la rueda con velocidad de giro y su orientación no es constante tal como se puede observar en la figura 17, varia a una razón con respecto al tiempo. Pueden funcionar como ruedas de tracción-dirección o solo como de dirección [9].

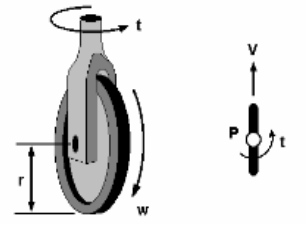

Figura 17. Ejemplo de ruedas orientables centradas [9]

- **Ruedas Orientables Descentradas o Castor:** pueden ser utilizadas como ruedas activas o pasivas. Poseen articulación de dirección, es decir, es orientable respecto al chasis del robot sin que su eje de dirección pase por el centro de rotación de la rueda. Giran alrededor del eje de la rueda con una velocidad y rotan alrededor del eje vertical ubicado a una distancia d desde el centro de la rueda con una velocidad tal como se muestra en la figura 18. Si d es libre, puede moverse en cualquier dirección, es decir se consideraría una rueda omnidireccional [9].

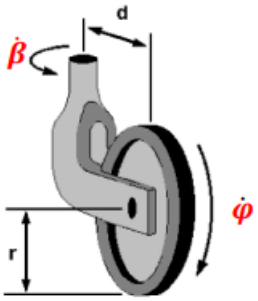

Figura 18. Ejemplo de ruedas orientables descentradas [9]

Las ruedas convencionales presentan una mayor capacidad de carga y una mayor resistencia a las imperfecciones del terreno en comparación con las configuraciones de ruedas especializadas. Pero debido a sus restricciones no holonómicas, no son ruedas verdaderamente omnidireccionales [9].

Ruedas especiales u omnidireccionales

Este tipo de ruedas poseen estructuras mecánicas únicas que incluyen rodillos o esferas, que permiten además del giro convencional también el giro lateral. Estas ruedas son concebidas de tal forma que presentan tracción activa en una dirección y movimiento pasivo en otra, lo que facilita una mayor capacidad de maniobra en entornos con obstáculos. En el sistema de desplazamiento de un robot móvil, estas ruedas pueden funcionar tanto como ruedas libres simples o como ruedas propulsoras. Tenemos tres tipos principales de ruedas especiales como: universal, mecanum y esférica [9].

- **Rueda Universal:** combina movimientos restringidos y libres durante su rotación. Como se muestra en la figura 19 está compuesta por pequeños rodillos que se encuentran pasivamente dispuestos a lo largo de su diámetro exterior y que están

conectados perpendicularmente al eje de rotación de la rueda. Estos rodillos pasivos tienen la capacidad de girar libremente en torno al eje de rotación, lo que genera un movimiento lateral de la rueda, De esta manera la rueda puede desplazarse en una dirección paralela a su propio eje, además de girar alrededor del mismo eje [9].

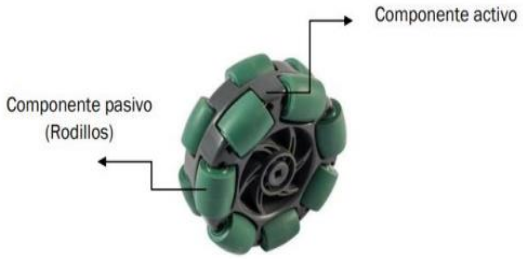

Figura 19. Ejemplo de rueda universal y sus componentes [9]

- **Rueda Mecanum o Sueca:** su diseño es parecido a la rueda universal, con la diferencia que los rodillos están montados en un ángulo distinto de 90 grados, usualmente de 45 grados. Este diseño de rueda posibilita una rotación controlada alrededor de su propio eje (con una velocidad determinada) y además permite otro tipo de rotación pasiva de los pequeños rodillos, como se muestra en la figura 20. Es capaz de girar en un plano vertical, así como también desplazarse en un plano horizontal [9].

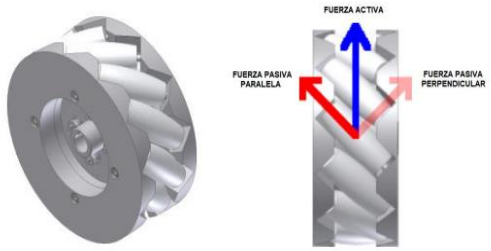

Figura 20. Ejemplo de rueda mecanum y sus fuerzas [9]

- **Rueda de Bola o Esférica:** como se puede observar en la figura 21 esta rueda no impone restricciones directas al movimiento, es decir, es omnidireccional como las ruedas castor, universal y mecanum. En este caso el eje de rotación de la rueda puede ubicarse en cualquier dirección arbitraria. Una alternativa para conseguir esta libertad de movimientos es mediante el empleo de un anillo activo impulsado por un motor y una caja de cambios que permiten transferir potencia a la bola a través de los rodillos

y la fricción, con lo cual esta puede girar en cualquier dirección de forma instantánea [9].

Figura 21. Ejemplo de rueda de bola o esférica [9]

El uso más común de las ruedas esféricas es como rueda pasiva, y en tal caso son llamadas ruedas caster o ruedas locas, cuando son utilizadas para este propósito, sirven como apoyo en el piso y para dar balance estático al robot [9].

2.2.2.3 Robot con accionamiento diferencial

El robot con accionamiento diferencial es una de las opciones de diseños más populares y preferidas por los investigadores para explorar nuevas estrategias de control debido a su cinemática simple. En la figura 22 se puede observar que es una estructura compuesta por dos ruedas fijas activas accionadas de forma independiente cada una por un motor y una rueda pasiva tipo castor o esférica que gira libremente según la velocidad y el movimiento del robot, con lo cual se evita que la plataforma bascule manteniendo el equilibrio y la estabilidad [9].

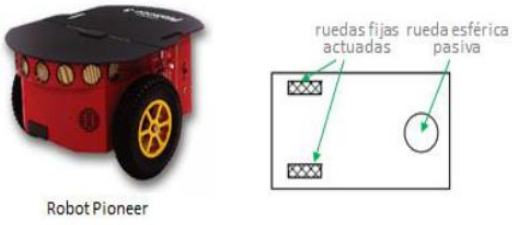

Figura 22. Robot móvil Pioneer con accionamiento diferencial [9]

Las principales ventajas del robot se pueden resumir en que tienen una estructura mecánica simple, son de bajo costo de fabricación, su modelo cinemático es simple y los errores sistemáticos estos robots son sencillos de calibrar, pero presentan ciertas desventajas, como la dificultad para moverse en terrenos irregulares. Además su orientación puede cambiar

bruscamente si una de las ruedas activas pierde contacto con el suelo y solo está disponible el movimiento bidireccional [9].

2.2.3 Cinemática de los robots móviles

En términos generales, la cinemática es una subdisciplina de la mecánica que analiza el movimiento de los cuerpos materiales sin centrarse en sus masas, momentos de inercia ni en las fuerzas o torques que causan dicho movimiento. En el ámbito de la robótica móvil, la cinemática se enfoca en el estudio de la evolución de la postura, la velocidad y la aceleración de los robots, sin tener en cuenta las fuerzas o torques involucrados. Es importante destacar que las ecuaciones cinemáticas están condicionadas por la geometría fija del robot en relación con un sistema de referencia global inmóvil.

El proceso de derivar un modelo cinemático para el movimiento de un robot móvil sigue un enfoque de lo específico a lo general, donde el movimiento de cada rueda individual contribuye al desplazamiento total del chasis y, al mismo tiempo, impone restricciones en su movimiento. Las ruedas están dispuestas según la geometría del chasis, y por lo tanto, las restricciones de cada una se combinan para definir las limitaciones del movimiento general del chasis. Sin embargo, las fuerzas y restricciones de cada rueda deben ser expresadas en relación con un sistema de referencia claro y consistente. Este aspecto es especialmente relevante en robótica móvil debido a la naturaleza autónoma y dinámica del robot, lo que requiere una correspondencia precisa entre los sistemas de referencia globales y locales. El proceso comienza con la definición formal de estos sistemas de referencia, y luego se utiliza este marco para determinar la cinemática de cada rueda, lo que a su vez permite calcular la cinemática total del robot móvil.

2.2.3.1 Representación de la posición y orientación

Cuando se modela un robot móvil con ruedas, el objetivo es definir la relación entre el movimiento de su chasis y sus ruedas con respecto a un sistema de referencia global, de forma que se pueda determinar la postura (posición y orientación) del robot en el plano.

En este análisis, el robot móvil se modela como un cuerpo rígido sobre ruedas que se desplaza en un plano horizontal. La dimensionalidad total del chasis del robot en dicho plano es tres:

dos dimensiones para la posición en el plano (x, y) y una para la orientación θ alrededor del eje vertical (que es perpendicular al plano). Aunque existen grados de libertad adicionales y flexibilidad debido a los ejes de las ruedas, las articulaciones de las ruedas orientables centradas y las descentradas, al referirnos al chasis del robot nos centramos exclusivamente en el cuerpo rígido, sin considerar las articulaciones ni los grados de libertad internos de las ruedas y otras partes del robot.

Para definir la postura o pose del robot en el plano, se establece una relación entre el sistema de referencia global (inercial, absoluto o fijo) del plano (I) y el sistema de referencia local del robot (R), tal como se ilustra en la figura 23.

Los ejes X_1 y Y_1 establecen una base inercial arbitraria en el plano, que constituyen el sistema de referencia global. Para determinar la posición del robot, se selecciona un punto R (el centro de masa) sobre el chasis como el punto de referencia para la posición, el cual se corresponde con el origen del sistema de referencia local del robot, formado por los ejes móviles X_R y Y_R.

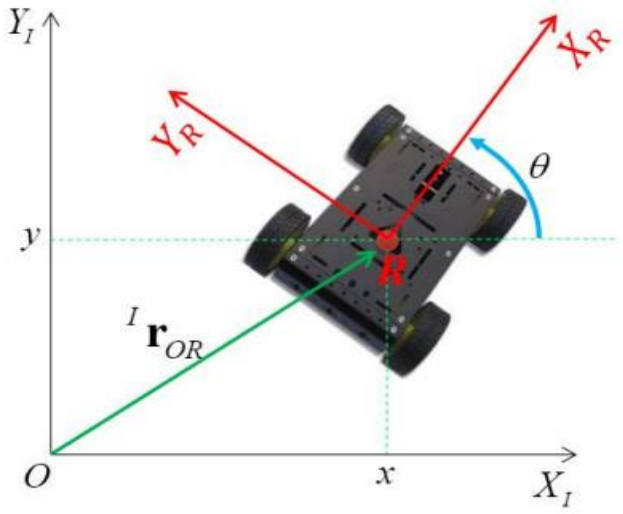

Figura 23. Representación de la postura del robot [9]

La posición de R en el sistema de referencia global se determina mediante las coordenadas x y y, mientras que la diferencia angular (o ángulo de orientación) entre los sistemas de referencia global y local se expresa mediante θ. La postura del robot puede describirse como un vector compuesto por estos tres elementos, como se muestra en (2.1), donde el superíndice I se utiliza para indicar que la postura del robot se refiere al sistema de referencia global.

$$\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{\theta} \end{bmatrix} \tag{2.1}$$

Para expresar el movimiento global del robot en función de las componentes del movimiento local, es necesario realizar un mapeo entre el movimiento a lo largo de los ejes del sistema de referencia global y el movimiento en los ejes del sistema de referencia local. Este mapeo depende de la postura actual del robot y se consigue mediante la matriz de rotación ortogonal alrededor del eje z, dada por:

$$R_{R_I}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

Esta matriz se puede emplear para transformar el movimiento del robot desde el sistema de referencia global $\{X_I, Y_I\}$ hacia el movimiento en el sistema de referencia local $\{X_R, Y_R\}$. Esta operación se representa de la siguiente manera:

$$\xi = R_{R_I}\xi \tag{2.3}$$

A partir de la ecuación anterior, también es posible determinar el movimiento del robot en el sistema de referencia global, basándose en el movimiento en su sistema de referencia local, mediante la siguiente expresión:

$$\xi = R_{R_I}\xi = (R_{R_I})^{-1}\xi \tag{2.4}$$

La estrategia consistirá en calcular primero la contribución de cada rueda del robot en su sistema de referencia local, ξ. No obstante, antes de esto, será necesario calcular la matriz de rotaciónI_{R_R}. En términos generales, calcular la inversa de una matriz puede ser complejo, pero en este caso es sencillo, ya que se trata simplemente de una transformación de ξa ξ, en la que se pueden aplicar las propiedades de la matriz de rotación.$R_{R_I} = (R_{R_I})^{-1} = (R_{R_I})^{T}$:

$$R_{R_I}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Este enfoque en el modelado cinemático puede ofrecer información sobre el movimiento de un robot al determinar el conjunto de movimientos posibles para cada tipo de chasis de robot. Para lograr esto, es necesario profundizar más, describiendo formalmente las restricciones al movimiento del robot impuestas por cada rueda.

2.2.3.2 Cinemática directa e inversa del robot

Como se ha mencionado, la cinemática vincula el movimiento de las ruedas (a través de los motores) con el movimiento del robot en el espacio cartesiano. En la robótica móvil con ruedas, se utiliza la cinemática diferencial, en la que se relacionan las velocidades en lugar de las posiciones u orientaciones, ya que las ruedas imponen restricciones sobre la velocidad del robot, y muchas de estas restricciones no son integrables (son restricciones no holonómicas). En términos generales, no es posible obtener relaciones de posición, ya que la velocidad no se puede integrar. Los tipos de análisis cinemático que deben resolverse para un robot móvil pueden clasificarse en directa e inversa.

Cinemática diferencial directa: dado un conjunto de características de las ruedas (como el radio, la ubicación geométrica en el robot, la distancia entre ellas, entre otros), un punto de referencia del robot (generalmente el centro geométrico), y las velocidades de giro de las ruedas, es posible determinar las velocidades del robot con respecto al sistema de referencia global. Es decir, permite calcular la velocidad cartesiana del robot $(\dot{x}, \dot{y}, \dot{\theta})$ a partir de las velocidades de las ruedas $(\dot{\varphi}_i, \dot{\beta}_i)$ y sus orientaciones β_i, tal como se muestra en la figura 24 Con esta información, se puede definir la posición del robot en el plano global.

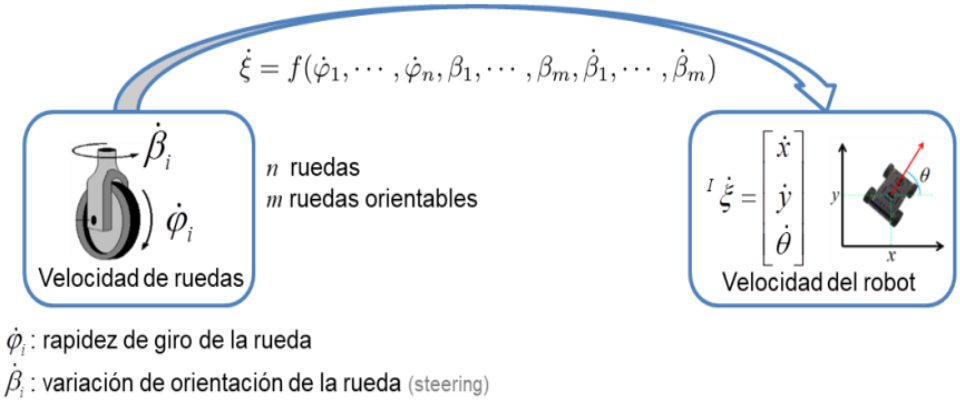

Figura 24. Representación de la cinemática diferencial directa [9]

Cinemática diferencial inversa: esta técnica permite calcular las velocidades $(\dot{\varphi}_i, \dot{\beta}_i)$ y las orientaciones β_i que deben aplicarse a las ruedas, para que el robot se desplace a una velocidad cartesiana deseada $(\dot{x}, \dot{y}, \dot{\theta})$, tal como se muestra en la figura 25.

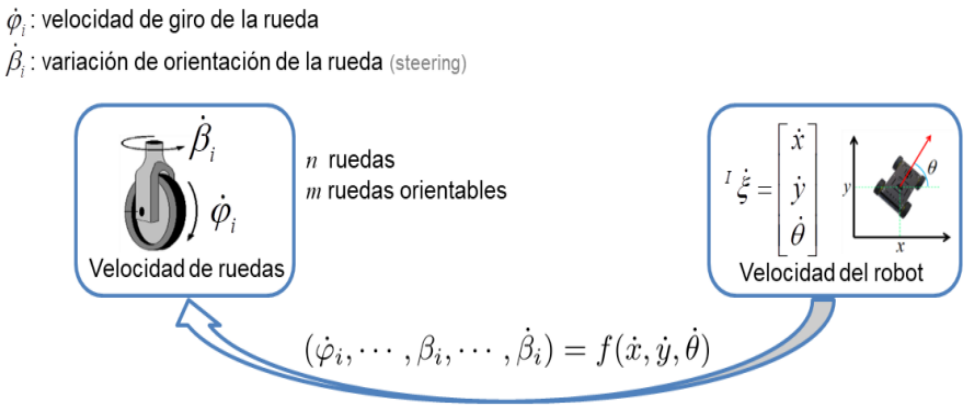

Figura 25. Representación de la cinemática diferencial inversa [9]

La cinemática establece la conexión entre la velocidad de giro y el cambio en la orientación de las ruedas con la velocidad cartesiana del robot. Esta relación está determinada por las restricciones que las ruedas imponen al movimiento del robot. Cada rueda establece restricciones (que pueden ser 0, 1 o 2) sobre la velocidad del robot, y estas restricciones dependen del tipo de rueda, así como de su posición y orientación en el chasis.

2.2.4 Seguimiento de trayectoria con robots móviles

El seguimiento de trayectorias consiste en generar un conjunto de puntos de referencia, definidos en términos de posición cartesiana x y y y orientación θ que el robot móvil debe alcanzar, durante su movimiento el robot evalúa continuamente su posición actual en relación con su posición deseada, ajustando su trayectoria para acercarse a la referencia. Este proceso se lleva a cabo en intervalos discretos denominados periodos de muestreo, donde se calculan las correcciones necesarias [11]. El éxito del seguimiento de trayectoria no solo depende del controlador, sino también de garantizar que las trayectorias estén libres de obstáculos, esto es fundamental para que los robots móviles puedan desplazarse sin interrupciones y logren un seguimiento progresivamente más preciso de la trayectoria deseada.

2.2.4.1 Tipos de trayectorias

Los robots móviles con ruedas pueden seguir diversas trayectorias, es importante tener en cuenta que la implementación exacta de estas trayectorias puede variar según el diseño y los algoritmos de control utilizados en cada robot especifico, A continuación, se describirán las trayectorias más comunes.

Trayectoria rectilínea: En esta trayectoria el robot móvil con rueda se desplaza en línea recta sin realizar giros, para seguir la trayectoria se utilizan sensores de distancia o retroalimentación de las ruedas para lograr mantener la rectitud de la trayectoria tal como se observa en la figura 26

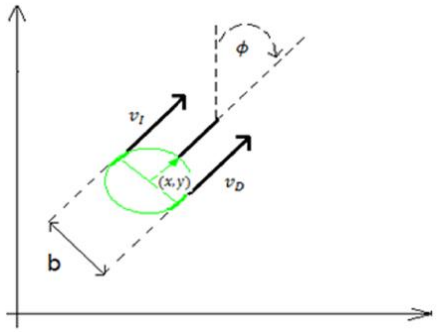

Figura 26. Ejemplo de trayectoria rectilínea [17]

Trayectoria curvilínea: Como se puede observar en la figura 27 el robot se desplaza en una trayectoria curva, lo cual puede lograrse mediante diferentes estrategias de control, uno de los métodos comunes para esto es el control de velocidad diferencial, donde se ajustan las velocidades de las ruedas izquierda y derecha para generar un giro suave.

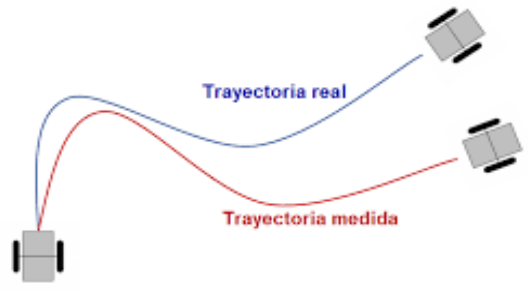

Figura 27. Ejemplo de un robot en una trayectoria curvilínea [16]

Trayectoria circular: El robot describe una trayectoria circular al girar continuamente con un radio fijo tal como se puede observar en la figura 28, esto se puede lograr al mantener una diferencia constante en las velocidades de las ruedas izquierda y derecha para producir una curva de radio constante.

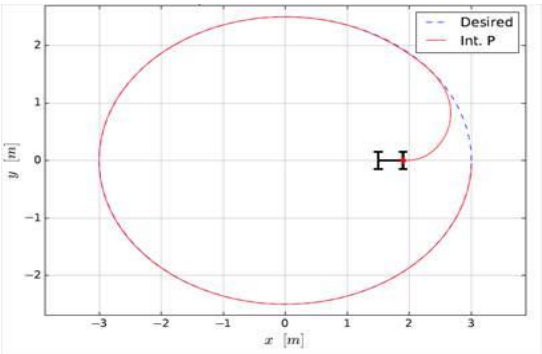

Figura 28. Ejemplo de un robot en una trayectoria [18]

Trayectoria elíptica: En esta trayectoria, el robot sigue una trayectoria en forma de elipse, esto se puede lograr mediante una combinación adecuada de velocidades y giros de las ruedas para generar curvas elípticas con radios de giro variables en la figura 29 se puede observar este tipo de trayectoria.

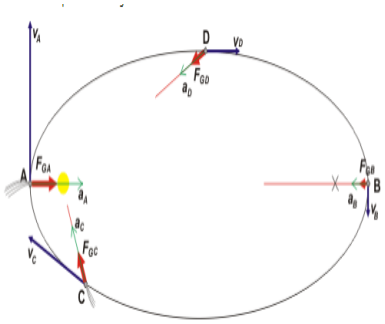

Figura 29. Ejemplo de una trayectoria elíptica [19]

Trayectoria poligonal: El robot sigue una trayectoria que está compuesta por segmentos rectos y giros a ángulos específicos tal como se muestra en la figura 30, para lograr cada segmento recto se debe mantener las velocidades constantes en las ruedas izquierda y derecha y los giros se realizan ajustando las velocidades de las ruedas para cambiar de dirección.

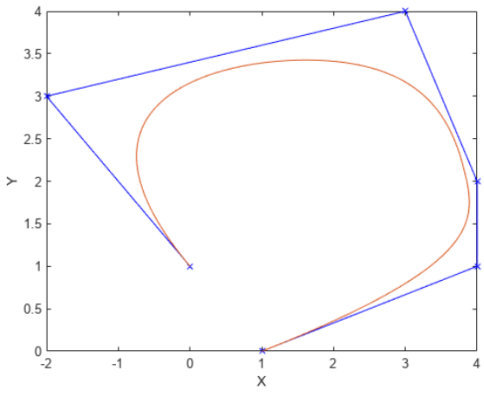

Figura 30. Ejemplo de una trayectoria poligonal [20]

Trayectoria de seguimiento de línea: En esta trayectoria el robot sigue una línea marcada en el suelo tal como se puede observar en la figura 31, utilizando sensores para detectar y ajustar su posición, los sensores de línea, como los sensores de reflexión infrarrojos, proporcionan retroalimentación para mantener el robot en la línea deseada.

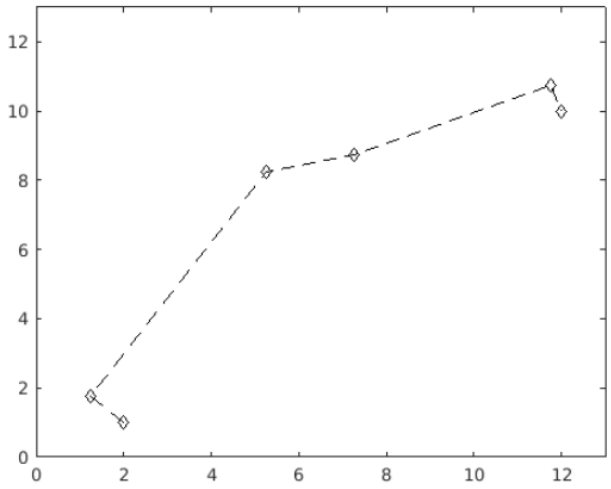

Figura 31. Ejemplo de una trayectoria de seguimiento de línea [21]

2.2.5 Modelado en el espacio de estados

La teoría de control moderna es aplicable a sistemas con múltiples entradas y salidas, que pueden ser tanto lineales como no lineales, y que pueden variar o no con el tiempo. En contraste, la teoría de control convencional se limita a sistemas de una sola entrada y una sola salida, que son invariables en el tiempo. Además, la teoría moderna se basa principalmente

en un enfoque en el dominio del tiempo, mientras que la teoría convencional utiliza el dominio de la frecuencia compleja [12].

2.2.5.1 Definiciones iniciales

Para profundizar en este tema se deben definir los conceptos de estado, variable de estado, vector de estado y espacio de estados.

Estado: Es un sistema dinámico, el estado es el conjunto mínimo de variables (denominadas variables de estado), que al ser conocidas en un momento $t = t_0$ junto con la información de la entrada para $t \geq t_0$ permiten determinar por completo el comportamiento del sistema en cualquier instante $t \geq t_0$

Variables de estado: Las variables de estado en un sistema dinámico constituyen el conjunto mínimo necesario de variables que determinan su estado. Si se requieren al menos n variables $x_1, x_2, \dots, x_n$ para describir completamente el comportamiento de un sistema dinámico de manera que conociendo la entrada para $t \geq t_0$ y especificando el estado inicial en $t = t_0$, se puede determinar el estado futuro del sistema, entonces estas n variables forman un conjunto de variables de estado

Cabe señalar que las variables de estado no tienen que ser necesariamente medibles u observables físicamente. Es posible elegir como variables de estado tanto aquellas que no representan cantidades físicas como las que no pueden medirse u observarse. Esta flexibilidad en la selección de variables de estado es una ventaja de los métodos en el espacio de estados. Sin embargo, en la práctica, es recomendable seleccionar variables de estado que pueden ser medidas físicamente, si es posible, ya que las leyes de control optimo requerirán retroalimentación de todas las variables de estado con una ponderación adecuada.

Vector de estado: Cuando se requieren n variables de estado para describir completamente el comportamiento de un sistema, estas n variables pueden agruparse como las componentes de un vector x, conocido como el vector de estado. Así el vector de estado es un vector que define de manera única el estado del sistema $x(t)$ en cualquier instante $t \geq t_0$, siempre que se conozca el estado en $t = t_0$, y se especifique la entrada $u(t)$ para $t \geq t_0$.

Espacio de estados: El espacio de estado es un espacio n-dimensional cuyas coordenadas están definidas por los ejes $x_1, x_2, \dots, x_n$, donde $x_1, x_2, \dots, x_n$, representan las variables de

estado. En este espacio cualquier estado del sistema se puede representar como un punto específico en el espacio de estados.

2.2.5.2 Ecuaciones en el espacio de estados

En el análisis del espacio de estados se estudian tres tipos de variables en el modelado de sistemas dinámicos: las variables de entrada, las variables de salida, y las variables de estado. La representación en el espacio de estados de un sistema no es única, aunque el número de variables de estado permanece constante para cualquier representación de un mismo sistema.

Para que un sistema dinámico conserve información de los valores de entrada en momentos previos $t \geq t_1$, debe incluir elementos que funcionen como memoria. En los sistemas de control en tiempo continuo, los integradores cumplen esta función de memoria y sus salidas pueden considerarse como las variables que describen el estado interno del sistema. Así las salidas de los integradores se emplean como variables de estado. El número de variables de estado necesarios para describir completamente la dinámica del sistema es igual al número de integradores que contiene.

Consideremos un sistema de múltiples entradas y múltiples salidas con n integradores, r entradas $u_1(t), u_2(t), \ldots\ldots, u_r(t)$, y m salidas $y_1(t), y_2(t), \ldots., y_m(t)$. Se define a las n salidas de los integradores como las variables de estado $x_1(t), x_2(t), \ldots., x_n(t)$. De este modo el sistema puede describirse mediante:

$$\begin{aligned} \dot{x}_1(t) &= f_1(x_1, x_2, \ldots, x_n;\ u_1, u_2, \ldots, u_r; t) \\ \dot{x}_2(t) &= f_2(x_1, x_2, \ldots, x_n;\ u_1, u_2, \ldots, u_r; t) \\ &\vdots \\ \dot{x}_n(t) &= f_n(x_1, x_2, \ldots, x_n;\ u_1, u_2, \ldots, u_r; t) \end{aligned} \tag{2.6}$$

Las salidas $y_1(t), y_2(t), \ldots, y_m(t)$ del sistema se pueden obtener mediante

$$\begin{aligned} y_1(t) &= g_1(x_1, x_2, \ldots, x_n;\ u_1, u_2, \ldots, u_r; t) \\ y_2(t) &= g_2(x_1, x_2, \ldots, x_n;\ u_1, u_2, \ldots, u_r; t) \\ &\vdots \\ y_m(t) &= g_m(x_1, x_2, \ldots, x_n;\ u_1, u_2, \ldots, u_r; t) \end{aligned} \tag{2.7}$$

Si se define

$$\boldsymbol{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix}, \qquad \mathbf{f}(\mathbf{x}, \mathbf{u}, t) = \begin{bmatrix} f_1(x_1, x_2, \dots, x_n;\ u_1, u_2, \dots, u_r; t) \\ f_2(x_1, x_2, \dots, x_n;\ u_1, u_2, \dots, u_r; t) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n;\ u_1, u_2, \dots, u_r; t) \end{bmatrix},$$

$$\boldsymbol{y}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix}, \qquad \mathbf{g}(\mathbf{x}, \mathbf{u}, t) = \begin{bmatrix} g_1(x_1, x_2, \dots, x_n;\ u_1, u_2, \dots, u_r; t) \\ g_2(x_1, x_2, \dots, x_n;\ u_1, u_2, \dots, u_r; t) \\ \vdots \\ g_m(x_1, x_2, \dots, x_n;\ u_1, u_2, \dots, u_r; t) \end{bmatrix}, \boldsymbol{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_r(t) \end{bmatrix}$$

Las ecuaciones (2.6) y (2.7) se convierten en

$$\dot{x}(t) = \mathbf{f}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{2.8}$$

$$\boldsymbol{y}(t) = \mathbf{g}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{2.9}$$

La ecuación (2.8) representa la ecuación de estado, mientras que la ecuación (2.9) corresponde a la ecuación de salida. Si las funciones vectoriales **f** y $\boldsymbol{g}$ incluyen el tiempo *t* de forma explícita, el sistema se clasifica como variante en el tiempo. Al linealizar las ecuaciones (2.8) y (2.9) en torno al estado de operación, se obtienen las ecuaciones de estado y de salida en su forma linealizada

$$\dot{x}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\boldsymbol{u}(t) \tag{2.10}$$

$$\boldsymbol{y}(t) = \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\boldsymbol{u}(t) \tag{2.11}$$

Donde $\mathbf{A}(t)$ se conoce como la matriz de estado, $\mathbf{B}(t)$ como la matriz de entrada, $\mathbf{C}(t)$ como la matriz de salida y $\mathbf{D}(t)$ como la matriz de transformación directa. La figura 32 muestra un diagrama de bloques que ilustra las ecuaciones (2.10) y (2.11)

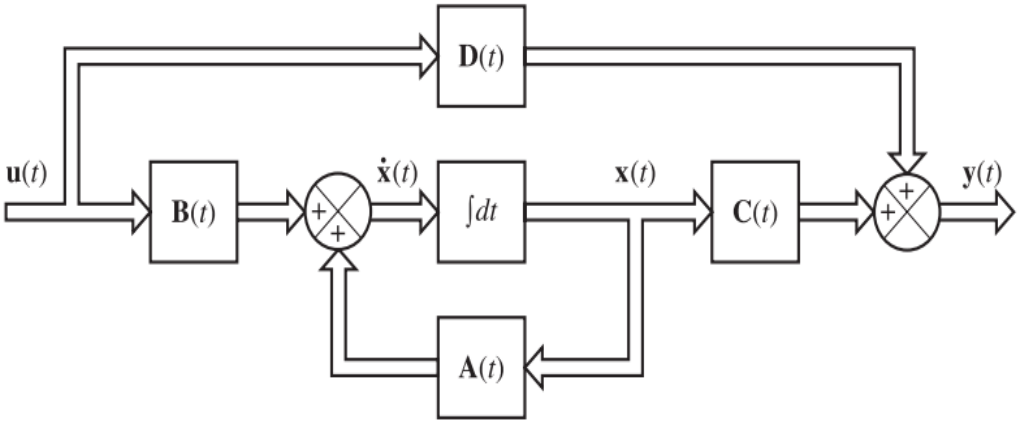

Figura 32. Diagrama de bloques de un sistema de control lineal en tiempo continúo representado en el espacio de estados [22]

Si las funciones vectoriales $\boldsymbol{f}$ y $\boldsymbol{g}$ no dependen explícitamente del tiempo t, el sistema se clasifica como invariante en el tiempo. En este caso, las ecuaciones (2.10) y (2.11) se simplifican a:

$$\dot{x}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\boldsymbol{u}(t) \tag{2.12}$$

$$\dot{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\boldsymbol{u}(t) \tag{2.13}$$

La ecuación (2.12) corresponde a la ecuación de estado del sistema lineal e invariante en el tiempo, mientras que la ecuación (2.13) representa la ecuación de salida de este mismo sistema.

