Desarrollo y validación de un sistema de control para vehículos autónomos en entornos urbanos a escala inspirados en la Ciudad de Guatemala

José Javier Monzón Paz

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo y validación de un sistema de control para vehículos autónomos en entornos urbanos a escala inspirados en la Ciudad de Guatemala

Trabajo de graduación presentado por José Javier Monzón Paz para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo y validación de un sistema de control para vehículos autónomos en entornos urbanos a escala inspirados en la Ciudad de Guatemala

Trabajo de graduación presentado por José Javier Monzón Paz para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

Vo.Bo.:

(f) _____

M. Sc. Carlos Esquit

Tribunal Examinador:

(f) _____

M.Sc. Carlos Esquit

(f) _____

M. Sc. Miguel Enrique Zea Arenales

(f) _____

Ing. Kurt Emmanuel Kellner

Fecha de aprobación: Guatemala, 13 de febrero de 2025.

Prefacio

Agradezco a Dios, por sus bendiciones a lo largo de mi trayecto universitario. Agradezco también profundamente a mis padres, por su apoyo constante y por darme la oportunidad de estudiar, por estar presentes en cada etapa y por motivarme a alcanzar mis metas. También quiero expresar mi gratitud a mis amigos, quienes estuvieron a mi lado compartiendo experiencias, aprendizajes y apoyándome en todo momento. Finalmente, agradezco a mi asesor MSc. Miguel Zea por su guía y valiosos consejos que fueron importantes para el desarrollo y la culminación de este trabajo de graduación.

Índice

Prefacio III

Lista de figuras VIII

Lista de cuadros IX

Resumen X

Abstract XI

1. Introducción 1

2. Antecedentes 3

3. Justificación 8

4. Objetivos 9
4.1. Objetivo general 9
4.2. Objetivos específicos 9

5. Alcance 10

6. Marco teórico 11
6.1. Vehículos autónomos 11
6.2. Modelo cinemático de un robot móvil diferencial 12
6.3. Control de robots móviles con ruedas 15
6.4. Modelo uniciclo para un robot móvil con ruedas 16
6.5. Cinemática diferencial 17
6.6. Cinemática diferencial inversa 17
6.7. Pololu 3Pi+ 17
6.8. Sistemas de captura de movimiento (MoCap) 19
6.9. Algoritmos de control para vehículos autónomos 21
6.10. Filtro de Kalman extendido 22
6.11. Controlador PID con acercamiento exponencial 25

6.12. Odometría 26
6.13. Conversión de cuaterniones de rotación a ángulos de Euler 26

7. Diseño de una infraestructura vial representativa de la Ciudad de Guatemala 28
7.1. Elección de calles y zonas representativas 28
7.1.1. Redondel de la Plaza del Obelisco 28
7.1.2. Carretera a Muxbal 29
7.1.3. Avenida Reforma 29
7.1.4. Zona 1 30
7.2. Diseño de la infraestructura representativa 31
7.3. Validación en simulación de la infraestructura diseñada 36
7.4. Diseño del controlador 38
7.5. Validación del controlador en simulación 39
7.6. Discusión de resultados de simulación 45

8. Implementación del controlador en tiempo real en la plataforma Pololu 3Pi+ 46
8.1. Propuesta de soluciones a las limitantes encontradas 46
8.2. Configuración WiFi del ESP32 47
8.3. Servidor para monitoreo de datos y depuración del programa 48
8.4. Implementación de funcionalidades en ESP32 49
8.4.1. Conexión al servidor del Robotat 49
8.4.2. Obtención de la pose del marcador del vehículo por medio del sistema de captura de movimiento 50
8.4.3. Función de conversión de cuaterniones de rotación a ángulos de Euler . 52
8.5. Detección de zonas muertas del sistema de captura de movimiento en la plataforma del Robotat 53
8.6. Fusión de sensores por medio de filtros complementarios 54
8.6.1. Filtro pasa bajas *leaky integrator* 55
8.6.2. Filtro pasa altas complementario 56
8.6.3. Resultados de estimación de pose utilizando un filtro complementario y dos fuentes de datos 57
8.6.4. Análisis de las limitaciones encontradas en el uso del filtro complementario para fusión de sensores 58
8.7. Fusión de sensores por medio de filtro de Kalman extendido 59
8.7.1. Resultados de estimación de pose utilizando el FKE y dos fuentes de datos 62
8.7.2. Análisis de los resultados y las limitaciones encontradas en el uso del filtro de Kalman para fusión de sensores 63

9. Validación del controlador en la infraestructura diseñada 65
9.0.1. Mejoras en el sistema de captura de movimiento para reducir el efecto de las zonas muertas 65
9.0.2. Resultados de correr las rutas presentadas en la simulación inicial sobre la plataforma del Robotat con el controlador en tiempo real implementado en el Pololu 66

10. Conclusiones **72**

11. Recomendaciones **73**

12. Bibliografía **74**

13. Anexos **76**

13.1. Repositorio de código de controlador y funcionalidades desarrolladas para el ESP32 76

Lista de figuras

1. Vehículo a escala desarrollado en la Universidad de Tecnología de Orsay [4] . 7

2. Diagrama de modelo cinemático de robot diferencial con dos ruedas[10] 13
3. Diagrama de modelo uniciclo de robot móvil con ruedas [13] 16
4. Pololu 3Pi+ [15] 18
5. Pololu 3Pi+ vista superior [15] 19
6. Pololu 3Pi+ vista inferior [15] 19
7. Cámara PrimeX41 [17] 20
8. Arquitectura de redes neuronales convolucionales [18] 21
9. Algoritmos de maquina de vectores de soporte [19] 21
10. Filtro de Kalman extendido [20] 22
11. Algoritmo de Dijkstra [21] 22
12. Diagrama de flujo del filtro de Kalman 25

13. Vías designadas 31
14. Bosquejo 2D dimensionado de la infraestructura en Autodesk Inventor 32
15. Sólido de la infraestructura con carriles delimitados en Autodesk Inventor . . 33
16. Pieza de prueba en forma de rompecabezas 33
17. Plano de prueba en tamaño A0 34
18. Diseño para impresión en lona vinílica 35
19. Calles y zonas representativas de la Ciudad de Guatemala dentro de la infraestructura diseñada 36
20. Mapa digital de la infraestructura diseñada 37
21. Ruta de prueba por el redondel 37
22. Ruta de prueba por zona con curvas pronunciadas 38
23. Ruta de prueba por carril auxiliar y atravesando la avenida 38
24. Simulación de controlador con trayectoria calculada y velocidad máxima de 0.2 m/s 41
25. Errores de simulación de controlador con trayectoria calculada y velocidad máxima de 0.2 m/s 41
26. Simulación de desempeño del controlador con trayectoria calculada y velocidad máxima de 0.2 m/s 42

27. Simulación de controlador con trayectoria calculada y velocidad máxima de 0.7 m/s 42
28. Errores de simulación de controlador con trayectoria calculada y velocidad máxima de 0.7 m/s 43
29. Simulación de desempeño del controlador con trayectoria calculada y velocidad máxima de 0.7 m/s 43

30. Distribución de las seis cámaras de captura de movimiento en la plataforma del Robotat 53
31. Zonas muertas de detección sobre la plataforma del Robotat 54
32. Implementación de filtro complementario para estimación de pose del vehículo sobre la plataforma del Robotat 57

33. 4 marcadores para ayudar al sistema de captura de movimiento en las zonas muertas de la plataforma del Robotat 66
34. Comparación entre primera ruta ideal planificada y la ruta ejecutada por el agente mediante el controlador en tiempo real implementado 67
35. Comparación entre segunda ruta ideal planificada y la ruta ejecutada por el agente mediante el controlador en tiempo real implementado 68
36. Comparación entre tercera ruta ideal planificada y la ruta ejecutada por el agente mediante el controlador en tiempo real implementado 68
37. Errores de ejecución de controlador con primera trayectoria calculada y velocidad máxima de 0.7 m/s 69
38. Errores de ejecución de controlador con segunda trayectoria calculada y velocidad máxima de 0.7 m/s 69
39. Errores de ejecución de controlador con tercera trayectoria calculada y velocidad máxima de 0.7 m/s 70

Lista de cuadros

1. Comparación de errores en diferentes trayectorias y con distintas velocidades . 44

2. Comparación entre pose real y pose estimada utilizando filtros complementarios en posición aleatoria 1 57
3. Comparación entre pose real y pose estimada utilizando filtros complementarios en posición aleatoria 2 57
4. Comparación entre pose real y pose estimada utilizando filtros complementarios en posición aleatoria 3 58
5. Comparación entre pose real y pose estimada utilizando filtros complementarios en posición aleatoria 4 58
6. Comparación entre pose real y pose estimada utilizando filtro Kalman en posición aleatoria 1 62
7. Comparación entre pose real y pose estimada utilizando filtro Kalman en posición aleatoria 2 62
8. Comparación entre pose real y pose estimada utilizando filtro Kalman en posición aleatoria 3 62
9. Comparación entre pose real y pose estimada utilizando filtro Kalman en posición aleatoria 4 63

10. Comparación de errores en diferentes trayectorias en simulación y validación experimental 70

Resumen

Este trabajo presenta el desarrollo y validación de un sistema de control para vehículos autónomos en una infraestructura a escala que representa entornos urbanos complejos inspirados en la Ciudad de Guatemala. Para el control del movimiento del vehículo, se implementaron dos estrategias principales: un controlador PID con acercamiento exponencial y un filtro de Kalman extendido (EKF). El controlador PID permitió el seguimiento de trayectorias predefinidas mediante el control de velocidad y orientación del robot móvil, mientras que el EKF se utilizó para la estimación precisa de la posición y orientación del vehículo a partir de datos provenientes de los *encoders* del agente robótico y un sistema de captura de movimiento.

Los resultados iniciales obtenidos en simulación con MATLAB, considerando la latencia de comunicación entre MATLAB y el ESP32, demostraron que el controlador PID era capaz de seguir trayectorias predefinidas con un error promedio de 0.025 m y un error máximo de 0.126 m a una velocidad baja de 0.2 m/s. Sin embargo, a una velocidad mayor de 0.7 m/s, los errores se incrementaron significativamente, alcanzando un promedio de 0.201 m y un máximo de 0.442 m, evidenciando limitaciones asociadas al sistema de comunicación inalámbrica. Para abordar estas deficiencias, se implementó el controlador en tiempo real dentro del ESP32, lo que eliminó el impacto de la latencia en la comunicación y permitió una evaluación más precisa del desempeño del sistema. En la etapa experimental final, el controlador implementado en el ESP32 mostró mejoría en su desempeño. A 0.2 m/s, se mantuvieron errores promedio de 0.020 m y un error máximo de 0.110 m, mientras que a 0.7 m/s los errores promedio se redujeron a 0.180 m y el error máximo a 0.410 m, debido a la optimización en los parámetros del controlador y a la precisión mejorada en la estimación de pose mediante el filtro de Kalman extendido.

Este proyecto contribuye al campo de la navegación autónoma mediante la validación de un método de control en un entorno urbano representativo. Los hallazgos de este trabajo proporcionan una base para futuras investigaciones en vehículos autónomos en entornos de recursos limitados, con potencial de escalabilidad a contextos reales, promoviendo soluciones de movilidad segura y eficiente en países latinoamericanos con infraestructuras viales complejas.

Abstract

This work presents the development and validation of a control system for autonomous vehicles in a scaled infrastructure representing complex urban environments inspired by Guatemala City. Two main strategies were implemented for vehicle motion control: a PID controller with an exponential approach and an Extended Kalman Filter (EKF). The PID controller enabled the tracking of predefined trajectories by controlling the speed and orientation of the robotic agent, while the EKF was used for precise estimation of the vehicle's position and orientation based on data from the agent's encoders and a motion capture system.

Initial results obtained from simulations in MATLAB, considering the communication latency between MATLAB and the ESP32, demonstrated that the PID controller could track predefined trajectories with an average error of 0.025 m and a maximum error of 0.126 m at a low speed of 0.2 m/s. However, at a higher speed of 0.7 m/s, the errors increased significantly, reaching an average of 0.201 m and a maximum of 0.442 m, highlighting limitations associated with the wireless communication system. To address these deficiencies, the controller was implemented in real-time within the ESP32, eliminating the impact of communication latency and enabling a more accurate evaluation of system performance. In the final experimental stage, the controller implemented in the ESP32 showed improved performance. At 0.2 m/s, average errors of 0.020 m and a maximum error of 0.110 m were maintained, while at 0.7 m/s, average errors were reduced to 0.180 m and maximum errors to 0.410 m, thanks to optimized controller parameters and improved pose estimation accuracy via the Extended Kalman Filter.

This project contributes to the field of autonomous navigation through the validation of a control method in a representative urban environment. The findings of this work provide a foundation for future research on autonomous vehicles in resource-constrained settings, with potential scalability to real-world contexts, promoting safe and efficient mobility solutions in Latin American countries with complex road infrastructures.

CAPÍTULO 1

Introducción

La conducción autónoma ha surgido como una tecnología innovadora que promete revolucionar el transporte, ofreciendo soluciones que mejoran la seguridad y eficiencia en la movilidad. Sin embargo, su implementación en entornos urbanos de países como Guatemala enfrenta desafíos significativos debido a la falta de infraestructura vial adecuada y a las condiciones complejas de tránsito que caracterizan las ciudades latinoamericanas. En estos contextos, los vehículos autónomos deben ser capaces de adaptarse a situaciones dinámicas y responder a obstáculos inesperados para garantizar un desempeño seguro y efectivo.

Este trabajo se enfoca en desarrollar y validar un sistema de control para vehículos autónomos en una infraestructura a escala que represente condiciones urbanas no ideales, tomando como referencia zonas emblemáticas de la Ciudad de Guatemala. La simulación de estas áreas, permite evaluar la capacidad de los algoritmos de control para operar en un entorno realista. Estos entornos plantean desafíos que prueban la adaptabilidad y precisión del sistema de control en condiciones de tráfico y topografía adversas.

Para lograr esto, se diseñó una infraestructura a escala y se implementó un algoritmo de control y percepción mediante la integración de múltiples sensores. Las funcionalidades desarrollados incluyen técnicas de fusión de sensores y métodos de control reactivos que permiten al vehículo autónomo navegar a lo largo de una trayectoria planificada. La plataforma experimental utilizada incluye el agente robótico Pololu 3pi+, que interactúa con el ecosistema Robotat de la Universidad del Valle de Guatemala y módulos de captura de movimiento y procesamiento para una navegación adecuada.

El trabajo forma parte de un proyecto más amplio que busca desarrollar sistemas autónomos y teleoperados para robots móviles. Dentro de este marco, el enfoque general del proyecto completo abarca tanto la simulación y el control remoto de los robots como el desarrollo de estrategias de navegación autónoma. Específicamente, este trabajo se centró en la implementación y validación de sistemas de control autónomo y estimación de estados, abordando las limitaciones inherentes a los entornos complejos y los recursos computacionales restringidos que caracterizan a los agentes robóticos y la infraestructura con la que se trabajó. A diferencia del trabajo complementario, enfocado principalmente en la teleopera-

ción de los robots y el diseño de la infraestructura experimental, este trabajo se distinguió por integrar estrategias de control autónomo y validar su desempeño tanto en simulación como en pruebas experimentales.

CAPÍTULO 2

Antecedentes

En las últimas décadas, el campo de la ingeniería mecatrónica ha impulsado significativamente la investigación y el desarrollo de sistemas de navegación autónoma para vehículos robóticos. Algunos de estos avances se han centrado en abordar los desafíos que presentan los entornos urbanos complejos y no ideales, como los que se encuentran en países latinoamericanos donde la infraestructura vial puede ser deficiente. En este contexto, se han realizado diversos estudios y proyectos que han contribuido al conocimiento y mejoramiento de estos sistemas. Por ejemplo, en la Universidad del Valle de Guatemala se han desarrollado dos soluciones que abordan la infraestructura a escala, algoritmos de visión por computadora y el control de vehículos autónomos a escala. A nivel internacional, instituciones como la Universidad de Tecnología de Orsay y la Universidad Autónoma de Occidente en Cali han llevado a cabo investigaciones que proporcionan resultados favorables en la enseñanza de las tecnologías de conducción autónomas y los sitemas de navegación en ambientes de tipo intersección.

Estos trabajos similares realizados se detallan a continuación.

Pruebas a escala de algoritmos básicos de visión de computadora y control para vehículos autónomos a escala

Este proyecto de graduación presenta un acercamiento a la tecnología de vehículos autónomos en entornos de recursos limitados. El enfoque del proyecto se centra en la creación de un vehículo autónomo a escala utilizando los agentes robóticos Pololu 3pi+ como plataforma base, en conjunto con la infraestructura disponible en el entorno del Robotat de la universidad [1].

A lo largo del trabajo se llevó a cabo una investigación sobre algoritmos de control empleados en la actualidad, destacando el uso de los algoritmos Pure Pursuit y Stanley. Estos algoritmos fueron seleccionados por su eficacia y versatilidad en la navegación autónoma, y fueron evaluados mediante simulaciones en Matlab para verificar su funcionamiento. Posteriormente, se procedió a la implementación de los algoritmos de control en los agentes robóticos Pololu 3pi+ utilizando las herramientas disponibles en el entorno de Robotat.