2.2.5.3 Correlación entre funciones de transferencia y ecuaciones en el espacio de estados

A continuación, se explicará cómo obtener la función de transferencia de un sistema con una única entrada y una única salida a partir de sus ecuaciones en el espacio de estados. Se considerará el sistema cuya función de transferencia se obtiene a través de:

$$\frac{Y(s)}{U(s)} = G(s) \tag{2.14}$$

Este sistema se describe en el espacio de estados mediante las siguientes ecuaciones:

$$\dot{x}(t) = \mathbf{A}\mathbf{x} + \mathbf{B}\boldsymbol{u} \tag{2.15}$$

$$y(t) = \mathbf{C}\mathbf{x} + \mathrm{D}u \tag{2.16}$$

donde x representa el vector de estado, u la entrada, e y la salida. Las transformadas de Laplace de las ecuaciones (2.15) y (2.16) se obtienen a través de:

$$s\boldsymbol{X}(s) - x(0) = \mathbf{A}\mathbf{X}(\mathbf{s}) + \mathbf{B}U(s) \quad (2.17)$$

$$Y(s) = \mathbf{C}\mathbf{x}(\mathbf{s}) + \mathrm{D}u(\mathbf{s}) \quad (2.18)$$

Dado que la función de transferencia se definió previamente como el cociente entre la transformada de Laplace de la salida y la de la entrada, bajo la suposición de condiciones iniciales nulas, $x(0)$ en la ecuación (2.17) se considera cero. Por lo tanto, se obtiene:

$$s\boldsymbol{X}(s) - \mathbf{A}\mathbf{X}(s) = \mathbf{B}U(\mathrm{s})$$

O también

$$(s\mathbf{I} - \mathbf{A})\mathbf{X}(\mathbf{s}) = \mathbf{B}U(\mathrm{s})$$

Premultiplicando ambos lados de esta última ecuación por $(\mathrm{s}\mathbf{I} - \mathbf{A})^{-1}$, se obtiene:

$$\boldsymbol{X}(s) = (\mathrm{s}\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}U(s) \quad (2.19)$$

Al sustituir la ecuación (2.19) en la ecuación (2.18), se obtiene:

$$Y(s) = [\boldsymbol{C}(\mathrm{s}\mathbf{I} - \mathbf{A})^{-1}\boldsymbol{B} + D]U(s) \quad (2.20)$$

Al comparar la ecuación (2.20) con la ecuación (2.14) se puede observar que:

$$G(s) = \boldsymbol{C}(\mathrm{s}\mathbf{I} - \mathbf{A})^{-1}\boldsymbol{B} + D \quad (2.21)$$

Esta es la expresión de la función de transferencia en términos de $\mathbf{A}, \mathbf{B}. \mathbf{C}$ y D.

Se puede notar que el segundo término de la ecuación (2.21) incluye $(\mathrm{s}\mathbf{I} - \mathbf{A})^{-1}$. Por lo tanto, $G(s)$ se expresa como

$$G(s) = \frac{Q(s)}{|s\mathbf{I} - \mathbf{A}|}$$

Donde $Q(s)$ es un polinomio en s. Por tanto, $|s\mathbf{I} - \mathbf{A}|$ corresponde al polinomio característico de $G(s)$. En otras palabras, los valores propios de $\mathbf{A}$ son iguales a los polos de $G(s)$.

Matriz de transferencia A continuación, se considera un sistema con múltiples entradas y salidas. Suponiendo que existen r entradas $u_1, u_2, \dots, u_r$ y m salidas $y_1, y_2, \dots, y_m$. Se define:

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \qquad \boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_r \end{bmatrix}$$

La matriz de transferencia $G(s)$ establece la relación entre la salida $\boldsymbol{Y}(s)$ con la entrada $\mathbf{U}(s)$, o también

$$\boldsymbol{Y}(\mathrm{s}) = \mathbf{G}(\boldsymbol{s})\mathbf{U}(\boldsymbol{s})$$

donde $G(s)$ esta dada por

$$\boldsymbol{G}(s) = \boldsymbol{C}(\mathrm{s}\mathbf{I} - \mathbf{A})^{-1}\boldsymbol{B} + D$$

El cálculo de esta ecuación es igual al de la ecuación (22). Dado que el vector de entrada u tiene dimensión r y el vector de salida y tiene dimensión m, la matriz de transferencia $G(s)$ es una matriz de $m \times r$

2.2.5.4 Representación en el espacio de estados de sistemas de ecuaciones diferenciales escalares

Un sistema dinámico compuesto por un numero finito de parámetros concentrados se describe mediante un conjunto de ecuaciones diferenciales, donde el tiempo es la variable independiente. Usando notación matricial, una ecuación diferencial de orden n puede ser representada como una ecuación diferencial matricial de primer orden, si n elementos del vector corresponden a un conjunto de variables de estado, la ecuación diferencial matricial se considera una ecuación de estado. En esta sección se presentarán métodos para obtener representaciones en el espacio de estados de sistemas de tiempo continuo.

Considerando el siguiente sistema de n-esimo orden:

$$y^{(n)} + a_1\, y^{(n-1)} + \cdots + a_{n-1}\dot{y} + a_n y = u \qquad (2.22)$$

Si se asume que el conocimiento de $y(0), \dot{y}(0), \ldots,\ y^{(n-1)}\,(0)$ junto con la entrada $u(t)$ para $t \geq 0$, determina completamente el comportamiento futuro del sistema, se pueden considerar $y(t), \dot{y}(t), \ldots,\ y^{(n-1)}(t)$ como un conjunto de n variables de estado. Matemáticamente, esta elección de variables de estado es muy conveniente. No obstante, en la práctica, debido a que los términos que contienen derivadas de orden superior no son exactos y a los efectos de

ruido inherentes en cualquier situación práctica, esta elección de variables de estado puede no ser lo más adecuado.

Si se define

$$x_1 = y$$

$$x_2 = \dot{y}$$

$$\vdots$$

$$x_n = y^{(n-1)}$$

Por lo tanto, la ecuación (2.22) se puede expresar como

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

$$\dot{x}_2 = x_3$$

$$\vdots$$

$$\dot{x}_{n-1} = x_n$$

$$\dot{x}_n = a_n x_1 - \cdots - a_1 x_n + u$$

O también

$$\dot{\boldsymbol{x}} = \mathbf{Ax} + \mathbf{Bu} \qquad (2.23)$$

Donde

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & \cdots & -a_1 \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

La salida se determina a través

$$y = [1 \quad 0 \quad \cdots \quad 0] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

O también

$$y = \mathbf{Cx} \tag{2.24}$$

donde

$$\boldsymbol{C} = [1 \quad 0 \quad \cdots \quad 0]$$

Cabe señalar que en la ecuación 17, D es igual a cero. La ecuación (24) representa la ecuación de estado como una ecuación diferencial de primer orden, mientras que la ecuación (25) corresponde a la ecuación de salida, que es algebraica. Es importante destacar que esta es la representación en el espacio de estados de la función de transferencia del sistema

$$\frac{Y(s)}{U(s)} = \frac{1}{s^n + a_1 s^{n-1} + \cdots + a_{n-1}s + a_n}$$

También se puede obtener utilizando las ecuaciones (2.23) y (2.24)

Si la ecuación diferencial del sistema incluye derivadas de la función de excitación, como

$$y^{(n)} + a_1 y^{(n-1)} + \cdots + a_{n-1}\dot{y} + a_n y = b_0 u^{(n)} + b_1 u^{(n-1)} + \cdots + b_{n-1}\dot{u} + b_n u \tag{2.25}$$

El principal desafío al definir las variables de estado en este caso es la presencia de términos derivados. Las variables de estado deben seleccionarse de manera que eliminen las derivadas de u en la ecuación de estado. Una manera de obtener tanto una ecuación de estado como una ecuación de salida consiste en definir las siguientes n variables como un conjunto de n variables de estado

$$\begin{aligned}
x_1 &= y - \beta_0 u \\
x_2 &= \dot{y} - \beta_0 \dot{u} - \beta_1 u = \dot{x}_1 - \beta_1 u \\
x_3 &= \ddot{y} - \beta_0 \ddot{u} - \beta_1 \dot{u} - \beta_2 u = \dot{x}_2 - \beta_2 u \\
&\vdots \\
x_n &= y^{(n-1)} - \beta_0 u^{(n-1)} - \beta_1 u^{(n-2)} - \cdots - \beta_{n-2}\dot{u} - \beta_{n-1} u = \dot{x}_{n-1} - \beta_{n-1} u
\end{aligned} \tag{2.26}$$

donde β_0, β_1, β_2, …, β_{n-1} se obtienen a partir de

$$\begin{aligned}
\beta_0 &= b_0 \\
\beta_1 &= b_1 - a_1\beta_0 \\
\beta_2 &= b_2 - a_1\beta_1 - a_2\beta_2 \\
\beta_3 &= b_3 - a_1\beta_2 - a_2\beta_1
\end{aligned} \tag{2.27}$$

$$\vdots$$

$$\beta_{n-1} = b_{n-1} - a_1\beta_{n-2} - \cdots - a_{n-2}\beta_1 - a_{n-1}\beta_0$$

Al elegir estas variables de estado, se asegura la existencia y unicidad de la solución para la ecuación de estado, es importante considerar que esta no es la única posible elección de un conjunto de variables de estado. Con esta selección actual de variables de estado, se obtiene:

$$\begin{aligned} \dot{x}_1 &= x_2 + \beta_1 u \\ \dot{x}_2 &= x_3 + \beta_2 u \\ &\vdots \\ \dot{x}_{n-1} &= x_n + \beta_{n-1} u \\ \text{s}\dot{x}_n &= -a_n x_1 - a_{n-1} x_2 - \cdots - a_{n-1}\beta_1 - a_{n-1}\beta_0 \end{aligned} \tag{2.28}$$

Donde β_0 esta dado por

$$\beta_n = b_n - a_1\beta_{n-1} - \cdots - a_{n-1}\beta_1 - a_{n-1}\beta_0$$

Expresado en términos de ecuaciones matriciales. La ecuación (2.28) y la ecuación de salida pueden escribirse como

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_n & -a_{\boldsymbol{n-1}} & -a_{\boldsymbol{n-2}} & \cdots & -a_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{n-1} \\ \beta_n \end{bmatrix} u$$

$$y = [1 \quad 0 \quad \cdots \quad 0] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \beta_0 u$$

O también

$$\dot{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\boldsymbol{u} \tag{2.29}$$

$$y = \mathbf{C}\mathbf{x} + \mathrm{D}u \tag{2.30}$$

Donde

$$\mathbf{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix}, \qquad \mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & \cdots & -a_1 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{n-1} \\ \beta_n \end{bmatrix}, \quad \boldsymbol{C} = [1 \quad 0 \quad \cdots \quad 0], \qquad D = B_0 = b_0$$

En esta representación en el espacio de estado las matrices A y C son idénticas a las del sistema en la ecuación (2.22). Las derivadas en el término derecho de la ecuación (2.25) únicamente afectan elementos de la matriz $\boldsymbol{B}$**.** Cabe destacar que la representación en el espacio de estados para la función de transferencia se puede determinar también mediante las ecuaciones (2.29) y (2.30)

$$\frac{Y(s)}{U(s)} = \frac{b_0 s^n + b_1 s^{n-1} + b_{n-1} s + b_n}{s^n + a_1 s^{n-1} + \cdots + a_{n-1} s + a_n}$$

2.2.5.5 Análisis de sistemas de control en el espacio de estado

Considerando un sistema definido por

$$y^{(n)} + a_1 y^{(n-1)} + \cdots + a_{n-1}\dot{y} + a_n y = b_0 u^{(n)} + b_1 u^{(n-1)} + \cdots + b_{n-1}\dot{u} + b_n u \quad (2.31)$$

donde u representa la entrada y y es la salida. Esta ecuación también se puede expresar de la siguiente manera

$$\frac{Y(s)}{U(s)} = \frac{b_0 s^n + b_1 s^{n-1} + \cdots + b_{n-1} s + b_n}{s^n + a_1 s^{n-1} + \cdots + a_{n-1} s + a_n} \quad (2.32)$$

A continuación, se muestran las representaciones en el espacio de estados del sistema definido por las Ecuaciones (2.31) o (2.32), en las formas canónica controlable, canónica observable y canónica diagonal (o de Jordan).

Forma canónica controlable. La siguiente representación en el espacio de estados es conocida como la forma canónica controlable:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_n & -a_{n-1} & -a_{n-2} & \cdots & -a_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u \qquad (2.33)$$

$$y = [b_n - a_n b_0 \vdots b_{n-1} - a_{n-1} b_0 \vdots \cdots \vdots b_1 - a_1 b_0] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + b_0 u \qquad (2.34)$$

La forma canónica controlable es fundamental para analizar el método de asignación de polos en el diseño de sistemas de control.

Forma canónica observable. Esta representación en el espacio de estados se conoce como la forma canónica observable:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} 0 & 0 & \cdots & 0 & -a_n \\ 1 & 0 & \cdots & 0 & -a_{n-1} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_n - a_n b_0 \\ b_{n-1} - a_{n-1} b_0 \\ \vdots \\ b_1 - a_1 b_0 \end{bmatrix} u \qquad (2.35)$$

$$y = [0 \quad 0 \quad \cdots \quad 0 \quad 1] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + b_0 u \qquad (2.36)$$

Es importante notar que la matriz de estado de $n \times n$ en la ecuación de estado obtenida a través de la Ecuación (2.35) es la transpuesta de la ecuación de estado definida por la Ecuación (2.33).

Forma canónica diagonal. Considérese un sistema representado por la función de transferencia dada en la Ecuación (2.32), donde el polinomio del denominador tiene únicamente raíces distintas. En este caso, la ecuación (2.32) se puede expresar como:

$$\begin{aligned} \frac{Y(s)}{U(s)} &= \frac{b_0 s^n + b_1 s^{n-1} + \cdots + b_{n-1} s + b_n}{(s + p_1)(s + p_2) \cdots (s + p_n)} \\ &= b_0 + \frac{C_1}{s + p_1} + \frac{C_2}{s + p_2} + \cdots + \frac{C_n}{s + p_n} \end{aligned} \qquad (2.37)$$

La forma canónica diagonal de la representación en el espacio de estados de este sistema está determinada por

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \vdots\\ \dot{x}_n\end{bmatrix}=\begin{bmatrix}-p_1 & \cdot & \cdot & 0\\ \vdots & -p_2 & \ddots & \vdots\\ 0 & \cdot & \cdot & -p_n\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ \vdots\\ x_n\end{bmatrix}+\begin{bmatrix}1\\ 1\\ \vdots\\ 1\end{bmatrix}u \tag{2.38}$$

$$y=[C_1 \quad C_2 \quad \cdots \quad C_n]\begin{bmatrix}x_1\\ x_2\\ \vdots\\ X_n\end{bmatrix}+b_0u \tag{2.39}$$

Forma canónica de Jordán Aquí se analiza el caso en que el polinomio denominador en la Ecuación (2.32) presenta raíces múltiples. En estas condiciones, la forma canónica diagonal debe ajustarse, adoptando la forma canónica de Jordan. Supongamos, por ejemplo, que todos los valores p_i, excepto los tres primeros, son distintos entre sí, es decir, $p_1 = p_2 = p_3$. Con esto, la expresión factorizada de $Y(s)/U(s)$ se convierte en la siguiente.

$$\frac{Y(s)}{U(s)}=\frac{b_0s^n+b_1s^{n-1}+\cdots+b_{n-1}s+b_n}{(s+p_1)^3(s+p_4)(s+p_5)\cdots(s+p_n)}$$

La descomposición en fracciones simples de esta última ecuación se transforma en

$$\frac{Y(s)}{U(s)}=b_0+\frac{C_1}{(s+p_1)^3}+\frac{C_2}{(s+p_1)^2}+\frac{C_3}{s+p_1}+\frac{C_4}{s+p_4}+\cdots+\frac{C_n}{s+p_n}$$

Se obtiene una representación en el espacio de estados de este sistema en su forma canónica de Jordan utilizando

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\\ \vdots\\ \dot{x}_n\end{bmatrix}=\begin{bmatrix}-p_1 & 1 & 0 & 0 & \cdots & 0\\ 0 & -p_1 & 1 & 0 & \cdots & 0\\ 0 & 0 & -p_1 & 0 & \cdots & 0\\ 0 & \cdots & 0 & -p_4 & \ddots & 0\\ \vdots & \cdot & \vdots & \cdot & \cdot & \cdot\\ 0 & \cdots & 0 & 0 & \cdot & -p_n\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\\ \vdots\\ x_n\end{bmatrix}+\begin{bmatrix}0\\ 0\\ 1\\ 1\\ \vdots\\ 1\end{bmatrix}u \tag{2.40}$$

$$y=[C_1 \quad C_2 \quad \cdots \quad C_n]\begin{bmatrix}x_1\\ x_2\\ \vdots\\ x_n\end{bmatrix}+b_0u \tag{2.41}$$

Considere el sistema que se define mediante

$$\frac{Y(s)}{U(s)} = \frac{s+3}{s^2+3s+2}$$

Determine las representaciones en el espacio de estados en la forma canónica controlable, la forma canónica observable y la forma canónica diagonal.

Forma canónica controlable

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}0 & 1\\ -2 & -3\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}0\\ 1\end{bmatrix}u(t)$$

$$y(t) = \begin{bmatrix}3 & 1\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix}$$

Forma canónica observable

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}0 & -2\\ 1 & -3\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}3\\ 1\end{bmatrix}u(t)$$

$$y(t) = \begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix}$$

Forma canónica diagonal

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}-1 & 0\\ 0 & -2\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}1\\ 1\end{bmatrix}u(t)$$

$$y(t) = \begin{bmatrix}-2 & -1\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix}$$

2.2.5.6 Controlabilidad

Un sistema se considera controlable en el tiempo t_0 si es posible llevarlo desde cualquier estado inicial $x(t_0)$a cualquier otro estado, utilizando un vector de control sin restricciones, en un intervalo de tiempo finito.

Se considera que un sistema descrito por la ecuación (2.29) es de estado controlable en $t = t_0$, si es posible generar una señal de control sin restricciones que permita transferir el estado inicial a cualquier estado final en un intervalo de tiempo finito $t_0 \leq t \leq t_1$. Si todos los estados son controlables, el sistema se considera completamente controlable.

Si el sistema es completamente controlable, entonces, para cualquier estado inicial $x(0)$, debe cumplirse que el rango de la matriz $n \times n$ debe ser n

$$[\mathbf{B} \vdots \ \mathbf{AB} \vdots \ \cdots \vdots \ \mathbf{A}^{n-1}\boldsymbol{B}]$$

A partir de este análisis, se puede deducir la condición para la controlabilidad completa del estado de la siguiente manera. El sistema descrito por la ecuación (1.12) es completamente controlable si y solo si los vectores $\mathbf{B}, \mathbf{AB}, \cdots, \mathbf{A}^{\boldsymbol{n-1}}\boldsymbol{B}$ son linealmente independientes, o si la matriz $n \times n$ es de rango n

2.2.6 Diseño de controladores en el espacio de estados

El diseño de controladores en el espacio de estados es una metodología moderna en la ingeniería de control que permite analizar y desarrollar sistemas desde una perspectiva matemática y estructurada. El análisis en el espacio de estados incluye herramientas y técnicas para estudiar la estabilidad, la respuesta transitoria y las características bajo condiciones específicas. Para plantas en forma de variables de fase, la representación y el diseño del controlador se simplifican, facilitando la implementación de estrategias como la realimentación de estados. Este enfoque resulta especialmente útil en aplicaciones donde se requieran respuestas precisas

2.2.6.1 Diseño del controlador

Esta sección muestra cómo introducir parámetros adicionales en un sistema para que podamos controlar la ubicación de todos los polos del lazo cerrado [13]. Un sistema de control por retroalimentación de orden n tiene una ecuación característica de lazo cerrado de orden n de la forma

$$s^n + a_{n-1}s^{n-1} + \cdots + a_1 s + a_0 = 0 \qquad (2.42)$$

Dado que el coeficiente de la mayor potencia de s es la unidad, existen n coeficientes cuyos valores determinan la ubicación de los polos del sistema en el lazo cerrado. Así, si podemos introducir n parámetros ajustables en el sistema y relacionarlos con los coeficientes en la ecuación (2.42), todos los polos del sistema de lazo cerrado pueden ser ubicados en cualquier lugar deseado.

Topología para la colocación de polos

Para sentar las bases del enfoque, consideremos una planta representada en el espacio de estados por

$$\dot{\boldsymbol{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u \qquad (2.43a)$$

$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} \qquad (2.43b)$$

y mostrada pictóricamente en la figura 33 y 34, donde las líneas delgadas son escalares y las líneas gruesas son vectores.

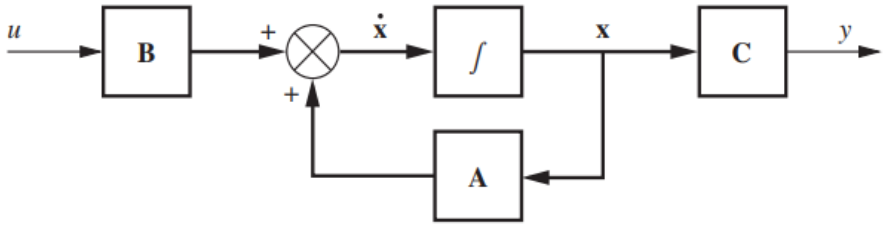

Figura 33. Representación en espacio de estados de una planta [23]

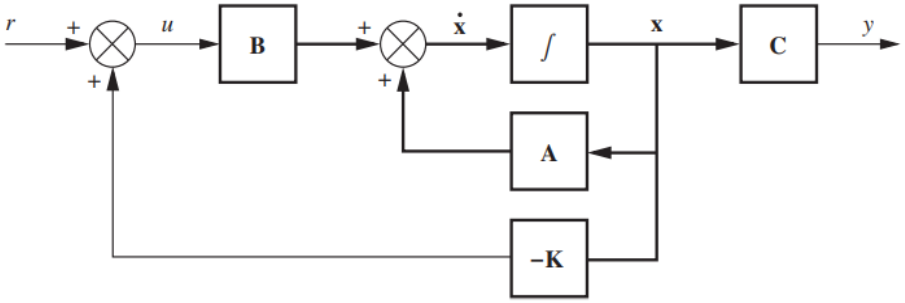

Figura 34. Planta con retroalimentación de estados [23]

En un sistema típico de control por retroalimentación, la salida, y, se retroalimenta al punto de suma. Es en este momento cuando la topología del diseño cambia. En lugar de retroalimentar y, ¿qué pasaría si retroalimentamos todas las variables de estado? Si cada variable de estado se retroalimenta al control, u, a través de una ganancia k_i, habría n ganancias k_i que podrían ajustarse para obtener los valores deseados de los polos del lazo cerrado. La retroalimentación a través de las ganancias k_i se representa en la figura 34 por el vector de retroalimentación $-K$.

Las ecuaciones de estado para el sistema de lazo cerrado de la figura 33 pueden escribirse por inspección como

$$\dot{\boldsymbol{x}} = \mathbf{Ax} + \mathbf{B}u = \mathbf{Ax} + \mathbf{B}(-\mathbf{Kx} + \mathbf{r}) = (\mathbf{A} - \mathbf{BK})\mathbf{x} + \mathbf{Br} \qquad (2.44a)$$

$$\boldsymbol{y} = \boldsymbol{Cx} \qquad (2.44b)$$

Antes de continuar, se debe tener una idea de cómo se implementa realmente el sistema de retroalimentación de la figura 34. Como, por ejemplo, supongamos un gráfico de flujo de señales de planta en forma de variable de fase, como se muestra en la figura 33. Luego, cada variable de estado se retroalimenta a la entrada de la planta, u, a través de una ganancia k_i, como se muestra en la figura 36. Aunque cubriremos otras representaciones más adelante en el capítulo, la forma de variable de fase, con su típica matriz de sistema de compañero inferior, o la forma canónica del controlador, con su típica matriz de sistema de compañero superior, ofrece la evaluación más simple de las ganancias de retroalimentación. En la discusión que sigue, utilizamos la forma de variable de fase para desarrollar y demostrar los conceptos. Los problemas al final del capítulo te darán la oportunidad de desarrollar y probar los conceptos para la forma canónica del controlador.

El diseño de retroalimentación de variables de estado para la colocación de polos en lazo cerrado consiste en igualar la ecuación característica de un sistema de lazo cerrado, como la que se muestra en la figura 36, a una ecuación característica deseada y luego encontrar los valores de las ganancias de retroalimentación, k_i..

Si una planta como la que se muestra en la figura 35 es de alto orden y no está representada en forma de variable de fase o en forma canónica del controlador, la solución para las k_i puede ser compleja. Por lo tanto, es recomendable transformar el sistema a cualquiera de estas formas, diseñar las k_i y luego transformar el sistema de nuevo a su representación original

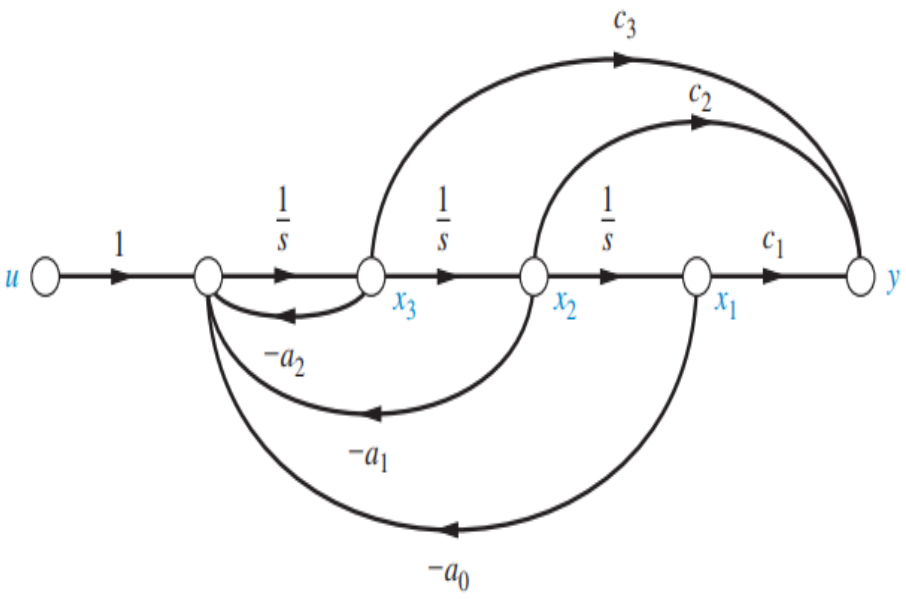

Figura 36. Representación en variables de fase para la planta [23]

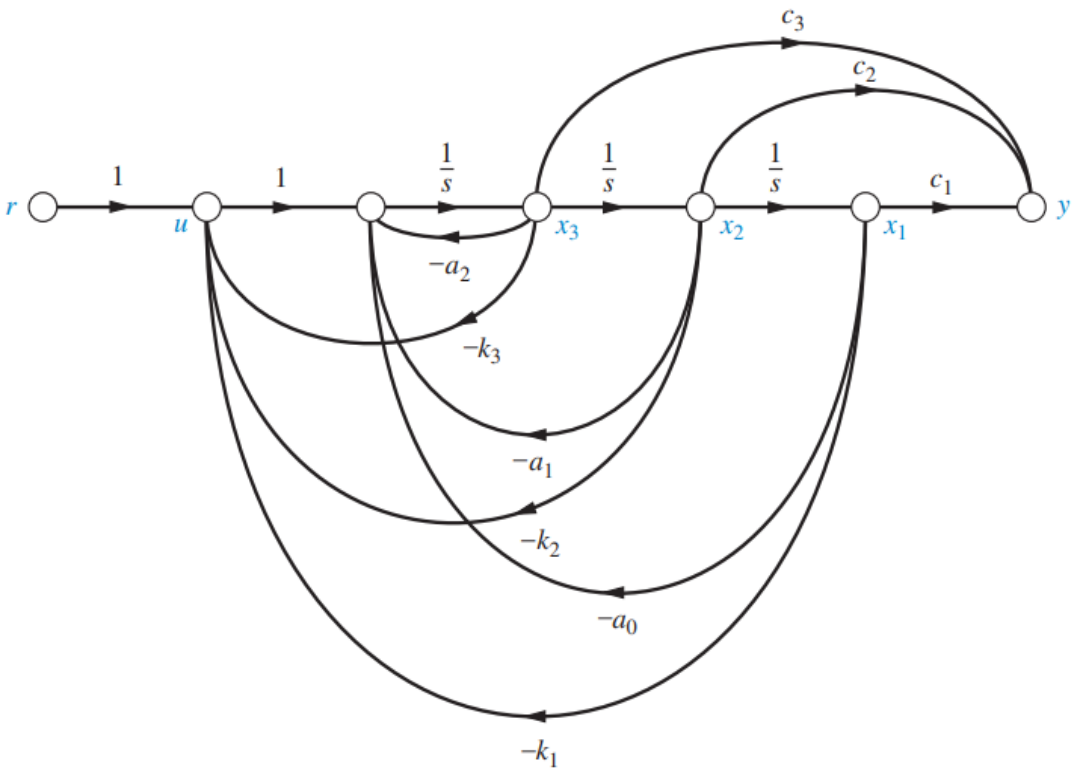

Figura 35. Planta con retroalimentación de variables de estado [23]

Colocación de polos para plantas en forma de variables de fase

Para aplicar la metodología de colocación de polos a plantas representadas en forma de variable de fase, seguimos los siguientes pasos:

1. Representar la planta en forma de variable de fase.
2. Retroalimentar cada variable de fase a la entrada de la planta a través de una ganancia k_i.
3. Encontrar la ecuación característica para el sistema de lazo cerrado representado en el Paso 2.
4. Decidir sobre las ubicaciones de todos los polos del lazo cerrado y determinar una ecuación característica equivalente.
5. Igualar los coeficientes correspondientes de las ecuaciones características de los Pasos 3 y 4 y resolver para k_i.

Siguiendo estos pasos, la representación en forma de variable de fase de la planta se da por la ecuación (2.43), con:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-1} \end{bmatrix}; \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}:$$

$$\boldsymbol{C} = [C_1 \quad C_2 \quad \cdots \quad C_n] \tag{2.45}$$

La ecuación característica de la planta es, por lo tanto,

$$s^n + a_{n-1}s^{n-1} + \cdots + a_1 s + a_0 = 0 \tag{2.46}$$

Ahora forma el sistema de lazo cerrado retroalimentando cada variable de estado a u, formando

$$u = -\mathbf{Kx} \tag{2.47}$$

Donde

$$\mathbf{K} = [\boldsymbol{k_1} \quad \boldsymbol{k_2} \quad \cdots \quad \boldsymbol{k_n}] \tag{2.48}$$

Las k_i son las ganancias de retroalimentación de las variables de fase. Usando la ecuación (2.44a) con las ecuaciones (2.45) y (2.48), la matriz del sistema, $\mathbf{A} - \mathbf{BK}$, para el sistema de lazo cerrado es

$$\mathbf{A} - \mathbf{BK} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -(a_0 + k_1) & -(a_1 + k_2) & -(a_2 + k_3) & \cdots & -(a_{n-1} + k_n) \end{bmatrix} \quad (2.49)$$

Dado que la ecuación (2.49) está en forma de variable de fase, la ecuación característica del sistema de lazo cerrado se puede escribir por inspección como

$$\det\big(sI - (A - BK)\big) = s^n + (a_{n-1} + k_n)s^{n-1} + (a_{n-2} + k_{n-1})s^{n-2} + \cdots (a_1 + k_2)s + (a_0 + k_i) = 0 \quad (2.50)$$

Observa la relación entre las ecuaciones (2.46) y (2.50). Para plantas representadas en forma de variable de fase, podemos escribir por inspección la ecuación característica del lazo cerrado a partir de la ecuación característica del lazo abierto, sumando el k_i correspondiente a cada coeficiente.

Ahora, supongamos que la ecuación característica deseada para la colocación adecuada de polos es

$$s^n + d_{n-1}s^{n-1} + d_{n-2}s^{n-2} + \cdots + d_2 s^2 + d_1 s + d_0 = 0 \quad (2.51)$$

donde los d_i son los coeficientes deseados. Igualando las ecuaciones (2.50) y (2.51), obtenemos

$$d_i = a_i + k_{i+1} \qquad i = 0, 1, 2, \cdots, n-1 \quad (2.52)$$

De donde se obtiene

$$k_{i+1} = d_i - a_i \quad (2.53)$$

Ahora que hemos encontrado el denominador de la función de transferencia en lazo cerrado, vamos a encontrar el numerador. Para los sistemas representados en forma de variable de fase, aprendimos que el polinomio del numerador se forma a partir de los coeficientes de la matriz de acoplamiento de salida, C. Dado que las figuras 35 y 36 están ambas en forma de

variable de fase y tienen la misma matriz de acoplamiento de salida, concluimos que los numeradores de sus funciones de transferencia son los mismos.

Veamos un ejemplo de diseño**.** Dada la planta

$$G(s) = \frac{20(s+5)}{s(s+1)(s+4)} \tag{2.54}$$

Comenzamos calculando la ecuación característica deseada del lazo cerrado. Usando los requisitos de la respuesta transitoria, los polos del lazo cerrado son $5.4 \pm j7.2$. Dado que el sistema es de tercer orden, debemos seleccionar otro polo del lazo cerrado. El sistema de lazo cerrado tendrá un cero en -5, igual que el sistema en lazo abierto. Podríamos seleccionar el tercer polo del lazo cerrado para cancelar el cero del lazo cerrado. Sin embargo, para demostrar el efecto del tercer polo y el proceso de diseño, incluyendo la necesidad de simulación, elegimos -5.1 como la ubicación del tercer polo del lazo cerrado.

Ahora, dibujemos el diagrama de flujo de señales para la planta. El resultado se muestra en la figura 37. Luego, retroalimentamos todas las variables de estado al control, u, a través de las ganancias k_i, como se muestra en la figura 38.

Escribiendo las ecuaciones de estado del sistema de lazo cerrado a partir de la figura 38, tenemos

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -k_i & -(4+k_2) & -(5+k_3) \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} r \tag{2.55a}$$

$$y = [100 \quad 20 \quad 0]x \tag{2.55b}$$

Comparando las ecuaciones (2.55) con las ecuaciones (2.44), identificamos la matriz del sistema de lazo cerrado como

$$\mathbf{A} - \mathbf{BK} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -k_i & -(4+k_2) & -(5+k_3) \end{bmatrix} \tag{2.56}$$

Para encontrar la ecuación característica del sistema de lazo cerrado, forma

$$\det\big(s\boldsymbol{I} - (\boldsymbol{A} - \boldsymbol{BK})\big) = s^3 + (5+k_3)s^2 + (4+k_2)s + k_1 = 0 \tag{2.57}$$

Esta ecuación debe coincidir con la ecuación característica deseada.

$$s^3 + 15.9s^2 + 136.08s + 413.1 = 0 \qquad (2.58)$$

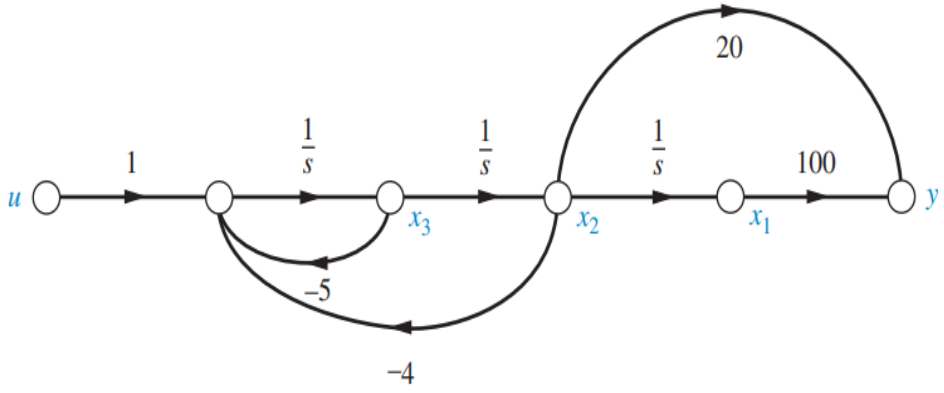

Figura 37. Representación en variables de fase para la planta del ejemplo 2.42 [23]

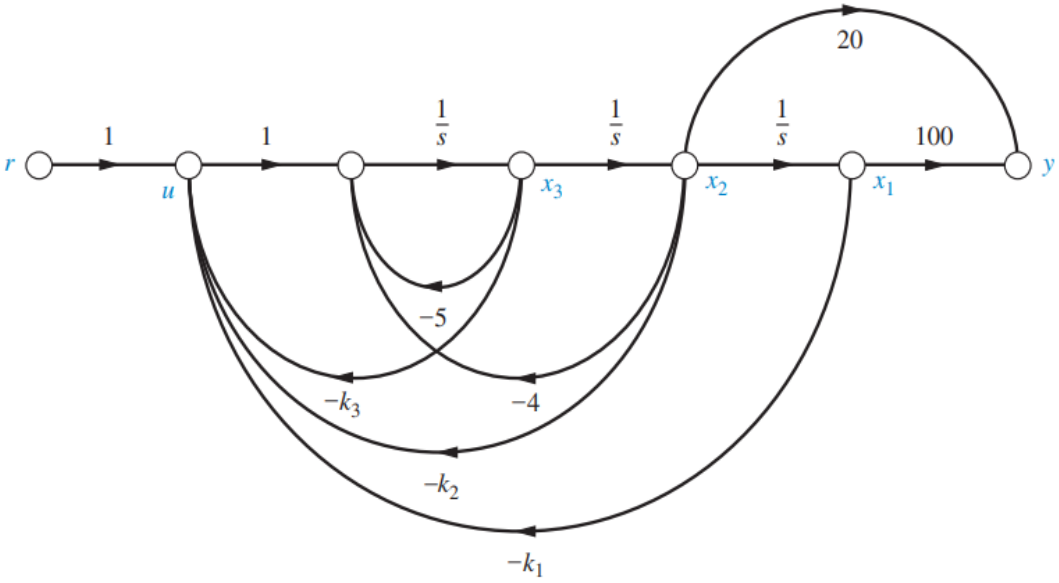

Figura 38. Planta con retroalimentación de variables de estado [23]

formado por los polos $-5.4 + j7.2$, $-5.4 - j7.2$, y -5.1, que fueron previamente determinados.

Igualando los coeficientes de las ecuaciones (2.57) y (2.58), obtenemos:

$$k_1 = 413.1;\ k_2 = 132.08;\ k_3 = 10.9 \qquad (2.59)$$

Finalmente, el término del cero de la función de transferencia de lazo cerrado es el mismo que el término del cero del sistema en lazo abierto, o $(s + 5)$

Usando las ecuaciones (2.55), obtenemos la siguiente representación en el espacio de estados del sistema de lazo cerrado:

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -413.1 & -136.08 & -15.9 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} r \qquad (2.60a)$$

$$y = [100 \quad 20 \quad 0]x \qquad (2.60b)$$

La función de transferencia es

$$T(s) = \frac{20(s + 5)}{s^3 + 15.9s^2 + 136.08s + 413.1} \qquad (2.61)$$

La figura 39, es una simulación del sistema de lazo cerrado, muestra un sobrepaso del 11.5% y un tiempo de establecimiento de 0.8 segundos. Un rediseño con el tercer polo cancelando el cero en 5 dará como resultado un rendimiento que cumple con los requisitos.

Dado que la respuesta en estado estacionario se aproxima a 0.24 en lugar de la unidad, existe un gran error en estado estacionario.

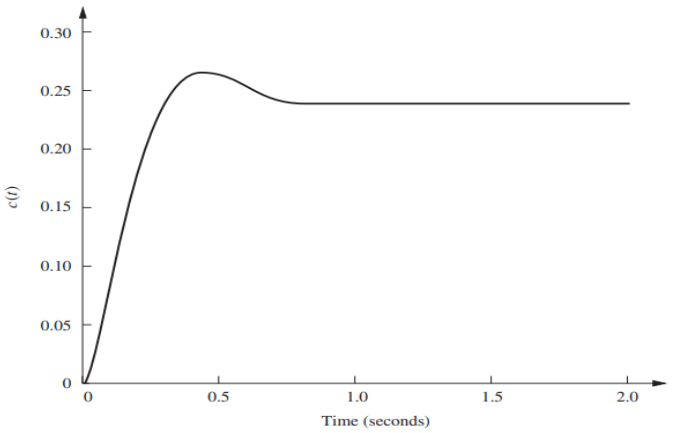

Figura 39. Simulación del sistema en lazo cerrado del ejemplo 2.42 [23]

2.2.7 Lenguajes de programación

Un lenguaje de programación se compone de un conjunto de instrucciones que permiten la interactúan entre los seres humanos y las computadoras, estos lenguajes posibilitan la comunicación efectiva entre las personas y los sistemas informáticos a través de algoritmos e instrucciones escritas en una sintaxis que las computadoras entienden e interpretan en lenguaje de máquina [14].

Los lenguajes de programación permiten a las computadoras procesar de forma rápida y eficiente grandes cantidades de información, existen muchos lenguajes de programación tal como se observan en la figura 40, que se utilizan hoy en día para diversas aplicaciones entre los lenguajes de programación más populares son C++, Visual Basic, Ruby, Java, JavaScript, Python, Matlab entre otros [14].

Figura 40. Ejemplos de lenguajes de programación [25]

Matlab: Es una plataforma de programación y calculo numérico utilizada por millones de ingenieros y científicos para el análisis de datos, el desarrollo de algoritmos y la creación de modelos. Matlab combina un entorno de trabajo optimizado para el análisis iterativo y los procesos de diseño con un lenguaje de programación que permite expresar operaciones matemáticas con matrices y arrays directamente [15].

Figura 41. Software Matlab [27]

Python: Es un lenguaje de programación de alto nivel, basado en la orientación a objetos y con una semántica dinámica integrada, diseñada principalmente para el desarrollo de aplicaciones web y software. Su sencillez lo convierte en una opción accesible para aprender, gracias a una sintaxis clara y enfocada en la legibilidad. Esto permite a los desarrolladores interpretar y entender el código en Python con mayor facilidad en comparación con otros lenguajes [16]

Figura 42. Software Python [29]

Scratch: Es un lenguaje de programación visual creado por el Instituto de Tecnología de Massachusetts, que facilita la creación de proyectos interactivos, como juegos, historias y animaciones de forma intuitiva y entretenida. Su principal característica es una interfaz gráfica que utiliza bloques de código que se conectan como piezas de un rompecabezas, eliminando la necesidad de escribir código en lenguajes complejos. Esto lo convierte en una herramienta perfecta para principiantes y niños que tienen interés por el mundo de la programación [17]

Figura 43. Software Scratch [31]

2.2.8 Tipos de comunicación

2.2.8.1 Bluetooth

Bluetooth es una tecnología que ofrece una vía sencilla para la comunicación entre dispositivos y la conexión a internet sin necesidad de cables, su objetivo principal es simplificar la sincronización de datos, la tecnología Bluetooth es de pequeña escala, bajo

costo y se caracteriza por usar enlaces de radio de corto alcance entre móviles y otros dispositivos, esta tecnología opera en la banda de 2.4 GHz también presenta la capacidad de penetrar paredes lo que la hace fundamental para actividades móviles. Los datos se pueden intercambiar a velocidades de hasta 1 Mbit/s, un esquema de frecuency hop permite a los dispositivos comunicarse incluso en áreas donde existe una gran interferencia electromagnética [18].

2.2.8.2 Cable USB

La tecnología USB se basa en una arquitectura de tipo serial, es una interfaz de entrada/salida mucho más rápida que los puertos seriales estándar, los cables USB están diseñados para operar en 4 hilos, incluyendo dos de alimentación y dos para datos. El bus del USB es capaza de mantener tres niveles de transferencia de información HIGH SPEED a 480 Megabits/sec, FULL SPEED a 12 Megabits y LOW SPEED a 1.5 Megabits/sec, tanto los drivers USB como la recepción de los datos y cables garantizan una interfaz que elimina el ruido que puede provocar error en los datos. Para la comunicación con los dispositivos se requieren de tres cosas como la detección del dispositivo USB en la conexión y desconexión, al existir un nuevo dispositivo conectado este debe ser capaz de descubrir como intercambiar información con este, provee un mecanismo que habilita el software para comunicación entre el hardware del USB y la aplicación que quiere acceder a los periféricos USB [19].

2.2.8.3 WIFI

WI-FI es una tecnología de red inalámbrica que permite la conexión de dispositivos electrónicos a una red sin necesidad de cables físicos, utilizando frecuencias de radio. Esta tecnología, conocida como red inalámbrica de área local (WLAN) posibilita que dispositivos como smartphones, tablets y ordenadores se conecten a internet o se compraren entre sí. Las redes WI-FI operan mediante la transmisión de ondas de radio en diferentes frecuencias 2.4 GHz, 5 GHz y 6 GHz, proporcionando conectividad a distintas velocidades. En general, las frecuencias más altas ofrecen mayores velocidades, mientras que las más bajas, como 2.4 GHz, permiten un alcance más amplio a menor velocidad [20].

2.2.8.4 Radiofrecuencia

La radiofrecuencia es un tipo de radiación electromagnética empleada en diversas aplicaciones, desde las comunicaciones inalámbricas hasta la medicina y los dispositivos tecnológicos de uso cotidiano, Esta tecnología utiliza ondas electromagnéticas para transferir datos y energía entre diferentes puntos. Su relevancia en el mundo moderno radica en su papel fundamental en la conectividad inalámbrica, como en dispositivos móviles, redes de alta velocidad y la transmisión global de datos. La radiofrecuencia facilita la interconexión de sistemas y dispositivos, siendo un pilar esencial para la comunicación y el avance tecnológico en nuestra sociedad [21]

2.3 MARCO TEÓRICO

En [22], se desarrolló un control de un robot móvil que permite llegar a su destino enfatizando la generación de velocidad de la variable y cambio en el sentido de la velocidad, la trayectoria que se diseñó está en base al comportamiento del vehículo en sus condiciones iniciales y finales de posicionamiento, al combinar ambos sentidos el robot se puede movilizar describiendo la trayectoria o ruta más recomendable para llegar a su destino en una distancia y tiempo más corto. Aplicando la técnica de control por realimentación utilizando salidas flat se obtuvieron como resultado un buen control del robot móvil permitiendo dar la trayectoria deseada al sistema y las señales de control correcta para el buen desempeño del sistema.

En [23], se presenta un control de seguimiento de trayectorias de robots móviles con estructura cinemática de tipo uniciclo, el sistema de control está diseñado en dos partes, la primera en un control de la cinemática y la segunda en un controlador para la dinámica, la estructura del control que se realizó combina la liberalización por medio de un lazo de retroalimentación basado en el modelo cinemático nominal y la red neuronal que está basada en funciones de base radial directa ajustando el peso para el control dinámico. El neuro-controlador propuesto y el ajuste de los pesos de conexión representaron una buena solución en aplicaciones de seguimiento y control de trayectorias, este sistema se lo experimento en un robot móvil Pioneer 2DX y mediante los resultados se pudo comprobar el buen funcionamiento de este controlador.

En [24], se presentó una propuesta simplificada del algoritmo de control super-twisting con el fin de lograr el seguimiento de trayectorias de robots móviles terrestres y aéreos incluso cuando existen incertidumbres y perturbaciones externas en el entorno, este algoritmo es muy popular debido a la robustez y convergencia que posee. Para el diseño del control de modo deslizante para el seguimiento de trayectorias se construyó una función fuerte de Lyapunov, la principal contribución es la simplificación de las ganancias al reducir a un solo parámetro de ajuste de las ganancias de control, sin embargo en el super-twisting las ganancias dependen de la cota de perturbación lo cual es desconocida en la mayoría de casos, lo que provoca que se sobreestime el valor de la ganancia incrementando el esfuerzo de control, para evitar esto se utiliza una versión adaptativa del super-twisting para que permitan el auto ajuste de las ganancias, Además se implementó la edometría para obtener la posición real de los robots, mientras la computadora genera las trayectorias de los robots y calcula las leyes de control, las señales enviadas de control al robot es por medio del protocolo Bluetooth. El rendimiento del control propuesto super-twisting adaptativo simplificado (SAST) se pudo evidenciar por medio de los resultados ya que aseguraron la convergencia exponencial a cero de la dinámica del error de seguimiento a pesar de que existan perturbaciones e incertidumbres externas.

CAPITULO III

3.1 Componentes de la propuesta

En este apartado se hará una breve descripción de los componentes electrónicos y electromecánicos que componen la estructura física del robot móvil con ruedas mBot Neo. Se describirá el principio de funcionamiento y las características técnicas de los sensores, actuadores y las placas electrónicas que permiten la movilidad del robot mediante la aplicación de algoritmos de control. También se mencionarán los programas, las instrucciones y los datos que hacen que un robot realice tareas específicas, al modificar el software o los algoritmos se puede cambiar el comportamiento del robot móvil.

3.1.1 Componentes físicos

Los componentes físicos en la robótica son los elementos del hardware que permiten interactuar entre el mundo real y los sistemas computacionales. Estos componentes se dividen en categorías claves como sensores actuadores controladores y estructuras mecánicos. Cada uno cumple un rol específico para el funcionamiento integral del sistema. A continuación, se mencionarán todos los componentes físicos de la propuesta

3.1.1.1 Robot móvil con ruedas mBot Neo

El mBot Neo es un robot móvil con ruedas de última generación el cual está diseñado para la informática y la educación STEAM. En la figura 44 se puede visualizar el robot móvil, este robot funciona con el controlador CyberPi el cual es un microcontrolador potente y versátil, junto con una gama de sensores y actuadores el mBot Neo es capaz de comunicarse por Wifi y además es muy útil ya que permite realizar diversas aplicaciones en diferentes temas de codificación, robótica, ciencia de datos e inteligencia artificial.

Figura 44. mBot Neo [39]

3.1.1.2 Tarjeta de expansión

La placa de expansión es un tablero que se le conecta al mBot Neo, esta placa está diseñada para poder ampliar la funcionalidad del robot móvil con ruedas, ya que nos permite a través de sus puertos conectar varios sensores, actuadores y además esta placa es capaz de suministrar energía adicional a los dispositivos conectados con el propósito de que funcionen de manera correcta y segura. Esta placa electrónica tiene varias características tal como se puede observar en la figura 45 cuenta con dos puertos de multifunción utilizados para conectar y controlar servo motores y tiras LED, cuenta con dos puertos para conectar y controlar motores DC, dos puertos para conectar y controlar motores codificadores, un puerto para conectar fácilmente el mBot Neo al módulo CyberPi y además es compatible con la programación Python [25].

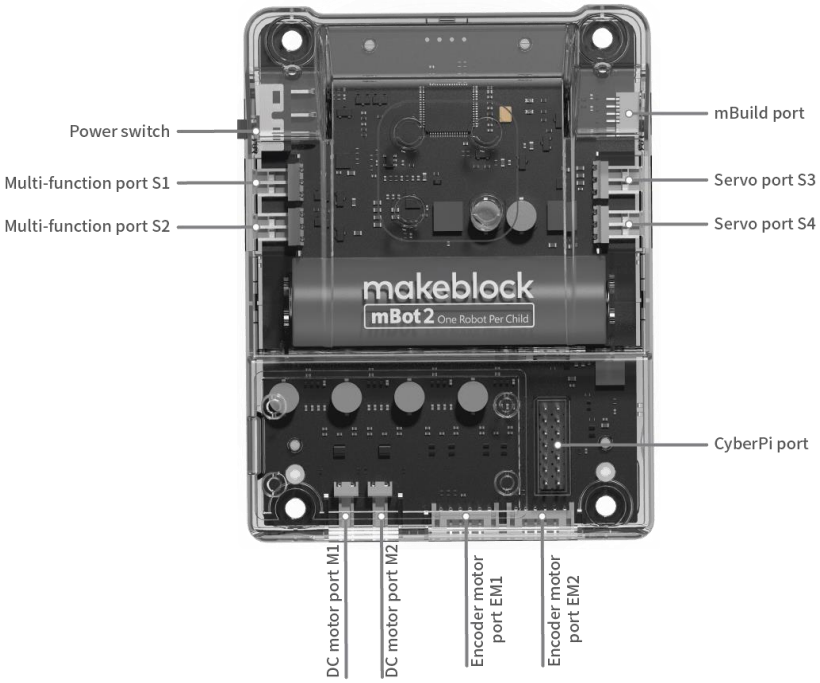

Figura 45. Placa de expansión del mBot Neo [40]

La placa de expansión está equipada con una batería de iones de litio recargable incorporada que puede suministrar energía al controlador CyberPi, en la tabla 1 se pueden ver las especificaciones técnicas de esta placa,

Especificación	Descripción
Microprocesador	CD32F403
Batería	Voltaje 3.7 V, Amperaje 2500 mAh Batería de polímero de iones de litio Vatios hora por batería 9.25 wh
Tensión y corriente de entrada	5V, 2000 mA (carga rápida) 5V, 500 mA (carga en funcionamiento)
Tensión y corriente de salida	5V. 6ª
Duración de la batería	3 a 6 horas
Tiempo de carga	80 minutos (en modo de carga rápida)
Modo de comunicación	Comunicación serial: entre el tablero de control principal y el tablero de extensión Señales digitales: en el puerto del servo digital Señales PWM: en el puerto del motor DC
Versión del Hardware	V1.0

Tabla 1. Especificaciones de mBot2/Neo Shield [25]

3.1.1.3 Sensores

Los sensores son los ojos y oídos de los robots ya que les permiten percibir y comprender el entorno que la rodea de forma autónoma, al transformar los estímulos físicos como la luz, la temperatura, el sonido, la distancia o la presión y convertirlos en señales eléctricas comprensibles les permiten a los robots interactuar con el mundo de forma inteligente y adaptarse a las diferentes condiciones que se le presentan en tiempo real. La capacidad que poseen los sensores es fundamental para que los robots puedan realizar tareas complejas desde la navegación en espacios desconocidos hasta la interacción con los seres humanos

Sensor ultrasónico

En la figura 46 se puede visualizar al sensor ultrasónico 2, este se puede utilizar para detectar la distancia entre un obstáculo y el. El transmisor de la izquierda transmite ondas ultrasónicas y el receptor recibe las ondas ultrasónicas reflejadas, este sensor ultrasónico puede detectar distancia de 5 a 300 cm. El valor de salida de los 300 cm es cuando la distancia detectada esta fuera del rango de salida y el valor de error es de $\pm 5\%$. Además, este sensor ha mejorado en cuanto a su carcasa, chip y leds azules, para aumentar el potencial de expresión e interacción de emociones en comparación con sensores ultrasónicos fabricados anteriormente [26].

Figura 46. Sensor ultrasónico 2 [41]

.

Sensor RGB cuádruple

El sensor RGB utiliza luz visible como luces de relleno, lo que reduce significativamente la interferencia de la luz ambiental, además proporciona la función de reconocimiento de colores, con cuatro sensores de luz puede admitir más escenarios de programación. En la figura 47 se puede observar que este sensor RGB tiene una carcasa de plástico para mejorar la durabilidad y calidad, su rango de detección es de 5 a 15 mm desde el objeto a detectar [27]. Este sensor también tiene la capacidad de no solo identificar colores, sino también de rastrear líneas para ayudar al mBot2 a seguir un trazado o detectar cruces o giros de 90° al mismo tiempo

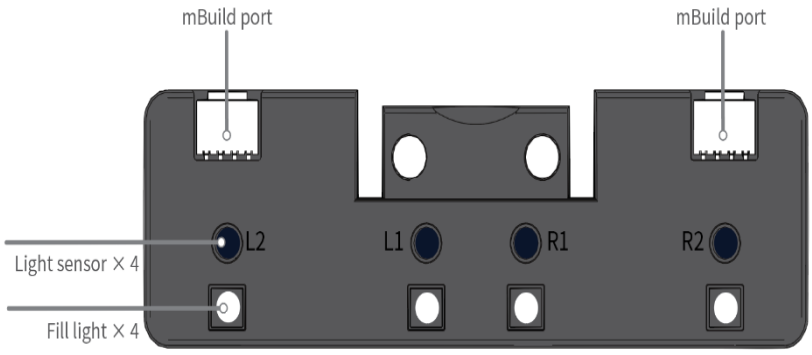

Figura 47. Sensor RGB [42]

3.1.1.4 Actuadores

Los actuadores son los músculos de los robots que les permiten accionar convirtiendo la energía en movimientos mecánicos, esto les da la capacidad a los robots de realizar tareas como desplazarse, manipular objetos y adaptarse a diversos entornos. Los actuadores tienen

un papel muy importante en la robótica moderna debido a que son una parte integral de los sistemas de control y esto les permite a los robots trabajar con precisión y eficiencia.

Motores

Los motores son actuadores que le permiten al robot móvil con ruedas moverse y realizar varias tareas, estos motores están equipados cada uno con un codificador óptico tal como se muestra en la figura 48, esto sirve para convertir el movimiento rotatorio en una señal digital, gracias a estos codificadores se puede controlar la velocidad y la posición angular en la que gira el motor.

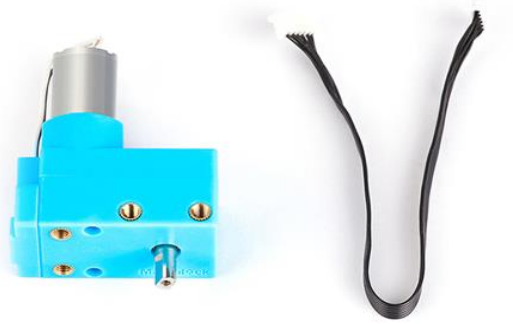

Figura 48. Motor Encoder [43]

En la tabla 2 se pueden ver las especificaciones técnicas de este tipo de motor. Para poder controlar los motores codificadores se pueden utilizar varios comandos que permitan establecer y leer la posición y velocidad, pero este tipo de comandos va depender del software en el que se esté realizando la programación ya sea en mBlock, Scratch o Python

Tensión nominal	7.4 V
Corriente sin carga	240 mA
Corriente de carga	≤ 750 mA
Velocidad sin carga	350 ± 5% (RPM)
Velocidad con carga (robot mBot Neo)	200 ± 5% (RPM)
Par de arranque	5 kg cm
Par nominal	800 g cm
Longitud del eje de salida	9 mm
Potencia	3.7 W
Angulo de rotación del codificador	360°

Tabla 2. Datos técnicos del motor encoder [28].

3.1.1.5 Módulo controlador CyberPi

Un CyberPi es un pequeño controlador programable desarrollado independiente por Makeblock con una estructura compacta y puertos integrados. Tiene una pantalla a todo color que proporciona interfaces de usuario fáciles de usar para la interacción hombre máquina, cuenta con el sistema CyberOS que le permite ejecutar programas predefinidos, a través del joystick y los botones integrados se puede configurar el idioma del sistema y actualizarlo. El CyberPi tiene un puerto Micro USB tipo C para conectarlo a PC para suministrarle energía y comunicación, un puerto para conectar módulos electrónicos y un puerto para conectar a palcas de extensión, en la tabla 3 se pueden ver las especificaciones técnicas del módulo CyberPi. además, tiene varios sensores integrados como sensor de luz y giroscopio que proporcionan múltiples tipos de salidas de datos. Para la comunicación inalámbrica se puede utilizar el módulo bluetooth y wifi integrado.

Chip	ESP32-WROVER-B
Procesador	Procesador principal Xtensa LX6 de 32 bits de doble núcleo Frecuencia de reloj 240 MHz
Memoria Integrada	ROM 448 KB SRAM 520 KB
Memoria Extendida	FLASH SPI 8 megas PSRAM 8 megas
Versión del hardware	V1.0

Tabla 3. Datos técnicos del CyberPi [29]

El módulo CyberPi tiene muchos sensores y funciones como un micrófono., un altavoz, una palanca de mando, en la figura 49 se puede visualizar cada uno de sus componentes.

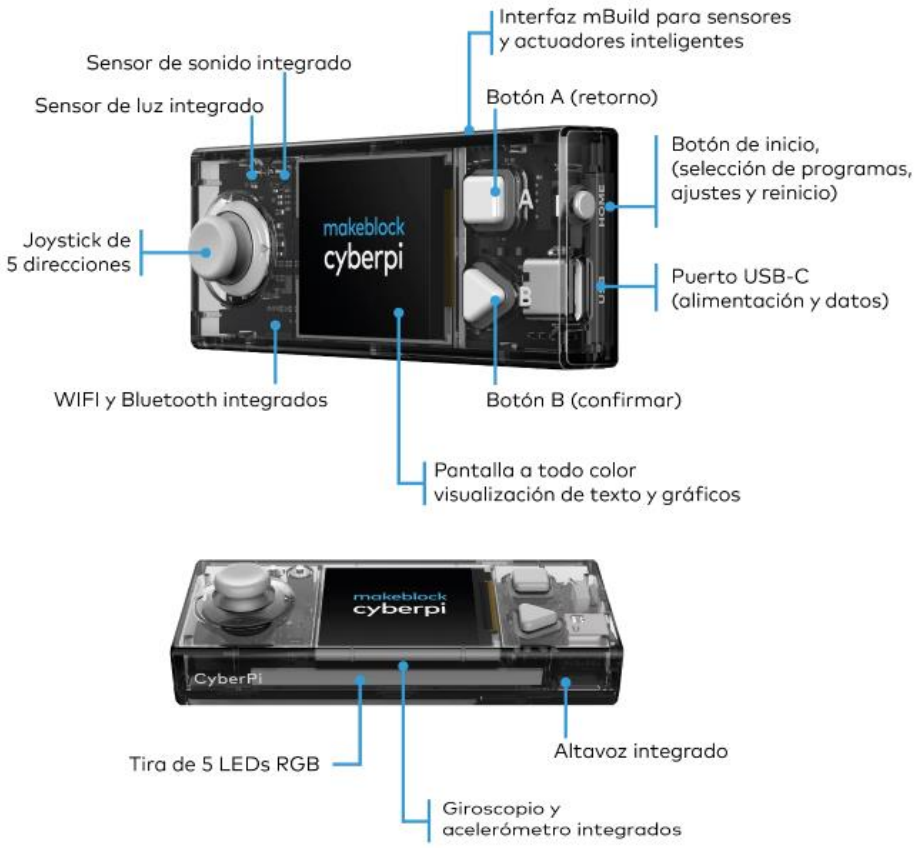

Figura 49. CyberPi y sus componentes [44]

Todos los robots funcionan con sensores, los sensores pueden compararse con los sentidos como el gusto, tacto, olfato, oído, vista. El mBot Neo ve su entorno a través de estos sensores, dentro del CyberPi podemos encontrar varios sensores los cuales se describirán a continuación

Sensor de luz: El sensor de luz detecta la cantidad de luz que hay a su alrededor y la convierte en una señal eléctrica que una computadora pueda entender. Con esto se puede programar el robot para que realice acciones basadas en la cantidad de luz que detecta, por ejemplo, el robot móvil puede seguir una línea en el suelo pero que se detenga si se acerca a un lugar oscuro. Al detectar cambios en la intensidad de la luz el robot puede evitar obstáculos como paredes o personas.

Sensor de sonido: Este sensor captura los sonidos a su alrededor y los transforma en señales eléctricas que una computadora pueda entender y esto los evaluada en términos de tono y potencia. Con esto se puede programar el mBot Neo como por ejemplo que responda a

comandos de voz para realizar diferentes acciones en respuesta a palabras o frases específicas y además al robot se lo puede programar para que se mueva hacia un lugar donde haya mucho ruido o que se detenga si hay silencio.

Giroscopio y acelerómetro: El giroscopio mide la rotación del módulo CyberPi alrededor de sus ejes, es fundamental para mantener el equilibrio y realizar movimientos precisos. El acelerómetro mide el cambio en la velocidad o aceleración esto permite detectar cambios en la orientación de cuando se inclina o se mueve. El giroscopio y el acelerómetro se los se los puede utilizar en conjunto para controlar la estabilidad del mBot Neo, como por ejemplo que detecten los giros o rotaciones y así el robot pueda mantener una posición estable en el entorno que se está moviendo.