La validación de los algoritmos de control se realizó a través de pruebas físicas en las cuales se definieron cuatro trayectorias: un círculo, un cuadrado y dos generadas por los métodos D* y DSD. Los resultados obtenidos mostraron una capacidad satisfactoria del Pololu 3pi+ para seguir las trayectorias definidas, lo que confirma la viabilidad de los algoritmos de control seleccionados en este entorno a escala.

Además de la navegación autónoma básica, el proyecto exploró la integración de algoritmos de visión de computadora utilizando el módulo OpenMV cam H7. Estos algoritmos fueron diseñados para la detección de señales de tránsito y el seguimiento de carriles por medio de la detección de las líneas, lo que amplía las capacidades del vehículo autónomo para operar de manera más inteligente y segura en entornos urbanos.

Al analizar los resultados específicos, se encontró que el controlador lateral basado en el algoritmo Stanley, complementado con un PID, demostró un rendimiento satisfactorio en la mayoría de las situaciones de prueba. Sin embargo, se identificaron desafíos en trayectorias con cambios de dirección cerrada, donde el algoritmo de control mostró limitaciones en su capacidad para realizar cambios bruscos de dirección.

En cuanto a la detección de señales de tránsito, se logró una detección satisfactoria de señales de alto y de peatones, lo que demuestra la eficacia del módulo OpenMV cam H7 como sensor óptico complementario. No obstante, se observó una limitación en la velocidad de detección de los algoritmos de visión de computadora mientras el vehículo estaba en movimiento, lo que sugiere áreas para futuras mejoras en la eficiencia del sistema. Otra limitante que presentó el módulo de la cámara es la baja memoria RAM que posee, haciendo que la detección de distintos objetos sea complicada.

Implementación de infraestructura a escala para la evaluación de algoritmos por visión de computador para vehículos autónomos

Este trabajo desarrolla una manera detallada para el acercamiento a la conducción autónoma mediante la integración de la visión por computadora. Este proyecto se propone desarrollar una infraestructura a escala que permita realizar pruebas de visión de computadora en vehículos autónomos utilizando el Polulu 3Pi+ como plataforma base [2].

En primer lugar, se llevó a cabo el diseño y la fabricación de la infraestructura vial a escala. Utilizando una escala de reducción de 1:18.75 basada en la relación de medidas de un vehículo tipo sedán y el Pololu 3Pi+. De igual forma, se tomó en cuenta el tamaño estándar de las calles centroamericanas para definir el ancho de las calles a escala, las cuales se fabricaron con MDF. Se crearon carreteras rectas, con curvas, señales de tránsito y semáforos que reproducen fielmente un entorno vial a escala reducida. Esta fase de fabricación aseguró la precisión y fidelidad del entorno para realizar pruebas realistas y efectivas que representaran un entorno urbano adecuado para las pruebas posteriores.

En cuanto a la parte de visión por computadora, se empleó la OpenMV Cam H7 como sensor principal. Se implementaron algoritmos de detección de carriles y objetos utilizando técnicas avanzadas como la transformada de Hough y filtros de color RGB565. Estos algoritmos permitieron la detección y clasificación precisa de objetos clave en el entorno vial, como líneas de carril, señales de tránsito y estados de semáforos. Además, se aprovechó la plataforma de Edge Impulse para entrenar y optimizar modelos de machine learning, lo que resultó en un modelo TensorFlow Lite capaz de detectar cinco clases de objetos con un alto

grado de precisión y eficiencia computacional.

Las conclusiones obtenidas destacan el éxito en todas las etapas del proyecto. La infraestructura vial a escala se diseñó y fabricó con precisión, proporcionando un entorno realista para las pruebas de los algoritmos de visión por computadora. Los algoritmos implementados demostraron un rendimiento alto en la detección y clasificación de objetos, con altos porcentajes de validación y eficiencia computacional. Además, se logró una integración exitosa entre la infraestructura, los algoritmos de visión por computadora y el vehículo autónomo a escala. Esta integración permite realizar pruebas realistas y efectivas del funcionamiento del vehículo autónomo en un entorno vial simulado, lo que facilita la evaluación y mejora de los algoritmos de control y visión por computadora. Estos resultados representan un avance significativo en el campo de la conducción autónoma, con el potencial de impulsar el desarrollo de soluciones más seguras y eficientes para la movilidad en entornos urbanos y de recursos limitados.

Desarrollo de un sistema para la navegación autónoma de ambientes urbanos tipo intersección y su evaluación en la plataforma Duckietown

Este proyecto presenta un aporte mediante la implementación de un Sistema de Navegación Autónoma basado en la plataforma Duckietown [3]. Este sistema, destinado a un robot diferencial equipado con una cámara monocular, demuestra la capacidad de identificar y tomar decisiones respecto a objetos de interés en ambientes urbanos, tales como semáforos, peatones, señales de tránsito, vehículos y señalizaciones viales.

Un aspecto destacado de este proyecto radica en la integración de algoritmos de inteligencia artificial y aprendizaje de máquina, junto con algoritmos de planificación clásica, como Rapid-exploring Random Tree. Esto permite un enfoque integral que aborda tanto la detección de objetos como el control del vehículo, basándose en las directrices de la Administración Nacional de Seguridad del Tráfico en las Carreteras. El proceso de detección de objetos se lleva a cabo mediante un proceso de entrenamiento de redes neuronales utilizando la biblioteca TensorFlow, empleando imágenes capturadas durante la operación remota del robot en el entorno urbano.

Para el control del vehículo, se diseñó un algoritmo que, basado en la información proporcionada por el detector de objetos y otros subsistemas del vehículo, toma decisiones sobre la navegación en intersecciones, siguiendo las reglas de tráfico establecidas. La implementación modular de este algoritmo, utilizando las librerías de Duckietown, garantiza una adaptabilidad y compatibilidad óptimas con la plataforma de pruebas.

La validación del sistema se llevó a cabo mediante pruebas de funcionamiento en el robot diferencial, utilizando el software Docker para su compilación y ejecución. Los resultados obtenidos destacan un tiempo de detección de objetos inferior a 1 segundo, junto con un rendimiento alto en la navegación de intersecciones, lo que culmina en la creación de un sistema automatizado de navegación autónoma a escala en el entorno del Robot Operating System (ROS), utilizado ampliamente en la comunidad robótica internacional.

En las conclusiones de este trabajo, se destacan varios aspectos positivos. En primer lugar, se destaca el éxito del sistema de detección de objetos, el cual se basó en una recolección adecuada de muestras de entrenamiento para la red neuronal, resultando en un detector con un rendimiento notable. Sin embargo, se identificaron áreas para futuras mejoras, como la

optimización del rendimiento de la red neuronal mediante un aumento en el número de muestras de entrenamiento, especialmente para clases específicas de objetos. Por otro lado, se observó que el principal desafío del sistema radica en el tiempo de procesamiento, debido a las limitaciones computacionales de la plataforma utilizada. Esto resalta la necesidad de explorar soluciones para mejorar la eficiencia computacional, especialmente en entornos donde el tiempo de respuesta es crítico. Finalmente, se concluye que el algoritmo de toma de decisiones desarrollado demostró ser capaz de analizar el entorno y tomar decisiones autónomas para cruzar intersecciones de manera segura, destacando la utilidad del sistema ROS.

Una plataforma modelo a escala de código abierto para la enseñanza de las tecnologías de vehículos autónomos

Esta investigación se basa en la elaboración de plataformas educativas destinadas a la enseñanza de tecnologías asociadas con los Vehículos Autónomos [4]. El estudio se basa en la propuesta y desarrollo de una plataforma de vehículo a escala de código abierto diseñada específicamente para enseñar los conceptos básicos de las tecnologías de vehículos autónomos, adaptada para estudiantes universitarios y técnicos.

El diseño de la plataforma se basó en la utilización de principios de código abierto para fomentar la colaboración y la mejora continua. Se seleccionaron cuidadosamente los componentes mecánicos y electrónicos para garantizar su adecuación y funcionalidad en el contexto de la enseñanza de los vehículos autónomos. Esto incluyó la elección de actuadores, sensores y unidades de procesamiento que replicaran de manera realista los sistemas presentes en los vehículos autónomos a escala real.

Una parte importante del proceso fue el desarrollo de la arquitectura de comunicación y control del vehículo. Se diseñó un sistema distribuido basado en la red CAN (Controller Area Network), utilizada en la industria automotriz, para permitir una comunicación eficiente entre los diferentes componentes del vehículo. Esto implicó la implementación de nodos de comunicación distribuidos que estructuraban las funciones complejas en bloques funcionales unitarios.

Posteriormente, se procedió a la fabricación y montaje de la plataforma utilizando técnicas como la impresión 3D y el mecanizado CNC para garantizar la precisión y durabilidad de los componentes. Se realizaron pruebas rigurosas para verificar el correcto funcionamiento de todos los sistemas integrados y se realizaron ajustes según fuera necesario. Paralelamente al desarrollo técnico de la plataforma, se crearon recursos educativos detallados y prácticos para su implementación en el proceso de enseñanza. Estos recursos incluyeron materiales didácticos, ejercicios prácticos y proyectos de investigación que permitieron a los estudiantes explorar y experimentar con los conceptos clave de los vehículos autónomos en un entorno controlado.

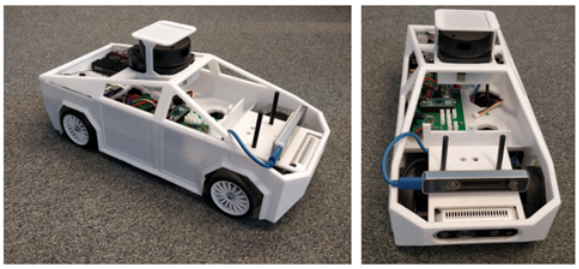

Figura 1: Vehículo a escala desarrollado en la Universidad de Tecnología de Orsay [4]

Los logros de este trabajo destacan el desarrollo de una plataforma a escala que permite a los estudiantes probar y desarrollar nuevas tecnologías que controlen a los vehículos autónomos. Los investigadores resaltan la ventaja del bajo costo y la seguridad que presenta su plataforma a escala, ya que un vehículo de tamaño real resulta ser muy costoso y hasta peligroso de manipular en un entorno cerrado. Esta plataforma también ofrece una amplia variedad de sensores y actuadores, lo que la hace versátil a la hora de probar algoritmos, a la vez que permite recolectar una amplia variedad de información del ambiente en el que se encuentra trabajando.

CAPÍTULO 3

Justificación

En países latinoamericanos como Guatemala, la educación e infraestructura vial sufren de un déficit de calidad, provocando así múltiples accidentes y percances día a día [5], los cuales podrían ser fácilmente evitados si existiera un ambiente ordenado, eficiente y coordinado entre conductores y su entorno. Es por estas carencias que los vehículos autónomos que se encuentran actualmente en el mercado no pueden funcionar con total seguridad, ya que fueron diseñados para operar en un entorno más ordenado y con infraestructura de calidad. Sus algoritmos de control no están diseñados para tomar en cuenta las diversas variables como motoristas navegando entre carriles, conductores que no respetan las señalizaciones de tránsito, mala delimitación de los carriles, peatones que cruzan la calle donde no deben, agujeros en el pavimento u otros obstáculos dentro de la vía pública. Poder diseñar y validar un método de control adecuado y representativo que cumpla con la seguridad y efectividad del vehículo autónomo y que pueda tomar en cuenta todas estas variables y perturbaciones, permitiría escalarlo en un futuro a un vehículo real e implementarlo en vehículos autónomos comerciales.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar y validar un sistema de control para vehículos autónomos en entornos urbanos no ideales, basado en una infraestructura vial representativa de la Ciudad de Guatemala.

4.2. Objetivos específicos

- Diseñar una infraestructura a escala representativa de la infraestructura vial de la Ciudad de Guatemala, basándose en calles conocidas y altamente transitadas.
- Implementar un método de control de vehículos autónomos y evaluar sus resultados de desempeño para determinar su eficacia dentro de ambientes no ideales.
- Validar el método de control elegido dentro del ecosistema Robotat y la infraestructura representativa creada.

CAPÍTULO 5

Alcance

El alcance de este trabajo se enfocó en el diseño, construcción y validación de una infraestructura a escala que reproduce aspectos característicos de un entorno urbano inspirado en la Ciudad de Guatemala, con el fin de probar y evaluar un sistema de control para vehículos autónomos. La infraestructura fue diseñada para simular condiciones reales, incluyendo intersecciones, curvas pronunciadas y áreas de tráfico denso, lo cual permitió replicar desafíos comunes de la movilidad en entornos urbanos complejos y no ideales.

Se desarrollaron e implementaron algoritmos de control y percepción, los cuales fueron probados en el agente robótico Pololu 3pi+ en conjunto con el sistema de captura de movimiento del ecosistema Robotat. Este entorno permitió la evaluación de las capacidades del sistema para realizar seguimiento de trayectorias y adaptarse a cambios en las condiciones del entorno..

Este proyecto se limita a una infraestructura a escala, por lo que los resultados obtenidos sirven como base experimental para futuros desarrollos y optimizaciones en entornos a mayor escala. Además, el trabajo se centró en probar la viabilidad del sistema de control y su adaptabilidad a los retos urbanos, dejando abierta la posibilidad de integrar mejoras en los algoritmos y explorar otras aplicaciones en escenarios reales.

CAPÍTULO 6

Marco teórico

Es importante comprender los conceptos teóricos de la autonomía vehicular, resaltando la importancia de los modelos cinemáticos y los métodos de control aplicados a robots móviles con ruedas. Dentro de estos conceptos, es importante analizar el comportamiento de los robots móviles diferenciales bajo modelos representativos, abordando tanto la cinemática directa como la cinemática inversa, que son necesarias para comprender los movimientos y trayectorias de los vehículos. Por otro lado, es importante conocer las características del robot que se utilizará, al igual que las especificaciones técnicas del sistema de captura de movimiento disponible en el Robotat. Por último, es necesario abordar los distintos métodos y algoritmos que se utilizan para el control de vehículos autónomos y su implementación.

6.1. Vehículos autónomos

Un vehículo autónomo se caracteriza por ser capaz de conducirse entre un punto de inicio y un punto final, con la menor intervención humana posible. Actualmente, no existen vehículos completamente autónomos, aunque se han desarrollado sistemas avanzados de asistencia al conductor que minimizan la necesidad de intervención por parte del usuario durante el trayecto. La seguridad de los ocupantes es uno de los aspectos más importantes en los vehículos autónomos. Sin embargo, a pesar de la confiabilidad de muchos de estos sistemas, su desempeño óptimo puede verse afectado por diversos factores, como la carencia de una infraestructura vial adecuada que facilite su navegación [6].

La manera en la que estos vehículos funcionan es por medio de una serie de pasos que permite conducirse segura y eficientemente de un lugar a otro. En primer lugar, los vehículos autónomos deben de tener una percepción completa de su entorno para que los algoritmos de toma de decisiones y de conducción funcionen adecuadamente. Mientras más información disponible se tenga, mejor puede ser el resultado de los algoritmos. Datos útiles e importantes para el correcto funcionamiento de los vehículos incluyen la delimitación de los carriles, señales de tránsito, ubicación y distancia de vehículos cercanos, ubicación de

obstáculos y peatones, entre otros. Esta información sobre el entorno es recabada por medio de distintos sensores instalados en el chasis.

Con la información de su entorno, los vehículos autónomos son capaces de definir la trayectoria de conducción óptima y segura entre el punto de partida y el destino deseado. Esto se calcula en conjunto con información del lugar por medio de mapas, vías y regulaciones. Por otro lado, los vehículos autónomos deben de ser capaces de reaccionar ante perturbaciones y obstáculos externos que afectan la trayectoria calculada, manteniendo siempre como prioridad la seguridad de los ocupantes y de los objetos y/o personas del entorno.

El control de los vehículos autónomos se puede separar en dos categorías importantes [7]. La primera categoría es el control longitudinal, el cual incluye el control del movimiento rectilíneo del vehículo, es decir la aceleración y desaceleración de este. La segunda categoría es el control lateral, el cual incluye el control de la dirección del vehículo por medio del giro de su sistema direccional. Estos dos controles en conjunto permiten conducir el vehículo de la misma manera en la que lo haría un conductor convencional, tomando en cuenta el plano de movimiento disponible para el cual están diseñados los vehículos.

Actualmente existen 5 niveles de autonomía que definen la capacidad de conducción de los vehículos autónomos según la Sociedad de Ingenieros de la Automoción [8].Las legislaciones y regulaciones para su uso dependiendo del nivel de autonomía varían entre regiones y países. Estas categorías son las siguientes:

- **Nivel 0:** sin autonomía en la conducción. Todas las intervenciones son por parte del conductor.
- **Nivel 1:** asistencia en la conducción por medio de sistemas para mantener el carril, control de la velocidad o frenado de emergencia.
- **Nivel 2:** autonomía parcial, en la cual el vehículo tiene control longitudinal y lateral de la conducción, pero no posee detección de objetos ni respuesta a los mismos.
- **Nivel 3:** autonomía condicionada, en la cual el vehículo posee el control longitudinal, lateral, detección de objetos y respuesta a su entorno. En esta categoría el vehículo es capaz de realizar cambios de carril sin intervención del conductor, pero este debe de estar atento para intervenir en casos de emergencia.
- **Nivel 4:** autonomía elevada, en la cual el vehículo tiene el control de la conducción y sistemas de respaldo para actuar en consecuencia al entorno, pero el conductor todavía puede intervenir y retomar el control de así desearlo.
- **Nivel 5:** autonomía completa, en la cual los vehículos ya no poseen de pedales de acelerador y freno ni de timón, por lo que el usuario no puede intervenir en ningún momento y la conducción es exclusivamente controlada por el vehículo.