Con el giroscopio, el acelerómetro y el algoritmo de detección de movimiento integrado en el módulo CyberPi se puede implementar un control basado en el movimiento, A continuación, se mostrarán las diferentes rotaciones del controlador CyberPi, En la figura 50 se puede observar la dirección en la que se agita el CyberPi, en este caso va desde los -179 a 180 grados.

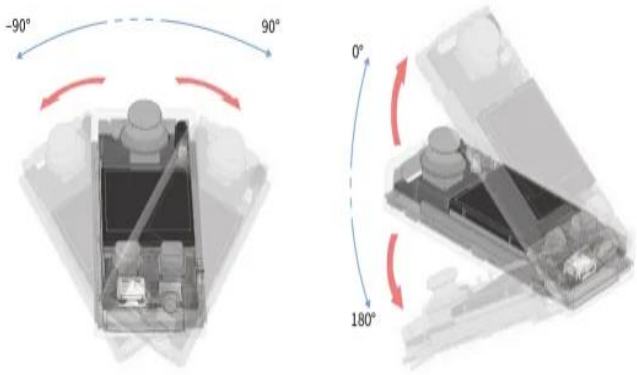

Figura 50. Dirección agitación del módulo CyberPi [45]

En la figura 51 se puede visualizar el ángulo de inclinación del módulo CyberPi, esto hace referencia al ángulo entre el plano horizontal y el eje y, en este caso va desde los -90 a 90 grados.

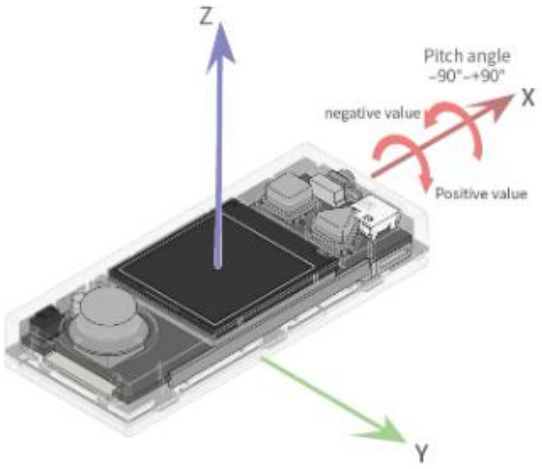

Figura 51. Angulo inclinación del módulo CyberPi [45]

En la figura 52 se puede observar el ángulo de giro del módulo CyberPi, este ángulo de balanceo hace referencia al ángulo entre el eje x y el plano horizontal, en este caso va desde los -179 a 180 grados.

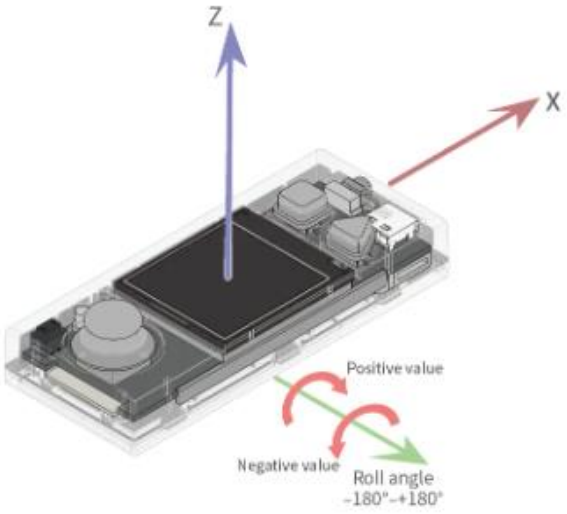

Figura 52. Angulo de giro del módulo CyberPi [45]

En la figura 53 se puede observar el ángulo de guiñada del controlador CyberPi, esto hace referencia al ángulo en el que el módulo CyberPi gira alrededor del eje z. En este caso puede ir desde -180 a 180 grado

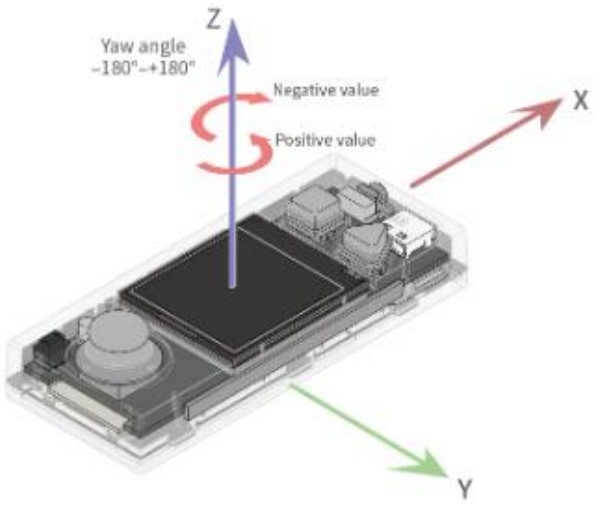

Figura 53. Angulo de guiñada del módulo CyberPi [45]

3.1.2 Componentes Lógicos

En este apartado se hará una descripción de todos los componentes lógicos que se utilizaran para programar el robot móvil con ruedas mBot Neo y pueda realizar las tareas de manera coordinada y eficiente. Estos componentes son esenciales para garantizar el desarrollo y el correcto funcionamiento de la implementación, ya que facilitan la coordinación y el control de las actividades realizadas por los componentes físicos.

3.1.2.1 Entorno de programación mBlock y lenguaje Scratch

mBlock es una plataforma diseñada para ofrecer una gran experiencia educativa a los estudiantes y aficionados con el mundo de la robótica con el fin de que desarrollen habilidades computacionales desde cosas básicas hasta profesionales. Este software cuenta con tecnologías emergentes como IoT y AI, todo esto proporciona une entorno de aprendizaje dinámico para que los estudiantes puedan programar robots de acuerdo a su imaginación y creatividad.

Para programar el mBot Neo se utilizará el software mBlock 5 el cual está basado en lenguaje Scratch 3.0 y orientado a la educación STEAM. Scratch 3.0 es la nueva generación de Scratch que incluye un editor de sonido y más bloques de programación. mBlock 5 utiliza Scratch 3.0 como base para crear el software de programación tanto visual como basado en texto. En la figura 54 se puede visualizar las diferentes áreas de trabajo y la configuración de la pantalla principal del software.

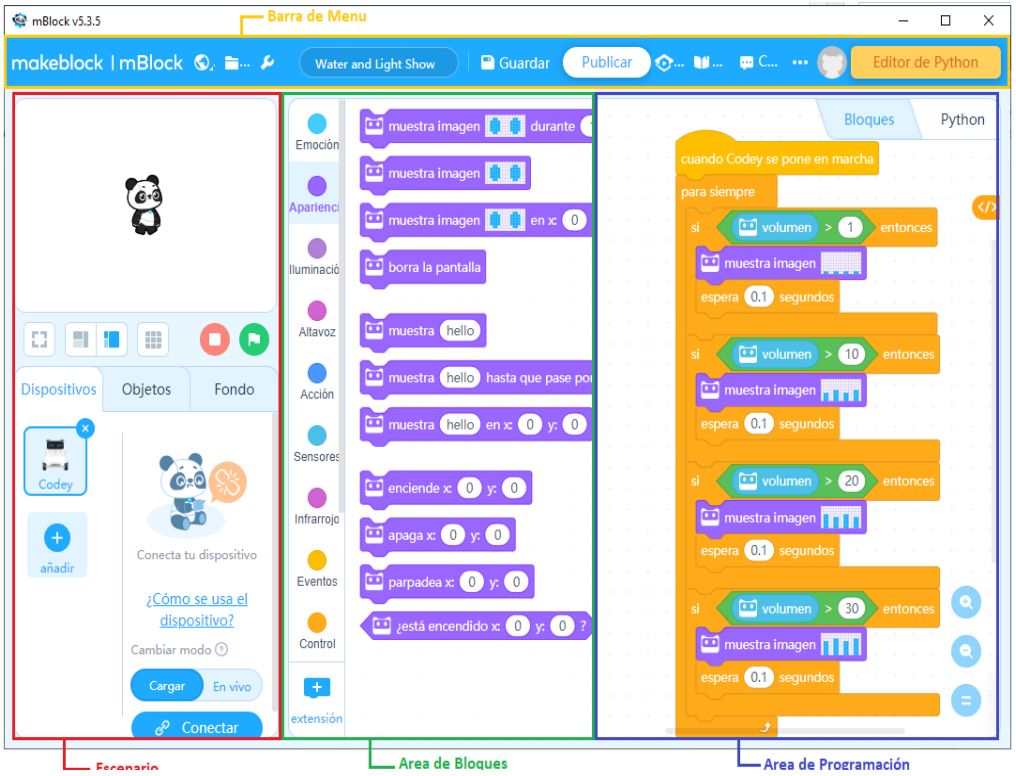

Figura 54. Software mBlock [Fuente: Autor]

A continuación, se explican los diferentes elementos que conforman el entorno de programación de Scratch:

- **Escenario:** En esta área se puede conectar el robot móvil mBot Neo y otros dispositivos al computador, también permite observar los proyectos creados y además se pueden personalizar los fondos y configurar los objetos.
- **Área de bloques:** En esta área se pueden encontrar los diferentes bloques que se van a utilizar para programar el robot, los cuales se encuentran agrupados por categoría y color dependiendo de cada una de sus funcionalidades
- **Área de programación:** En esta área se crea el programa, en el cual se arrastran los bloques de código para realizar la programación del mBot Neo o cualquier otro robot.

El entorno mBlock basado en Scratch es una herramienta muy valiosa que cuenta con muchos bloques de codificación y por medio de su interfaz visual se podrá observar la categorización de los bloques por colores de acuerdo a la función que cumplen ya sean de movimiento, control, sensores, sonido entre otros, esto permitirá programar el robot móvil con ruedas al momento de realizar varios movimientos al seguir una trayectoria.

3.1.2.2 Lenguaje de programación Python

Python es un lenguaje de programación muy utilizado en todo el mundo ya que es fácil de aprender y además es eficiente para desarrollar cualquier tipo de programa como aplicaciones web, ciencia de datos entre otros. Python es un lenguaje de código abierto que se puede ejecutar en diversas plataformas como por ejemplo en mBlock. En 2018 se implementó un editor de código en mBlock 5 como complemento a la programación basada en bloques para realizar la transición de la programación basada en bloques a la programación de código en tiempo real esto permite a los estudiantes aprender MicroPython y programar en Python [30].

El editor mBlock-Python proporciona un entorno real en Python, esto permite experimentar diversas bibliotecas, como por ejemplo la biblioteca CyberPi tiene varias API para controlar el hardware incorporado como los elementos electrónicos y motores del robot mBot Neo. Las bibliotecas CyberPi hacen referencia a dos bibliotecas independientes que son MicroPython y Python. Estas bibliotecas permiten que el código que se compila en MicroPython se pueda ejecutar como código Python al instante. En la figura 55 se puede visualizar las diferentes áreas de trabajo del entorno de programación de Python.

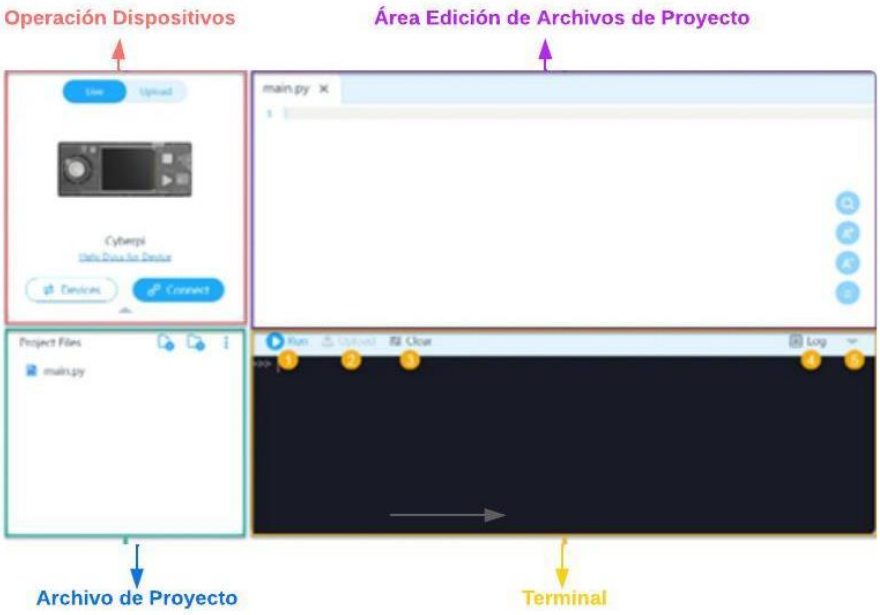

Figura 55. Entorno de trabajo Python [Fuente: Autor]

A continuación, se explican los diferentes elementos que conforman el entorno de programación de Python:

- **Operación de dispositivos:** En esta área se puede conectar o cambiar dispositivos y además permite cambiar el modo de programación de un dispositivo.
- **Archivos del proyecto:** En esta área se puede crear, renombrar o eliminar un archivo o carpeta y también permite agregar un archivo o carpeta local y agregar un archivo de la biblioteca de recursos.
- **Área de edición de archivos del proyecto:** En esta área se compila o modifica el código y se pueden abrir varios archivos del proyecto
- **Terminal:** En esta área se ejecutan los comandos y se puede visualizar la información sobre la ejecución del programa.

3.1.2.3 Lenguaje de programación Matlab

Matlab es una herramienta versátil en el mundo de la programación ya que se la utiliza en diversos campos de la ingeniería y ciencias, este software permite realizar cálculos complejos, visualizar datos de forma intuitiva. En Matlab se pueden desarrollar diversas aplicaciones como la simulación y análisis de sistemas dinámicos que son excelentes para realizar pruebas experimentales de algoritmos en entornos virtuales. Además, cuenta con una amplia variedad de toolboxes y funciones como control, procesamiento de señales y visión por computadora que están diseñadas para las áreas de ingeniería

Con Matlab se pueden crear modelos de sistema de control y probar el algoritmo en entornos virtuales sin la necesidad del hardware físico. Estoy es muy útil para desarrollar el modelo cinemático del mBot Neo, ya que por medio de este software se podrá simular el comportamiento del robot en una trayectoria bajo diferentes condiciones en tiempo real de forma virtual tal como se puede observar en la figura 56. Gracias a la capacidad de Matlab se podrán realizar varias trayectorias para el robot móvil con ruedas con el propósito de poder visualizar si el robot las sigue correctamente y como responde ante las perturbaciones. Con esto se podrá comparar la trayectoria real que sigue el robot con la trayectoria deseada, esto permitirá evaluar los posibles errores y ajustar los parámetros del controlador.

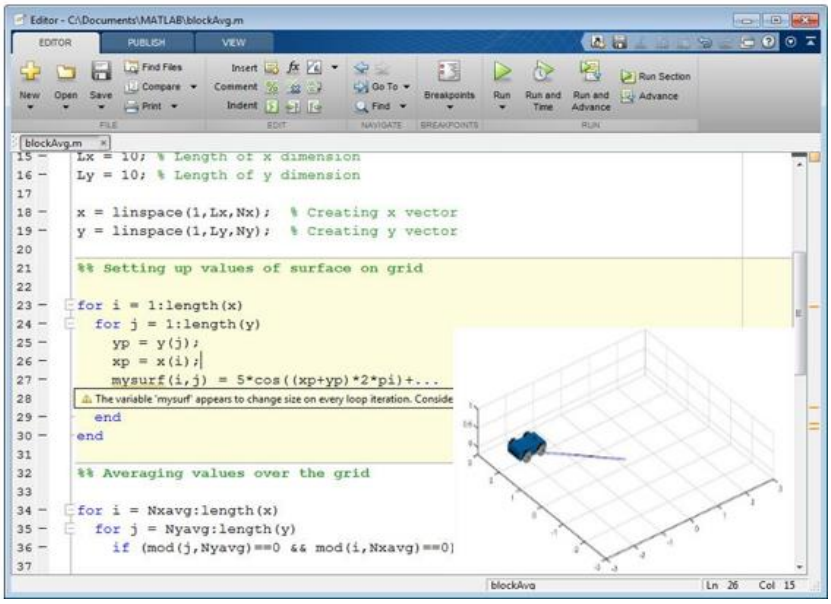

Figura 56. Ejemplo de programación y simulación de un robot móvil en Matlab [Fuente: Autor]

3.1.2.4 App Designer

Es una herramienta de Matlab que permite crear interfaces graficas de usuario de manera intuitiva y visual. App Designer se la puede utilizar para desarrollar aplicaciones interactivas que combinan controles gráficos como botones, deslizadores, cuadros de texto entre otros con capacidades de programación. Una característica muy importante es que su entorno visual simplifica el diseño de interfaces sin necesidad de escribir el código desde cero para elementos gráficos y además permite combinar la interactividad de las GUIs, con el poder computacional de Matlab, esto permite ejecutar algoritmos complejos de manera accesible para los usuarios. En la figura 57 se puede visualizar las diferentes áreas de trabajo y las configuraciones de App Designer

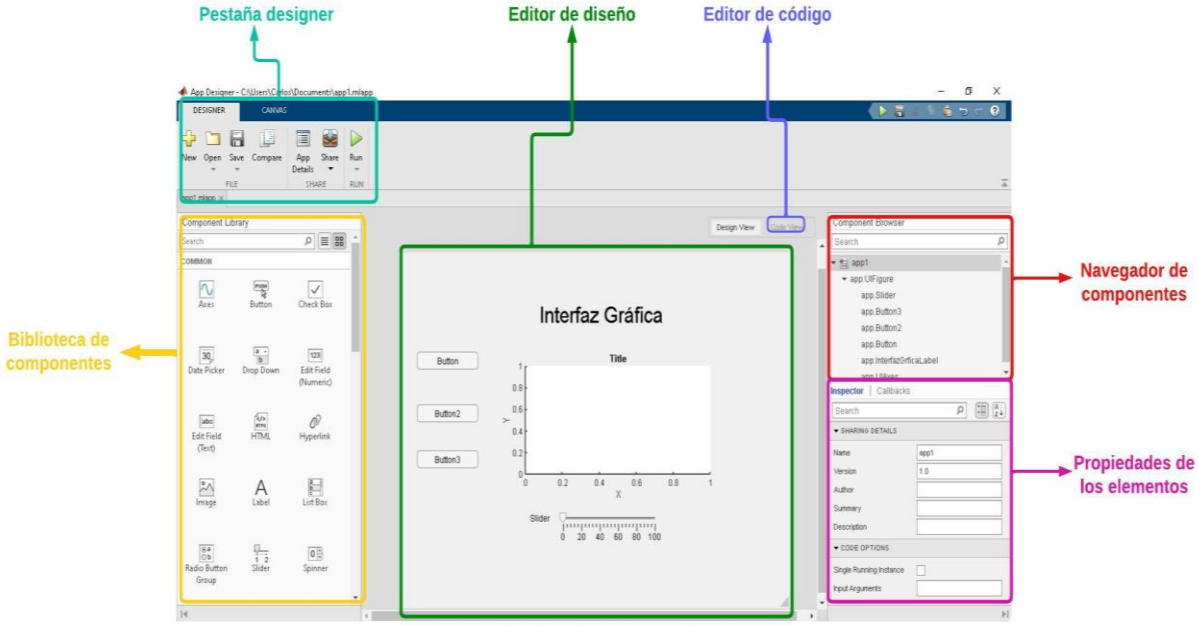

Figura 57. Entorno de trabajo de App Designer [Fuente: Autor]

A continuación, se explican los principales elementos que conforman el entorno de programación de App Designer

- **Biblioteca de componentes:** En esta área se encuentran los diferentes componentes disponibles para el diseño de la aplicación, los componentes están clasificados en componentes comunes, contenedores, herramientas de figuras e instrumentación.
- **Pestaña designer:** Esta área es un menú que nos permite realizar varias funciones como crear, abrir, guardar y ejecutar una aplicación en la que se está trabajando.
- **Editor de diseño;** Espacio donde se colocan y configuran los componentes gráficos como botones, gráficos, cuadro de texto, etiquetas y deslizadores.
- **Editor de código:** En esta área se escribe el comportamiento de los componentes y se implementa la lógica de la aplicación.
- **Navegador de componentes:** En esta área se muestra la lista que organiza los componentes de la interfaz gráfica y facilita su gestión.
- **Propiedades de los elementos:** En esta área se pueden ajustar características de los componentes como tamaño, color, etiquetas y eventos.

3.2 Diseño de la propuesta

Se presentará un enfoque al desarrollo de un controlador para el seguimiento de trayectorias de un robot móvil con ruedas basado en linealización de realimentación y variables de estado. Se analizará el modelo cinemático del mBot Neo para determinar las ecuaciones de la cinemática directa e inversa del robot que nos permitirán obtener el movimiento y velocidad de cada una de las ruedas y se aplicara la técnica de linealización el cual transforma el sistema no lineal del robot en un sistema lineal facilitando el desarrollo e implementación del controlador. Se realizarán varias pruebas experimentales y simulaciones utilizando el software Matlab y posteriormente se implementa el algoritmo en el entorno de programación mBlock para ejecutarlo en el robot mBot Neo.

3.2.1 Diseño e implementación del hardware

En este aparatado se indicarán todas las conexiones de los diferentes componentes del robot móvil mBot Neo para ponerlo en funcionamiento.

3.2.1.1 Conexiones de la tarjeta de expansión con los componentes

Para poner en funcionamiento el mBot Neo primero se debe conectar la tarjeta de expansión mCore con cada uno de los componentes, como los actuadores y sensores. Tal como se muestra en la figura 58 para poder conectar los motores con la placa de expansión se deben utilizar dos cables codificadores, ya que un motor va a la derecha y otro a la izquierda del chasis. Para conectar el sensor ultrasónico y el RGB se utilizan los cables mBuild. Cada uno de estos sensores y actuadores van conectados en diferentes puertos y pines. A continuación, se detallará la conexión de estos elementos con la placa de expansión y los pines asociados.

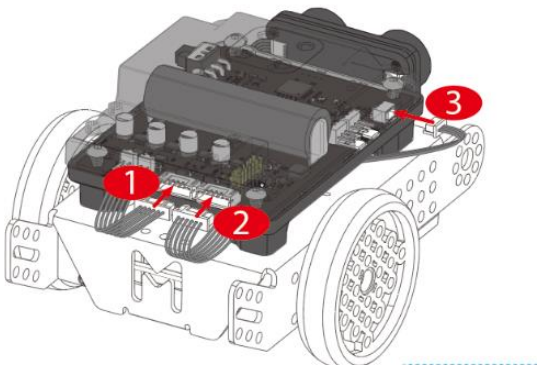

Figura 58. Montaje de la placa de expansión con el chasis del robot [47]

La placa mCore tiene dos puertos para motores DC, estos puertos se los puede identificar en la placa como M1 y M2 y permite conectar un motor para la derecha y otro motor para la izquierda. Estos motores se los utiliza para el movimiento de las ruedas del mBot Neo. Los sensores y módulos de Makeblock utilizan cables RJ25 con colores para identificar fácilmente las conexiones. Esta placa de expansión cuenta con cuatro puertos RJ25 etiquetados como Port 1, Port 2, Port 3, y Port 4, normalmente el sensor ultrasónico va conectado al Port 3, y el RGB puede ir conectado al Port1 o Port4. En la figura 59 se puede visualizar la conexión de estos elementos.

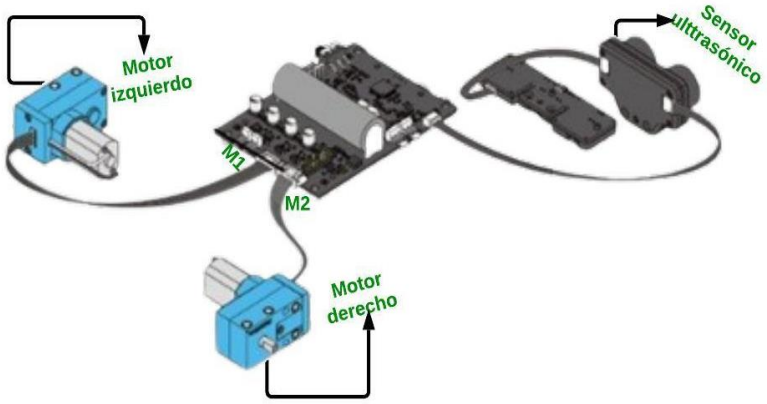

Figura 59. Conexión de la placa de expansión con los motores y actuadores [Fuente: Autor]

Dado que la placa de expansión está basada en Arduino se la puede programar como un Arduino Uno, sin embargo, como la placa mCore tiene muchos componentes integrados dentro de la propia placa esto limita la disponibilidad de algunos pines que normalmente estarían libres en una placa Arduino estándar, ya que han sido reservados para dichos componentes. A continuación, en la tabla 4 se detallan los pines asociados a los diferentes componentes.

Puertos y Componentes	**Pines asociados**
PORT_1	D11, D12
PORT_2	D9, D10
PORT_3	A2, A3
PORT_4	A0, A1
M1	D6, D7
M2	D5, D4
Sensor de luz	A6
Buzzer	D8
LEDS RGB	D13

Emisor de infrarrojos	D3
Receptor de infrarrojos	D2
Sensor ultrasónico	A2, A3
Sensor RGB	D11

Tabla 4. Mapeo de puertos y pines asociados a los componentes del mBot Neo [31]

3.2.1.2 Conexión de la placa mCore con el módulo CyberPi

La conexión entre el módulo CyberPi y la placa de expansión combina las capacidades de procesamiento avanzado como la programación y conectividad del CyberPi con la versatilidad y compatibilidad de la placa mCore para controlar los motores y sensores. Para realizar la conexión entre ambos elementos se la puede hacer de dos formas mediante los pines RX/TX o mediante la conexión I2C. El CyberPi tiene pines RX/TX expuestos a través de un módulo de expansión o conectores específicos y la placa mCore tiene pines RX/TX internos accesibles o se puede usar un puerto RJ25. En la figura 60 se puede visualizar como van a ir conectados ambos elementos, Para la primera conexión se conecta el TX del CyberPi con el RX del mCore y luego se conecta el RX del CyberPi con el TX del mCore. Para realizar la segunda conexión I2C se conecta el SDA del CyberPi con el SDA del mCore y luego se conecta el SCL del CyberPi con el SCL de la placa mCore.

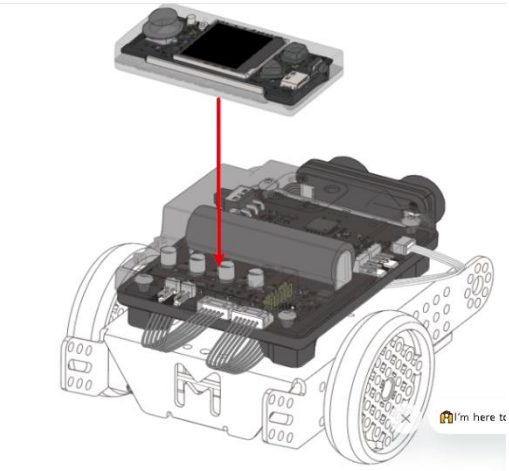

Figura 60. Conexión del módulo CyberPi con la placa mCore [47]

El módulo CyberPi con su capacidad de ejecutar programas complejos y conectarse a redes actúa como el controlador principal, mientras que la placa de expansión sirve como una unidad periférica encargada de interactuar con el hardware robótico. A continuación, en la

tabla 5 y tabla 6 se detallan los pines utilizados para realizar las dos formas de conexión entre el CyberPi y la placa mCore

Componentes	Pin TX	Pin RX
CyberPi	TX del módulo expansión	RX del módulo expansión
mCore	Pin D1(TX)	Pin D0(RX)

Tabla 5. Conexión mediante UART [Fuente: Autor]

Componentes	Pin TX	Pin RX
CyberPi	SDA del módulo expansión	SCL del módulo expansión
mCore	Pin A4 (SDA)	Pin A5 (SCL)

Tabla 6. Conexión mediante I2C [Fuente: Autor]

Una vez que se conectan todos los componentes físicos como los motores, sensores, la placa de expansión y el módulo CyberPi al chasis tal como se muestra en la figura 61, el robot mBot Neo estaría listo para funcionar y empezar a ser programado de acuerdo a las necesidades del usuario

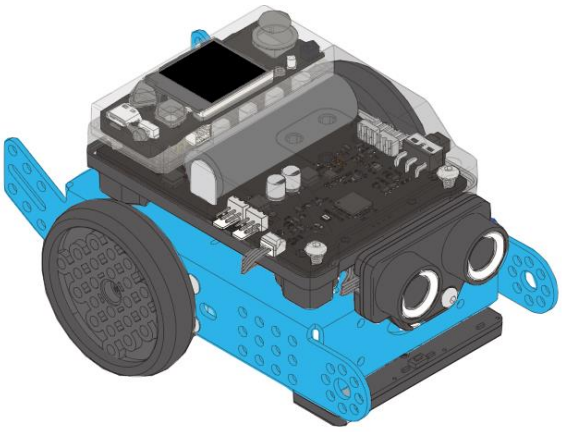

Figura 61. MBot Neo con todos sus elementos conectados [47]

3.2.1.3 Tipos de comunicación

El tipo de comunicación que utilizan los motores es mediante el control por señal PWM y pines digitales Los motores DC conectados a los puertos M1 y M2 se controlan mediante un puente H integrado (L298N) que usa señales PWM (modulación por ancho de pulso) para regular la velocidad y pines digitales para la dirección del giro (hacia adelante o hacia atrás). Los sensores conectados a través de los puertos RJ25 utilizan diferentes tipos de comunicación según el tipo de sensor. La comunicación del sensor ultrasónico es mediante señales digitales de Trigger y Echo, el microcontrolador envía un pulso Trigger al sensor y

este devuelve un pulso Echo cuya duración depende de la distancia medida. Para el sensor RGB el tipo de comunicación es mediante señal digital serial, controlados mediante un único pin digital utilizando un protocolo serial especifico como el de los LEDs WS2812

La conexión entre el CyberPi y la placa mCore puede establecerse mediante protocolos de comunicación digitales como el UART o el I2C. El protocolo de comunicación serial UART permite el intercambio de datos a través de los pines TX y RX, esto facilita el envió de comandos o datos en tiempo real entre el CyberPi y la tarjeta de expansión. El protocolo de comunicación síncrona I2C utiliza dos líneas SDA y SCL para transmitir datos entre un dispositivo maestro y uno o varios dispositivos esclavos. En esta configuración el CyberPi generalmente actúa como maestro enviando comandos y recibiendo datos desde la placa mCore

3.2.1.4 Modelamiento cinemático del robot mBot Neo

En el campo de la robótica la cinemática se encarga de analizar cómo cambia la velocidad, postura y aceleración de un robot móvil, sin tener en cuenta el torque o la fuerza. Las ecuaciones cinemáticas están determinadas por la configuración geométrica fija del robot en relación con un sistema de referencia global fijo.

El desplazamiento del robot móvil con ruedas puede describirse como una rotación en torno a un punto específico, conocido como (ICR) Centro Instantáneo de Rotación o (ICC) Centro Instantáneo de Curvatura. Este punto ICR corresponde al lugar donde se intersectan los ejes de todas las ruedas en torno al cual el robot gira en un instante determinado. Su ubicación relativa al chasis del robot puede cambiar con el tiempo.

El mBot Neo, es un robot móvil de configuración diferencial el cual cuenta con dos ruedas motorizadas y una rueda libre que proporciona estabilidad al chasis. Las dos ruedas motorizadas están controladas de forma independiente por los motores codificadores, permitiendo el control preciso de la velocidad y la dirección del robot. Las dos ruedas fijas motorizadas están colocadas en un eje común. La velocidad de cada una de las ruedas fijas está controlada por un motor DC independiente. Las ruedas del robot tienen un radio r (3.2 cm par el mBot) y la distancia entre las mismas es de $2b$ (6.5 cm para el mBot).

De acuerdo a la figura 62 las variables de entrada son la velocidad de la rueda derecha v_d y la velocidad de la rueda izquierda v_i, en donde para el análisis se asume que $v_d > v_i$, R es el radio instantáneo de la trayectoria de conducción del robot y que corresponde a la distancia entre el centro del chasis que es el punto medio entre las ruedas activas y el punto ICR. En cada instancia de tiempo, ambas ruedas tienen la misma velocidad angular ω alrededor del ICR.

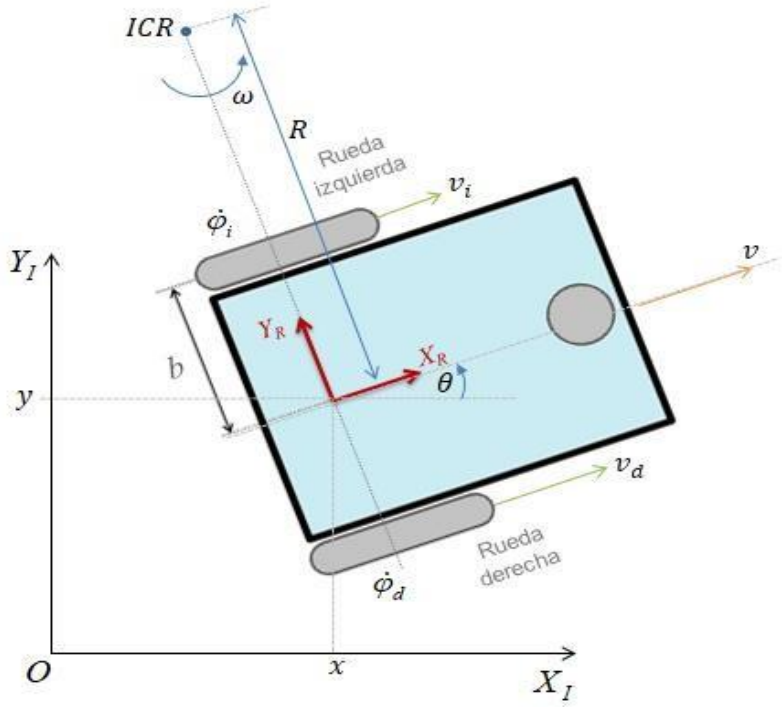

Figura 62. Representación geométrica del robot móvil mBot Neo [9]

A continuación, se describen las diferentes variables y parámetros de la representación geométrica del robot móvil con ruedas:

- (x, y): Coordenadas de la posición del robot en un plano.
- θ: Ángulo de orientación del robot con respecto a un marco de referencia fijo.
- v: Velocidad lineal del robot.
- ω: Velocidad angular del robot.

Las velocidades lineales de cada rueda se obtienen a partir de:

$$v_d = \omega(R + b) \qquad v_i = \omega(R - b) \tag{3.1}$$

Resolviendo las dos ecuaciones anteriores es posible determinar ω y R (radio de curvatura) en función de las velocidades lineales de las ruedas

$$\omega = \frac{v_d - v_i}{2b} \qquad R = \frac{b(v_d + v_i)}{v_d - v_i} \tag{3.2}$$

La velocidad tangencial del robot se calcula entonces como

$$v = \omega R = \frac{v_d + v_i}{2} \tag{3.3}$$

Las velocidades tangenciales de las ruedas son $v_i = r\dot{\varphi}_i$ y $v_d = r\dot{\varphi}_d$ donde $\dot{\varphi}_i$ y $\dot{\varphi}_d$ son las velocidades angulares izquierda y derecha de las ruedas alrededor de sus ejes, respectivamente. Reemplazando estos valores en las expresiones v y ω de las ecuaciones (3.2) y (3.3) se obtiene:

$$v = \frac{r\dot{\varphi}_d + r\dot{\varphi}_i}{2} = \frac{r}{2}\dot{\varphi}_d + \frac{r}{2}\dot{\varphi}_i \tag{3.4}$$

$$\omega = \frac{r\dot{\varphi}_d - r\dot{\varphi}_i}{2b} = \frac{r}{2b}\dot{\varphi}_d - \frac{r}{2b}\dot{\varphi}_i \tag{3.5}$$

Las ecuaciones (3.4) y (3.5) representan la cinemática directa del robot en coordenadas locales. La cinemática directa en el sistema inercial o global quedaría:

$$\dot{x} = v\,cos\theta \qquad \dot{y} = v\,sin\theta \qquad \dot{\theta} = \omega \tag{3.6}$$

$$\begin{aligned} \dot{x} &= \frac{r}{2}(\dot{\varphi}_d cos\theta + \dot{\varphi}_i cos\theta) \\ \dot{y} &= \frac{r}{2}(\dot{\varphi}_d sin\theta + \dot{\varphi}_i sin\theta) \\ \dot{\theta} &= \frac{r}{2b}(\dot{\varphi}_d - \dot{\varphi}_i) \end{aligned} \tag{3.7}$$

A partir de las ecuaciones de la cinemática directa se puede determinar las velocidades de las ruedas del robot utilizando las siguientes ecuaciones que corresponden a la cinemática inversa del robot:

$$\dot{\varphi}_d = \frac{1}{r} = (v + b\omega) \tag{3.8}$$

$$\dot{\varphi}_i = \frac{1}{r} = (v - b\omega) \tag{3.9}$$

3.2.2 Diseño e implementación del software

En este apartado se analizará el algoritmo de control que se aplicará en el desarrollo del controlador, para esto primero se implementará el algoritmo en el entorno de Matlab, que permitirá corregir errores y realizar ajustes en los parámetros, además cuenta con una herramienta como App Designer que va permitir crear una interfaz gráfica que muestre el comportamiento del robot móvil a lo largo de una trayectoria.

3.2.2.1 Algoritmo de control basado en linealización de realimentación y variables de estado

El control del movimiento de los robots móviles con ruedas en un entorno sin obstáculos se lo puede realizar controlando el movimiento desde una posición inicial hasta una posición final o mediante el seguimiento de trayectoria. En la robótica móvil una trayectoria es una línea que el robot necesita recorrer en el espacio de coordenadas Para el seguimiento de trayectoria se debe aplicar el uso de un controlador no suave o que varié en el tiempo debido a que el sistema controlado no es lineal

Con las ecuaciones del modelo cinemático en el accionamiento diferencial las salidas planas son salidas reales del sistema x y y. Se puede demostrar que las entradas y la orientación del robot se puede representar como funciones de x y y, y sus derivadas. Sabemos que $\dot{x}$ y $\dot{y}$ se pueden interpretar como coordenadas cartesianas de la velocidad de traslación del robot. Por lo tanto, la velocidad de traslación se obtiene mediante la suma pitagórica de sus componentes

$$v(t) = \sqrt{\dot{x^2}(t) + \dot{y^2}(t)}$$

Un robot con ruedas impulsado diferencialmente siempre se mueve en la dirección de su orientación, lo que significa que la tangente de la orientación es igual al cociente de los componentes cartesianos de la velocidad de traslación

$$\varphi(t) = tan^{-1}\left(\frac{\dot{y}(t)}{\dot{x}(t)}\right)$$

Linealización de realimentación

La idea de la linealización por retroalimentación es introducir alguna transformación normalmente en la entrada del sistema que haga que la salida sea lineal, Para realizar la linealización de retroalimentación se deben elegir las salidas planas adecuadas, cuyo número sea igual al número de entradas. Las salidas planas deben diferenciarse y las derivadas obtenidas deben verificarse para detectar la presencia funcional de las entradas. El sistema de ecuaciones se resuelve para las derivadas más altas de las entradas individuales. Para obtener las entradas reales del sistema, se debe utilizar una cadena de integradores en las derivadas de entrada. Las derivadas de salida por otro lado sirven como nuevas entradas al sistema. Dado que el sistema obtenido se vuelve lineal se puede utilizar una amplia selección de posibles leyes de control en estas nuevas entradas.

En el caso de un robot móvil con ruedas y accionamiento diferencial, las salidas planas son x y y. Su primera derivada según el modelo cinemático es

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

$$\dot{y} = v\sin\varphi$$

En las primeras derivadas, solo la velocidad traslación v aparece y la diferenciación continua

$$\ddot{x} = \dot{v}\cos\varphi - \dot{v}\dot{\varphi}\sin\varphi$$

$$\ddot{y} = \dot{v}\sin\varphi - \dot{v}\dot{\varphi}\cos\varphi$$

En las segundas derivadas ambas velocidades v y $\omega = \dot{\varphi}$ están presentes. Ahora el sistema de ecuaciones se reescribe de modo que las segundas derivadas de las salidas planas se describen como funciones de las derivadas más altas de los valores individuales, las entradas v y ω en este caso:

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\end{bmatrix} = \begin{bmatrix}\cos\varphi & -v\sin\varphi\\ \sin\varphi & v\cos\varphi\end{bmatrix}\begin{bmatrix}\dot{v}\\ \omega\end{bmatrix} = \boldsymbol{F}\begin{bmatrix}\dot{v}\\ \omega\end{bmatrix} \qquad (3.10)$$

La matriz $\boldsymbol{F}$ se ha introducido, que no es singular si $v \neq 0$. Por lo tanto, el sistema de ecuaciones se puede resolver para $\dot{v}$ y ω:

$$\begin{bmatrix}\dot{v}\\ \omega\end{bmatrix} = \boldsymbol{F}^{-1}\begin{bmatrix}\ddot{x}\\ \ddot{y}\end{bmatrix} = \begin{bmatrix}\cos\varphi & \sin\varphi\\ -\dfrac{\sin\varphi}{v} & \dfrac{\cos\varphi}{v}\end{bmatrix}\begin{bmatrix}\ddot{x}\\ \ddot{y}\end{bmatrix} \tag{3.11}$$

La solución ω de la ecuación (3.11) es la entrada real del robot, mientras que la solución $\dot{v}$ debe integrarse antes de usarse como entrada. El sistema lineal recién obtenido tiene entradas $[u_1, u_2]^T = [\ddot{x}, \ddot{y}]^T$ y los estados $z = [\,x, y, \dot{x}, \dot{y}]^T$. La dinámica del nuevo sistema puede ser descrito convenientemente por la representación en el espacio de estado:

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dot{y}\\ \ddot{y}\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\\ y\\ \dot{y}\end{bmatrix} + \begin{bmatrix}0 & 0\\ 1 & 0\\ 0 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix} \tag{3.12}$$

O en forma compacta como

$$\dot{z} = \boldsymbol{Az} + \boldsymbol{Bu} \tag{3.13}$$

El sistema (3.13) es controlable porque su matriz de controlabilidad tiene rango completo

$$\boldsymbol{Q_c} = [\boldsymbol{B}, \boldsymbol{AB}] \tag{3.14}$$

Por lo tanto, el controlador de estado existe para un polinomio característico elegido arbitrariamente del bucle cerrado. Un requisito adicional es diseñar la ley de control de modo que el robot siga una trayectoria de referencia. En el contexto de sistemas planos se proporciona una trayectoria de referencia para las salidas planas, en este caso $X_{\mathrm{ref}}(t)$ y $Y_{\mathrm{ref}}(t)$. Luego se puede obtener fácilmente la referencia para el estado del sistema $z_{\mathrm{ref}}(t) = \left[x_{\mathrm{ref}}, \dot{x}_{\mathrm{ref}}, y_{\mathrm{ref}}, \dot{y}_{\mathrm{ref}}\right]^T$ y la entrada del sistema $\boldsymbol{u}_{\mathrm{ref}} = \left[\ddot{x}_{\mathrm{ref}}, \ddot{x}_{ref}\right]^T$. La ecuación (3.13) también se puede escribir para las señales de referencia:

$$\dot{z}_{\mathrm{ref}} = \boldsymbol{Az}_{\mathrm{ref}} + \boldsymbol{Bu}_{\mathrm{ref}} \tag{3.15}$$

El error entre los estados reales y los estados de referencia se define como $\tilde{z} = z - z_{\mathrm{ref}}$ restando la ecuación (3.15) de la ecuación (3.13) da como resultado

$$\dot{z} = A\tilde{z} + B(u - u_{\mathrm{ref}}) \tag{3.16}$$

La ecuación (3.16) describe la dinámica del error de estado. Esta dinámica debe ser estable y apropiadamente rápida. Una forma de prescribir la dinámica de bucle cerrado es requerir polos de bucle cerrado específicos. Ya se ha demostrado que (A, B) son controlables y por lo

tanto seleccionando adecuadamente una matriz de ganancia de control constante $\boldsymbol{K}$ se pueden lograr ubicaciones arbitrarias de los polos de lazo cerrado en el semiplano de lazo izquierdo del plano complejo s. La ecuación (3.16) se puede reescribir

$$\dot{\tilde{z}} = (\boldsymbol{A} - \boldsymbol{BK})\tilde{z} + \boldsymbol{BK}\tilde{z} + \boldsymbol{B}(\boldsymbol{u} - \boldsymbol{u}_{\text{ref}}) = (\boldsymbol{A} - \boldsymbol{BK})\tilde{z} + \boldsymbol{B}(\boldsymbol{K}\tilde{\boldsymbol{z}} + \boldsymbol{u} - \boldsymbol{u}_{\text{ref}}) \qquad (3.17)$$

Si el último término de la ecuación (3.17) es cero, loe errores de estado convergen a 0 con la dinámica prescrita, dada por la matriz del sistema de bucle cerrado $(\boldsymbol{A} - \boldsymbol{BK})$ Forzar este término a 0 define la ley de control para este enfoque.

$$\boldsymbol{u}(t) = \boldsymbol{K}(\boldsymbol{z}_{\text{ref}}(t) - z(t)) + \boldsymbol{u}_{\text{ref}}(t) \qquad (3.18)$$

La representación esquemática del sistema de control completo se da en la figura 63

Debido a la forma específica de las matrices $\boldsymbol{A}$ y $\boldsymbol{B}$ en la ecuación (3.12) donde u_1 solo influye en los estados z_3 y z_4, la matriz de ganancia de control toma una forma especial

$$\boldsymbol{K} = \begin{bmatrix} k_1 & k_2 & 0 & 0 \\ 0 & 0 & k_3 & k_4 \end{bmatrix} \qquad (3.19)$$

La ley de control (3.19) por lo tanto se puede descomponer completamente

$$\begin{aligned} u_1(t) = \ddot{x}(t) &= k_1\big(\boldsymbol{x}_{\text{ref}}(t) - \boldsymbol{x}(t)\big) + k_2(\dot{x}_{\text{ref}}(t) - \dot{x}(t)) + \ddot{x}_{\text{ref}}(t) \\ u_2(t) = \ddot{y}(t) &= k_3\,(y_{\text{ref}}\big((t) - \boldsymbol{y}(t)\big) + k_4(\dot{y}_{\text{ref}}(t) - \dot{y}(t)) + \ddot{x}_{\text{ref}}(t) \end{aligned} \qquad (3.20)$$

Y los errores $e_x, e_y, \dot{e}_x, \dot{e}_y$ se pueden obtener de la siguiente forma

$$e_x = \big(x_{ref} - x\big) \qquad \dot{e}_x = \big(\dot{x}_{ref} - \dot{x}\big)$$

$$e_y = \big(y_{ref} - y\big) \qquad \dot{e}_y = \big(\dot{y}_{ref} - \dot{y}\big)$$

El siguiente diagrama de la figura 63 representa el control que se le aplica al robot móvil con ruedas para que el sistema siga la trayectoria deseada de forma precisa

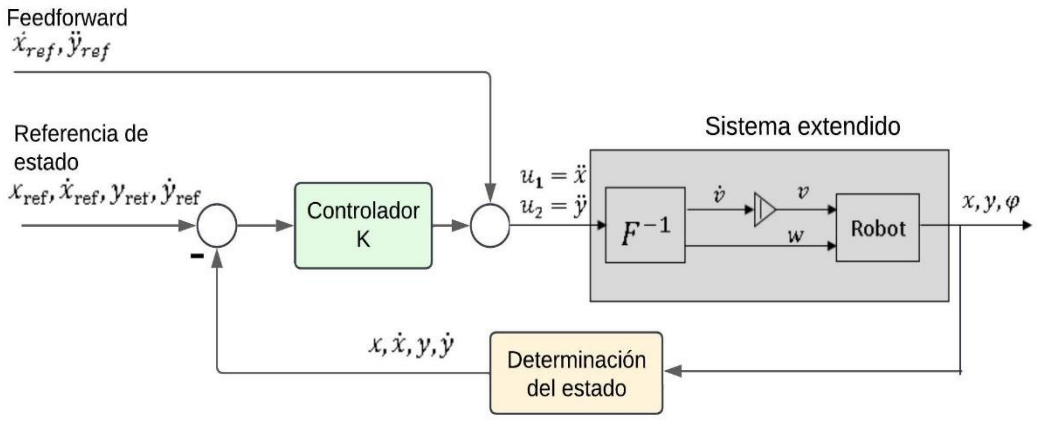

Figura 63. Diagrama de control de linealización [Fuente: Autor]

A continuación, se dará una explicación de cómo funciona este diagrama de control para el seguimiento de trayectoria de un robot móvil.

Las entradas $x_{ref}, \dot{x}_{ref}, y_{ref}, \dot{y}_{ref}$ son la referencia de estado, que representan las trayectorias deseadas del robot, las entradas van al primer sumador y se encuentran con el estado real del sistema $x, \dot{x}, y, \dot{y}$ que vienen de la determinación de estados. Por lo tanto, en el primer sumador se calcula el error de estado, es decir la diferencia entre los valores deseado y los valores reales y como salida del sumador obtenemos el error en posición y velocidad, esto es enviado a la entrada del controlador K, el controlador K aplica una ley de control basada en el error de estado para generar señales de control. Las salidas del controlador K son las correcciones generadas en base al error de estado. Estas correcciones son necesarias para compensar desviaciones entre la trayectoria deseada y el estado actual del robot. La salida del controlador K entran al segundo sumador y se encuentran con el feedforward , $\dot{x}_{ref}, \ddot{y}_{ref}$ que representan las velocidades deseadas en los ejes (x, y) que permiten mejorar el desempeño anticipado para seguir la trayectoria, Por lo tanto, el segundo sumador combina las señales del feedforward con las señales del controlador K y como salida del sumador obtenemos las señales de control u_1, u_2, estas señales se envían al siguiente bloque que es el sistema extendido. Este sistema está compuesto por dos bloques, el primero es la transformación F^{-1}, las entradas de este bloque son las señales de control u_1, u_2 en este bloque se traduce las señales de aceleración en señales físicas que el robot pueda entender, la transformación F^{-1} también toma en cuenta las características físicas del robot como su

modelo cinemático y como salida se obtienen los comandos de movimientos para el robot como la velocidad lineal (v) y la velocidad angular (w). Luego el segundo bloque (Robot) tiene como entradas v y w, el robot utiliza estos comandos v y w para moverse en el espacio y como salida obtenemos los nuevos valores de(x, y)y la orientación φ . Estas salidas entran al bloque de (determinación de estado) y como salida vamos a tener la actualización de los estados reales del sistema $x, \dot{x}, y, \dot{y}$ y todo eso se va estar realimentando y se envían de vuelta al primer sumador cerrando el lazo de realimentación

El diagrama de flujo que se visualiza en la figura 64 representa el proceso de seguimiento de trayectoria para un sistema de control basado en la técnica de linealización por retroalimentación y el uso de variables de estado. Este diagrama es una herramienta útil para comprender los pasos necesarios para guiar al robot hacia una trayectoria deseada, asegurando que siga una secuencia lógica para minimizar errores y garantizar un comportamiento eficiente, mediante la comparación constante entre los estados deseados y los estados reales del sistema, a través de la retroalimentación se corrigen las desviaciones en tiempo real asegurando la precisión.

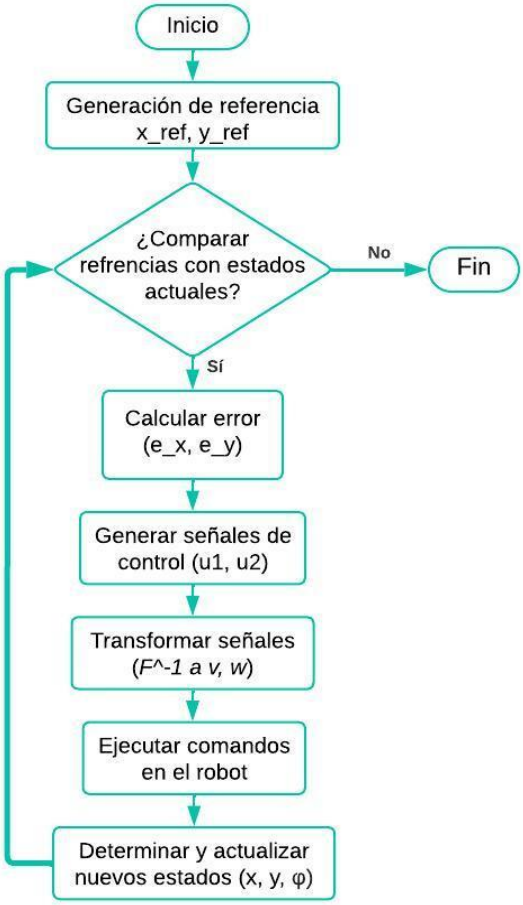

Figura 64. Flujograma de control [Fuente: Autor]

3.2.2.2 Implementación del algoritmo de control en Matlab

En este apartado se presenta el desarrollo del algoritmo de control implementado en Matlab, el cual se centra en el control de seguimiento de trayectoria para un robot móvil, basado en el modelo cinemático y una representación de espacio de estados. A continuación, se presentan las formulas aplicadas del algoritmo de control en el entorno de Matlab

Sistema linealizado en espacio de estados

Para la representación linealizada de un sistema no lineal, se utiliza una aproximación que modela el sistema mediante un conjunto de matrices $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ en espacio de estados. La matriz $\boldsymbol{A}$ describe como las velocidades y posición del robot están relacionados entre sí, $\boldsymbol{B}$ describe como las entradas (aceleración y velocidad angular) afectan a las velocidades y posiciones del robot y $\boldsymbol{C}$ define como se mide el estado del sistema

La matriz ***K*** se utiliza en el controlador de retroalimentación de estado, para calcular la matriz ***K*** se utiliza el método de Acker, el cual está basado en la ubicación de polos deseados del sistema cerrado y también con la matriz de controlabilidad. Los polos son los valores de s en el dominio complejo que determinan el comportamiento dinámico del sistema como la velocidad con la que se estabiliza. En la figura 65 se visualiza un fragmento del código que se aplicó.

```
% Matrices del sistema linealizado
app.A = [0, 1; 0, 0];
app.B = [0; 1];
app.C = [1, 0];

% Configuración de los polos del controlador
desPoles = [-2-1i; -2+1i];
app.K = acker(app.A, app.B, desPoles);
```

Figura 65. Código de programación en Matlab para la matrices y método Acker [Fuente: Autor]

Matriz de transformación *F*

La matriz ***F*** que transforma las aceleraciones $\ddot{x}$ y $\ddot{y}$ en los comandos $\dot{v}$ y w es la siguiente

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \boldsymbol{F} \begin{bmatrix} \dot{v} \\ \omega \end{bmatrix}$$

Cuando

$$\boldsymbol{F} = \begin{bmatrix} \cos\varphi & -v\sin\varphi \\ \sin\varphi & v\cos\varphi \end{bmatrix}$$

Para obtener las entradas $\dot{v}$ y w a partir de las aceleraciones $\ddot{x}$ y $\ddot{y}$, se usa la inversa de la matriz ***F***

$$\begin{bmatrix} \dot{v} \\ \omega \end{bmatrix} = \boldsymbol{F}^{-1} \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix}$$

En el siguiente código de la figura 66 se calculan las entradas de control necesarias para el sistema del robot móvil basado en la orientación actual y las entradas deseadas, se implementa la matriz de transformación ***F*** que relaciona las velocidades, se determina los incrementos necesarios en la velocidad lineal y velocidad angular para aproximarse a las referencias deseadas.

Este fragmento se encarga de transformar las referencias de movimiento globales en comandos concretos que el robot puede ejecutar para el seguimiento de trayectoria.

```
% Cálculo de entradas
F = [cos(app.q(3)), -app.v * sin(app.q(3));
     sin(app.q(3)),  app.v * cos(app.q(3))];
vv = F \ uu;
app.v = app.v + app.Ts * vv(1);
u = [app.v; vv(2)];
```

Figura 66. Código de programación en Matlab para la matriz de transformación F [Fuente: Autor]

Representación del sistema en el espacio de estados

El sistema de ecuaciones en el espacio de estados es:

$$\dot{z} = \boldsymbol{Az} + \boldsymbol{Bu}$$

Donde z es el vector de estados, que incluye las posiciones y velocidades del robot y u son las entradas de control ($\dot{v}$ y w)

La matriz $\boldsymbol{A}$ describe la relación entre los estados y sus derivadas, mientras que la matriz $\boldsymbol{B}$ describe como las entradas $u_1 = \dot{v}$ y $u_2 = w$afectan a las derivadas de los estados.

En el siguiente fragmento de código de la figura 67 el sistema es simulado mediante el cálculo de las velocidades y las posiciones utilizando las ecuaciones de estado, pero no se refleja directamente como una matriz de estado en el espacio de estados, sin embrago la actualización del estado del robot es similar a la formulación en el espacio de estado.

```
% Simulación del movimiento del robot
dq = [u(1) * cos(app.q(3)); u(1) * sin(app.q(3)); u(2)];
app.q = app.q + app.Ts * dq;
app.q(3) = wrapToPi(app.q(3));
```

Figura 67. . Código de programación en Matlab para representación de sistemas en el espacio de estados [Fuente: Autor]

Controlador basado en la referencia

El controlador busca minimizar el error entre el estado actual $z(t)$ y la referencia z_{ref}. La ley de control se describe como

$$\boldsymbol{u}(t) = \boldsymbol{K}(\boldsymbol{z}_{\text{ref}}(t) - z(t)) + \boldsymbol{u}_{\text{ref}}(t)$$

Donde ***K*** es la matriz de ganancias del controlador

La ley de control descompuesta es

$$u_1(t) = \ddot{x}(t) = k_1\big(\boldsymbol{x}_{\text{ref}}(t) - \boldsymbol{x}(t)\big) + k_2(\dot{x}_{\text{ref}}(t) - \dot{x}(t)) + \ddot{x}_{\text{ref}}(t)$$

$$u_2(t) = \ddot{y}(t) = k_3\,(y_{\text{ref}}\big((t) - \boldsymbol{y}(t)\big) + k_4(\dot{y}_{\text{ref}}(t) - \dot{y}(t)) + \ddot{x}_{\text{ref}}(t)$$

Estas ecuaciones representan como se calculan las entradas u_1 (aceleración en x) y u_2 (aceleración en y) a partir de los errores de posición y velocidad. En el fragmento de código de la figura 68 se puede ver la forma del controlador que actúa sobre los errores de la trayectoria de referencia, se ajusta las entradas de control en función del error entre la posición y velocidad actual con las de referencia para reducir el error en el sistema

```
% Error y control
ez1 = zRef1 - z1;
ez2 = zRef2 - z2;
uu = [app.ddxRef(k); app.ddyRef(k)] + [app.K * ez1; app.K * ez2];
```

Figura 68. Código de programación en Matlab para el control basado en la referencia [Fuente: Autor]

3.2.2.3 Diseño de la interfaz gráfica

Para crear la interfaz gráfica utilice la herramienta App Designer de Matlab, esta herramienta es muy útil ya que cuenta con varios elementos como botones, deslizantes, etiquetas, entre otros, por medio de App Designer se puede visualizar y controlar el robot en el seguimiento de trayectorias específicas.

Para crear la interfaz gráfica primero hay que abrir Matlab y en la pestaña Home seleccionar la opción App Designer, esto abrirá el entorno de diseño de aplicaciones la cual muestra dos secciones principales. La primera sección es la vista de diseño donde se colocan y configuran los elementos de la interfaz gráfica, la segunda sección es el editor de código donde se programa la lógica de la aplicación. Para realizar la simulación del seguimiento de trayectoria se necesitan componentes visuales como ejes gráficos, controles interactivos, etiquetas y campos de texto.

Cada componente tiene diferentes configuraciones y propiedades específicas como el tamaño, color y la posición en la que van a ir ubicados. La interacción de los componentes

de la interfaz con la simulación se realiza mediante la programación en Matlab. El propósito de esta interfaz gráfica es que el usuario pueda elegir el tipo de trayectoria que desea que siga el robot y visualizar en tiempo real las velocidades lineales y angulares en cada tramo y las métricas de errores promedio y acumulados. A continuación, se dará una descripción de cómo se creó y configuro los elementos principales de la interfaz enfocada en el seguimiento de trayectoria para un robot móvil.

Título principal

Figura 69. Título principal en la interfaz gráfica [Fuente: Autor]

Para crear el título principal “Seguimiento de Trayectoria” como la figura 69, en la interfaz gráfica se utilizó el siguiente fragmento de código de la figura 70 en el cual utiliza varias funciones que se describirán a continuación

```
% Create App Title
app.TitleLabel = uilabel(app.UIFigure);
app.TitleLabel.Text = 'Seguimiento de Trayectoria';
app.TitleLabel.FontSize = 20;
app.TitleLabel.FontWeight = 'bold';
app.TitleLabel.HorizontalAlignment = 'center';
app.TitleLabel.Position = [430 480 280 45]; % Ajusta la posición y tamaño según el diseño
```

Figura 70. Creación del título principal [Fuente: Autor]

Con “uilabel” se crea una etiqueta de texto, luego “text” muestra el contenido en la etiqueta, con la función “FontSize y FontWeight” se puede ajustar el tamaño y grosor del texto. después con “HorizontalAlignment” se alinea el texto horizontalmente en este caso en el centro y finalmente con “position” se establece la ubicación y dimensiones de la etiqueta

Menú desplegable para selección de trayectoria

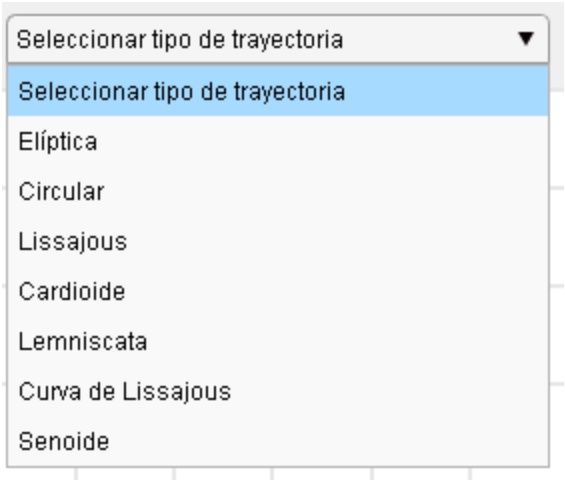

Figura 71. Menú desplegable de la interfaz gráfica [Fuente: Autor]

Para crear el menú desplegable como la figura 71 y el usuario pueda elegir el tipo de trayectoria que desea crear, se utilizó el siguiente fragmento de código de la figura 72 en el cual utiliza varias funciones que se describirán a continuación

```
% Create TrajectoryDropDown
app.TrajectoryDropDown = uidropdown(app.UIFigure);
app.TrajectoryDropDown.Items = {'Seleccionar tipo de trayectoria', 'Elíptica', 'Circular'
app.TrajectoryDropDown.Position = [50 430 50 25];
app.TrajectoryDropDown.Value = 'Seleccionar tipo de trayectoria'; % Valor inicial
```

Figura 72. Creación del menú desplegable [Fuente: Autor]

Con la función “uidropdown” se crea un menú desplegable con varias opciones, luego “items” define la lista de opciones disponibles, en este caso son todas las trayectorias que se van a simular, después “position” especifica la ubicación y dimensiones que va a tener el menú y finalmente con “value” se define el valor inicial seleccionado y se escoge el tipo de trayectoria

Botón de acción

Figura 73. Botón de acción de la interfaz gráfica [Fuente: Autor]

Al momento de iniciar la simulación de la trayectoria seleccionada se debe implementar un botón como en la figura 73 que permita ejecutar dicha acción, para realizar esa acción se utilizó el siguiente fragmento de código de la figura 74 en el cual utiliza varias funciones que se describirán a continuación

```
% Create PlotButton
app.PlotButton = uibutton(app.UIFigure, 'push');
app.PlotButton.ButtonPushedFcn = createCallbackFcn(app, @PlotButtonPushed, true);
app.PlotButton.Position = [550 420 100 25];
app.PlotButton.Text = 'Ejecutar';
```

Figura 74. Creación del botón de acción

Con la función “uibutton” se genera un botón interactivo, luego en “position” se determina la ubicación y tamaño del botón y finalmente con “text” se coloca el texto que aparecerá en el botón en este caso es la palabra ejecutar.