6.2. Modelo cinemático de un robot móvil diferencial

Los robots móviles con ruedas, a pesar de ser robots de base flotante, pueden ser descritos y controlados solamente con su cinemática. Esto es una ventaja ya que no es necesario aplicar

técnicas de dinámica de cuerpos rígidos. Una característica de estos robots es que con tres ruedas se garantiza estabilidad, pero con más de tres ruedas requieren de un sistema de suspensión. La ventaja de las ruedas en esta clase de robots es que dependiendo del tipo de rueda se pueden tener distintos grados de libertad. Entre estas clases de ruedas pueden encontrarse:

- Rueda estándar, que permite dos grados de libertad
- Rueda giratoria, que permite 3 grados de libertad
- Rueda sueca, que permite 3 grados de libertad

El movimiento de un robot móvil con ruedas está definido por las restricciones de rodamiento y de no deslizamiento que existen entre las ruedas y el suelo.

Los robots móviles con ruedas se clasifican como sistemas no holonómicos. Un sistema no holonómico se refiere a un tipo de sistema en el cual las restricciones de movimiento no pueden ser integradas como las restricciones de un sistema holonómico. En otras palabras, en un sistema no holonómico, el movimiento no puede ser descrito únicamente por su posición y orientación actuales, sino que también depende de las restricciones cinemáticas y dinámicas. En este caso, las restricciones están dadas por las ruedas. Esto quiere decir que una misma configuración puede producir múltiples poses finales distintas.

Un aspecto a considerar en el modelo cinemático de los robots móviles con ruedas es que la velocidad de las mismas es la combinación de todas las velocidades que experimenta el robot durante su movimiento. Debido a que cada rueda induce tanto una restricción de rodamiento y una restricción de no deslizamiento, es necesario apilar las restricciones de cada rueda en el modelo cinemático.

El modelo cinemático de un robot diferencial con dos ruedas como el robot con el que se trabajará, se puede deducir de la siguiente manera bajo las dimensiones mostradas en la Figura2. Este modelo sigue el desarrollo propuesto en [9].

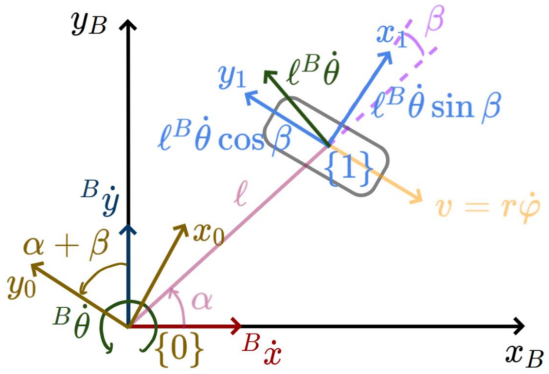

Figura 2: Diagrama de modelo cinemático de robot diferencial con dos ruedas[10]

El modelo se basa en la configuración del robot móvil con ruedas mostrado en (1).

$$^{I}\xi = {}^{I}Rot_{B}(\theta)A^{+}Bv = f(^{I}\xi, v), \tag{1}$$

En donde las matrices A y B son las matrices mostradas en (2) y (3).

$$A = \begin{bmatrix} J_1 \\ C_1 \end{bmatrix}, \tag{2}$$

$$B = \begin{bmatrix} J_2 \\ C_2 \end{bmatrix}. \tag{3}$$

En el caso de este modelo, ya que se tienen dos ruedas, las matrices de restricción se muestran en las ecuaciones 4, (5), (5) y (7).

$$J_1 = \begin{bmatrix} J_{11} \\ J_{12} \end{bmatrix}, \tag{4}$$

$$C_1 = \begin{bmatrix} C_{11} \\ C_{12} \end{bmatrix}, \tag{5}$$

$$J_2 = I_{2\times 2}, \tag{6}$$

$$C_2 = 0_{2\times 2}. \tag{7}$$

La información geométrica de instalación de las ruedas se define con las matrices (8) y (9).

$$J_{1i} = \begin{bmatrix} \sin(\alpha+\beta) & \cos(\alpha+\beta) & l\cos(\beta) \end{bmatrix}, \tag{8}$$

$$C_{1i} = \begin{bmatrix} \cos(\alpha+\beta) & \sin(\alpha+\beta) & l\sin(\beta) \end{bmatrix}. \tag{9}$$

Se definen las restricciones de no deslizamiento y de rodamiento de las ruedas como el vector de velocidades con respecto al marco inercial de la rueda (10).

$$^{1}\mathbf{v} = \begin{bmatrix} {}^{1}v_x \\ {}^{1}v_y \end{bmatrix} = \begin{bmatrix} 0 \\ -r\dot{\varphi} \end{bmatrix}. \tag{10}$$

Al tomar en cuenta que la velocidad de la rueda es la combinación de todas las velocidades que experimenta el robot, se obtiene (11).

$$^{1}\mathbf{R}_{B}\,{}^{B}\mathbf{v} + l^{B}\dot{\theta}\begin{bmatrix} \sin\beta \\ \cos\beta \end{bmatrix} = \begin{bmatrix} 0 \\ -r\dot{\varphi} \end{bmatrix}. \tag{11}$$

Tomando en cuenta la información geométrica de los puntos de instalación de las ruedas y el marco inercial de la base del robot, se obtiene (12).

$$\begin{bmatrix} \cos(\alpha+\beta) & \sin(\alpha+\beta) \\ -\sin(\alpha+\beta) & \cos(\alpha+\beta) \end{bmatrix} \begin{bmatrix} {}^{B}\dot{x} \\ {}^{B}\dot{y} \end{bmatrix} + \begin{bmatrix} l^{B}\dot{\theta}\sin(\beta) \\ l^{B}\dot{\theta}\cos(\beta) \end{bmatrix} = \begin{bmatrix} 0 \\ -r\dot{\varphi} \end{bmatrix}. \tag{12}$$

Aplicando las operaciones para ambas ruedas, el modelo del robot diferencial de dos ruedas es el mostrado en (13).

$$f({}^{I}\xi, v) = \begin{pmatrix} \frac{r\cos(\theta)(\phi_L+\phi_R)}{2} \\ \\ \frac{r\sin(\theta)(\phi_L+\phi_R)}{2} \\ \\ -\frac{r(\phi_L-\phi_R)}{2l} \end{pmatrix}. \tag{13}$$

6.3. Control de robots móviles con ruedas

La manera más común y eficiente de controlar robots móviles con ruedas se basa en una arquitectura jerárquica de control [11]. Esta arquitectura se compone de tres niveles que permiten el control por medio de pasos escalados, desde un alto nivel hasta un bajo nivel.

El primer nivel de la arquitectura se basa en un modelo de orden reducido de un sistema dinámico simple que sea representativo para el comportamiento de un sistema completo y complejo. Este nivel de control se basa principalmente en técnicas de control moderno e incluso inteligencia artificial.

El segundo nivel está compuesto por un sistema completo con actuadores ideales. Este es normalmente un sistema dinámico que modela al mecanismo del sistema a controlar. Por último, el tercer nivel está compuesto por los actuadores del sistema que permiten el movimiento del mecanismo modelado en el nivel anterior.

Esta arquitectura jerárquica permite que cada nivel genere las referencias necesarias por la capa de control del nivel subsecuente. Esta técnica de control permite control comportamientos complicados debido a la naturaleza de sistemas no holonómicos. Técnicas de control modernas convencionales como el control LTI no funcionan adecuadamente con sistemas no holonómicos, ya que al linealizar el modelo del sistema, este resulta no ser completamente controlable.

En el caso del robot diferencial de dos ruedas, la arquitectura jerárquica se compone de los siguientes tres niveles:

- Nivel 1: modelo uniciclo que representa el comportamiento no holonómico del robot con ruedas.
- Nivel 2: el modelo cinemático del robot móvil deducido en (13).

- Nivel 3: motores de las ruedas que son los actuadores del robot. Estos se controlan generalmente con controladores PID de velocidad. El lazo cerrado lo permiten los *encoders* de las ruedas.

6.4. Modelo uniciclo para un robot móvil con ruedas

El modelo uniciclo [12] es una simplificación utilizada para describir el movimiento de un robot con ruedas en una superficie plana. Se basa en suposiciones simplificadas que permiten una representación matemática más manejable del comportamiento del robot y su movimiento en el plano.

En este modelo, se considera que el robot es un cuerpo rígido que se desplaza en un plano bidimensional, representado por un par de coordenadas (x, y) que denotan su posición en el espacio, y una orientación angular θ que indica su dirección respecto a un marco de referencia global como se muestra en la Figura 3. Este modelo asume que las ruedas del robot se mueven sin deslizamiento y se pueden controlar independientemente.

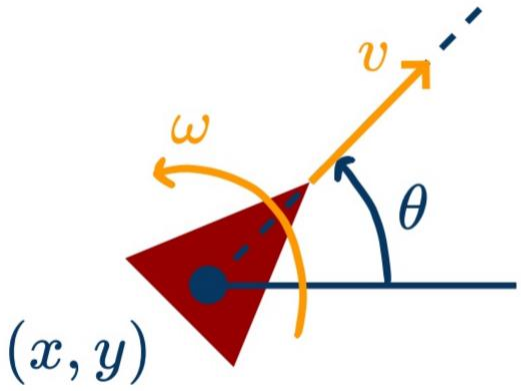

Figura 3: Diagrama de modelo uniciclo de robot móvil con ruedas [13]

El modelo uniciclo describe el comportamiento del robot con ecuaciones cinemáticas simples que relacionan las velocidades lineales y angulares de las ruedas con la velocidad lineal y angular del robot como un cuerpo rígido. Estas ecuaciones se muestran en (14), (15) y (16).

$$\dot{x} = v\cos(\theta), \tag{14}$$

$$\dot{y} = v\sin(\theta), \tag{15}$$

$$\dot{\theta} = \omega. \tag{16}$$

El modelo uniciclo simplifica la dinámica del movimiento del robot al ignorar aspectos como la inercia del robot, la fricción de las ruedas, la resistencia del aire y las irregularidades del terreno. A pesar de estas simplificaciones, el modelo uniciclo sigue siendo útil para el diseño de algoritmos de control de movimiento y planificación de trayectorias como los que se buscan validar.

6.5. Cinemática diferencial

La cinemática [14] en robots móviles con ruedas es el proceso matemático que describe la relación entre las variables de control del robot, como las velocidades de las ruedas, y las variables que describen la posición y orientación del robot en un espacio tridimensional. Esto permite describir la pose del robot como una función de la configuración.

Para un robot móvil diferencial, su cinemática directa se basa en el modelo cinemático diferencial. Este modelo describe cómo las velocidades y orientaciones de las ruedas se relacionan con la velocidad lineal y angular del robot como un cuerpo rígido. Esta relación se puede observar en (17).

$$\xi = f(\dot{\varphi}_1, \ldots, \dot{\varphi}_n, \beta_1, \ldots, \beta_m, \dot{\beta}_1, \ldots, \dot{\beta}_m). \tag{17}$$

6.6. Cinemática diferencial inversa

La cinemática diferencial inversa [14] en robots móviles con ruedas es el proceso matemático que describe la relación entre las variables que describen la posición y orientación del robot en un espacio tridimensional, con las variables de control del robot, como las velocidades de las ruedas. Este modelo es el inverso al de la cinemática directa, ya que permite describir la configuración del robot como una función de la pose deseada del robot.

En el caso de un robot móvil, la cinemática inversa implica calcular las velocidades y orientaciones de las ruedas para lograr una posición y orientación deseada del robot. Esta relación se puede observar en 18.

$$(\dot{\varphi}_1, \ldots, \beta_i, \ldots, \dot{\beta}_i) = f(\dot{x}, \dot{y}, \dot{\theta}). \tag{18}$$

6.7. Pololu 3Pi+

La empresa Pololu Robotics and Electronics [15] fabrica a los robots de dos ruedas Pololu 3Pi+, mostrados en la Figura 4. Estos robots se caracterizan por su bajo costo y gran versatilidad de programación para ejecutar distintas tareas. El Pololu 3Pi+ está basado en el microcontrolador ATmega32U4 compatible con Arduino, haciendo que su programación sea sencilla. La alimentación del Pololu 3Pi+ es por medio de 3 baterías AAA, las cuales pueden ser recargables ya que el robot posee un circuito de carga para las mismas.

Figura 4: Pololu 3Pi+ [15]

Para el control de velocidad de las ruedas del robot, este posee *encoders* en las ruedas, con los que se puede generar un lazo de control cerrado. Entre los demás sensores que están instalados en el Pololu, se encuentran sensores de línea, sensores de impacto y una unidad de medición inercial.

El tamaño del robot es otra característica destacable del mismo. Este posee un diámetro de 9.8 centímetros y una altura de 3.9 centímetros. Debido a su tamaño pequeño, el peso no supera los 83 gramos sin los accesorios, por lo que los motores de las ruedas tampoco se ven forzados para poder moverlo. Las ruedas tienen un diámetro de 3.2 centímetros y permiten alcanzar una velocidad lineal máxima de 1.5 metros sobre segundo. Los sensores y características se observan en las Figuras 5 y 6.

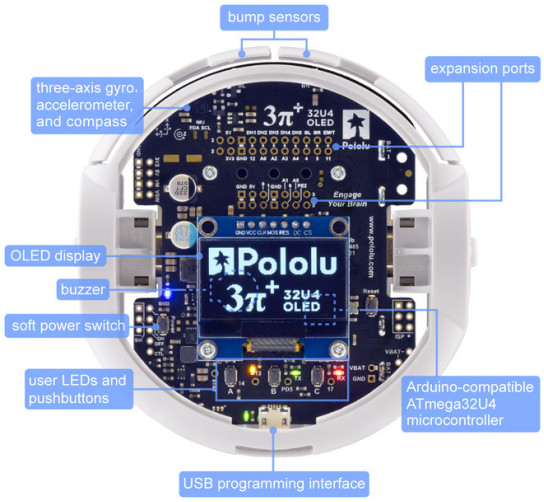

Figura 5: Pololu 3Pi+ vista superior [15]

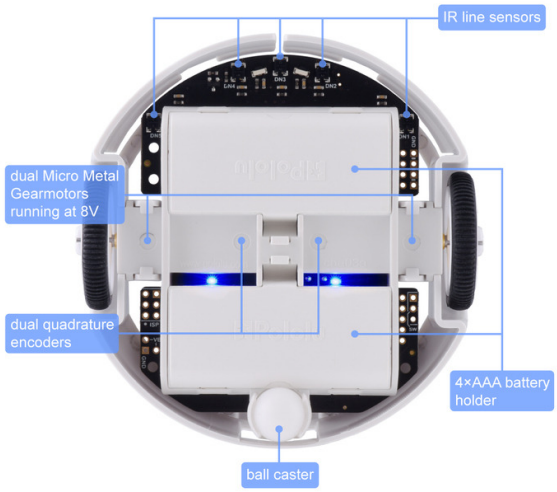

Figura 6: Pololu 3Pi+ vista inferior [15]

6.8. Sistemas de captura de movimiento (MoCap)

La captura de movimiento [16], identificada también por sus siglas MoCap, es un proceso que consiste en registrar digitalmente el movimiento de un cuerpo. Existen distintas tecnologías que permiten capturar digitalmente el movimiento. Entre estas se encuentran las siguientes:

- **Tecnología óptica pasiva:** con la cual se colocan marcadores reflectivos en los cuerpos u objetos. Estos marcadores reflejan la luz infrarroja emitida en el entorno de captura y unas cámaras registran el movimiento.

- **Tecnología óptica activa:** funciona muy similar a la tecnología óptica pasiva, pero en este caso los marcadores colocados en los cuerpos u objetos emiten la luz infrarroja en lugar de reflejarla. En este caso, los marcadores requieren una fuente de alimentación externa, por lo que no es una tecnología tan versátil como la óptica pasiva.
- **Tecnología libre de marcadores:** la cual consiste en un sistema de cámaras especializadas en detectar la profundidad del entorno y los objetos situados en el mismo. En conjunto con un software, el sistema es capaz de detectar el movimiento y distribución de los objetos y digitalizarlo.
- **Tecnología inercial:** la cual no requiere de cámaras que capturen el movimiento sino que unidades de medición inercial IMU son las encargadas de registrar el movimiento. Estas unidades están compuestas principalmente por sensores giroscópicos, magnetrómetros y acelerómetros que permiten medir la velocidad de rotación de distintos puntos al rededor de los ejes de referencia.

En la Universidad del Valle de Guatemala se encuentra instalado un sistema de captura de movimiento óptico pasico, compuesto por 6 cámaras Prime x41 fabricadas por la empresa OptiTrack [17], como la de la Figura 7. Estas cámaras tienen las siguientes especificaciones:

- Resolución de 4.1 Mega Pixeles.
- Exactitud de 0.1 mm en tres dimensiones.
- Exactitud de 0.5 grados de rotación en tres ejes.
- Tasa de refresco nominal de 180 cuadros por segundo.
- Tasa de refresco máxima de 250 cuadros por segundo.