Grafica principal para las trayectorias

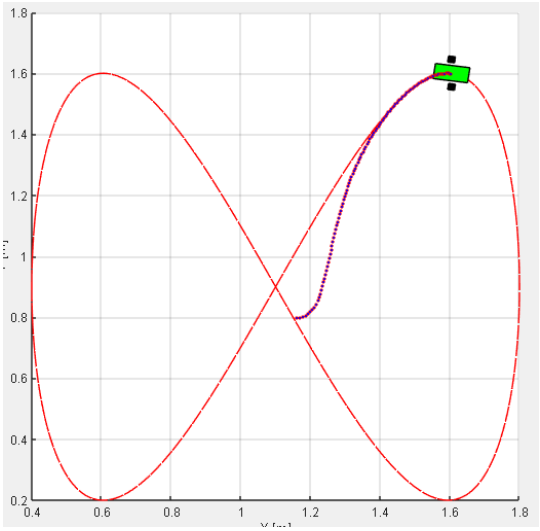

Figura 75. Eje principal para representar las trayectorias en la interfaz gráfica [Fuente: Autor]

Para crear esto se utilizaron los ejes del plano cartesiano como se muestra en la figura 75, los cuales permiten visualizar la trayectoria que se ha seleccionado y el movimiento del robot a través de dicha trayectoria, para esto se utilizó el siguiente fragmento de código de la figura 76 en el cual utiliza varias funciones que se describirán a continuación

```
% Create UIAxes
app.UIAxes = uiaxes(app.UIFigure);
app.UIAxes.Position = [15 75 300 350];
grid(app.UIAxes, 'on');
```

Figura 76. Creación del componente de los ejes [Fuente: Autor]

Con la función “uiaxes” se crea un conjunto de ejes donde se pueden graficar datos, y con “position” se especifica las dimensiones y la ubicación de los ejes

Etiquetas para mostrar métricas

Figura 77. Creación de las métricas en la interfaz gráfica [Fuente: Autor]

Para crear las métricas como se muestran en la figura 77, se utilizan las etiquetas que permiten visualizar la información dinámica en tiempo real como el tiempo transcurrido, las velocidades lineales, angulares y los errores, para realizar esto se utilizó los siguientes fragmentos de códigos de la figura 78, 79 y 80 que comparten las mismas funciones para mostrar las métricas.

```
% Create ElapsedTimeLabel
app.ElapsedTimeLabel = uilabel(app.UIFigure);
app.ElapsedTimeLabel.Position = [50 4 300 40]; % Posición y tamaño
app.ElapsedTimeLabel.Text = 'Tiempo Actual de recorrido: 0.00 s';
```

Figura 78. Creación de etiquetas para el tiempo transcurrido [Fuente: Autor]

```
% Etiquetas de velocidad lineal y angular
app.LinearVelocityLabel = uilabel(app.UIFigure, 'Position', [530 10 200 80], 'Text', 'Velocidad Lineal A
app.LinearVelocityLabel.FontSize = 12;
app.LinearVelocityLabel.FontWeight = 'bold'; % Texto en negrita
app.AngularVelocityLabel = uilabel(app.UIFigure, 'Position', [740 10 200 80], 'Text', 'Velocidad Angular
app.AngularVelocityLabel.FontSize = 12;
app.AngularVelocityLabel.FontWeight = 'bold'; % Texto en negrita
```

Figura 79. Creación de etiquetas para velocidad lineal y angular [Fuente: Autor]

```
% Crear etiquetas para errores
%app.PositionErrorLabel = uilabel(app.UIFigure, 'Position', [600 30 200 100], 'Text', 'Error de Posición: 0 m');
app.AccumulatedErrorLabel = uilabel(app.UIFigure, 'Position', [50 3 300 100], 'Text', 'Error Acumulado de Posición: 0
app.AccumulatedErrorLabel.FontSize = 12;
app.AccumulatedErrorLabel.FontWeight = 'bold'; % Texto en negrita
app.AverageErrorLabel = uilabel(app.UIFigure, 'Position', [280 3 300 100], 'Text', 'Error Promedio de Posición: 0 m')
app.AverageErrorLabel.FontSize = 12;
app.AverageErrorLabel.FontWeight = 'bold'; % Texto en negrita
```

Figura 80. Creación de etiquetas para error promedio y acumulado

Para realizar todas estas etiquetas, se utilizaron la función de "uilabel" que permite crear etiquetas para mostrar el texto dinámico, la función "text" muestra el contenido inicial que se le asigna a cada etiqueta y "position" ajusta la ubicación de las etiquetas

Gráficas para las velocidades lineales y angulares

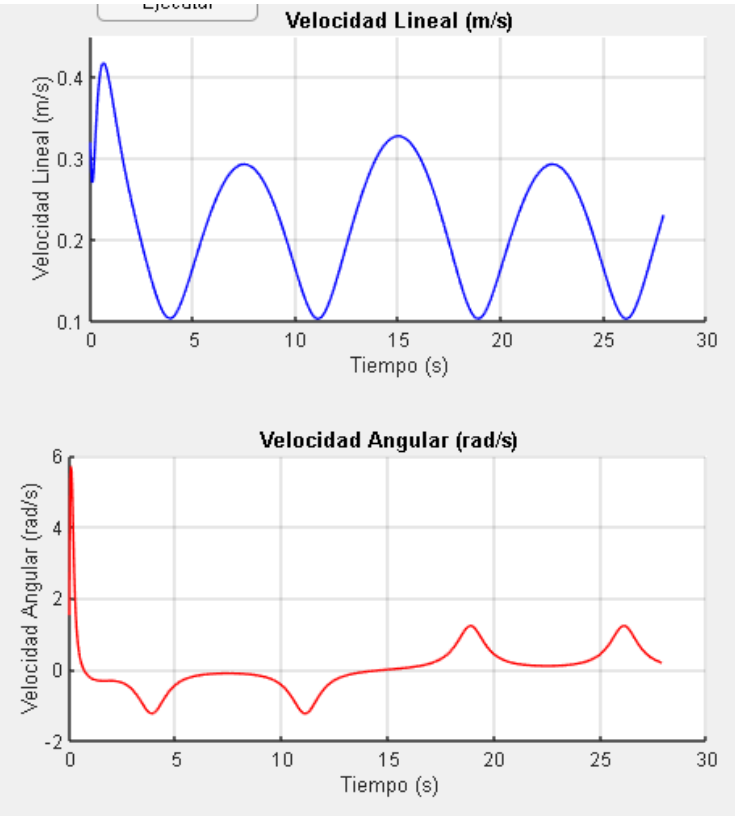

Figura 81. Creación de las métricas en la interfaz gráfica [Fuente: Autor]

Para crear las gráficas de la figura 81, se utilizó los ejes del plano cartesiano, las cuales permiten visualizar las velocidades lineales y angulares del robot a lo largo del tiempo, para

esto se utilizó el siguiente fragmento de código de la figura 82 en el cual utiliza varias funciones que se describirán a continuación

```
% Crear Ejes para la Velocidad Lineal
app.LinearVelocityAxes = uiaxes(app.UIFigure);
app.LinearVelocityAxes.Position = [500 280 250 150];
app.LinearVelocityAxes.Title.String = 'Velocidad Lineal (m/s)';
xlabel(app.LinearVelocityAxes, 'Tiempo (s)');
ylabel(app.LinearVelocityAxes, 'Velocidad Lineal (m/s)');
grid(app.LinearVelocityAxes, 'on');

% Crear Ejes para la Velocidad Angular
app.AngularVelocityAxes = uiaxes(app.UIFigure);
app.AngularVelocityAxes.Position = [500 100 250 150];
app.AngularVelocityAxes.Title.String = 'Velocidad Angular (rad/s)';
xlabel(app.AngularVelocityAxes, 'Tiempo (s)');
ylabel(app.AngularVelocityAxes, 'Velocidad Angular (rad/s)');
grid(app.AngularVelocityAxes, 'on');
```

Figura 82. Creación de la gráfica para la velocidad lineal y angular [Fuente: Autor]

Con la función “uiaxes” se generan las áreas para graficar los datos de la velocidad lineal y angular, luego con “Title.String” se establece el título de cada gráfica, con “position” se ajusta la posición y dimensiones de cada conjunto de ejes, con “LinearVelocityAxes” se representa la velocidad lineal en metros por segundo, y con “AngularVelocityAxes” se representa la velocidad angular en radianes por segundo y con la función “grid” se habilitan las cuadriculas para facilitar la visualización e interpretación de los datos.

Una vez de haber configurado todos los botones, los deslizantes, los ejes de las gráficas, y las etiquetas en el entorno de App Designer, la interfaz gráfica quedaría tal como se muestra en la figura 83. En la parte izquierda de la interfaz está ubicado el deslizante para seleccionar el tipo de trayectoria y el eje principal donde se van a graficar las trayectorias que el robot debe seguir y en la parte inferior se muestran las etiquetas del error promedio, acumulado y el tiempo transcurrido que se demora el robot en seguir la trayectoria. En la parte derecha de la interfaz se muestran los ejes para la velocidad lineal y angular las cuales se muestran en el instante que el robot va siguiendo la trayectoria y en la parte inferior se pueden visualizar las etiquetas para la velocidad lineal representada en m/s y la etiqueta de la velocidad angular en rad/s. Toda esta interfaz permite simular el seguimiento de trayectoria de un robot móvil

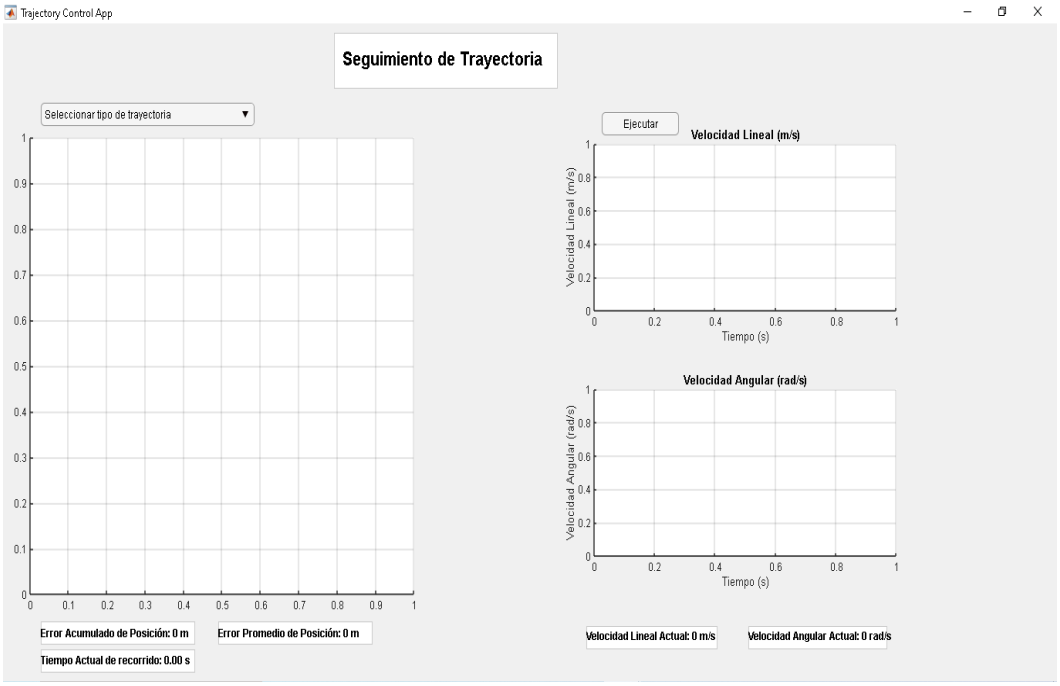

Figura 83. Interfaz para el seguimiento de trayectoria de un robot móvil [Fuente: Autor]

3.3 Pruebas y resultados

En este apartado, se presentan todos los resultados obtenidos en las diversas pruebas realizadas. Se realizaron pruebas en el entorno de Matlab por medio de App Designer, la cual permitió simular el seguimiento de trayectoria de un robot móvil, dentro de estas simulaciones se tomó en cuenta las métricas como las velocidades y errores que presenta el robot en cada instante que va avanzando, con el fin de demostrar que el software funciona correctamente.

3.3.1 Pruebas en Matlab-App Designer

Para realizar las pruebas en el entorno simulado utilice la herramienta App Designer, el cual por medio de la interfaz gráfica se podrá evaluar la precisión, estabilidad y capacidad del robot móvil para seguir la trayectoria. Para comprobar que la interfaz funcione correctamente se realizaron diferentes pruebas en 5 tipos de trayectorias en el entorno simulado. En la figura 84 se puede visualizar que hay una trayectoria de referencia punteada de color rojo la cual el robot debe seguir, a medida que avanza se van a ir mostrando la velocidad lineal y angular del robot en cada instante de la trayectoria y además se mostrara los errores de posición. Con estos datos se podrá interpretar mejor el desempeño del controlador aplicado

.

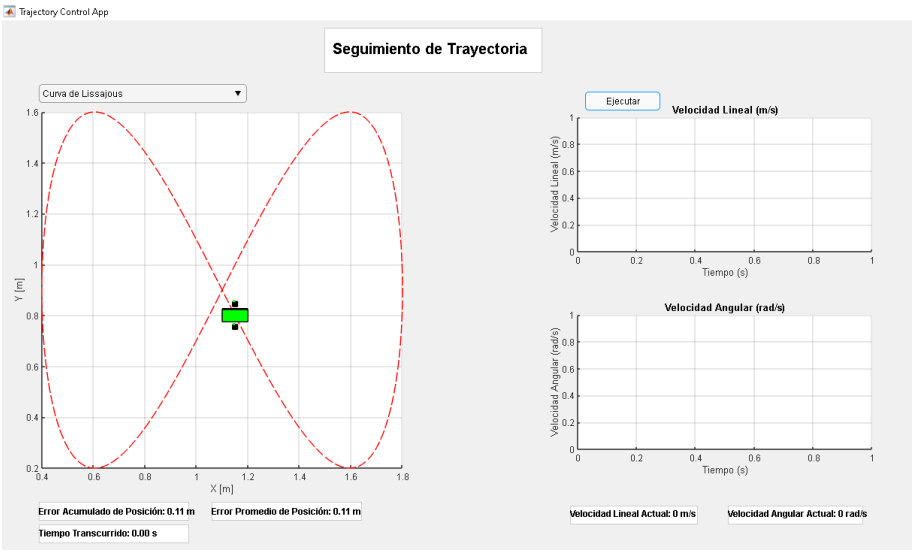

Figura 84. Interfaz para la simulación del seguimiento de trayectoria [Fuente: Autor]

Trayectoria de una curva de Lissajous

La primera prueba que se realizó fue con la curva de Lissajous, por medio de la interfaz gráfica se puede visualizar el comportamiento del robot móvil a través de la trayectoria, en la figura 85 se puede observar que cada vez que avanza el robot se va marcando con una línea morada la trayectoria deseada que el robot a seguido

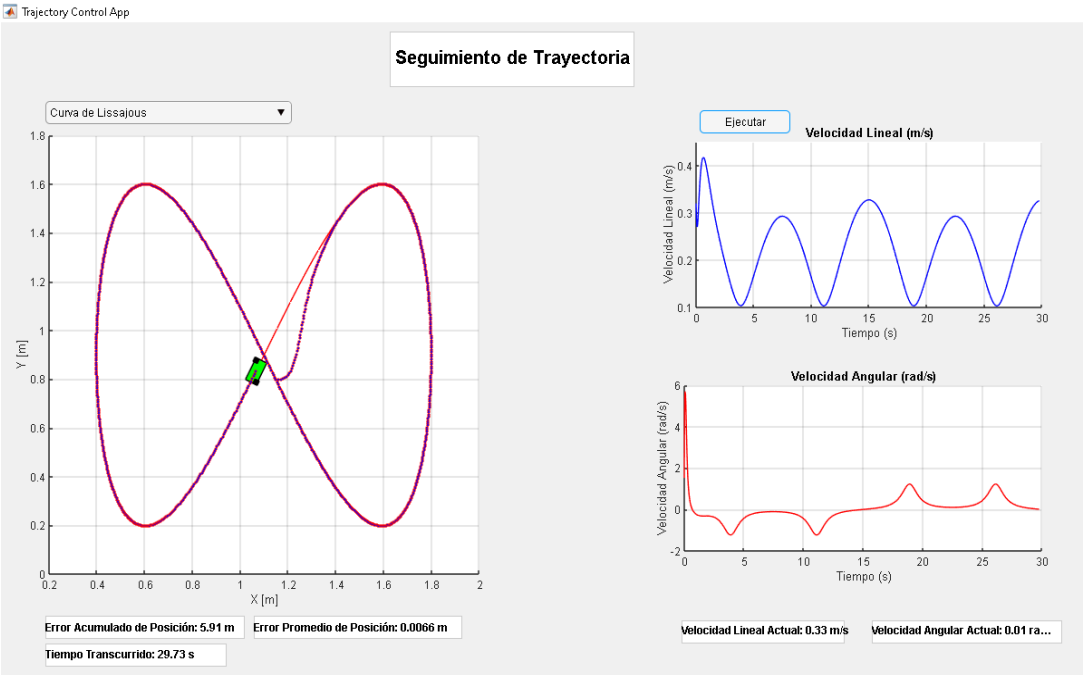

Figura 85. Seguimiento de trayectoria de un robot móvil en la curva de Lissajous [Fuente: Autor]

Por medio de las gráficas de las velocidades se puede interpretar que en el último tramo de la trayectoria la velocidad lineal es de 0.33 m/s, esto nos indica que el robot está en un tramo recto y no detecta mucha curvatura, por lo tanto, la velocidad angular ha disminuido a 0.01rad/s esto indica que el robot ha alcanzado la estabilidad y requiere de pocos ajustes en su orientación. También se puede observar que el error promedio es de 0.0068 m, lo cual indica que se ha podido mantener un control para que el robot móvil pueda seguir la trayectoria deseada, sin embargo, el error acumulado es de 5.91 m lo cual es considerable debido al tiempo transcurrido y las posibles oscilaciones iniciales. Como podemos ver el comportamiento de las velocidades refleja que el robot se adapta a los cambios de curvatura y logra seguir la trayectoria con un error promedio bajo, con esto se puede comprobar la eficacia del controlador en esta trayectoria.

Trayectoria Circular

La segunda prueba que se realizó en el entorno de simulación fue con la trayectoria circular, por medio de la interfaz se puede observar el comportamiento del robot móvil a lo largo de toda la trayectoria, en la figura 86 se puede visualizar que cada vez que el robot avanza va marcando una línea continua de color morada indica la trayectoria deseada que el robot ha logrado seguir

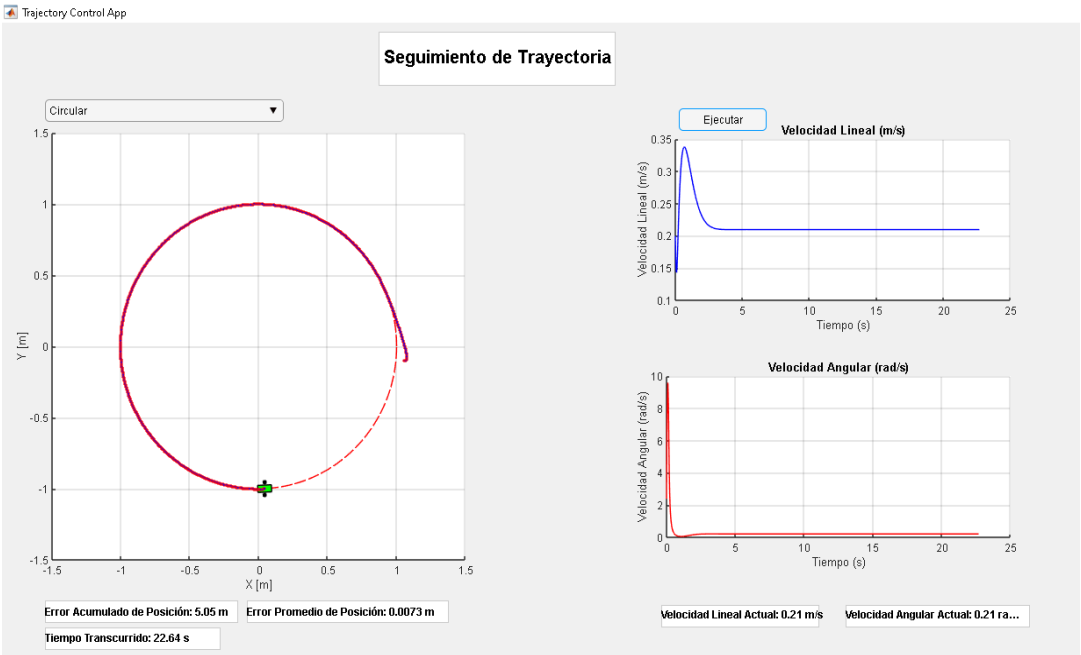

Figura 86. Seguimiento de trayectoria circular [Fuente: Autor]

A través de las gráficas de velocidades se puede interpretar que los picos iniciales altos tanto en la velocidad lineal como angular muestran que el robot corrige su posición y orientación rápidamente para alinearse con la trayectoria circular, una vez alineado las correcciones disminuyen y el movimiento se estabiliza. En este tramo de la trayectoria la velocidad lineal se estabiliza con un valor de 0.21 m/s esto ocurre porque la trayectoria circular requiere un movimiento uniforme sin grandes cambios de velocidad lineal. La velocidad angular se estabiliza en un valor constante de 0.21 rad/s esto refleja el giro uniforme necesario para seguir la trayectoria circular. En este caso el error promedio es de 0.0073 m lo cual indica que el robot se desvía muy poco y logra mantenerse cerca de la trayectoria deseada durante la mayor parte del tiempo. El error acumulado es de 5.55 m este valor indica cómo se comporta el robot móvil y todas las pequeñas desviaciones en el transcurso de la trayectoria

Trayectoria Cardioide

La tercera prueba que se realizo fue con la trayectoria de cardiode, esta es una curva cerrada que incluye un giro interno y una zona externa más amplia, por medio de la interfaz se puede visualizar el comportamiento del robot móvil al seguir esta trayectoria. En la figura 87 se puede observar que cada vez que el robot va avanzando marca una línea continua de color morado que indica la trayectoria deseada que el robot logra seguir

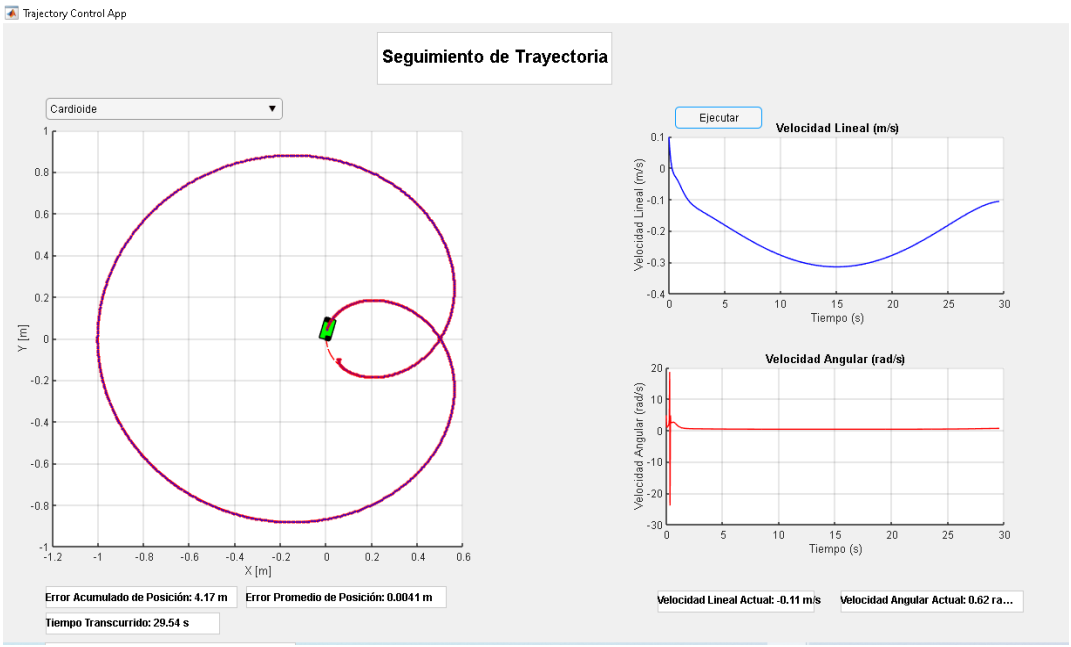

Figura 87. Seguimiento de la trayectoria cardioide [Fuente: Autor]

Por medio de las gráficas de velocidades se puede interpretar que en este tramo de la trayectoria la velocidad lineal es de -0.11 m/s, esto nos indica que el robot móvil retrocede en zonas críticas de la trayectoria especialmente en las curvas cerradas para ajustar el seguimiento, este movimiento es estratégico en zonas donde la orientación es más importante que el avance. En cambio, la velocidad angular es de 0.62 rad/s lo que implica que el robot está girando constantemente en sentido antihorario, la velocidad angular alta permite al robot móvil seguir las curvas cerradas de la trayectoria sin desviarse. Se puede observar que el error acumulado es de 4.17 m que es relativamente bajo y refleja las desviaciones durante toda la trayectoria. El error promedio es de 0.0041 m lo que indica una alta precisión en cada instante, con esto se demuestra que el controlador es capaz de seguir trayectorias complejas como la cardioide con desviaciones mínimas

Trayectoria Elíptica

La cuarta prueba que se realizó en el entorno de simulación, fue con la trayectoria elíptica, por medio de la interfaz gráfica se puede visualizar el comportamiento del robot en cada instante de la trayectoria. En la figura 88 se puede observar que en cada momento que el robot avanza comienza a marcar una línea continua de color morada indicando la trayectoria deseada que el robot ha logrado seguir

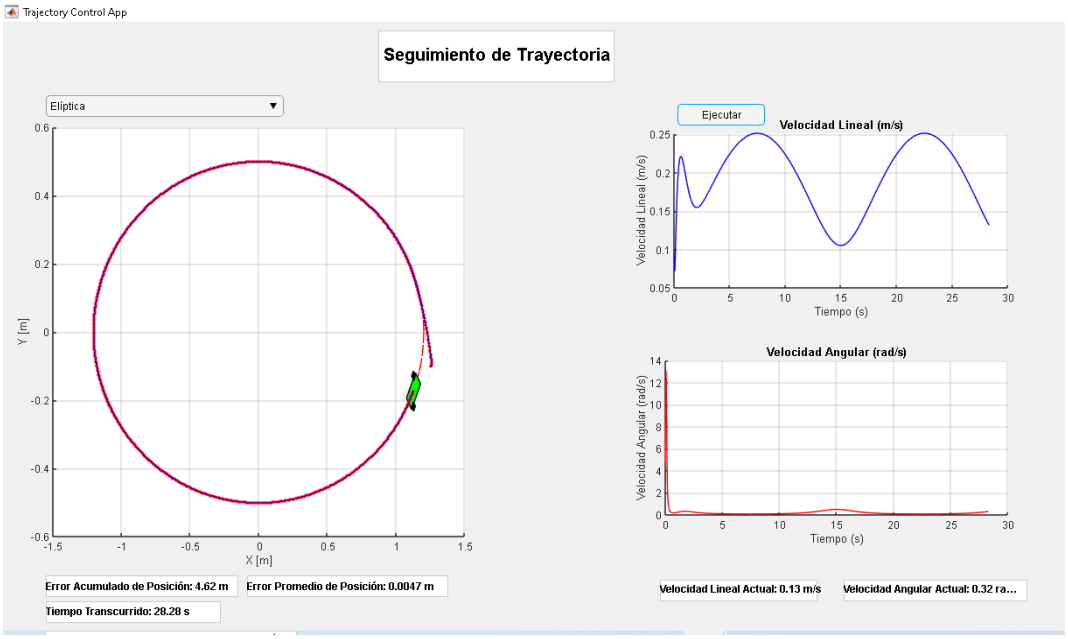

Figura 88. Seguimiento de la trayectoria elíptica [Fuente: Autor]

A través de la gráfica de velocidades se puede interpretar que en este tramo de la trayectoria elíptica la velocidad lineal es baja de 0.13 m/s, esto nos indica que el robot está avanzando lentamente para asegurar un seguimiento preciso debido a que el robot empieza a girar y la velocidad angular es de 0.32 rad/s, esto nos indica que la velocidad angular aumenta en zonas de mayor curvatura para que el robot pueda girar con mayor rapidez reflejando una transición más uniforme. Además, se puede observar que el error promedio es de 0.0047 m lo cual indica que se puede mantener un control estable para que el robot siga la trayectoria elíptica sin desviarse y el error acumulado es de 4.62 m lo cual es considerable debido a las oscilaciones iniciales. Estos valores permiten al robot móvil seguir la trayectoria deseada y se comprueba que el control aplicado es eficaz

Trayectoria Senoide

La quinta prueba que se realizo fue con una trayectoria senoide, en la cual por medio de la interfaz gráfica se puede analizar el comportamiento del robot móvil a lo largo de toda la trayectoria. En la figura 89 se puede observar que cada vez que el robot avanza va marcando la trayectoria con una línea continua de color morado que muestra la trayectoria deseada que el robot ha logrado seguir

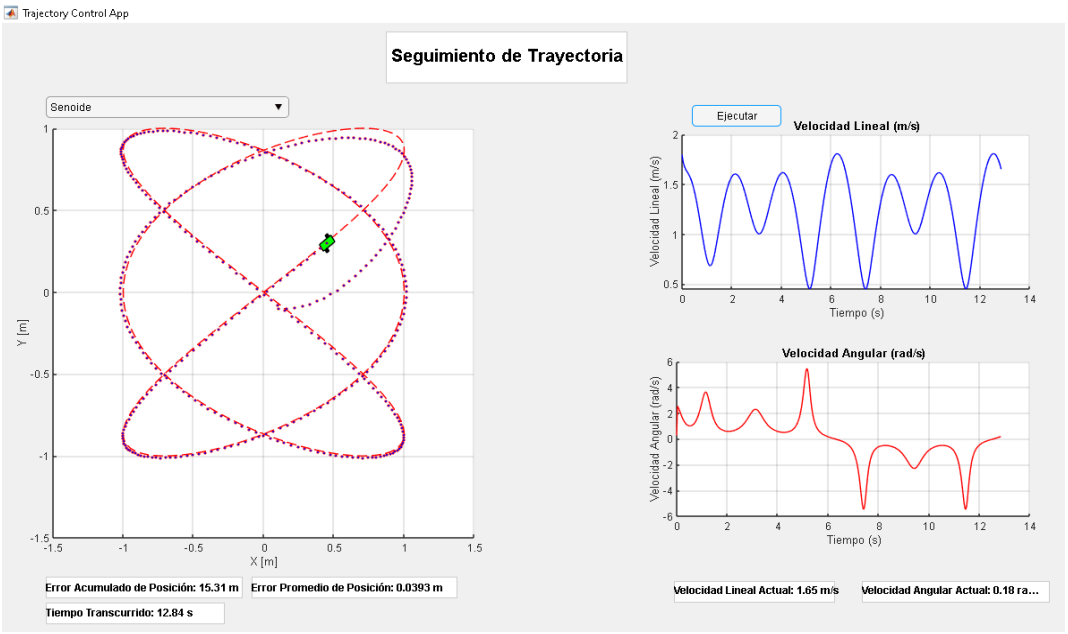

Figura 89. Seguimiento de una trayectoria senoide [Fuente: Autor]

Con las gráficas de velocidades que nos muestra la interfaz, se puede interpretar que la velocidad lineal es de 1.65 m/s, este valor indica que el robot se encuentra en un tramo relativamente recto, por lo tanto, la velocidad lineal va aumentando a medida que avanza. La velocidad angular es de 0.18 rad/s, este valor es bajo por lo que indica que el robot está haciendo un giro leve o se encuentra en una parte de la trayectoria donde hay poca curvatura. El error promedio es de 0.0393 m, esto indica que el robot puede adaptarse con precisión y se desvía muy poco de la trayectoria que va siguiendo. El error acumulado es de 15.31m, este valor indica todos los errores de las áreas donde el robot no logro adaptarse a cambios bruscos a lo largo de la trayectoria. Se debe tener en cuenta si la trayectoria es larga este valor acumulado puede ser aceptable, sin embargo, si la trayectoria es corta este valor podría indicar que existen problemas en el sistema de control.

Trayectoria de Lissajous

Para la sexta prueba que se realizó en el entorno de simulación fue con la trayectoria de Lissajous, esta trayectoria es compleja debido a que combina variaciones constantes en dirección y curvatura. Por medio de la interfaz se puede analizar el comportamiento del robot móvil a lo largo de toda la trayectoria. En la figura 90 se puede observar cada vez que el robot avanza va marcando una línea continua de color morado, esto muestra la trayectoria deseada que el robot ha seguido.