Figura 7: Cámara PrimeX41 [17]

El rango de cobertura por cada cámara es de hasta 100 pies con los marcadores pasivos y hasta 150 pies con marcadores activos. Esto se traduce en una cobertura de hasta 290,000 pies cúbicos por cámara. De igual forma, cada cámara tiene la capacidad de procesamiento de imagen que le permite detectar el centro de los marcadores antes de enviar la información al software. Esto reduce significativamente la latencia y tiempo de digitalización del movimiento.

6.9. Algoritmos de control para vehículos autónomos

Los algoritmos de control de vehículos autónomos se basan en el uso simultáneo de distintos algoritmos más específicos. Entre estos algoritmos se encuentran algoritmos de percepción para detectar el entorno de los vehículos, algoritmos de mapeo y localización simultánea que permiten que el vehículo construya un mapa del entorno mientras se localiza dentro de ese mapa y algoritmos de planificación de trayectorias para determinar la mejor ruta para conducirse.

En cuanto a los algoritmos de percepción, usualmente se utilizan redes neuronales convolucionales. Estas redes son efectivas en tareas de visión por computadora, como la detección de objetos en imágenes. Funcionan mediante la convolución de filtros sobre la imagen de entrada para extraer características relevantes, seguido de capas de agrupación y clasificación para detectar objetos específicos.

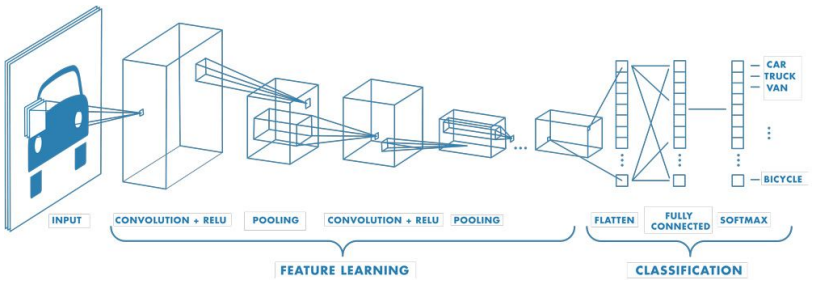

Figura 8: Arquitectura de redes neuronales convolucionales [18]

Por otro lado, también se suelen utilizar algoritmos de histogramas de gradientes orientados HOG en conjunto con algoritmos de máquina de vectores de soporte SVM [19]. los algoritmos HOG calculan la dirección del gradiente de la intensidad de píxeles en regiones de la imagen y luego utilizan estas direcciones para construir un histograma de gradientes. Los algoritmos SVM se entrenan utilizando estos histogramas para clasificar diferentes partes de la imagen como pertenecientes a un objeto específico o no.

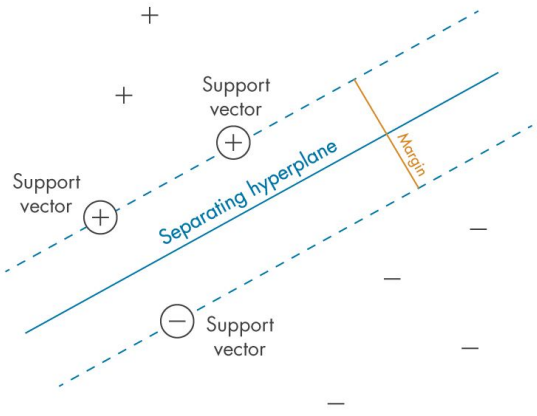

Figura 9: Algoritmos de maquina de vectores de soporte [19]

Para los algoritmos de localización, usualmente se emplean los filtros de Kalman extendidos. Estos son una extensión del filtro de Kalman clásico que puede manejar sistemas no lineales. Este se utiliza para estimar la posición del vehículo y la ubicación de los objetos en un entorno desconocido utilizando datos de sensores y un modelo del movimiento del vehículo.

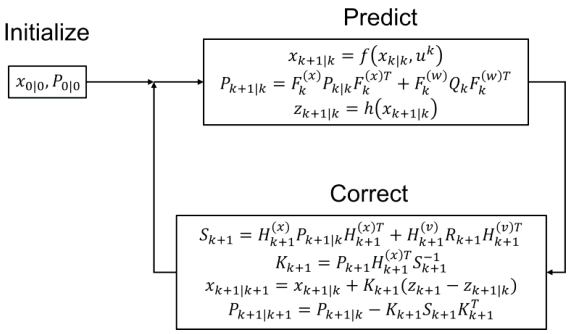

Figura 10: Filtro de Kalman extendido [20]

Por último, entre los algoritmos de planificación de trayectorias, el más utilizado es el algoritmo de Dijkstra [21]. Este es un algoritmo de búsqueda de caminos que encuentra la ruta más corta entre dos puntos en un grafo ponderado.

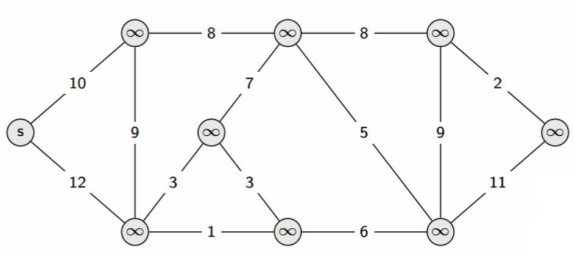

Figura 11: Algoritmo de Dijkstra [21]

6.10. Filtro de Kalman extendido

El filtro de Kalman extendido (EKF) es una extensión del filtro de Kalman (KF) que se creó para tratar sistemas no lineales. El filtro de Kalman es ideal para sistemas lineales y gaussianos, pero el filtro de Kalman extendido se adapta a la no linealidad de otros sistemas mediante la linealización local del sistema alrededor de la estimación actual [22]. Este método permite la aplicación del algoritmo de Kalman en situaciones donde el modelo del sistema o las observaciones no pueden ser descritos de manera lineal.

Rudolf E. Kalman desarrolló el filtro de Kalman en 1960 y este se convirtió en una herramienta importante para estimar estados en sistemas dinámicos. El filtro de Kalman extendido se creó debido a la necesidad de estimación en sistemas no lineales, que son comunes en aplicaciones como la robótica, la navegación y el control [23]. La principal característica del filtro de Kalman extendido es su capacidad de manejar la no linealidad

mediante la aproximación local de las funciones no lineales a través de series de Taylor, utilizando matrices Jacobianas para la predicción y actualización.

Desde el punto de vista de la optimización, el filtro de Kalman extendido busca reducir la incertidumbre en la estimación del estado del sistema, minimizando el error cuadrático medio entre el estado verdadero del sistema y la estimación del estado que se obtiene a través del filtro.

La minimización del error de estimación se busca calculando la ganancia de Kalman para que minimice la traza de la matriz de covarianza *a posteriori*. Esto implica que el EKF ajusta la ganancia de Kalman para equilibrar adecuadamente la confianza entre las predicciones del modelo y las observaciones reales, con el objetivo de obtener la estimación más precisa posible del estado del sistema.

El filtro de Kalman extendido se basa en un sistema no lineal descrito por el siguiente modelo discreto [24], dado por (19) y (20).

$$x_{k+1} = \mathcal{F}(x_k, u_k, w_k), \tag{19}$$

$$y_k = \mathcal{H}(x_k) + v_k. \tag{20}$$

En donde w_k y v_k son los ruidos de proceso y de observación del sistema, los cuales no están correlacionados entre sí.

El FKE se separa en dos etapas, las cuales son predicción y corrección. En ambas etapas se hace una estimación previo a la lectura de los sensores y una estimación luego de la lectura de los sensores. Estas estimaciones se conocen como estimaciones *a priori* y *a posteriori*, representadas por $\hat{x}_{k|k-1}$ y $\hat{x}_{k|k}$ respectivamente.

De igual forma, en ambas etapas se hace una estimación *a priori* y *a posteriori* de la covarianza, representadas por $\hat{P}_{k|k-1}$ y $\hat{P}_{k|k}$ respectivamente.

La etapa de predicción comienza con la estimación *a priori* del vector de estados del sistema, dado por (21)

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Bu_k. \tag{21}$$

En cuanto a la estimación *a priori* de la covarianza, esta está dada por (22).

$$P_{k|k-1} = AP_{k-1|k-1}A^T + FQ_wF^T, \tag{22}$$

en donde las matrices A, B y F están dadas por los jacobianos $\frac{\partial \mathcal{F}}{\partial x}$, $\frac{\partial \mathcal{F}}{\partial u}$ y v respectivamente.

La matriz Q_w es la matriz de los valores de varianzas de las mediciones del sensor o sensores empleados para la predicción.

La etapa de corrección comienza calculando al ganancia de Kalman L dada por (23).

$$L_k = P_{k|k-1} C^T S_k^{-1}, \tag{23}$$

utilizando el valor de la estimación *a priori* de la covarianza en $S_{k^{-1}}$ dado por (24) y se conoce como la matriz de covarianza de la innovación.

$$S_k = Q_v + C P_{k|k-1} C^T, \tag{24}$$

en donde la matriz C está dada por el jacobiano (25).

$$\frac{\partial \mathcal{H}}{\partial x}, \tag{25}$$

y la matriz Q_v es la matriz de los valores de varianzas de las mediciones del sensor o sensores empleados para la corrección.

Luego se calcula la estimacion *a posteriori* del vector de estados del sistema dado por (26).

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + L_k z_k, \tag{26}$$

en donde z_k está dado por (27).

$$z_k = y_k - C\hat{x}_{k|k-1}. \tag{27}$$

Por último, se calcula la estimación *a posteriori* de la covarianza, dada por (28).

$$P_{k|k} = P_{k|k-1} - L_k C P_{k|k-1}. \tag{28}$$

En resumen, el funcionamiento del filtro de Kalman extendido se puede mostrar bajo el siguiente diagrama de bloques de la Figura 12.

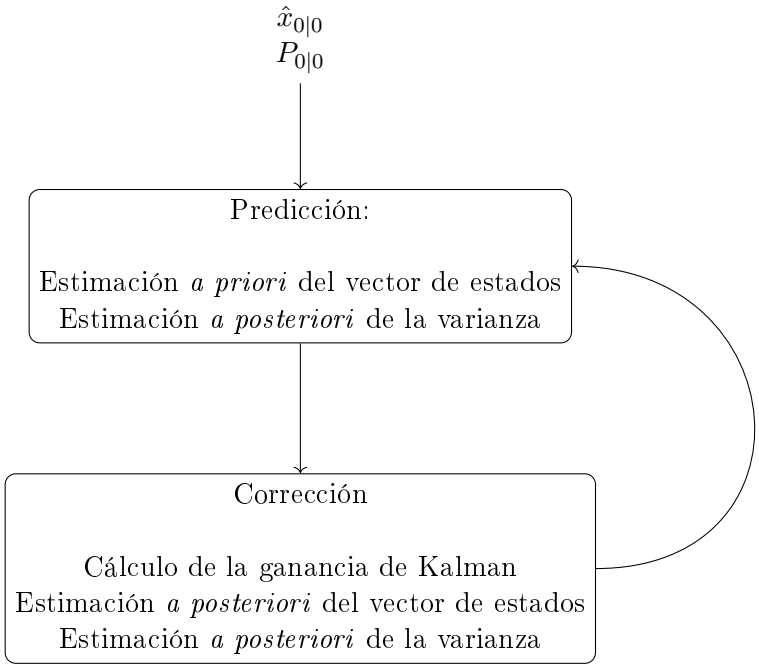

Figura 12: Diagrama de flujo del filtro de Kalman

6.11. Controlador PID con acercamiento exponencial

El PID [25] de acercamiento exponencial se modela bajo la ecuación 29.

$$v = -k(e_p)e_p = -\frac{v_0\left(1 - e^{-\alpha e_p^2}\right)}{e_p}e_p, \tag{29}$$

en donde el error de posición e_p está dado por la norma de la diferencia entre la posición deseada en x y la posición actual en x, y la posición deseada en y y la posición actual en y. v_0 representa la velocidad lineal máxima deseada y α es el coeficiente de ajuste exponencial del controlador, el cual permite determinar la suavidad y velocidad de la convergencia del error lo más cercano a cero posible. El valor de dicho coeficiente es de $\alpha = 1000$.

En cuanto al PID de orientación, este se modela bajo la ecuación 30.

$$\omega = PID(e_o) = k_{P_o}e_o + k_{I_o}\int_0^t e_o(\tau)d\tau + k_{D_o}\dot{e}_o, \tag{30}$$

en donde el error de orientación e_o está dado por la diferencia angular entre la orientación deseada del robot y la orientación actual del mismo. Para poder evitar errores en los signos de los ángulos, es importante acotar los valores de dicho error como se muestra en la ecuación 31.

$$e_o = \text{atan2}\left(\frac{\sin e_0}{\cos e_0}\right). \tag{31}$$

6.12. Odometría

La odometría [26] es una técnica fundamental en robótica móvil que permite estimar la posición y orientación de un robot a lo largo del tiempo, basándose en mediciones internas de movimiento. Esta estimación se logra mediante la integración de datos proporcionados por sensores como *encoders* en las ruedas, giroscopios y acelerómetros, que registran desplazamientos y rotaciones del robot.

En robots con tracción diferencial, la odometría se basa en el conteo de pulsos de los *encoders* de cada rueda para calcular la distancia recorrida y el cambio de orientación. Este método es ampliamente utilizado debido a su simplicidad y bajo costo, aunque es susceptible a errores acumulativos por deslizamientos o irregularidades en el terreno.

Para mejorar la precisión de la odometría, se han desarrollado métodos de calibración en [27] que corrigen errores sistemáticos. Por ejemplo, Navarro et al [27] proponen una metodología sencilla que no requiere instrumentos especiales para calibrar los parámetros cinemáticos de un robot diferencial, logrando reducir errores en la estimación de la posición.

Además, la odometría se complementa con otros sistemas de percepción, como la odometría visual, que utiliza imágenes capturadas por cámaras para estimar el movimiento del robot. Esta técnica es especialmente útil en entornos donde los sensores tradicionales presentan limitaciones, proporcionando información adicional que mejora la estimación de la posición y orientación del robot.

6.13. Conversión de cuaterniones de rotación a ángulos de Euler

Para convertir la orientación representada por un cuaternión en ángulos de Euler, se transformó el cuaternión en una matriz de rotación y luego se extrajeron los ángulos de rotación correspondientes en los ejes x, y, y z, basado en el proceso que se presenta en [28].

El cuaternión utilizado fue representado como:

$$q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}, \tag{32}$$

donde q_0 fue el componente escalar y (q_1, q_2, q_3) los componentes vectoriales. Se aseguró que el cuaternión q fuera unitario para representar una rotación válida, cumpliendo así que su norma fuera 1.

$$\|q\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1. \tag{33}$$

Se procedió a convertir el cuaternión unitario en una matriz de rotación R de 3×3, capaz de operar sobre un vector tridimensional para realizar la rotación correspondiente. La matriz de rotación asociada al cuaternión q se definió como:

$$R(Q) = \begin{pmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & 2(q_0^2 + q_3^2) - 1 \end{pmatrix} \tag{34}$$

Para obtener los ángulos de Euler en la secuencia de rotación xyz, se extrajeron los ángulos de rotación alrededor de los ejes x (roll), y (pitch), y z (yaw) a partir de los elementos de la matriz de rotación.

El ángulo de rotación alrededor del eje x, conocido como roll (ϕ), fue calculado como:

$$\phi = -\left(\operatorname{atan2}\left(q_3 - q_1, q_0 - q_2\right) - \operatorname{atan2}\left(q_1 + q_3, q_2 + q_0\right)\right) \tag{35}$$

El ángulo de rotación alrededor del eje y, conocido como pitch (θ), fue calculado como:

$$\theta = \arccos\left((q_0 - q_2)^2 + (q_3 - q_1)^2 - 1\right) - \frac{\pi}{2} \tag{36}$$

Finalmente, el ángulo de rotación alrededor del eje z, conocido como yaw (ψ), se obtuvo a partir de:

$$\psi = \operatorname{atan2}\left(q_3 - q_1, q_0 - q_2\right) + \operatorname{atan2}\left(q_1 + q_3, q_2 + q_0\right) \tag{37}$$

Durante el proceso, se consideraron posibles singularidades, especialmente cuando el ángulo de pitch (θ) se aproximaba a $\pm\frac{\pi}{2}$. Esta condición podría llevar a un bloqueo de gimbal, donde se perdería un grado de libertad en la rotación.

CAPÍTULO 7

Diseño de una infraestructura vial representativa de la Ciudad de Guatemala

Para poder llevar a cabo pruebas del sistema de control en entornos urbanos, fue necesario diseñar una infraestructura vial a escala que representara las características urbanas de la Ciudad de Guatemala. Dentro del contexto de este trabajo, la infraestructura debe de simular las condiciones urbanas que reflejan los desafíos que caracterizan a las calles guatemaltecas, con el fin de verificar el desempeño de los algoritmos de control.

7.1. Elección de calles y zonas representativas

Para el diseño de la infraestructura, se eligieron cuatro calles o zonas importantes en el casco urbano de la capital, que a la vez representaran desafíos para los vehículos autónomos. Las cuatro zonas elegidas para este trabajo son las siguientes.