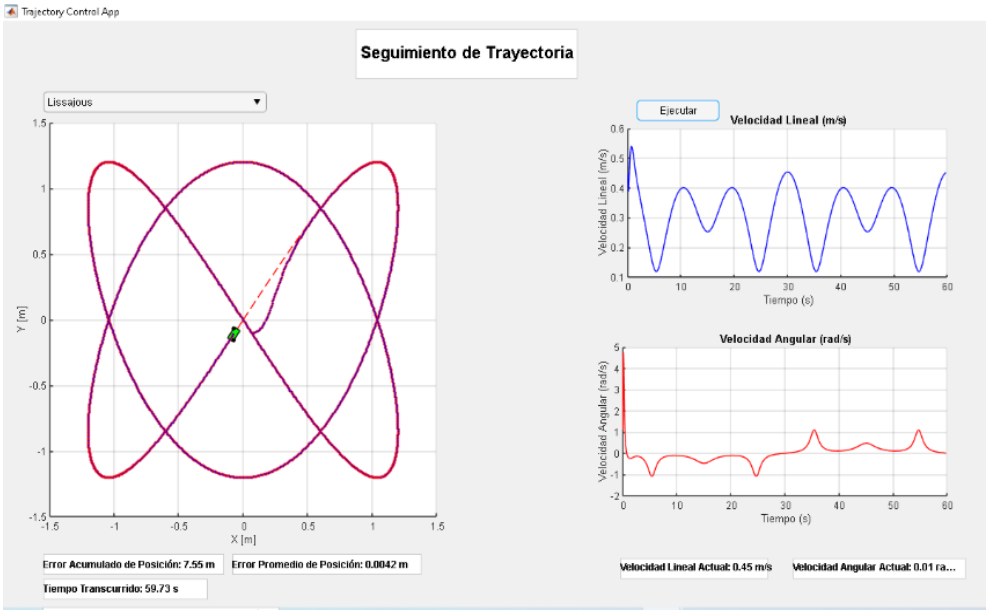

Figura 90. Seguimiento de una trayectoria Lissajous [Fuente: Autor]

Por medio de la interfaz se muestran las gráficas de velocidades, con esto se puede interpretar que la velocidad lineal actual es de 0.45m/s, lo cual indica que el robot se encuentra en un tramo donde el desplazamiento es relativamente recto y constante. La velocidad angular actual es de 0.01 rad/s, esto indica que el robot se encuentra en un tramo donde no existe mucha curvatura, sin embargo, se puede observar que la velocidad angular ha tenido picos altos, esto significa que ha pasado por tramos de mayor curvatura como los cambios de dirección en los extremos de la curva. El error promedio es de 0.0042 m, es bajo considerando la complejidad de la trayectoria, esto indica que el control es preciso con pocas desviaciones entre la trayectoria deseada y de referencia. El error acumulado es de 7.55 m, esto indica la suma de todos los errores instantáneo a lo largo de la trayectoria, este error es razonable, ya que la trayectoria de Lissajous incorpora giros frecuentes y cambios en la dirección lo que aumenta la posibilidad de pequeñas desviaciones.

Resultados

La formulación matemática del algoritmo de linealización de realimentación y variables de estado, que se le aplicó a un robot móvil con ruedas en el seguimiento de trayectorias fue implementado en Matlab para realizar simulaciones y analizar el rendimiento del controlador. Para comprobar el rendimiento del algoritmo de control en el seguimiento de trayectoria se realizaron varias pruebas en el software sobre 6 trayectoria diferentes. Para evaluar el desempeño del controlador se ha tomado en cuenta la métrica del error promedio de posición, este se calcula promediando los errores absolutos de posición generado a lo largo de toda la trayectoria. El error absoluto representa la distancia entre la posición teórica generada por el algoritmo y la posición real del robot en cada punto de la trayectoria. Si se toma en cuenta que el algoritmo de linealización de realimentación y variables de estado es un poco determinista, ya que en algunos casos los resultados de posición pueden ser predecibles y en otros no. La respuesta en la ejecución del algoritmo puede coincidir entre diversas ejecuciones tomando en cuenta la misma trayectoria. Tomando como referencia lo mencionado anteriormente se realizarán 5 ejecuciones para cada una de las 6 trayectorias evaluadas, con el propósito de calcular y comparar el error promedio obtenido en cada ejecución. En la tabla 7 se presentan los valores del error promedio y la desviación estándar correspondiente a las 5 ejecuciones realizadas para cada una de las 6 trayectorias.

	Error Promedio de Posición (%)					
Ejecución	**Curva Lissajouns**	**Circular**	**Cardioide**	**Elíptica**	**Senoide**	**Lissajouns**
1	0.0068	0.0073	0.0073	0.0047	0.0393	0.0042
2	0.0055	0.0080	0.0075	0.0070	0.0390	0.0064
3	0.0040	0.0073	0.0066	0.0064	0.0394	0.0058
4	0.0060	0.0077	0.0075	0.0070	0.0394	0.0076
5	0.0055	0.0079	0.0079	0.0059	0.0385	0.0076
Media (%)	0.0056	0.0076	0.0073	0.0062	0.0391	0.0063
Desviación estándar (%)	0.0010	0.0003	0.0004	0.0009	0.0003	0.0014

Tabla 7. Errores promedios de los 6 tipos de trayectorias de referencia [Fuente: Autor]

A través de los resultados obtenidos se puede interpretar que en la mayoría de las trayectorias el error promedio es bajo, por lo tanto, el algoritmo basado en la linealización de realimentación y variables de estado es confiable. Sin embargo, la desviación estándar es importante, debido a que una desviación baja como la trayectoria circular (0.0003%) o la trayectoria senoide con (0.0003%) indica que el algoritmo es consistente.

En las trayectorias de menor error promedio como la curva de Lissajouns o la elíptica el algoritmo puede seguir estas trayectorias con mayor precisión, a diferencia de la trayectoria senoide que tiene un error promedio alto de 0.0391% esto indica que es una trayectoria un poco desafiante y compleja de seguir para el algoritmo de control. La variabilidad de errores que se presentan indican que el algoritmo va responder de forma distinta de acuerdo a la complejidad de la trayectoria, mostrando un desempeño superior en aquellas de forma geométrica más sencillas.

El algoritmo de linealización de realimentación y variables de estado muestra el error promedio más bajo en la trayectoria de la curva de Lissajouns con 0.0056% y el más alto en la curva de senoide con 0.0391 %. Esta diferencia en los errores promedios da a entender que las trayectorias con cambios más suaves y predecibles de anticipar son más fáciles de seguir para el algoritmo de control.

3.3.2 Pruebas reales para el algoritmo de control con el mBot Neo

Para implementar el algoritmo de linealización de realimentación y variables de estado en el mBot Neo el código de Matlab fue llevado al software mBlock, el cual cuenta con el entorno de editor Python, se realizaron varios ajustes en el código de simulación para adaptarlo a Python, luego ese código se lo almaceno en el módulo CyberPi del mBot Neo, para posteriormente realizar las pruebas reales sobre el entorno físico.

En la prueba física que se realizó para probar el algoritmo de control, se utilizó una pista de madera de 1.80m de ancho por 1.50m de largo y para marcar la trayectoria deseada que va siguiendo el mBot Neo se utilizó un marcador de color rojo. Tal como se muestra en la figura 91

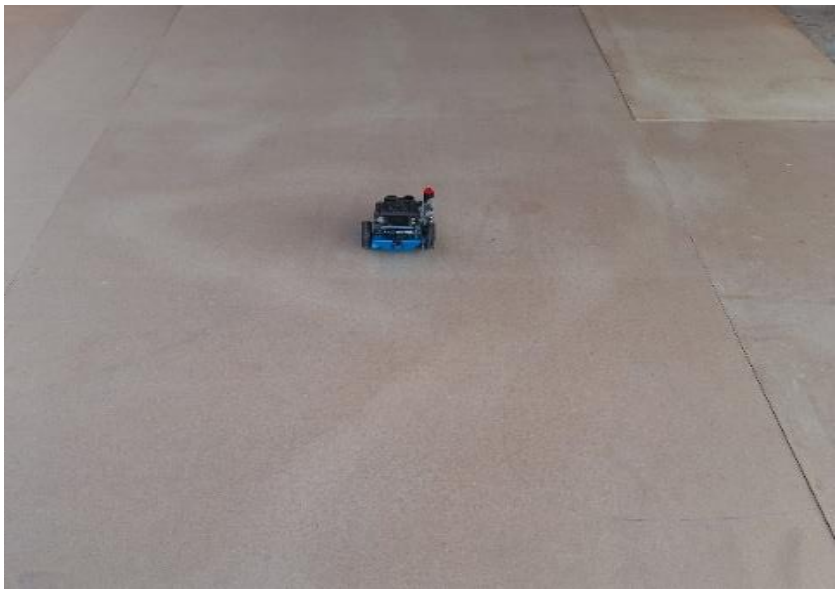

Figura 91. Entorno para realizar pruebas reales con el mBot Neo [Fuente: Autor]

La prueba física que se realizo fue con la curva de Lissajous, en la cual podemos observar en la figura 92 como el robot móvil mBot Neo se desplaza por la pista de madera va realizando la trayectoria de curva de Lissajous, en el cual va marcando de color rojo una línea continua de la trayectoria que ha seguido, con esto se puede comprobar que el algoritmo de control aplicado funciona correctamente.

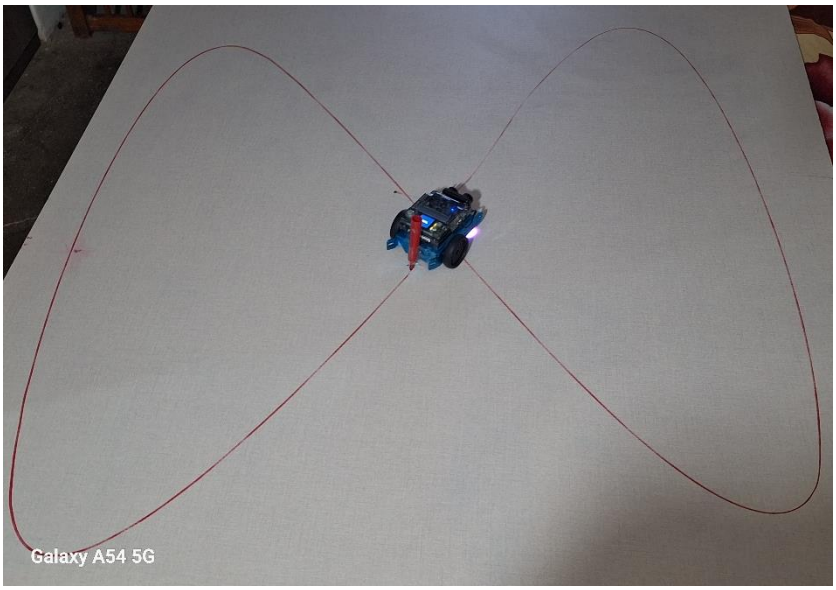

Figura 92. Seguimiento de trayectoria de la curva de Lissajous con el mBot Neo [Fuente: Autor]

La segunda prueba física que se realizo fue con una trayectoria circular, en la cual podemos observar en la figura 93 como el robot móvil mBot Neo se desplaza por la pista de madera y va realizando la trayectoria circular, en el cual va marcando de color rojo una línea continua de la trayectoria que ha seguido, con esto se puede comprobar que el algoritmo de control aplicado funciona correctamente en esta trayectoria.

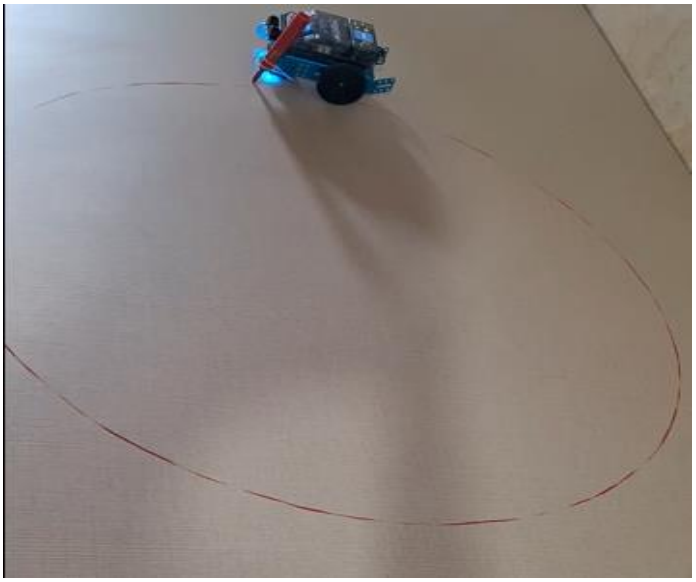

Figura 93. Seguimiento de trayectoria circular con el mBot Neo [Fuente: Autor]

La tercera prueba física que se realizo fue con la trayectoria cardioide, en la cual podemos observar en la figura 94 como el robot móvil mBot Neo se desplaza por la pista de madera y va realizando la trayectoria cardioide, en el cual va marcando de color rojo una línea continua de la trayectoria que ha seguido, con esto se puede comprobar que el algoritmo de control aplicado funciona correctamente en esta trayectoria.

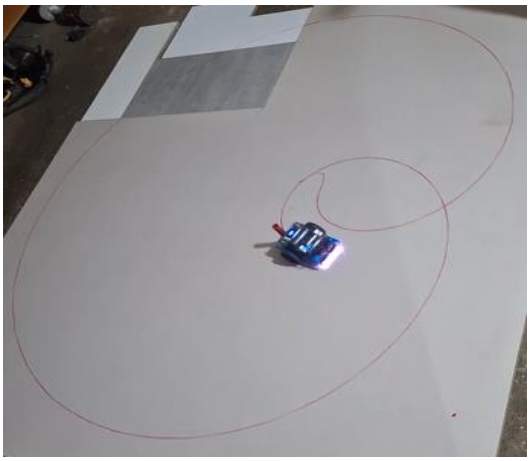

Figura 94. Seguimiento de trayectoria cardioide con el mBot Neo [Fuente: Autor]

La cuarta prueba física que se realizo fue con la trayectoria de Lissajous que es la más compleja, en la cual podemos observar en la figura 95 como el robot móvil mBot Neo se desplaza por la pista de madera y va realizando la trayectoria de Lissajous, en el cual va marcando de color azul una línea continua de la trayectoria que ha seguido, con esto se puede comprobar que el algoritmo de control aplicado en esta trayectoria funciona correctamente.

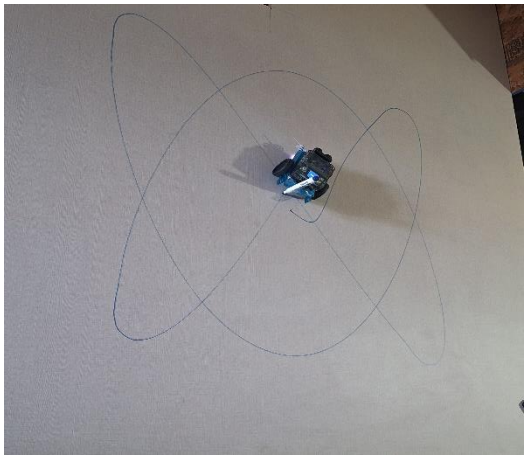

Figura 95. Seguimiento de trayectoria Lissajous con el mBot Neo [Fuente: Autor]

CAPITULO IV

4.1 Conclusiones y Recomendaciones

4.1.1 Conclusiones

- La comprensión de los fundamentos matemáticos fue un paso muy importante ya que esto permitió comprender la cinemática directa e inversa del robot móvil diferencial, por medio de estas ecuaciones se logró obtener la velocidad lineal y angular con la que el robot debe moverse a lo largo de una trayectoria.
- Matlab fue un software clave en esta propuesta, gracias a su entorno matemático se pudo desarrollar el algoritmo de control a través de las ecuaciones que se iban ingresando por medio de funciones y líneas de código. Además, cuenta con una herramienta muy valiosa como el App designer que se utilizó para implementar la interfaz gráfica y visualizar las diferentes simulaciones.
- La herramienta App Designer fue de gran ayuda, ya que permitió crear una interfaz gráfica que representa el seguimiento de trayectoria del robot móvil, por medio de sus elementos como los botones que permitieron ejecutar diferentes acciones, los planos cartesianos que permitieron graficar las trayectorias de referencia y la real, además las gráficas de velocidad lineal y angular que permitieron ver el comportamiento del robot cada vez que giraba o seguía segmentos de línea recta por un tramo de la trayectoria.
- En el entorno de simulación de App Designer se realizaron seis tipos de trayectorias para ver como reaccionaba el robot en diferentes instancias, como por ejemplo en la trayectoria de Lissajous el robot seguía la trayectoria deseada mostrando pocas desviaciones, a diferencia de la trayectoria senoide en la cual al robot le costaba adaptarse a las curvaturas y por esta razón tenía vari as desviaciones lo que aumentaba el valor de los errores acumulados.
- Por medio de las gráficas de las velocidades se pudo interpretar que cada vez que la velocidad angular aumenta y tiene picos altos, el robot gira con mayor rapidez con el propósito de ajustarse a la curvatura de la trayectoria, esto provoca una reducción en la velocidad lineal para mantener el control. Pero si la velocidad lineal aumenta significa que el robot está pasando por tramos rectos y por lo tanto disminuye la

velocidad angular debido a que no existen muchas curvaturas por el tramo en que va avanzando.

- La implementación de las métricas como el error promedio, el error acumulado, la velocidad lineal y angular son muy importantes ya que muestran datos que ayudan a comprender como va comportándose el robot móvil a lo largo de la trayectoria actualizándose en cada instante que avanza.
- Por medio del análisis de los resultados obtenidos, al ejecutar 5 veces las seis trayectorias se obtuvieron diferentes valores de error promedio, los cuales sirvieron para sacar la media de cada trayectoria y la desviación estándar, debido a que en la mayoría de trayectorias el error promedio y la desviación estándar son bajos se pudo comprobar que el controlador implementado en el software funciona correctamente
- Mediante la prueba reales que se realizaron con el mBot Neo en la curva de Lissajous se puede comprobar que el algoritmo de control aplicado funciona correctamente y solo presenta pocas desviaciones al momento de seguir un tramo de la trayectoria donde va girando el robot

4.1.2 Recomendaciones

- Se debe tener en cuenta que al implementar el algoritmo de control en Matlab se debe asignar correctamente las matrices y la función Acker ya que de esto depende los valores de la ganancia K y si se asigna mal, el controlador no va funcionar correctamente y se presentaran muchos errores al momento de realizar las pruebas de simulación
- Al momento de crear la interfaz gráfica en App Designer se recomienda agregar un botón que permita limpiar la pantalla al momento de terminar una trayectoria, ya que puede que en algún momento se colapse el Matlab y no se reinicien los valores de las métricas o se quede dibujada las trayectorias al iniciar una nueva simulación.
- Para armar el robot móvil mBot Neo es muy importante revisar el manual de guía y el datasheet de cada uno de sus componentes ya que estos archivos muestran información muy importante, como las conexiones de los componentes con las placas de expansión y el módulo CyberPi, ya que si conectamos un elemento en un puerto equivocado el robot no va funcionar y hasta podría dañarse algún componente.

- Para realizar la programación del mBot Neo es mejor utilizar el editor de Python que viene integrado en mBlock ya que esto permite agregar funciones y representar las matrices del controlador de una manera más rápida y entendible, debido a que en Scratch hay ciertas limitaciones que no permiten agregar todos los cálculos y ecuaciones que se necesitan para el controlador.
- Al momento de realizar las pruebas en físico con el mBot Neo, se recomienda que la pista este vacía y libre de obstáculos para que el robot pueda moverse libremente y no tenga muchas perturbaciones que alteren el seguimiento de la trayectoria con el propósito de comprobar si el controlador funciona correctamente.
- Se puede realizar una investigación más profunda sobre este algoritmo de control, con el propósito de mejorar la lógica de funcionamiento del controlador, ya que en algunas trayectorias a veces el robot se desviaba un poco en el entorno de simulación y presentaba oscilaciones como por ejemplo en la trayectoria senoide.

REFERENCIAS

[1] L. E. Solaque Guzmán, "SEGUIMIENTO DE TRAYECTORIAS CON UN ROBOT MÓVIL DE CONFIGURACIÓN DIFERENCIAL," 18 Mayo 2014. [Online]. Available: http://revistas.usbbog.edu.co/index.php/IngUSBmed/article/view/298/211.

[2] LOCUS ROBOTICS, "FLOTA LOCUS," 2015. [Online]. Available: https://locusrobotics.com/products/locus-origin/.

[3] M. Wise and M. Ferguson, "Fetch & Freight: Standard Platforms for Service Robot Applications," 2015. [Online]. Available: https://fetchrobotics.com/wp-content/uploads/2021/06/Fetch-and-Freight-Workshop-Paper.pdf.

[4] Á. ARANEDA MORALES, "Simulación y Análisis de algoritmos de planificación y rutas para robot," Enero 2020. [Online]. Available: https://repositorio.unab.cl/xmlui/bitstream/handle/ria/17975/a131453_Araneda_A_Simulacion_y_Analisis_de_algoritmos_2020_tesis.pdf?sequence=1.

[5] AETHON, "Aethon Changes Health Care," 2018. [Online]. Available: https://aethon.com/mobile-robots-for-healthcare/.

[6] M. Aguilera Hernández and M. Bautista, "Diseño y Control de Robots Móviles," 2007. [Online]. Available: http://www.mecamex.net/anterior/cong02/papers/art24.pdf.

[7] J. Cruz Muñoz, "Aplicación de control de movimientos y recopilación de datos de contexto para un robot móvil.," 2017. [Online]. Available: http://hdl.handle.net/11441/62254.

[8] V. R. Barrientos Sotelo, J. R. García Sánchez and R. Silva Ortigoza, "Robots Móviles: Evolución y Estado del Arte," 2007. [Online]. Available: https://www.redalyc.org/pdf/4026/402640448003.pdf.

[9] J. R. Figueroa Olmedo, W. M. Montalvo López and M. M. Bayas Sampedro, "Cinemática y Dinámica de Robots Móviles con Ruedas," Marzo 2023. [Online]. Available: https://ciladi.org/wp-content/uploads/Libro-Robots-VF3.pdf.

[10] J. A. Cook Meneses, "Diseño e implementación de un sistema de generación de trayectorias para un robot móvil utilizando control odométrico," Noviembre 2012. [Online]. Available: https://tesis.pucp.edu.pe/repositorio/bitstream/handle/20.500.12404/1629/COOK_MENESES_JORDI_ROBOT_CONTROL_ODOMETRICO.pdf?sequence=1&isAllowed=y.

[11] F. E. Vásquez Calero , "Diseño de un controlador para seguimiento de la trayectoria no lineal de robots móviles con regulación de la limitación de la velocidad a través de la variación de los parámetros," 2017. [Online]. Available: http://dspace.uazuay.edu.ec/handle/datos/7002.

[12] K. Ogata, Ingeniería de control moderna, Madrid: Pearson Educación , 2010.

[13] . N. Norman S, Control Systems Engineering, California: John Wiley & Sons Inc, 2015.

[14] M. López Mendoza, "LENGUAJES DE PROGRAMACIÓN," OpenWebinars, 16 Julio 2020. [Online]. Available: https://openwebinars.net/blog/que-es-un-lenguaje-de-programacion/.

[15] MathWorks, "MATLAB, Matemáticas. Gráficas. Programación.," [Online]. Available: https://la.mathworks.com/products/matlab.html. [Accessed 2023].

[16] AULA21 , "Python: qué es, para qué sirve y cómo se programa," [Online]. Available: https://www.cursosaula21.com/que-es-python/. [Accessed 2023].

[17] M. Martínez Canelo, "Qué es Scratch y para qué sirve," [Online]. Available: https://profile.es/blog/que-es-scratch/. [Accessed 2023].

[18] S. A. GONZALEZ VERGARA , "“TECNOLOGÍA BLUETOOTH”," AGOSTO 2008. [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/52079743/Tecnologia_Bluetooth-libre.pdf?1489012713=&response-content-

disposition=inline%3B+filename%3DTECNOLOGIA_BLUETOOTH_Tesis_para_obtener.pdf&Expires=1690946262&Signature=MHpRIRTbhVo-eLP0jm-BHzcJ4rmVEWbwn-m7kMAkj3.

[19] L. Lucero Molina and B. Pilapaña Anaguano, "Construcción de una tarjeta electrónica de adquisición de datos con comunicacion por puerto USB," Mayo 2010. [Online]. Available: https://bibdigital.epn.edu.ec/bitstream/15000/2107/1/CD-2886.pdf.

[20] Proofpoint, "¿Qué es el wifi?," [Online]. Available: https://www.proofpoint.com/es/threat-reference/wifi#:~:text=a%20las%20redes.-,%C2%BFQu%C3%A9%20significa%3F,red%20mediante%20frecuencias%20de%20radio.. [Accessed 2023].

[21] ITSQMEET, "¿Quieres ser un maestro de las redes? La radiofrecuencia es tu arma secreta," [Online]. Available: https://itsqmet.edu.ec/radiofrecuencia-la-clave-para-el-exito/#:~:text=La%20radiofrecuencia%20es%20una%20forma,m%C3%B3viles%20y%20conexiones%20Wi%2DFi.. [Accessed 2023].

[22] L. A. Velasco Mellado, "DISEÑO DE UN SISTEMA DE CONTROL BASADO EN LINEALIZACIÓN POR REALIMENTACIÓN PARA UN ROBOT MÓVIL TIPO ACKERMAN CON VELOCIDAD VARIABLE Y MOVIMIENTO EN DOBLE SENTIDO DESCRIBIENDO TRAYECTORIAS ÓPTIMAS," 2017. [Online]. Available: https://tesis.pucp.edu.pe/repositorio/bitstream/handle/20.500.12404/9753/VELASCO_LUIS_SISTEMA_CONTROL_ROBOT_MOVIL_ACKERMAN.pdf?sequence=1&isAllowed=y.

[23] F. Rossomando, C. Soria and R. Carelli, "Control de Robots Móviles con Incertidumbres Dinámicas usando Redes de Base Radial," *RIAI,* vol. 7, no. 4, pp. 28-35, 2010.

[24] S. V. GUTIÉRREZ MARTÍNEZ, "CONTROL DE ROBOTS MÓVILES PARA EL SEGUIMIENTO DE TRAYECTORIAS," Agosto 2021. [Online]. Available: http://eprints.uanl.mx/22156/1/1080315209.pdf.

[25] Makeblock, "Acerca de mBot2/Neo Shield," 28 Febrero 2023. [Online]. Available: https://support.makeblock.com/hc/en-us/articles/12739821665687-About-mBot2-Neo-Shield.

[26] Makeblock Education , "Ultrasonic Sensor 2," 02 Mayo 2022. [Online]. Available: https://education.makeblock.com/help/mbuild-ultrasonic-sensor-2/.

[27] Makeblock Education , "Quad RGB Sensor," 28 Septiembre 2022. [Online]. Available: https://education.makeblock.com/help/mbuild-quad-rgb-sensor/.

[28] Makeblock Education , "180 Optical Encoder Motor," 23 Deciembre 2023. [Online]. Available: https://education.makeblock.com/help/photoelectric-encoder-motor-180rpm/.

[29] Makeblock , "What is CyberPi?," 8 Febrero 2023. [Online]. Available: https://support.makeblock.com/hc/en-us/articles/12259780402199-What-is-CyberPi.

[30] Makeblock Help Center, "What is mBlock-Python Editor?," 13 Agosto 2020. [Online]. Available: https://www.yuque.com/makeblock-help-center-en/mcode/mblock-python.

[31] Á. Villanueva, "mCore: mapeo de puertos y programación," [Online]. Available: https://mecatronicalab.es/mcore-mapeo-de-puertos-y-programacion/?fbclid=IwY2xjawG502lleHRuA2FlbQIxMAABHWbQn7_FLDnGvt5EDSbyvs3CgLX4foAu7vTwypeLyqhY2fgdwaZVDwdljQ_aem_RvkRwkg88wxjLdWWvmTZ8A. [Accessed 5 Noviembre 2024].

[32] R. A. García García and M. Arias Montiel, "Prototipo virtual de un robot móvil multi-terreno para aplicaciones de búsqueda y rescate," Octubre 2016. [Online]. Available: https://www.researchgate.net/publication/309398090_Prototipo_virtual_de_un_robot_movil_multi-terreno_para_aplicaciones_de_busqueda_y_rescate.

[33] D. Hernández, "Este robot araña es una planta con patas," 15 Julio 2018. [Online]. Available: https://computerhoy.20minutos.es/noticias/tecnologia/este-robot-arana-es-planta-patas-277145.

[34] Direct INDUSTRY/Connect, "Robot de inspección de orugas SUGV," 2023. [Online]. Available: https://www.directindustry.es/prod/flir-systems/product-7945-2367386.html.

[35] K. Navarro, "Diseñan robot submarino para mantenimiento portuario," 5 Abril 2018. [Online]. Available: https://www.cienciamx.com/index.php/tecnologia/robotica/20738-robot-submarino-ceid-cetys.

[36] Becas Internacionales, "Robótica Aérea," 6 Junio 2022. [Online]. Available: https://www.becasinternacionales.net/curso/robotica-aerea-2571.

[37] J. A. Cook Meneses, "Diseño e implementación de un sistema de generación de trayectorias para un robot móvil utilizando control odométrico," Noviembre 2012. [Online]. Available: https://tesis.pucp.edu.pe/repositorio/bitstream/handle/20.500.12404/1629/COOK_MENESES_JORDI_ROBOT_CONTROL_ODOMETRICO.pdf?sequence=1&isAllowed=y.

[38] J. Quintana, "Movimientos," Agosto 2024. [Online]. Available: https://libros.catedu.es/books/cyberpi-y-mbot2/page/movimientos.

[39] G. Hernández Millán, L. H. Ríos Gonzales and M. Bueno López, "Implementación de un controlador de posición y movimiento de un robot móvil diferencial," 15 Febrero 2016. [Online]. Available: https://www.redalyc.org/journal/2570/257046835010/.

[40] J. Montesdeoca, L. Salinas, M. Toibero and R. Carelli, "Seguimiento de Trayectoria de un Robot Móvil Tipo Auto," 22 Octubre 2020. [Online]. Available: https://doi.org/10.33414/rtyc.37.157-165.2020.

[41] R. Cabrera , " PROBLEMAS RESUELTOS DE FÍSICA DEL CBC," [Online]. Available: https://ricuti.com.ar/no_me_salen/dinamica/d4FIS_12.html. [Accessed Noviembre 2023].

[42] MathWorks, "bsplinepolytraj," [Online]. Available: https://la.mathworks.com/help/robotics/ref/bsplinepolytraj.html?lang=en. [Accessed Diciembre 2023].

[43] MathWorks, "Seguimiento de rutas para un robot de tracción diferencial," [Online]. Available: https://la.mathworks.com/help/robotics/ug/path-following-for-differential-drive-robot.html. [Accessed Diciembre 2023].

[4] Pequeños Genios, "Como crear tu primer robot," [Online]. Available: https://xn--pequeosgenioscba-bub.com/lenguajesDeProgramacion2023.html. [Accessed Diciembre 2023].

[5] Reclu IT, "¿Qué es MatLab?," 23 Julio 2020. [Online]. Available: https://recluit.com/que-es-matlab/.

[6] Reclu IT, "¿Qué es MatLab?," 23 Julio 2020. [Online]. Available: https://recluit.com/que-es-matlab/.

[7] F. Ramírez, "LENGUAJE PYTHON: QUÉ ES, SUS CARACTERÍSTICAS Y USOS," 7 Abril 2023. [Online]. Available: https://itsoftware.com.co/content/lenguaje-python-que-es-sus-caracteristicas-y-usos/.

[8] PROMETEC, "Introducción a Scratch," [Online]. Available: https://www.prometec.net/scratch-introduccion/. [Accessed Diciembre 2023].

[9] Makeblock, "What is mBot Neo/mBot2?," 7 Abril 2021. [Online]. Available: https://support.makeblock.com/hc/en-us/articles/1500006183021-What-is-mBot-Neo-mBot2.

[10] Makeblock Help Center, "Python API Documentation for CyberPi," 28 Junio 2021. [Online]. Available: https://www.yuque.com/makeblock-help-center-en/mcode/cyberpi-api.

[11] Makeblock, "Assemble mBot Neo/mBot2," 7 Abril 2021. [Online]. Available: https://support.makeblock.com/hc/en-us/articles/1500006253942-Assemble-mBot-Neo-mBot2?fbclid=IwY2xjawG3_EZleHRuA2FlbQIxMAABHYa294o6InXyK2ewVXQTOQNuxo3ZvaK7P3YWa_Opd0R3kWZwillkzeb8QQ_aem_CakoxVqXieVqHBgUudBVBQ.