7.1.1. Redondel de la Plaza del Obelisco

El Obelisco es uno de los puntos más transitados de la Ciudad de Guatemala. Este redondel rodea al monumento a los próceres de la Independencia, construido en 1935 que, no sólo es un símbolo histórico, sino también un nodo principal en la red vial de la ciudad. Este redondel es el punto que delimita la separación de las zonas 9, 10, 13 y 14 de la capital, zonas en las cuales se encuentran varios comercios y distritos empresariales. Este redondel conecta la Avenida las Américas, la Avenida Reforma, el Bulevar Liberación y el Bulevar Los Próceres, lo que genera un flujo vehicular constante y denso. Este flujo vehicular aumenta significativamente en las horas pico de la ciudad. Estos horarios cubren desde las 5:45 hasta las 8:30 hs durante las mañanas y desde las 16:30 hasta las 20:30 hs durante las tardes. Durante estos horarios, se estima que se requieren de 1 a 1.5 horas para recorrer trayectos

de 10 a 15 kilómetros, con una velocidad promedio de 9 kilómetros por hora [29].

A parte del alto flujo vehicular por el redondel del Obelisco, este presenta otros desafíos para los vehículos autónomos. Este no solo funciona como un redondel convencional, sino que tiene semáforos que operan dentro del mismo haciendo que las reglas para determinar qué vehículo lleva la vía sean más complejas. Por otro lado, al rodear una plaza con un monumento histórico, es atravesado por diversos pasos de cebra para peatones. Estos pasos de cebra están distribuidos a lo largo del redondel y el flujo peatonal provoca que los conductores se detengan constantemente para permitir el cruce de las personas. El flujo peatonal aumenta significativamente cuando se realizan eventos en la plaza, como conciertos, exposiciones o incluso manifestaciones. Todos estos factores añaden una capa adicional de complejidad a la toma de decisiones en tiempo real de los algoritmos de control de los vehículos autónomos, ya que combina distintos casos que por sí solos ya son complejos.

7.1.2. Carretera a Muxbal

Muxbal es una zona residencial y comercial ubicada al este de la Ciudad de Guatemala. Esta zona se caracteriza por ser montañosa y estar rodeada de barrancos. Estas montañas y barrancos generan una topografía accidentada y compleja, por lo que la carretera tiene curvas cerradas con peraltes altos. Además, la elevación del terreno cambia conforme se avanza en la carretera, ya que el cambio de elevación total sobre el nivel del mar desde su punto de inicio hasta su punto final es de aproximadamente 100 metros. Este sector se ha convertido en una alternativa vial para transitar desde los municipios de Santa Catarina Pinula y Fraijanes hacia la ciudad y viceversa, evitando transitar por la Carretera Interamericana. Se estima que hasta 80 mil vehículos transitan por esta carretera diariamente, requiriendo de un tiempo aproximado de 45 minutos para atravesar la zona [30]. Por otro lado, al ser una zona montañosa, el clima es otro factor que aumenta la complejidad de manejar por esta carretera. La lluvia en la zona humedece el asfalto de la carretera, reduciendo el coeficiente de fricción de las llantas de los carros en las curvas pronunciadas. La neblina y la poca visibilidad también son comunes, lo cual representa un desafío para los vehículos autónomos que dependen de sensores basados en visión por computadora para la toma de decisiones. Estos factores climáticos provocan que los conductores tengan que reducir su velocidad al transitar en épocas lluviosas. Esto aumenta el tráfico que afecta a la carretera, ya que solamente tiene un carril disponible en cada sentido por lo que no se puede rebasar en todo el trayecto.

7.1.3. Avenida Reforma

La Avenida Reforma [31] es una de las arterias más importantes y emblemáticas de la Ciudad de Guatemala. Esta vía, que se extiende desde el centro de la ciudad hacia el sur, es conocida por su ancho de múltiples carriles. Esta calle divide a la zona 9 de la zona 10 de la ciudad y recorre desde la zona 4 hasta la zona 13. Esta avenida tiene 3 carriles de circulación en cada sentido, divididos por un arriate central ancho. En este arriate central se encuentra una ciclo vía, al igual que un caminamiento para peatones y jardines con monumentos y estatuas. La Avenida Reforma es una de las calles con mayor cantidad de carriles de todo el país. La avenida también tiene un carril auxiliar en cada sentido, con un ancho de dos

carriles. Esta cantidad de carriles, tomando en cuenta todas las entradas y salidas disponibles para los carriles auxiliares, complican la movilidad autónoma. Los cruces peatonales y de ciclistas también son un factor a tomar en cuenta en los algoritmos de control de los vehículos autónomos.

Ya que esta avenida se encuentra en las zonas empresariales de la ciudad, tiene varios elementos de orden vehicular como semáforos y pasos de cebra. Los semáforos regulan el flujo de vehículos en las calles que atraviesan a la avenida. Las calles principales que cruzan perpendicularmente a la avenida Reforma son la 13 Calle, la 12 Calle, la 10 Calle, la 6 Calle, la 5 Calle y la 2 Calle. Estos cruces perpendiculares a la avenida permiten la conectividad entre la zona 9 y la zona 10, por lo que el flujo vehicular que necesita atravesar la avenida es alto. Esto hace que los semáforos tengan que detener el tránsito sobre la avenida por tiempos largos y en periodos recurrentes.

Otra de las razones por las cuales la Avenida Reforma maneja un flujo vehicular alto es porque a sus lados se encuentran varios edificios de oficinas, bancos, embajadas, edificios gubernamentales y del ejército, comercios, restaurantes, hoteles, ministerios e incluso distritos médicos. Todo esto genera un flujo vehicular alto a lo largo de todo el día en esta avenida. En las horas pico, el tráfico llega a estar detenido por momentos, lo que complica no solo la movilidad sobre la avenida sino también en sus alrededores.

7.1.4. Zona 1

La Zona 1 de la Ciudad de Guatemala o también conocida como el Centro Histórico, es una de las áreas más antiguas y culturalmente significativas de la ciudad. Esta zona es famosa por sus calles estrechas y en forma de cuadrícula y su mezcla densa de tráfico vehicular y peatonal. Al ser una zona administrativa, comercial y turística, la Zona 1 experimenta un movimiento constante de personas que se movilizan en vehículos propios, transporte público, transporte alternativo como bicicletas o monopatines y un alto flujo peatonal. Todos estos elementos son un reto para los vehículos autónomos, ya que deben de ser robustos para poder identificar cada elemento del entorno y distinguirlo para poder tomar decisiones.

Una de las principales características de la Zona 1 es lo estrecho de sus calles, muchas de las cuales fueron diseñadas en una época en que la que los vehículos no existían y la movilidad urbana era considerablemente menor. Esta zona fue desarrollada entre los años 1776 y 1982, previo a la expansión de la ciudad hacia sus alrededores. La Municipalidad de Guatemala reporta que esta zona es una de las más transitadas por peatones en la ciudad, con un flujo diario estimado en más de 100,000 personas [29]. Además, la presencia de estructuras históricas y la necesidad de preservar el patrimonio arquitectónico hacen que la implementación de soluciones de infraestructura modernas sean casi imposibles. Esto provoca que los sistemas de control de vehículos autónomos deban adaptarse a entornos urbanos como este, en donde el desorden vehicular debe de tomarse en cuenta.

7.2. Diseño de la infraestructura representativa

Luego de haber elegido las calles y zonas a plasmar en la infraestructura se comenzó con el diseño de la misma. La infraestructura vial se diseñó con una escala de 1:18.75 [2], tomando en cuenta las dimensiones de un vehículo tipo sedán y las características del Pololu 3Pi+, que es la plataforma base utilizada. De igual forma, se tomó en cuenta el espacio de la plataforma del Robotat de 3.80 metros de ancho por 4.80 metros de alto.

El diseño comenzó con trazos a mano alzada, intentando representar las distintas calles y zonas elegidas, a la vez que buscaba que la interconexión de las rutas disponibles hiciera sentido y que se pudiera transitar desde cualquier punto del mapa hacia otro, evitando calles sin salida. El esbozo final al que se llegó es el que se muestra en la Figura 13.

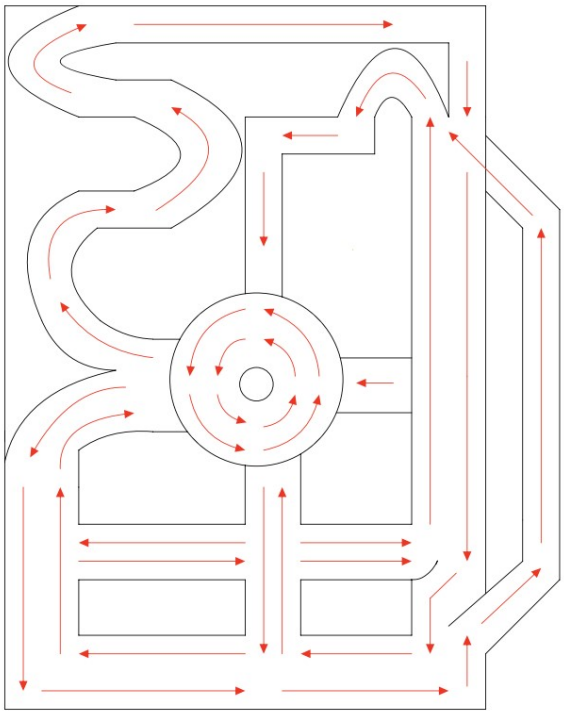

Figura 13: Vías designadas

Con el trazo a mano alzada se procedió a dimensionar la infraestructura en Autodesk Inventor. Para esto se hizo un bosquejo en dos dimensiones en el plano XZ. En este bosquejo se capturó la misma geometría propuesta en el trazo a mano alzada, dimensionándolo bajo la escala 1:18.75 propuesta en los antecedentes. Tomando en cuenta la delimitación total de la infraestructura para que ocupe todo el espacio disponible dentro de la plataforma del Robotat, se calcularon las dimensiones necesarias para los carriles. Con una altura máxima de 4800 milímetros y un ancho máximo de 3800 milímetros, cada carril tiene un ancho de 183 milímetros y el ancho de las líneas que delimitan los carriles es de 8 milímetros. El espacio que se dejó entre el límite de un carril y la carretera fue de 25 milímetros. Esto quiere decir que el ancho total de una calle de dos carriles debía de ser de 440 milímetros y el ancho de una calle de un carril debía de ser de 249 milímetros. En el bosquejo en Inventor

se colocaron dichas dimensiones en una calle de un carril y en una calle de dos carriles. Para las demás calles, se utilizó como referencia dichas dimensiones, por lo que todos los anchos de las calles del bosquejo dependen de estas únicas dimensiones. Esto quiere decir que si se desean cambiar las dimensiones de los carriles, únicamente se deben cambiar dos dimensiones principales. La dependencia de dichas dimensiones se puede observar en la Figura 14.

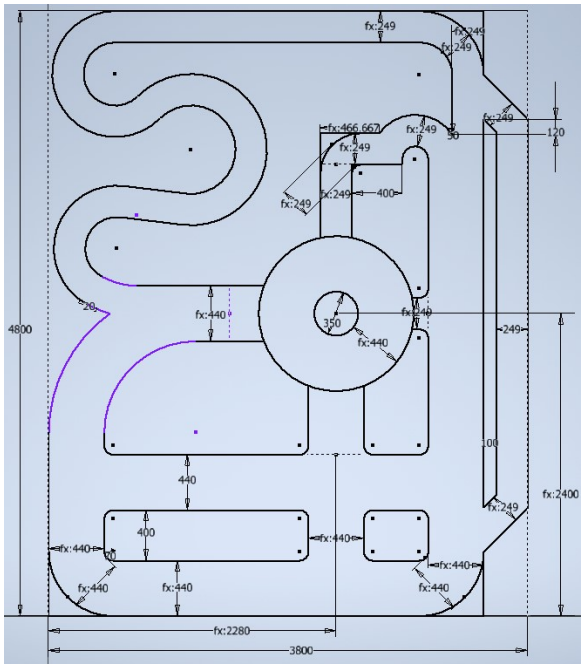

Figura 14: Bosquejo 2D dimensionado de la infraestructura en Autodesk Inventor

Al completar el bosquejo y asegurarse que cada carril cumpliera con las dimensiones correctas para garantizar la escala, se procedió a extruir el bosquejo para generar un sólido. Sobre el sólido se generó un segundo bosquejo 2D para dibujar las líneas que delimitan los carriles. La técnica utilizada para este proceso fue la de desfasar la misma geometría que delimitaba las calles, pero por 25 milímetros. Esto permitió no tener que re dibujar toda la geometría, sino aprovecharse del perímetro cerrado ya existente para mantener las mismas dimensiones y escalas en todas las secciones del bosquejo. Al completar este proceso se realizó una segunda extrusión sobre el bosquejo, para cortar 1 milímetro de profundidad en todas las líneas de delimitación. Este nuevo espacio cortado en el sólido se pintó de color amarillo para indicar la delimitación de los carriles y las calles. El resultado de este proceso se puede observar en la Figura 15.

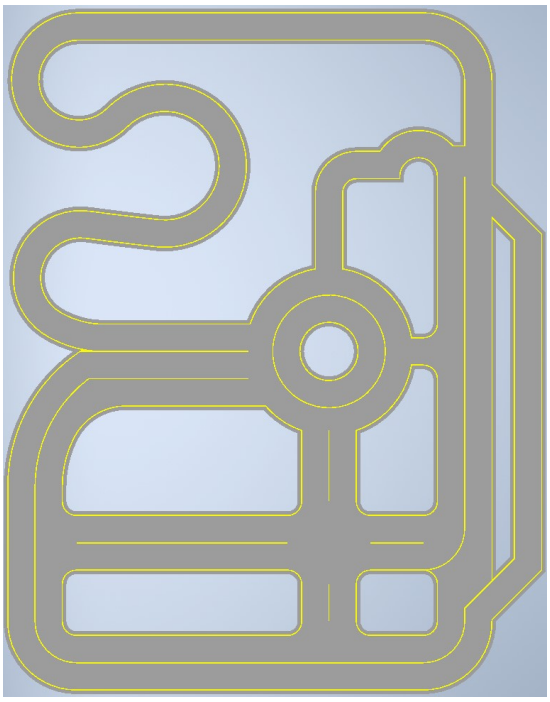

Figura 15: Sólido de la infraestructura con carriles delimitados en Autodesk Inventor

Para poder cubrir físicamente el área de la plataforma del Robotat, se optó primero por fabricar piezas de rompecabezas que, al unirlas todas, formaran la infraestructura diseñada. Se determinó que el tamaño máximo de cada pieza podía ser de 600 milímetros de alto por 450 milímetros de ancho, que es el tamaño máximo que la cortadora láser disponible en la Universidad permite cortar. Se pensó trabajar con planchas de MDF de 3 milímetros de grosor, pero por el área de trabajo necesaria, se calculó que se necesitarían 74 planchas. Al haber diseñado 10 piezas de prueba como la que se muestra en la figura 16, se determinó que este proceso no sería eficiente. Esto se debe a que al ser un área de trabajo colaborativa, iba a ser necesario armar todo el rompecabezas cada vez que se fuera a trabajar en el proyecto y remover todas las piezas al finalizar. Al ser 74 piezas, esto tomaría mucho tiempo, que se podría invertir en los demás objetivos del trabajo. Es por esto que se optó por dejar esta idea por un lado y buscar una alternativa distinta.

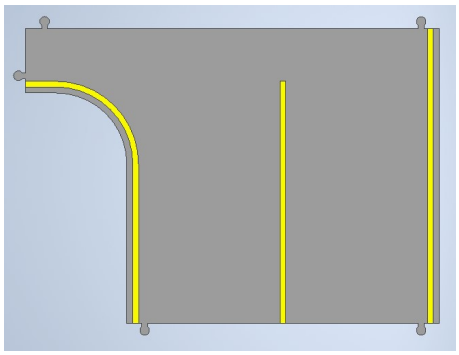

Figura 16: Pieza de prueba en forma de rompecabezas

La siguiente alternativa que se propuso fue imprimir segmentos de la carretera en planos de papel tamaño A0. Estos planos tienen un tamaño de 841 milímetros de ancho por 1189 milímetros de alto. Con este tamaño, la cuadrícula necesaria para dividir la infraestructura en partes iguales se reducía significativamente a 16 secciones. Esto significa que únicamente sería necesario imprimir 16 planos A0 para capturar por completo la geometría propuesta. Se decidió imprimir un plano de prueba como el que se muestra en la Figura 17, pero al intentar extenderlo el papel no se quedaba completamente plano sobre la superficie de la plataforma. Esto se debe a que por su tamaño, para poder almacenarlo dentro de la bodega del laboratorio, era necesario enrollar el plano. Al no poder dejar los planos completamente planos, se determinó que sería muy complicado poder lograr alinearlos todos y respetar las dimensiones y escalas de la infraestructura diseñada. Es por esto que también se optó por dejar esta idea y buscar una nueva alternativa.

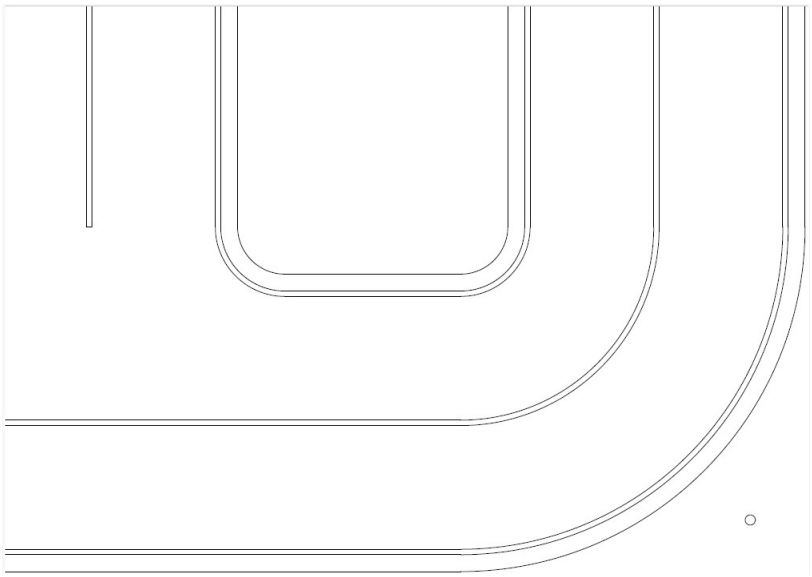

Figura 17: Plano de prueba en tamaño A0

La última alternativa que se propuso, fue imprimir la geometría completa en una lona de vinilo, como las que se utilizan en las mantas publicitarias. Se buscaron proveedores que pudieran imprimir en un material con acabado mate, para evitar reflexiones de luz sobre la lona que puedan afectar cámaras de visión por computadora, ya que esta implementación se pensó a futuro en los agentes Pololu 3Pi+. Se encontró un proveedor que podía trabajar en el material solicitado y con las dimensiones necesarias para cubrir toda la plataforma, por lo que se decidió trabajar de esta manera. Para esto, se modificó el plano general de la infraestructura para aprovechar el proceso de impresión. Se rellenaron de color verde, simulando áreas verdes, los espacios libres entre las calles y se colocó el logo de la Universidad. Esto se puede observar en la Figura 18.

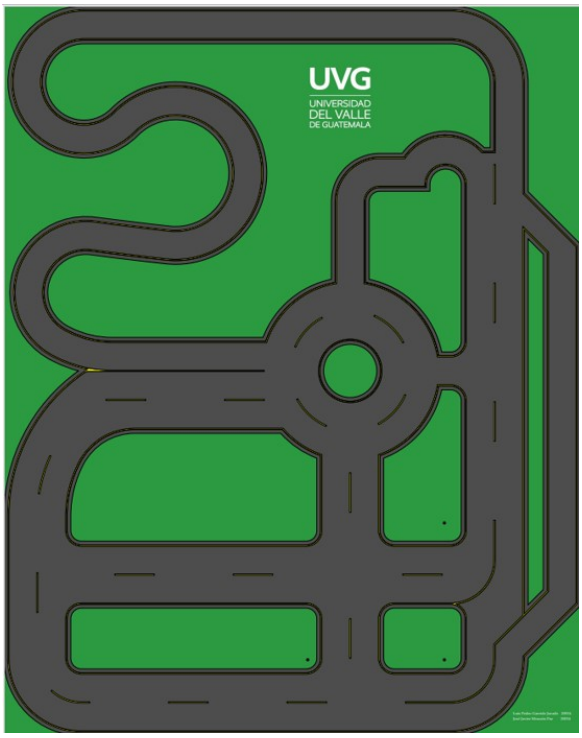

Figura 18: Diseño para impresión en lona vinílica

En esta infraestructura final diseñada, la distribución de las calles y zonas elegidas para representar a la Ciudad de Guatemala se pueden observar en la Figura 19. En color azul, se encuentra el área que representa a la carretera a Muxbal. En color rojo, se resalta el redondel que representa al Obelisco. En color amarillo, se muestra el área que representa a la Avenida Reforma y en color morado se señala el área que representa a la Zona 1.

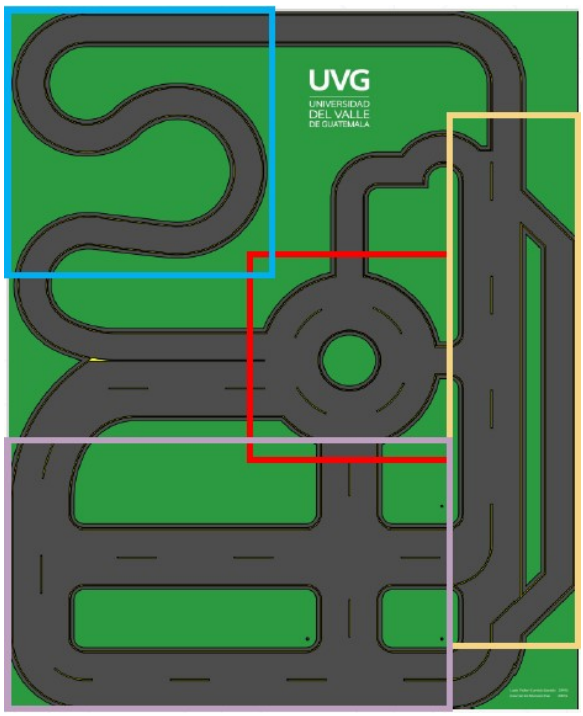

Figura 19: Calles y zonas representativas de la Ciudad de Guatemala dentro de la infraestructura diseñada

7.3. Validación en simulación de la infraestructura diseñada

Para validar la infraestructura diseñada se utilizó Matlab. En primer lugar, se obtuvo una versión digital de la carretera para poder realizar una simulación del desempeño del controlador en este entorno, previo a su implementación física. La versión digital del mapa se muestra en la Figura 20.

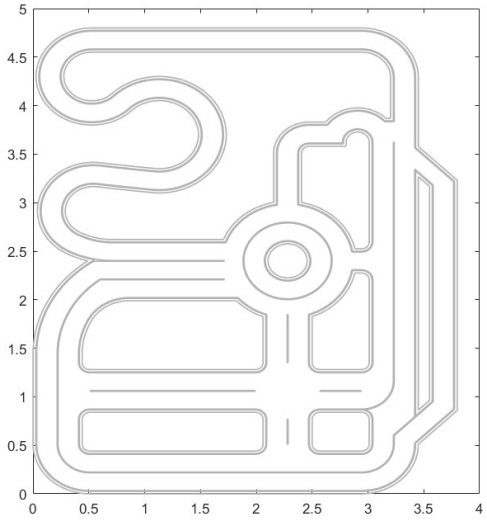

Figura 20: Mapa digital de la infraestructura diseñada

Para determinar el funcionamiento del controlador, se determinó la ruta entre dos puntos dentro del mapa para poder generar una trayectoria a seguir por el controlador. Para esta primera ruta, se seleccionó un punto inicial previo a ingresar al redondel por el sur y un punto final previo a ingresar al redondel por el este, para obligar a la ruta a pasar por el redondel y buscar pasar por la intersección de 4 vías. Esta ruta calculada se muestra en la Figura 21 con una línea punteada en color rojo.

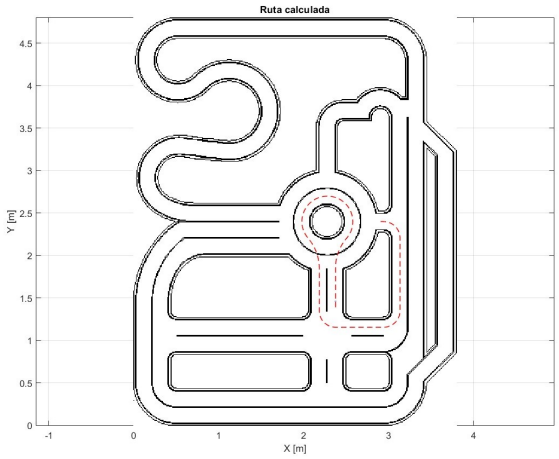

Figura 21: Ruta de prueba por el redondel

De igual forma, se seleccionó una segunda ruta que obligara al controlador a pasar por el área que representa a la carretera a Muxbal, para luego poder probar el desempeño del controlador en un área con curvas pronunciadas. Para esto, se seleccionó un punto inicial antes del comienzo de las curvas y un punto final previo a ingresar a la zona que representa

a la Avenida Reforma, esta ruta se muestra en la Figura 22.

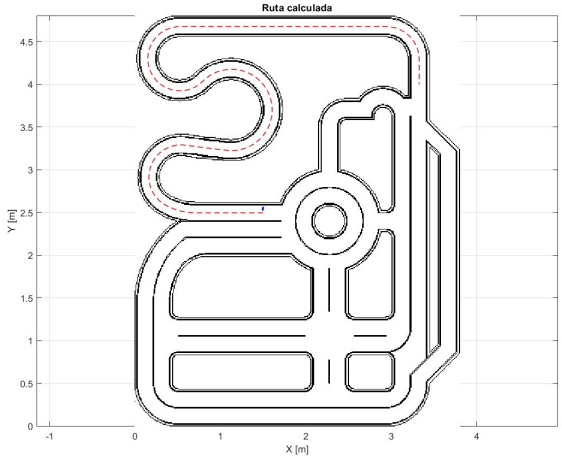

Figura 22: Ruta de prueba por zona con curvas pronunciadas

Por último, se seleccionó una ruta que debiera pasar por el carril auxiliar y el cruce de la avenida, atravesándola y no retornando sobre la misma como se muestra en la Figura 23. De esta manera, se buscó simular las calles que cruzan por completo a la Avenida Reforma, un escenario común en la Ciudad de Guatemala.

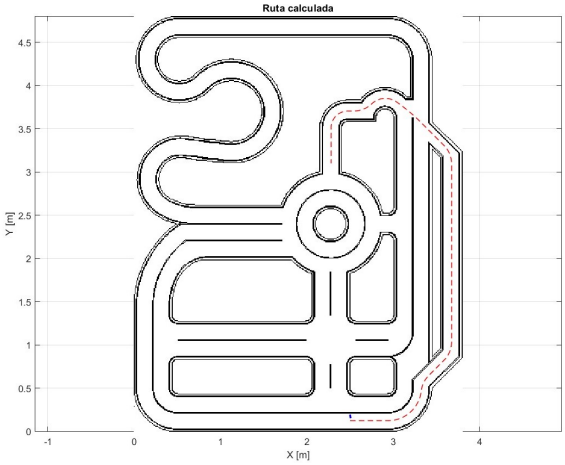

Figura 23: Ruta de prueba por carril auxiliar y atravesando la avenida

7.4. Diseño del controlador

El controlador elegido para comenzar las pruebas fue el PID con acercamiento exponencial. Este controlador fue elegido ya que permite corregir los problemas de convergencia de la velocidad lineal en comparación al PID de posición, al igual que permite seleccionar la

velocidad lineal máxima deseada y combinarlo con un PID de orientación. Con este controlador, se pueden calcular las velocidades lineales y angulares de referencia, para luego poder ser transformadas en velocidades angulares de las ruedas, siguiendo el modelo uniciclo del Pololu 3Pi+.

Los valores de las constantes del PID de orientación corresponden a $k_{P_o} = 5$, $k_{I_o} = 0.001$ y $k_{D_o} = 0.0001$. Estos valores, en conjunto con el coeficiente del PID de acercamiento exponencial $\alpha = 1000$, pueden ser modificados para cambiar el comportamiento y respuesta del controlador, pero para las pruebas en simulación, se utilizaron estos valores.

7.5. Validación del controlador en simulación

Al ser necesario correr una simulación en MATLAB, se debía utilizar un bucle FOR que permitiera simular el controlador por un tiempo definido por el usuario. Para esto se siguió la lógica que se muestra en el fragmento de Código 7.1. De esta manera, se podía calcular cada iteración e ir almacenando los datos necesarios para la validación.

```
% Pasos de tiempo para la simulacion
ts = 0.3;

% Tiempo de simulacion en segundos
tf = 40;

% Calculo de numero de iteraciones necesarias para la simulacion
t = 0:ts:tf;
N = length(t);

for k = 1:N-1
  % Bucle de simulacion
end
```

Código 7.1: Bucle de simulación en MATLAB

Para validar el funcionamiento del controlador diseñado en simulación y usando las rutas calculadas, se programó el controlador descrito como se observa en el fragmento de Código 7.2. Este código permitió calcular los valores de salida del controlador para cada iteración de la simulación, ya que se encuentra dentro del bucle FOR. Este ciclo de simulación del controlador se corrió tomando en cuenta el tiempo de retardo que existiría en la comunicación entre el servidor de captura de movimiento, Matlab y el envío de las velocidades al Polou. Este tiempo se estimó en 300 mili segundos, tomando el tiempo desde la solicitud de información al servidor del Robotat para la captura de movimiento, hasta la finalización del envío de las velocidades por medio de TCP al vehículo físico, luego de haber calculado la salida del controlador.

```
% Acercamiento exponencial
v0 = vMax;
alpha = 1000;

% PID orientacion
kp0 = 5;
ki0 = 0.001;
kd0 = 0;
```

```
  E0 = 0;
  e0_1 = 0;

  % Error de posicion
  e = [pxd - hx(k); pyd - hy(k)];

  $ Error de orientacion
  thetag = atan2(e(2), e(1));

  $ PID
  eP = norm(e);
  e0 = thetag - phi(k);
  e0 = atan2(sin(e0), cos(e0));

  % Control de velocidad lineal
  kP = v0 * (1-exp(-alpha*eP^2)) / eP;
  v = kP*eP;

  % Control de velocidad angular
  e0_D = e0 - e0_1;
  E0 = E0 + e0;
  w = kp0*e0 + ki0*E0 + kd0*e0_D;
  e0_1 = e0;

  % Combinacion de los controladores
  qpRef = [v; w];
```

Código 7.2: PID de acercamiento exponencial en MATLAB

Para poder convertir las velocidades de referencia, tanto lineal como angular, a las velocidades de cada rueda del Pololu 3Pi+, se implementó el modelo del uniciclo del robot, como se muestra en el fragmento de Código 7.3.

```
  % Extraccion de velocidades de referencia lineal y angular
  uRef(k) = qpRef(1);
  wRef(k) = qpRef(2);

  % Mapeo con modelo unicilo del robot
  wL(k) = (uRef(k)-39.5/1000*wRef(k))/(16/1000)*60/(2*pi);
  wR(k) = (uRef(k)+39.5/1000*wRef(k))/(16/1000)*60/(2*pi);

  % Acotacion a RPM maximo permitido
  wL(k) = max(min(wL, 800), -800);
  wR(k) = max(min(wR, 800), -800);
```

Código 7.3: Cálculo de velocidades de cada rueda en MATLAB

Con esta implementación del controlador en simulación en MATLAB, se pudo validar su correcto funcionamiento con las rutas previamente calculadas. Como se muestra en la Figura 24, con una velocidad máxima de 0.2 m/s, se obtiene un error máximo de 0.1 m aproximadamente. Este error ocurre en el inicio de la simulación, ya que la orientación del robot comienza con un valor aleatorio. Al comenzar a funcionar el controlador, este debe de corregir la orientación del vehículo para dirigirse en el sentido correcto, por lo que se da este error alto. Conforme avanza la simulación se puede observar que los errores disminuyen y no superan valores de 0.03 m ó 3 cm, como se puede observar en la Figura 25. Con la escala

que se trabajó, esto demostró que el vehículo nunca se saldría del carril y se mantendría centrado, siguiendo la trayectoria calculada.

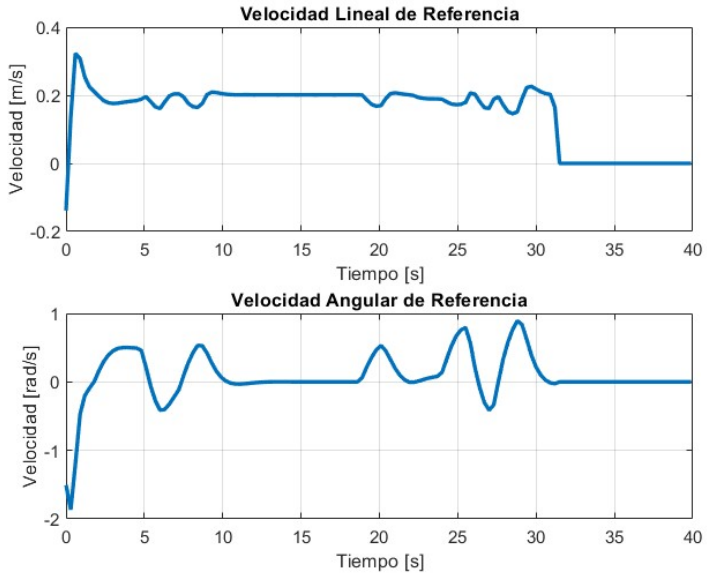

Figura 24: Simulación de controlador con trayectoria calculada y velocidad máxima de 0.2 m/s

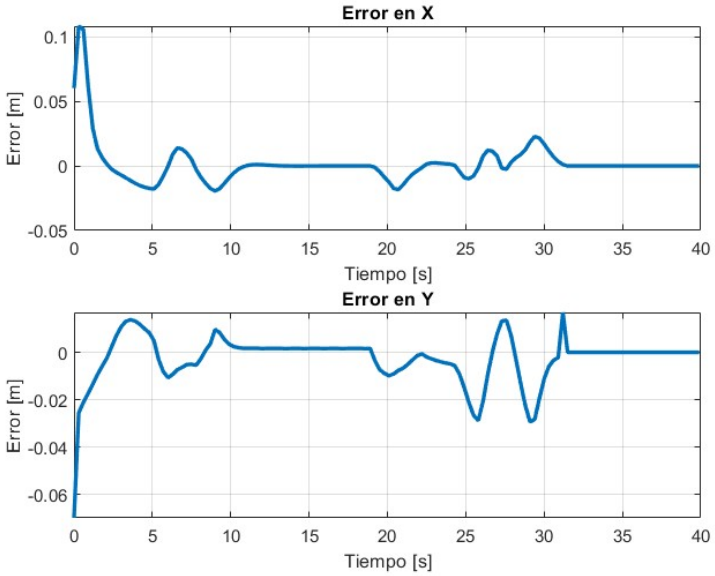

Figura 25: Errores de simulación de controlador con trayectoria calculada y velocidad máxima de 0.2 m/s

Al graficar la trayectoria del controlador sobre la trayectoria calculada, se puede observar que efectivamente, el desempeño del controlador cumple con navegar del punto de inicio al punto final, al igual que no salirse del carril. Esto se muestra en la Figura 26.