ANEXOS

ANEXO A-Código en Matlab

```
classdef TrajectoryControlApp < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure               matlab.ui.Figure
        TitleLabel             matlab.ui.control.Label  % Título
principal
        TrajectoryDropDown     matlab.ui.control.DropDown
        PlotButton             matlab.ui.control.Button
        UIAxes                 matlab.ui.control.UIAxes
        ElapsedTimeLabel   matlab.ui.control.Label % Nueva etiqueta para
el tiempo transcurrido
        LinearVelocityLabel    matlab.ui.control.Label % Nueva etiqueta
para velocidad lineal
        AngularVelocityLabel   matlab.ui.control.Label % Nueva etiqueta
para velocidad angular
        LinearVelocityAxes     matlab.ui.control.UIAxes % Ejes para la
velocidad lineal
        AngularVelocityAxes    matlab.ui.control.UIAxes % Ejes para la
velocidad angular
        PositionErrorLabel   matlab.ui.control.Label % Nueva etiqueta
para error de posición
        AccumulatedErrorLabel matlab.ui.control.Label % Nueva etiqueta
para error acumulado
        AverageErrorLabel    matlab.ui.control.Label % Nueva etiqueta
para error promedio
    end

    properties (Access = private)
        Ts = 0.033;          % Tiempo de muestreo
        t;                   % Vector de tiempo
        q;                   % Estado del robot
        xRef; yRef;          % Trayectorias de referencia
        dxRef; dyRef;        % Derivadas de referencia
        ddxRef; ddyRef;      % Derivadas de orden superior
        v;                   % Velocidad inicial
        K;                   % Ganancia de control
        A; B; C;             % Matrices del sistema linealizado
        carPlot;             % Handle del carrito para actualizar en
tiempo real
        wheel1Plot;          % Handle de la rueda trasera
        wheel2Plot;          % Handle de la rueda delantera
        currentTime; % Tiempo actual en segundos
        linearVelocityData; % Datos de la velocidad lineal
        angularVelocityData;% Datos de la velocidad angular
        positionErrors       % Vector para almacenar errores de posición
en cada paso
        accumulatedError     % Error acumulado
        averageError         % Error promedio
    end

    % Callbacks that handle component events
```

```
    methods (Access = private)

        % Setup the initial parameters and UI components
        function startupFcn(app)
            % Vector de tiempo
            app.t = 0:app.Ts:60;

            % Tiempo trancurrido
            app.currentTime = 0; % Inicializamos el tiempo transcurrido
en cero

            % Matrices del sistema linealizado
            app.A = [0, 1; 0, 0];
            app.B = [0; 1];
            app.C = [1, 0];

            % Configuración de los polos del controlador
            desPoles = [-2-1i; -2+1i];
            app.K = acker(app.A, app.B, desPoles);

            % Valor inicial de velocidad
            app.v = 0;
        end

        % Function to update reference trajectories
        function updateTrajectory(app)
            % Obtener el tipo de trayectoria seleccionado y parámetros
            selectedTrajectory = app.TrajectoryDropDown.Value;
            freq = 2 * pi / 30; % Frecuencia común
            a = 1.2; b = 0.5;   % Parámetros para algunas trayectorias

            % Configuración de trayectorias según la selección
            switch selectedTrajectory
                case 'Elíptica'
                    app.xRef = a * cos(freq * app.t);
                    app.yRef = b * sin(freq * app.t);
                    app.dxRef = -a * freq * sin(freq * app.t);
                    app.dyRef = b * freq * cos(freq * app.t);
                    app.ddxRef = -a * freq^2 * cos(freq * app.t);
                    app.ddyRef = -b * freq^2 * sin(freq * app.t);

                case 'Circular'
                    radius = 1;
                    app.xRef = radius * cos(freq * app.t);
                    app.yRef = radius * sin(freq * app.t);
                    app.dxRef = -radius * freq * sin(freq * app.t);
                    app.dyRef = radius * freq * cos(freq * app.t);
                    app.ddxRef = -radius * freq^2 * cos(freq * app.t);
                    app.ddyRef = -radius * freq^2 * sin(freq * app.t);

                case 'Lissajous'
                    freq_x = freq;
                    freq_y = 3 * pi / 30;
                    amplitude = 1.2;
                    app.xRef = amplitude * sin(freq_x * app.t);
```

```
                    app.yRef = amplitude * sin(freq_y * app.t);
                    app.dxRef = freq_x * amplitude * cos(freq_x * app.t);
                    app.dyRef = freq_y * amplitude * cos(freq_y * app.t);
                    app.ddxRef = -freq_x^2 * amplitude * sin(freq_x *
app.t);
                    app.ddyRef = -freq_y^2 * amplitude * sin(freq_y *
app.t);

                case 'Cardioide'
                     radius = 0.5;
                     a = 1;
                    app.xRef = radius * (a * cos(freq * app.t) - cos(2 *
freq * app.t));
                    app.yRef = radius * (a * sin(freq * app.t) - sin(2 *
freq * app.t));
                    app.dxRef = -radius * (a * freq * sin(freq * app.t) -
2 * freq * sin(2 * freq * app.t));
                    app.dyRef = radius * (a * freq * cos(freq * app.t) -
2 * freq * cos(2 * freq * app.t));
                    app.ddxRef = -radius * (a * freq^2 * cos(freq *
app.t) - 4 * freq^2 * cos(2 * freq * app.t));
                    app.ddyRef = -radius * (a * freq^2 * sin(freq *
app.t) - 4 * freq^2 * sin(2 * freq * app.t));

                case 'Lemniscata'
                    a = 1;
                    app.xRef = a * sqrt(2) * cos(freq * app.t) ./
(sin(freq * app.t).^2 + 1);
                    app.yRef = a * sqrt(2) * cos(freq * app.t) .*
sin(freq * app.t) ./ (sin(freq * app.t).^2 + 1);
                    app.dxRef = -a*sqrt(2)*freq*sin(freq*t) .*
((sin(freq*t).^2 + 1).^2) - 2*a*sqrt(2)*freq*cos(freq*t) .*
(sin(freq*t).^2 + 1).^2;
                    app.dyRef = a*sqrt(2)*freq*cos(freq*t) .*
((sin(freq*t).^2 + 1).^2) - 2*a*sqrt(2)*freq*sin(freq*t) .*
(sin(freq*t).^2 + 1).^2;
                    app.ddxRef = -2*a*sqrt(2)*freq^2*cos(freq*t) .*
((sin(freq*t).^2 + 1).^3) + 8*a*sqrt(2)*freq^2*sin(freq*t) .*
((sin(freq*t).^2 + 1).^3);
                    app.ddyRef = -2*a*sqrt(2)*freq^2*sin(freq*t) .*
((sin(freq*t).^2 + 1).^3) - 8*a*sqrt(2)*freq^2*cos(freq*t) .*
((sin(freq*t).^2 + 1).^3);
        % Añade las derivadas como en los valores de la trayectoria

               case 'Curva de Lissajous'
                    app.xRef = 1.1 + 0.7 * sin(freq * app.t);
                    app.yRef = 0.9 + 0.7 * sin(2 * freq * app.t);
                    app.dxRef = freq * 0.7 * cos(freq * app.t);
                    app.dyRef = 2 * freq * 0.7 * cos(2 * freq * app.t);
                    app.ddxRef = -freq^2 * 0.7 * sin(freq * app.t);
                    app.ddyRef = -4 * freq^2 * 0.7 * sin(2 * freq *
app.t);

              case 'Senoide'
                   a = 1;
                   b = 1.5;
```

```
                    c = 1;
                    app.xRef = a * sin(b * app.t);
                    app.yRef = a * sin(c * app.t);
                    app.dxRef = a * b * cos(b * app.t);
                    app.dyRef = a * c * cos(c * app.t);
                    app.ddxRef = -a * b^2 * sin(b * app.t);
                    app.ddyRef = -a * c^2 * sin(c * app.t);
                end
            end

            % Button pushed function: PlotButton
            function PlotButtonPushed(app, event)

            % Verificar si se ha seleccionado una trayectoria válida
              selectedTrajectory = app.TrajectoryDropDown.Value;
              if strcmp(selectedTrajectory, 'Seleccionar tipo de
trayectoria')
              uialert(app.UIFigure, 'Por favor, selecciona un tipo de
trayectoria antes de continuar.', 'Error');
             return;

        end

                % Actualizar trayectoria de referencia
                updateTrajectory(app);
                app.q = [app.xRef(1) + 0.05; app.yRef(1) - 0.1; 0]; %
Posición inicial
                z1 = [app.q(1); app.dxRef(1)];
                z2 = [app.q(2); app.dyRef(1)];
                app.v = sqrt(z1(2)^2 + z2(2)^2);

                % Graficar trayectoria
                plot(app.UIAxes, app.xRef, app.yRef, 'r--'); hold(app.UIAxes,
'on');
                app.UIAxes.XLabel.String = 'X [m]';
                app.UIAxes.YLabel.String = 'Y [m]';
                grid(app.UIAxes, 'on');


                % Definir el carrito como un cuadrado con ruedas
                carWidth = 0.1; % Ancho del carrito
                carHeight = 0.05; % Altura del carrito
                carX = [carWidth/2, carWidth/2, -carWidth/2, -carWidth/2];
                carY = [carHeight/2, -carHeight/2, -carHeight/2,
carHeight/2];

                % Coordenadas de las ruedas
                wheelOffset = carHeight / 2 + 0.02; % Separación de las
ruedas del cuerpo
                wheelRadius = 0.01; % Radio de las ruedas
                wheelX = [-wheelRadius, wheelRadius, wheelRadius, -
wheelRadius];
                wheelY = [wheelRadius, wheelRadius, -wheelRadius, -
wheelRadius];
```

```
            % Dibujar el carrito y las ruedas en la posición inicial
            theta = app.q(3);
            R = [cos(theta), -sin(theta); sin(theta), cos(theta)];
            carPos = R * [carX; carY];
            wheel1Pos = R * [wheelX; wheelY] + [0; -wheelOffset];
            wheel2Pos = R * [wheelX; wheelY] + [0; wheelOffset];

            % Dibujar el carrito
            app.carPlot = fill(app.UIAxes, app.q(1) + carPos(1,:),
app.q(2) + carPos(2,:), 'g');
            app.wheel1Plot = fill(app.UIAxes, app.q(1) + wheel1Pos(1,:),
app.q(2) + wheel1Pos(2,:), 'k');
            app.wheel2Plot = fill(app.UIAxes, app.q(1) + wheel2Pos(1,:),
app.q(2) + wheel2Pos(2,:), 'k');


            % Bucle de simulación
            for k = 1:length(app.t)

                % Actualizar tiempo actual
                app.currentTime = app.t(k); % Tiempo en el índice actual
                app.ElapsedTimeLabel.Text = sprintf('Tiempo Transcurrido:
%.2f s', app.currentTime);

                % Estados de referencia
                zRef1 = [app.xRef(k); app.dxRef(k)];
                zRef2 = [app.yRef(k); app.dyRef(k)];

                % Calcular errores de posición para cada paso
                positionError = sqrt((app.q(1) - app.xRef(k))^2 +
(app.q(2) - app.yRef(k))^2);
                app.positionErrors(k) = positionError;

                % Calcular error acumulado y promedio
                app.accumulatedError = sum(app.positionErrors);
                app.averageError = mean(app.positionErrors(1:k));

                % Actualizar etiquetas de errores
                %app.PositionErrorLabel.Text = sprintf('Error de
Posición: %.2f m', positionError);
                app.AccumulatedErrorLabel.Text = sprintf('Error Acumulado
de Posición: %.2f m', app.accumulatedError);
                app.AverageErrorLabel.Text = sprintf('Error Promedio de
Posición: %.4f m', app.averageError);

                % Error y control
                ez1 = zRef1 - z1;
                ez2 = zRef2 - z2;
                uu = [app.ddxRef(k); app.ddyRef(k)] + [app.K * ez1; app.K
* ez2];

                % Cálculo de entradas
                F = [cos(app.q(3)), -app.v * sin(app.q(3)); ...
                     sin(app.q(3)),  app.v * cos(app.q(3))];
                vv = F \ uu;
```

```
                app.v = app.v + app.Ts * vv(1);
                u = [app.v; vv(2)];

                % Simulación del movimiento del robot
                dq = [u(1) * cos(app.q(3)); u(1) * sin(app.q(3)); u(2)];
                app.q = app.q + app.Ts * dq;
                app.q(3) = wrapToPi(app.q(3));

                % Actualizar el carrito en tiempo real
                theta = app.q(3);
                R = [cos(theta), -sin(theta); sin(theta), cos(theta)];
                carPos = R * [carX; carY];
                wheel1Pos = R * [wheelX; wheelY] + [0; -wheelOffset];
                wheel2Pos = R * [wheelX; wheelY] + [0; wheelOffset];

                % Actualizar gráficos
                set(app.carPlot, 'XData', app.q(1) + carPos(1,:),
'YData', app.q(2) + carPos(2,:));
                set(app.wheel1Plot, 'XData', app.q(1) + wheel1Pos(1,:),
'YData', app.q(2) + wheel1Pos(2,:));
                set(app.wheel2Plot, 'XData', app.q(1) + wheel2Pos(1,:),
'YData', app.q(2) + wheel2Pos(2,:));

                % Dibujar una marca en el camino (punto rojo)
                plot(app.UIAxes, app.q(1), app.q(2), 'ro', 'MarkerSize',
2, 'MarkerFaceColor', 'b');

                % Actualizar estados para el controlador
                z1 = [app.q(1); u(1) * cos(app.q(3))];
                z2 = [app.q(2); u(1) * sin(app.q(3))];

                % Actualizar etiquetas de velocidad
                app.LinearVelocityLabel.Text = sprintf('Velocidad Lineal
Actual: %.2f m/s', app.v);
                app.AngularVelocityLabel.Text = sprintf('Velocidad
Angular Actual: %.2f rad/s', vv(2));

                % Guardar velocidades para graficar
                app.linearVelocityData(k) = app.v;
                app.angularVelocityData(k) = vv(2);

                % Graficar velocidades en tiempo real
                plot(app.LinearVelocityAxes, app.t(1:k),
app.linearVelocityData(1:k), 'b');
                plot(app.AngularVelocityAxes, app.t(1:k),
app.angularVelocityData(1:k), 'r');

                pause(app.Ts);
            end
            hold(app.UIAxes, 'off');
        end
    end

    % Component initialization
    methods (Access = private)
```

```
        % Create UIFigure and components
        function createComponents(app)
            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 700 500];
            app.UIFigure.Name = 'Trajectory Control App';

            % Crear un panel para contener el título
            titlePanel = uipanel(app.UIFigure);
            titlePanel.Position = [430 470 270 60]; % Ajusta la posición
y tamaño según necesites
            titlePanel.BackgroundColor = 'white'; % Fondo blanco
            titlePanel.BorderType = 'line'; % Tipo de borde

            % Create App Title
            app.TitleLabel = uilabel(app.UIFigure);
            app.TitleLabel.Text = 'Seguimiento de Trayectoria';
            app.TitleLabel.FontSize = 20;
            app.TitleLabel.FontWeight = 'bold';
            app.TitleLabel.HorizontalAlignment = 'center';
            app.TitleLabel.Position = [425 480 280 45]; % Ajusta la
posición y tamaño según el diseño

           % Create TrajectoryDropDown
           app.TrajectoryDropDown = uidropdown(app.UIFigure);
           app.TrajectoryDropDown.Items = {'Seleccionar tipo de
trayectoria', 'Elíptica', 'Circular', 'Lissajous', 'Cardioide',
'Lemniscata', 'Curva de Lissajous', 'Senoide'};
           app.TrajectoryDropDown.Position = [50 430 50 25];
           app.TrajectoryDropDown.Value = 'Seleccionar tipo de
trayectoria'; % Valor inicial

            % Create PlotButton
            app.PlotButton = uibutton(app.UIFigure, 'push');
            app.PlotButton.ButtonPushedFcn = createCallbackFcn(app,
@PlotButtonPushed, true);
            app.PlotButton.Position = [550 420 100 25];
            app.PlotButton.Text = 'Ejecutar';

            %Panel para tiempo
            timePanel = uipanel(app.UIFigure);
            timePanel.Position = [50 10 200 25]; % Posición y tamaño del
panel
            timePanel.BackgroundColor = 'white'; % Fondo blanco
            timePanel.BorderType = 'line'; % Borde

            % Create ElapsedTimeLabel
            app.ElapsedTimeLabel = uilabel(app.UIFigure);
            app.ElapsedTimeLabel.Position = [50 4 300 40]; % Posición y
tamaño
            app.ElapsedTimeLabel.Text = 'Tiempo Actual de recorrido: 0.00
s'; % Texto inicial
            app.ElapsedTimeLabel.FontSize = 12;
            app.ElapsedTimeLabel.FontWeight = 'bold'
```

```
            % Create UIAxes
            app.UIAxes = uiaxes(app.UIFigure);
            app.UIAxes.Position = [15 75 300 350];
            grid(app.UIAxes, 'on');

            % Crear Ejes para la Velocidad Lineal
            app.LinearVelocityAxes = uiaxes(app.UIFigure);
            app.LinearVelocityAxes.Position = [500 280 250 150];
            app.LinearVelocityAxes.Title.String = 'Velocidad Lineal
(m/s)';
            xlabel(app.LinearVelocityAxes, 'Tiempo (s)');
            ylabel(app.LinearVelocityAxes, 'Velocidad Lineal (m/s)');
            grid(app.LinearVelocityAxes, 'on');

            % Crear Ejes para la Velocidad Angular
            app.AngularVelocityAxes = uiaxes(app.UIFigure);
            app.AngularVelocityAxes.Position = [500 100 250 150];
            app.AngularVelocityAxes.Title.String = 'Velocidad Angular
(rad/s)';
            xlabel(app.AngularVelocityAxes, 'Tiempo (s)');
            ylabel(app.AngularVelocityAxes, 'Velocidad Angular (rad/s)');
            grid(app.AngularVelocityAxes, 'on');

            % Mostrar figura
            app.UIFigure.Visible = 'on';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';

            % Crear panel para "Velocidad Lineal"
            linearVelPanel = uipanel(app.UIFigure);
            linearVelPanel.Position = [530 35 180 25]; % Posición y
tamaño del panel
            linearVelPanel.BackgroundColor = 'white'; % Fondo blanco
            linearVelPanel.BorderType = 'line'; % Borde

            % Crear panel para "Velocidad Angular"
            angularVelPanel = uipanel(app.UIFigure);
            angularVelPanel.Position = [740 35 210 25]; % Posición y
tamaño del panel
            angularVelPanel.BackgroundColor = 'white'; % Fondo blanco
            angularVelPanel.BorderType = 'line'; % Borde



            % Etiquetas de velocidad lineal y angular
            app.LinearVelocityLabel = uilabel(app.UIFigure, 'Position',
[530 10 200 80], 'Text', 'Velocidad Lineal Actual: 0 m/s');
            app.LinearVelocityLabel.FontSize = 12;
            app.LinearVelocityLabel.FontWeight = 'bold'; % Texto en
negrita
            app.AngularVelocityLabel = uilabel(app.UIFigure, 'Position',
[740 10 200 80], 'Text', 'Velocidad Angular Actual: 0 rad/s');
            app.AngularVelocityLabel.FontSize = 12;
```

```
            app.AngularVelocityLabel.FontWeight = 'bold'; % Texto en
negrita

            % Panel para Error Acumulado
            accumulatedErrorPanel = uipanel(app.UIFigure);
            accumulatedErrorPanel.Position = [50 40 220 25]; % Ajusta
según tu diseño
            accumulatedErrorPanel.BackgroundColor = 'white'; % Fondo
blanco
            accumulatedErrorPanel.BorderType = 'line'; % Tipo de borde

            % Panel para Error Promedio
            averageErrorPanel = uipanel(app.UIFigure);
            averageErrorPanel.Position = [280 40 230 25]; % Ajusta según
tu diseño
            averageErrorPanel.BackgroundColor = 'white'; % Fondo blanco
            averageErrorPanel.BorderType = 'line'; % Tipo de borde

            % Crear etiquetas para errores
            %app.PositionErrorLabel = uilabel(app.UIFigure, 'Position',
[600 30 200 100], 'Text', 'Error de Posición: 0 m');
            app.AccumulatedErrorLabel = uilabel(app.UIFigure, 'Position',
[50 3 300 100], 'Text', 'Error Acumulado de Posición: 0 m');
            app.AccumulatedErrorLabel.FontSize = 12;
            app.AccumulatedErrorLabel.FontWeight = 'bold'; % Texto en
negrita
            app.AverageErrorLabel = uilabel(app.UIFigure, 'Position',
[280 3 300 100], 'Text', 'Error Promedio de Posición: 0 m');
            app.AverageErrorLabel.FontSize = 12;
            app.AverageErrorLabel.FontWeight = 'bold'; % Texto en negrita

        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = TrajectoryControlApp
            % Create and configure components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Call startup function
            startupFcn(app)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)
```

```
            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

Anexo 2 Código mBlock-Python implementación en entorno real

```
- import time, math, cyberpi, event, mbot2
-
- @event.start
- def on_start():
-     cyberpi.console.set_font(16)
-     cyberpi.console.println("Trayectoria")
-     cyberpi.console.println("Curva Lissajous")
-     cyberpi.console.println("Presione Boton A")
-
- # Parámetros y configuraciones iniciales
- Ts = 0.03
- freq = 2 * math.pi/30
- t = [i * Ts for i in range(int(30 / Ts) + 1)]
- r = 0.032
- b = 0.065
-
- # Referencias
- xRef = [1.1 + 0.7 * math.sin((freq * ti * (180 / 3.1416))
  / 180.0 * math.pi) for ti in t]
- yRef = [0.9 + 0.7 * math.sin((2 * freq * ti * (180 / 3.141
  6)) / 180.0 * math.pi) for ti in t]
- dxRef = [freq * 0.7 * math.cos((freq * ti * (180 / 3.1416)
  ) / 180.0 * math.pi) for ti in t]
- dyRef = [2 * freq * 0.7 * math.cos((2 * freq * ti * (180 /
   3.1416)) / 180.0 * math.pi) for ti in t]
- ddxRef = [-
  (freq**2) * 0.7 * math.sin((freq * ti * (180 / 3.1416)) /
  180.0 * math.pi) for ti in t]
- ddyRef = [-
  4 * (freq**2) * 0.7 * math.sin((2 * freq * ti * (180 / 3.1
  416)) / 180.0 * math.pi) for ti in t]
-
- # Función wrapToPi
- def wrapToPi(Angulo):
-     AngToPi = Angulo % (2*math.pi)
```

```
-       if AngToPi > math.pi:
-           AngToPi -= 2*math.pi
-       return AngToPi
-
-   # Lista para almacenar las posiciones del robot
-   posiciones_robot = []
-
-   # Función para mostrar los valores en CyberPi
-   @event.is_press('a')
-   def handle_button_press():
-       global v, z1, z2, q, posiciones_robot
-
-       # Vector de postura inicial
-       q = [xRef[0], yRef[0], 0]
-
-       # Vectores adicionales de estado inicial
-       z1 = [q[0], dxRef[0]]
-       z2 = [q[1], dyRef[0]]
-
-       # Velocidad inicial del robot
-       v = math.sqrt((z1[1]**2) + (z2[1]**2))
-
-       # Vector K
-       K = [5, 4]
-
-       cyberpi.console.set_font(12)
-       cyberpi.audio.set_vol(100)
-       cyberpi.audio.play('start')
-       time.sleep(1)
-       cyberpi.console.clear()
-
-       for i in range(len(t)):
-           zRef1 = [xRef[i], dxRef[i]]
-           zRef2 = [yRef[i], dyRef[i]]
-           ez1 = [zRef1[0] - z1[0], zRef1[1] - z1[1]]
-           ez2 = [zRef2[0] - z2[0], zRef2[1] - z2[1]]
-
-           # Cálculo de la matriz F
-           F = [[math.cos((q[2] * (180 / 3.1416)) / 180.0 * m
    ath.pi), -
    v * math.sin((q[2] * (180 / 3.1416)) / 180.0 * math.pi)],
```

```
-                [math.sin((q[2] * (180 / 3.1416)) / 180.0 * m
  ath.pi),  v * math.cos((q[2] * (180 / 3.1416)) / 180.0 * m
  ath.pi)]]
-
-           # Cálculo de uu
-           uu = [ddxRef[i] + K[0]*ez1[0] + K[1]*ez1[1], ddyRe
  f[i] + K[0]*ez2[0] + K[1]*ez2[1]]
-
-           # Cálculo de la matriz F invertida (pseudo-
  inversa)
-           detF = F[0][0] * F[1][1] - F[0][1] * F[1][0]
-           F_inv = [[F[1][1] / detF, -F[0][1] / detF],
-                    [-F[1][0] / detF, F[0][0] / detF]]
-
-           # Cálculo de vv
-           vv = [F_inv[0][0] * uu[0] + F_inv[0][1] * uu[1],
-                 F_inv[1][0] * uu[0] + F_inv[1][1] * uu[1]]
-
-           # Integración de la aceleración de traslación
-           v = v + Ts * vv[0]
-
-           # Entrada del robot (velocidad lineal y angular)
-           u = [v, vv[1]]
-
-           # Cálculo de dq
-           dq = [u[0] * math.cos((q[2] * (180 / 3.1416)) / 18
  0.0 * math.pi), u[0] * math.sin((q[2] * (180 / 3.1416)) /
  180.0 * math.pi), u[1]]
-
-           # Integración de Euler para actualizar la pose del
   robot
-           q[0] = q[0] + Ts * dq[0]
-           q[1] = q[1] + Ts * dq[1]
-           q[2] = q[2] + Ts * dq[2]
-
-           # Ajuste del ángulo de orientación para mantenerlo
   en [-pi, pi]
-           q[2] = wrapToPi(q[2])
-
-           # Actualización de los estados estimados del siste
  ma
-           z1 = [q[0], u[0] * math.cos((q[2] * (180 / 3.1416)
  ) / 180.0 * math.pi)]
```

```
-          z2 = [q[1], u[0] * math.sin((q[2] * (180 / 3.1416)
   ) / 180.0 * math.pi)]
-
-          # Almacenar la posición del robot
-          posiciones_robot.append((q[0], q[1]))
-
-          # Mostrar los valores en la consola del CyberPi
-          cyberpi.console.clear()
-
-          vx = v * math.cos((q[2] * (180 / 3.1416)) / 180.0
   * math.pi)
-          vy = v * math.sin((q[2] * (180 / 3.1416)) / 180.0
   * math.pi)
-          wd = (1 / 0.032) * ((math.cos((q[2] * (180 / 3.141
   6)) / 180.0 * math.pi) * vx + ((math.sin((q[2] * (180 / 3.
   1416)) / 180.0 * math.pi) * vy + 0.065 * u[1])))
-          wi = (1 / 0.032) * ((math.cos((q[2] * (180 / 3.141
   6)) / 180.0 * math.pi) * vx + ((math.sin((q[2] * (180 / 3.
   1416)) / 180.0 * math.pi) * vy - 0.065 * u[1])))
-          wdRMP = (wd * 60) / (2 * 3.1416)
-          wiRPM = (wi * 60) / (2 * 3.1416)
-          mbot2.drive_speed(wiRPM, -1 * wdRMP)
-          #time.sleep(Ts)
-
-      mbot2.EM_stop("ALL")
-      cyberpi.console.clear()
-      cyberpi.display.show_label("FINAL", 24, "center", inde
   x= 0)
-      cyberpi.audio.play('level-up')
-      time.sleep(1)
-
```

Anexo 3. Certificado de Análisis de plagio

CERTIFICADO DE ANÁLISIS
magister

DESARROLLO E IMPLEMENTACIÓN DE UN CONTROLADOR PARA EL SEGUIMIENTO DE TRAYECTORIA DE UN ROBOT MÓVIL CON RUEDAS BASADO EN LINEALIZACIÓN DE REALIMENTACIÓN Y VARIABLES DE ESTADO

3% Textos sospechosos

3% Similitudes
< 1% similitudes entre comillas
< 1% entre las fuentes mencionadas
< 1% Idiomas no reconocidos (ignorado)
7% Textos potencialmente generados por la IA (ignorado)

Nombre del documento: DESARROLLO E IMPLEMENTACIÓN DE UN CONTROLADOR PARA EL SEGUIMIENTO DE TRAYECTORIA DE UN ROBOT MÓVIL CON RUEDAS BASADO EN LINEALIZACIÓN DE REALIMENTACIÓN Y VARIABLES DE ESTADO.docx
ID del documento: 9370d79cec527d80b7338ffe2683fe8861bb8792
Tamaño del documento original: 12,1 MB
Autores: []

Depositante: Junior Rafael Figueroa Olmedo
Fecha de depósito: 2/12/2024
Tipo de carga: interface
fecha de fin de análisis: 2/12/2024

Número de palabras: 31.628
Número de caracteres: 210.670

Ubicación de las similitudes en el documento:

Fuentes principales detectadas

Nº	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	Documento de otro usuario #53f24a El documento proviene de otro grupo 19 fuentes similares	< 1%		Palabras idénticas: < 1% (132 palabras)
2	Documento de otro usuario #3101ef El documento proviene de otro grupo 17 fuentes similares	< 1%		Palabras idénticas: < 1% (130 palabras)
3	www.slideshare.net | Ingeniería de control: Tema 3. El método del espacio de estad... https://www.slideshare.net/slideshow/ingeniera-de-control-tema-3-el-mtodo-del-espacio-de-esta... 1 fuente similar	< 1%		Palabras idénticas: < 1% (192 palabras)
4	Documento de otro usuario #2e66d0 El documento proviene de otro grupo 7 fuentes similares	< 1%		Palabras idénticas: < 1% (114 palabras)
5	virtual.usalesiana.edu.bo https://virtual.usalesiana.edu.bo/web/contenido/dossier/22011/700.pdf	< 1%		Palabras idénticas: < 1% (63 palabras)

Fuentes con similitudes fortuitas

Nº	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	www.academia.edu | (PDF) Seguimiento de trayectorias de un robot movil diferenci... https://www.academia.edu/49092796/Seguimiento_de_trayectorias_de_un_robot_movil_diferenci...	< 1%		Palabras idénticas: < 1% (34 palabras)
2	tesis.pucp.edu.pe | Browsing Facultad de Ciencias e Ingeniería by Title https://tesis.pucp.edu.pe/repositorio/handle/20.500.12404/7/browse?type=title&offset=1751	< 1%		Palabras idénticas: < 1% (30 palabras)
3	www.doi.org https://www.doi.org/10.1109/TLA.2018.8327385	< 1%		Palabras idénticas: < 1% (19 palabras)
4	dspace.espol.edu.ec https://dspace.espol.edu.ec/bitstream/123456789/6118/33/Chapter_3.pdf	< 1%		Palabras idénticas: < 1% (21 palabras)
5	www.doi.org https://www.doi.org/10.1109/TLA.2016.7530409	< 1%		Palabras idénticas: < 1% (17 palabras)

Fuentes ignoradas

Estas fuentes han sido retiradas del cálculo del porcentaje de similitud por el propietario del documento.

Nº	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	Documento de otro usuario #b76bd4 El documento proviene de otro grupo	12%		Palabras idénticas: 12% (3751 palabras)
2	Documento de otro usuario #d6a1b8 El documento proviene de otro grupo	5%		Palabras idénticas: 5% (1587 palabras)

Nº		Descripciones	Similitudes	Ubicaciones	Datos adicionales
3		**Documento de otro usuario** #877fc7 El documento proviene de otro grupo	3%		Palabras idénticas: 3% (940 palabras)
4		**Documento de otro usuario** #65643a El documento proviene de otro grupo	< 1%		Palabras idénticas: < 1% (137 palabras)
5		**Documento de otro usuario** #6f5415 El documento proviene de otro grupo	< 1%		Palabras idénticas: < 1% (136 palabras)
6		**Documento de otro usuario** #e38556 El documento proviene de otro grupo	< 1%		Palabras idénticas: < 1% (137 palabras)
7		**Documento de otro usuario** #1dbda4 El documento proviene de otro grupo	< 1%		Palabras idénticas: < 1% (136 palabras)
8		**Documento de otro usuario** #cd0ae3 El documento proviene de otro grupo	< 1%		Palabras idénticas: < 1% (136 palabras)
9		**Documento de otro usuario** #17ea90 El documento proviene de otro grupo	< 1%		Palabras idénticas: < 1% (114 palabras)
10		**Documento de otro usuario** #1c98e6 El documento proviene de otro grupo	< 1%		Palabras idénticas: < 1% (113 palabras)

Fuentes mencionadas (sin similitudes detectadas) Estas fuentes han sido citadas en el documento sin encontrar similitudes.

1	http://revistas.usbbog.edu.co/index.php/IngUSBmed/article/view/298/211
2	https://locusrobotics.com/products/locus-origin/
3	https://fetchrobotics.com/wp-content/uploads/2021/06/Fetch-and-Freight-Workshop-Paper.pdf
4	https://repositorio.unab.cl/xmlui/bitstream/handle/ria/17975/a131453_Araneda_A_Simulacion_y_Analisis_de_algoritmos_2020_tesis.pdf?sequence=1
5	https://aethon.com/mobile-robots-for-healthcare/