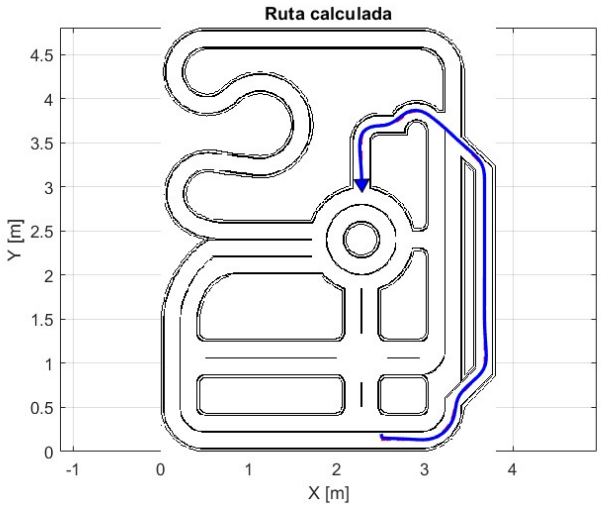

Figura 26: Simulación de desempeño del controlador con trayectoria calculada y velocidad máxima de 0.2 m/s

Al aumentar la velocidad máxima del controlador a 0.7 m/s, el desempeño del controlador en la simulación disminuyó significativamente. Como se muestra en la Figura 27, el error máximo aumenta a 0.4 m aproximadamente. Este error máximo ocurrió nuevamente al inicio de la simulación y fue debido a la orientación inicial aleatoria. El error también presentó el mismo comportamiento, disminuyendo conforme avanzaba la simulación, pero alcanzando valores de hasta 0.2 m o 20 cm como se observa en la Figura 28. Bajo la escala utilizada, esto demostró que con dicha velocidad máxima haría que el vehículo se saliera de los carriles.

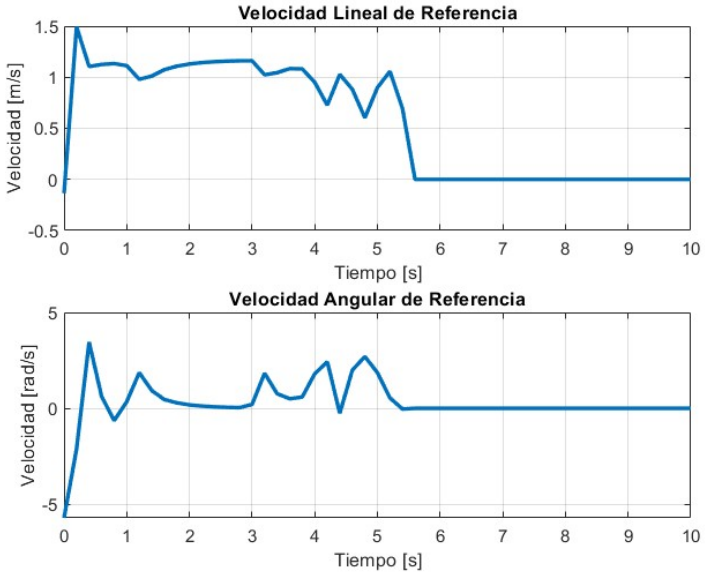

Figura 27: Simulación de controlador con trayectoria calculada y velocidad máxima de 0.7 m/s

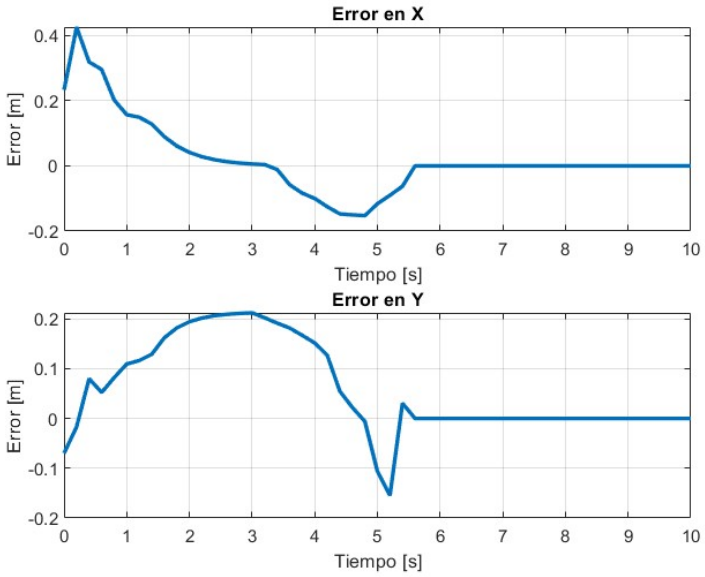

Figura 28: Errores de simulación de controlador con trayectoria calculada y velocidad máxima de 0.7 m/s

Al volver a graficar la trayectoria del controlador sobre la trayectoria calculada, se pudo observar que efectivamente, el desempeño del controlador sí cumplió con navegar del punto de inicio al punto final, pero se sale de los carriles y vías designadas. Esto se muestra en la Figura 29.

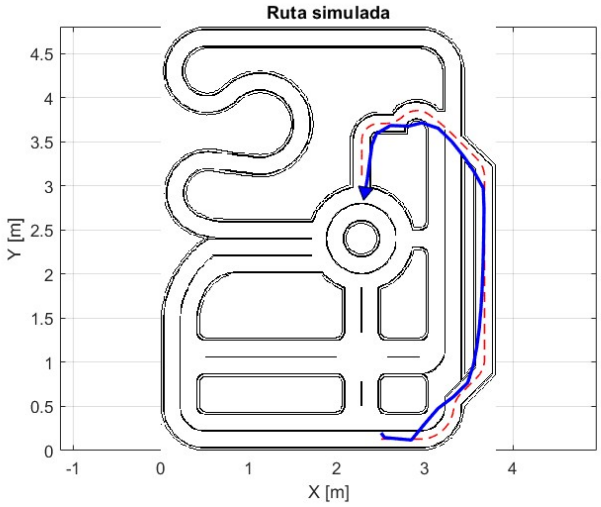

Figura 29: Simulación de desempeño del controlador con trayectoria calculada y velocidad máxima de 0.7 m/s

De la misma forma se simularon las otras dos trayectorias calculadas con velocidades de 0.2 m/s y 0.7 m/s. Los resultados de los errores obtenidos para todas las simulaciones se muestran en el Cuadro 1.

	Velocidad 0.2 m/s		**Velocidad 0.7** m/s	
	Error máximo (m)	**Error promedio** (m)	**Error máximo** (m)	**Error promedio** (m)
Trayectoria 1	0.126	0.025	0.423	0.195
Trayectoria 2	0.122	0.021	0.442	0.203
Trayectoria 3	0.124	0.024	0.437	0.201

Cuadro 1: Comparación de errores en diferentes trayectorias y con distintas velocidades

El comportamiento de los errores fue el mismo para todas las simulaciones. El error máximo siempre ocurrió al inicio de la simulación y luego disminuyó y se mantuvo con valores cercanos al error promedio. En el caso de la velocidad de 0.2 m/s, los errores promedios siempre mostraron que el vehículo no se saldría de los carriles, mientras que para la velocidad de 0.7 m/s, los errores promedios siempre mostraron que el vehículo se saldría de los carriles.

La limitante que afectó el no poder aumentar la velocidad lineal máxima del vehículo fue el tiempo que toma calcular el controlador en Matlab y enviar las señales al Pololu. En primer lugar, se lee la pose del robot con el sistema de captura de movimiento. Para esto, se envía una solicitud, por medio del protocolo de comunicación TCP, al servidor del Robotat del marcador que se desea leer. Esta solicitud es gestionada y retorna la pose del marcador en formato JSON. Esta pose se recibe como cuaterniones por lo que es necesario convertirla a la posición respectiva en x, y y el ángulo de orientación θ. Al tener la pose del marcador colocado sobre el vehículo, se pueden calcular las salidas de control necesarias para alcanzar el punto objetivo. Luego de calcular las salidas de control, estas son enviadas al vehículo por medio de otra comunicación por medio de TCP, pero esta vez con el ESP32 que está embebido en el vehículo.

Por naturaleza, el protocolo TCP es relativamente lento ya que este prioriza la integridad de los datos y se asegura que los mismos hayan llegado correctamente al destino. Por otro lado, las librerías de Matlab que gestionan la comunicación por medio de este protocolo no son tan eficientes. Matlab, al ser un lenguaje no compilado sino interpretado, es más lento en ejecutar todas las instrucciones y cálculos. Esto, sumado a los tiempos que se requieren para llevar a cabo las comunicaciones con el servidor del Robotat y el ESP32, hace que el controlador no corra en tiempo real como se espera.

Al tener una velocidad lineal máxima mayor a 0.2 m/s, el tiempo que transcurre entre la lectura de la pose del vehículo y el recibir las velocidades de control hace que el vehículo ya haya avanzado una distancia considerable, por lo que el error de posición y de orientación no representan el error real. Mientras más alta es la velocidad lineal máxima permitida, la distancia que recorre el vehículo entre los intervalos que recibe las velocidades de control es mayor, por lo que el controlador no es efectivo en estos casos. Para el caso de una velocidad de $0.7m/s$ y el tiempo estimado que transcurre entre la lectura de la pose, el cálculo del controlador y el envío de las velocidades al vehículo de 300 mili segundos, haría que el vehículo hubiera podido avanzar una distancia de hasta 21 cm entre cada iteración del controlador. Esto, bajo la escala utilizada equivaldría al 84 % del ancho de un carril. Al tener una velocidad lineal baja, esta distancia disminuye considerablemente, por lo que el controlador es capaz de funcionar correctamente.

7.6. Discusión de resultados de simulación

El desempeño del controlador PID mostró resultados significativos tanto en términos de efectividad como en la identificación de limitaciones inherentes al sistema. Las simulaciones evidenciaron que, a velocidades bajas de 0.2 m/s, el controlador logra mantener errores máximos relativamente pequeños y errores promedio en el rango de milímetros. Esto indica un alto grado de precisión y estabilidad en condiciones controladas. Sin embargo, al incrementar la velocidad a 0.7 m/s, los errores máximos aumentaron de manera notable, lo que señala una disminución en la capacidad del controlador para reaccionar eficientemente frente a trayectorias más exigentes.

Una de las principales limitaciones detectadas en las simulaciones fue el impacto del tiempo de transmisión de datos promedio de 300 ms asociado a la comunicación TCP. Este retardo temporal, aunque adecuado para garantizar la integridad de los datos, introduce una barrera significativa en la capacidad del sistema para responder en tiempo real a cambios abruptos en la trayectoria. Esto se observó con mayor claridad en trayectorias que incluían curvas pronunciadas, donde la necesidad de correcciones rápidas y precisas se ve limitada por la latencia acumulada. Si bien el protocolo TCP asegura la secuencialidad y la confiabilidad de la transmisión, su naturaleza introduce un desfase temporal que, en condiciones de mayor dinamismo, afecta negativamente el desempeño global del sistema.

En términos de desempeño dinámico, el controlador mostró un comportamiento adecuado en trayectorias rectas y en condiciones donde las tasas de cambio de posición y orientación eran moderadas. No obstante, al enfrentar trayectorias con mayor curvatura y velocidades elevadas, el desfase de 300 ms en la transmisión de datos se tradujo en desviaciones significativas entre la trayectoria ideal y la trayectoria simulada. Esto resaltó la necesidad de evaluar alternativas en los métodos de comunicación, como la adopción de protocolos de menor latencia, o de incorporar mecanismos de predicción dentro del controlador que puedan compensar de manera proactiva el desfase temporal.

CAPÍTULO 8

Implementación del controlador en tiempo real en la plataforma Pololu 3Pi+

En el capítulo anterior, se abordó el diseño y la validación de un controlador PID a través de simulaciones, con el objetivo de evaluar su desempeño en el contexto de este trabajo. Los resultados permitieron identificar fortalezas, como la capacidad de seguimiento preciso a bajas velocidades, y limitaciones, particularmente relacionadas con el retardo en la transmisión de datos y su impacto en la precisión del sistema a mayores velocidades.

En este capítulo, se documentó la implementación del controlador PID en la plataforma Pololu 3Pi+ dentro del ecosistema Robotat. Este proceso incluyó la configuración del hardware y la integración de los componentes del sistema, como la comunicación WiFi y el manejo de datos en tiempo real. Además, se realizaron pruebas experimentales para evaluar el comportamiento del controlador en condiciones reales, utilizando la infraestructura diseñada previamente. Con ello, se buscó analizar cómo las limitaciones identificadas en las simulaciones afectaron el desempeño en un entorno físico y establecer un marco para futuras optimizaciones.

8.1. Propuesta de soluciones a las limitantes encontradas

Para poder aumentar la velocidad lineal máxima del controlador, se buscó reducir el tiempo que toman las comunicaciones y los cálculos. Para esto, se utilizó el ESP32 conectado el Pololu. Este microcontrolador puede llegar hasta velocidades de reloj de 240 MHz y tiene capacidad de conexión WiFi, por lo que se pueden utilizar protocolos de comunicación TCP o incluso UDP.

Para poder implementar el controlador dentro del ESP32, era necesario desarrollar todas las funcionalidades que se corrían en Matlab pero ahora en el microcontrolador. Las funcio-

nalidades ya disponibles permitían enviar las velocidades de control al Arduino del Pololu por medio de comunicación serial, utilizando el formato CBOR[1]. De igual forma, se podían leer los datos de odometría de los *encoders* del Pololu, por lo que se tenían disponibles los valores de x, y y θ con respecto al punto inicial donde se encendiera el robot.

Para temas de depuración de código y poder monitorear variables del programa, se pensó en utilizar un servidor en una computadora portátil, ya que no sería posible usar una terminal serial para leer datos, porque el vehículo se estaría moviendo y no sería eficiente tener un cable conectado al ESP32 todo el tiempo. Como este servidor solo recibiría información para poder observarla, se optó por utilizar el protocolo UDP. Este protocolo acelera las comunicaciones al no establecer formalmente una conexión antes de transferir los datos y envía paquetes directamente al destino sin indicar el orden de dichos paquetes ni comprobar si han llegado. Esto no presenta un problema para la finalidad de este servidor ya que sólo se desean monitorear valores, no se utilizarían las variables directamente.

8.2. Configuración WiFi del ESP32

La comunicación principal del ESP32 debía ser con el servidor del Robotat, para poder recibir los valores de la pose del marcador del vehículo leídos con el sistema de captura de movimiento. Para esto, era necesario basar toda la comunicación bajo la red del servidor del Robotat. La red local establecida para el Robotat tiene una sub máscara de red en 255.255.255.0 y el servidor Robotat tiene una dirección IP de 192.168.50.200. Esto quiere decir que la red local la red está configurada para que todas las direcciones IP dentro del rango 192.168.50.1 a 192.168.50.254 estén en la misma red local. Esto significa que cualquier dispositivo con una dirección IP en este rango puede comunicarse directamente sin necesidad de pasar por un *gateway* o *router*.

Se asignó una dirección IP estática al ESP32 para asegurarse de que siempre tuviera la misma dirección IP en la red. Esto era necesario porque otros dispositivos en la red como el servidor Robotat y el servidor UDP necesitan conocer la dirección del ESP para establecer una comunicación. Si el ESP32 tuviera una IP dinámica asignada a través de DHCP, su dirección podría cambiar cada vez que se reinicia lo que complicaría la configuración de la comunicación con los demás dispositivos. La dirección IP estática definida para el ESP32 fue 192.168.50.100.

Como el servidor UDP estaría corriendo en una computadora portátil, se conectó la computadora a la red del Robotat y se verificó la dirección IP asignada a la computadora. Esta dirección fue 192.168.50.116 y se seleccionó el puerto 8888 para establecer al servidor.

El servidor del Robotat utiliza la dirección IP 192.168.50.200 y el puerto 1883. Este servidor ya estaba configurado y corriendo previo a este trabajo, por lo que solo se obtuvo la información necesaria para establecer la comunicación con el ESP32.

[1]El Concise Binary Object Representation (CBOR) es un formato de serialización binario eficiente en tamaño y procesamiento, diseñado para aplicaciones que requieren transmisión rápida de datos. [32]

8.3. Servidor para monitoreo de datos y depuración del programa

Como el propósito del servidor era recibir datos y desplegarlos para sustituir a una terminal serial, no se agregaron muchas funcionalidades al mismo. Sólo se buscaba eliminar la necesidad de una conexión física mediante cable, para facilitar la movilidad del vehículo durante las pruebas. Además, se quería poder implementar una depuración remota, donde se pudiera supervisar el funcionamiento del código desde cualquier ubicación dentro de la red local. Esto fue útil en los momentos que el Pololu se movía sobre la plataforma del Robotat ya que en todo momento se tuvo acceso a las variables deseadas.

El servidor fue implementado en Python utilizando la librería Socket, que permite la comunicación dentro de una red mediante la creación de *sockets* o puertos. Se configuró al servidor para escuchar en la dirección IP 0.0.0.0 y en el puerto 8888. Se utilizó esta dirección ya que permite que el servidor pueda recibir paquetes desde cualquier dirección IP. El protocolo implementado en el servidor fue UDP ya que no era necesario establecer una conexión continua entre el cliente y el servidor. En el caso del ESP32, este sólo debía enviar los paquetes de datos sin garantizar su entrega ya que la velocidad era más crítica que la confiabilidad de los datos, ya que se buscaba monitorearlos lo más cercano a tiempo real posible.

El servidor una vez en funcionamiento, entra en un bucle infinito donde espera recibir datos por el puerto especificado. Se implementó una lectura máxima de 1024 bytes por cada vez que ingresara información en el puerto, ya que no se esperaba monitorear demasiadas variables al mismo tiempo. Como las variables se querían enviar en formato JSON para que se recibieran de la forma nombre: valor, se implementó la decodificación de los bytes recibidos a una cadena de texto, la cual se desplegó junto con la información de la dirección IP del cliente que envió la información. Esto se pensó en el caso fuera necesario estar monitoreando datos de más de un ESP32 en simultáneo, para así poder identificar la procedencia de los datos. El Código 8.1 muestra el servidor implementado con las características mencionadas.

```
import socket
import json

UDP_IP = "0.0.0.0"
UDP_PORT = 8888

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind((UDP_IP, UDP_PORT))

try:
  while True:
    data, addr = sock.recvfrom(1024)
    data = json.loads(data.decode('utf-8'))
    print(f"Received message from {addr}: {data}")
except KeyboardInterrupt:
    print("Server closed")
finally:
  sock.close()
```

Código 8.2: Servidor UDP para recibir datos en Python

8.4. Implementación de funcionalidades en ESP32

8.4.1. Conexión al servidor del Robotat

Para poder establecer una conexión con el servidor del Robotat, se utilizó el mismo principio de comunicación cliente-servidor. En este caso, el cliente fue el ESP32 que enviaba solicitudes de datos al servidor del Robotat. Al estar el servidor del Robotat basado en una comunicación bajo el protocolo TCP, se debía implementar una conexión por medio de un puerto en el ESP32. Para esta conexión se utilizó la dirección IP y el número de puerto del servidor ya conocidos, para así poder abrir una comunicación por ese puerto. El Código 8.2 muestra la apertura de la comunicación entre el microcontrolador y el servidor, la cual se ejecuta en el ciclo de inicialización del ESP32 luego de asignar la dirección IP estática. En el código también se verificó que la conexión fuera exitosa por medio de la verificación del puerto creado. El fragmento de código únicamente muestra la función implementada para la conexión, incluyendo las variables declaradas y la forma de llamarla en el bucle de inicialización.

```
1  // Robotat server information
2  static const char *robotat_server_ip = "192.168.50.200";
3  static const int robotat_server_port = 1883;
4
5  int robotat_connect(void)
6  {
7    int sock = socket(AF_INET, SOCK_STREAM, 0);
8    if (sock < 0)
9    {
10     return -1;
11   }
12
13   struct sockaddr_in dest_addr;
14   dest_addr.sin_addr.s_addr = inet_addr(robotat_server_ip);
15   dest_addr.sin_family = AF_INET;
16   dest_addr.sin_port = htons(robotat_server_port);
17
18   int err = connect(sock, (struct sockaddr *)&dest_addr, sizeof(dest_addr)
     );
19   if (err != 0)
20   {
21     close(sock);
22     return -1;
23   }
24
25   return sock; // Return the socket descriptor for verification
26 }
27
28 // Connect to Robotat server
29 int robotat_sock = robotat_connect();
30 if (robotat_sock < 0)
31 {
32   return; // Stop execution if connection fails
33 }
```

Código 8.3.1: Conexión al servidor Robotat desde un ESP32

8.4.2. Obtención de la pose del marcador del vehículo por medio del sistema de captura de movimiento

Para poder obtener la pose del vehículo dentro de la plataforma del Robotat, fue necesario implementar una función que permitiera enviar la solicitud al servidor y procesar la información enviada de vuelta. Esta función se llamó *robotat_get_pose* y tomó como primer argumento de entrada el puerto de la conexión establecida en el Código 8.2. Otros argumentos de entrada fueron el número del marcador del que se desea obtener la pose y el arreglo donde se almacenarían los datos de la pose.

En primer lugar, se verificó que el puerto enviado fuera mayor a 0, para determinar que la conexión con el servidor del Robotat fuera una conexión abierta y correcta. Si esta conexión no existía, no se ejecutaba la función. En el caso de que la conexión con el servidor existiera y fuera correcta, se enviaba la solicitud en formato JSON. Este formato de solicitud ya existía y solo se utilizó dentro de la función. En el formato de solicitud se debía incluir el número de marcador a leer, como se muestra en el segmento de código 8.3. La variable *agent_id* es la que indica el marcador a leer.

```
1  // Create JSON request
2  sprintf(buffer, "{\"dst\":1,\"cmd\":1,\"pld\":%d}", agent_id);
3
4  // Send request to Robotat server
5  int send_result = send(sock, buffer, strlen(buffer), 0);
6  if (send_result < 0)
7  {
8    return -1;
9  }
```

Código 8.3.1.A: Envío de solicitud JSON al servidor Robotat desde un ESP32

Luego de enviar la solicitud al servidor, se estableció un *timeout* de 1 segundo para obtener una respuesta con la información de la pose. Si no se recibía la respuesta dentro del tiempo límite o si ocurría un error al recibir la información, la función dejaba de ejecutarse. (Ver segmento de Código 8.4)

```
1  // Set timeout for receiving data
2  timeout.tv_sec = 1;
3  timeout.tv_usec = 0;
4
5  // Wait for data to be available
6  FD_ZERO(&read_fds);
7  FD_SET(sock, &read_fds);
8  int ret = select(sock + 1, &read_fds, NULL, NULL, &timeout);
9  if (ret <= 0)
10 {
11   return -1;
12 }
```

Código 8.3.1.B: Establecimiento de *timeout* y recepción de datos

Al verificar que se tenía una respuesta, se procedió a leer los datos. Al leer los datos, se verificó que el formato de respuesta correspondiera a un JSON. De igual forma, si el formato no era el adecuado, la función dejaba de ejecutarse, como se muestra en el Código 8.5

```
  // Read data from server
  bytes_read = recv(sock, buffer, sizeof(buffer) - 1, 0);
  if (bytes_read <= 0)
  {
    return -1;
  }

  buffer[bytes_read] = 0;

  // Check if the response is a valid JSON string
  if (buffer[0] != '{')
    {
      return -1;
    }

    cJSON *response = cJSON_Parse(buffer);
    if (response == NULL)
    {
      return -1;
    }

```

Código 8.3.1.C: Recepción y validación de formato de datos del servidor

Al haber verificado el formato de la respuesta del servidor, se extrajo el objeto *data* y se verificó que contuviera 7 valores, ya que es lo que el servidor debe enviar como respuesta. Nuevamente, si ocurría un error durante el parseo de los datos o si el objeto *data* no contenía 7 elementos, la función dejaba de ejecutarse, como se muestra en el Código 8.6.

```
  // Parse received data
  cJSON *data = cJSON_GetObjectItem(response, "data");
  if (data == NULL || !cJSON_IsArray(data) || cJSON_GetArraySize(data) < 7)
  {
    cJSON_Delete(response);
    return -1;
  }
```

Código 8.3.1.D: Análisis de los datos recibidos

Al haber obtenido la respuesta válida del servidor, la función permitió extraer la información necesaria. Los primeros 3 elementos del objeto *data* correspondían a las coordenadas x, y y z del marcador leído. Los últimos cuatro elementos correspondían a los cuaterniones de orientación qw, qx, qy y qz. Estos valores fueron almacenados en sus variables correspondientes. La variable de los cuaterniones corresponde a un arreglo de 4 posiciones. Este arreglo de cuatro posiciones se envió a otra función que permite convertir los cuaterniones de orientación a ángulos de Euler. Por último, se liberó el espacio de memoria del objeto JSON y se retornó un valor de 0 para indicar que la función se había ejecutado correctamente. Al ocurrir errores dentro de la ejecución de estas funcionalidades, se retornaba un valor de -1 para indicar que no se había ejecutado correctamente la función. Esto se hizo para todas las veces que fuera necesario dejar de ejecutar la función, como se muestra en el Código 8.7.

```
  // Extract position and quaternion
  mocap_data[0] = cJSON_GetArrayItem(data, 0)->valuedouble; // X
  mocap_data[1] = cJSON_GetArrayItem(data, 1)->valuedouble; // Y
  mocap_data[2] = cJSON_GetArrayItem(data, 2)->valuedouble; // Z
```

```
  float q[4];
  q[0] = cJSON_GetArrayItem(data, 3)->valuedouble; // qw
  q[1] = cJSON_GetArrayItem(data, 4)->valuedouble; // qx
  q[2] = cJSON_GetArrayItem(data, 5)->valuedouble; // qy
  q[3] = cJSON_GetArrayItem(data, 6)->valuedouble; // qz

  // Convert quaternion to Euler angles (XYZ)
  float euler[3];
  quaternion_to_euler_xyz(q, euler);

  // Store Euler angles in mocap_data
  mocap_data[3] = euler[0]; // Roll
  mocap_data[4] = euler[1]; // Pitch
  mocap_data[5] = euler[2]; // Yaw

  cJSON_Delete(response);
  return 0;
```

Código 8.3.1.E: Extracción y conversión de datos de posición y orientación

8.4.3. Función de conversión de cuaterniones de rotación a ángulos de Euler

Para convertir el cuaternión de rotación a la secuencia de ángulos de Euler, se implementó el Código 8.8, siguiendo el proceso de [28].

```
void quaternion_to_euler_xyz(const float q[4], float euler[3])
{
  const float unity_norm_tol = 0.1f;
  const float angcomp_tol = 0.1f * (M_PI / 180.0f);

  // Check if the quaternion is a unit quaternion
  float norm_q = sqrtf(q[0]*q[0] + q[1]*q[1] + q[2]*q[2] + q[3]*q[3]);
  if (fabs(norm_q - 1.0f) > unity_norm_tol)
  {
    return;
  }

  // Calculate Euler angles (XYZ sequence)
  float t0 = q[3] - q[1];
  float t1 = q[0] - q[2];
  float t2 = q[1] + q[3];
  float t3 = q[2] + q[0];

  euler[2] = atan2f(t0, t1) + atan2f(t2, t3);
  euler[1] = acosf(t1*t1 + t0*t0 - 1.0f) - M_PI / 2.0f;
  euler[0] = -(atan2f(t0, t1) - atan2f(t2, t3));

  euler[0] = radiansToDegrees(euler[0]);
  euler[1] = radiansToDegrees(euler[1]);
  euler[2] = radiansToDegrees(euler[2]);

  // Check for singularities
  if (fabs(fabs(euler[1]) - M_PI / 2.0f) < angcomp_tol)
  {
    euler[2] = euler[2] + copysignf(1.0f, euler[1]) * euler[0];
```

```
31          euler[0] = 0.0f;
32        }
33    }
```

Código 8.3.3: Conversión de cuaterniones a ángulos de Euler (XYZ)

8.5. Detección de zonas muertas del sistema de captura de movimiento en la plataforma del Robotat

Por la distribución de las seis cámaras en la plataforma del Robotat, se detectaron zonas en las que las cámaras no podían determinar la pose de los marcadores utilizados. Las seis cámaras se encontraban en la periferia de la plataforma, con un ángulo de depresión que permitía cubrir la mayor área superficial de la plataforma en la posición en la que se encontraban. En la Figura 30 se muestra la ubicación de las cámaras del sistema de captura de movimiento. En la parte izquierda de la imagen se muestra la vista en planta de la plataforma, mientras que en la parte derecha se observa la vista frontal, en la cual se puede apreciar el ángulo de depresión de las cámaras para poder enfocar a la plataforma desde su ubicación de instalación.

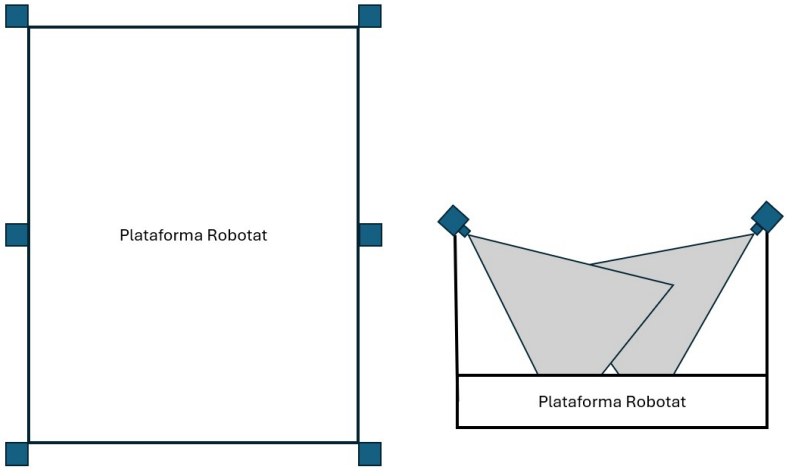

Figura 30: Distribución de las seis cámaras de captura de movimiento en la plataforma del Robotat

Debido a que las cámaras se encontraban instaladas sobre el perímetro de la plataforma y ya que estas cuentan con un campo de visión que tiene un ángulo de apertura de 51°, se generaban zonas donde no se podía detectar a los marcadores. Estas zonas, como se muestra en la Figura 31, se encontraban principalmente en las esquinas de la plataforma y en los bordes superior e inferior. En los bordes laterales no sucedía esto, ya que en los puntos medios de estos bordes si se encontraban instaladas unas cámaras que apuntaban directamente al borde opuesto por lo que si había un campo de visión disponible. Las zonas muertas tenían una altura aproximadamente de medio metro, lo que representó un área significativa bajo la escala de diseño de las calles y la infraestructura propuesta.

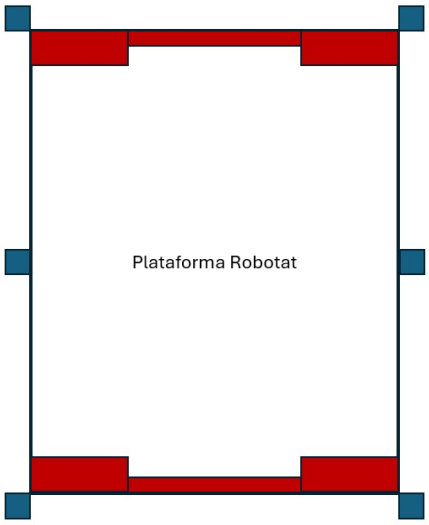

Figura 31: Zonas muertas de detección sobre la plataforma del Robotat

Debido a la falta de información del sistema de captura de movimiento y a que partes de las calles de la infraestructura diseñada pasan por estas zonas, el controlador no era capaz de funcionar al encontrarse dentro de estas áreas. Esto porque no se podía cerrar el lazo de control ya que no se tenía la retroalimentación necesaria para el controlador. Buscando una alternativa para poder obtener datos de posición y orientación que suplieran los datos del sistema de captura de movimiento, se optó por utilizar los datos de odometría obtenidos de los *encoders* de los motores de las ruedas del Pololu.

8.6. Fusión de sensores por medio de filtros complementarios

Tomando en cuenta que la frecuencia con la que se obtienen los datos del sistema de captura de movimiento no es la misma que con la que se obtienen los datos de la odometría se propuso plantear el problema de fusión de sensores desde la perspectiva de frecuencia. Esto también fue válido ya que la confiabilidad de los datos de ambas fuentes era distinta en distintos periodos de tiempo.

Tomando en cuenta la exactitud de las dos fuentes de datos, se sabía que los datos de odometría eran menos exactos que los datos del sistema de captura de movimiento. Esto quería decir que el error de los datos de la pose obtenidos a partir de la odometría era mayor. Esto tenía como consecuencia que los datos de odometría fueran exactos y confiables a corto plazo, ya que en un periodo extendido de tiempo la propagación del error haría que los datos se alejaran de la pose real del vehículo. Por otro lado, como el sistema de captura de movimiento tenía una exactitud mucho mayor, los datos que proporcionaba eran exactos y confiables a largo plazo. La poca propagación de error hacía que los datos no se alejaran de la pose real del vehículo.

La idea principal detrás de los filtros complementarios es el uso de un filtro pasa bajas para extraer la información de baja frecuencia, asociada a la fuente confiable a largo plazo,

y un filtro pasa altas para capturar la información de alta frecuencia, asociada a la fuente confiable a corto plazo. De esta forma se buscaba obtener una señal más precisa y confiable en cualquier situación. Esto contemplaba los casos en los que el vehículo se encontraría en una zona donde no se pudieran obtener datos del sistema de captura de movimiento, en donde solo se utilizarían los datos de odometría para determinar la pose del vehículo hasta obtener nuevamente un dato más confiable al salir de la zona muerta.

Se definieron las funciones de transferencia de los filtros como $H_{lp}(s)$ para el filtro pasa bajas y $H_{hp}(s)$ para el filtro pasa altas, de manera que se cumpliera la siguiente condición:

$$H_{lp}(s) + H_{hp}(s) = 1. \tag{38}$$

Esta condición aseguró que la suma de las señales filtradas de ambas fuentes abarcara todo el espectro de frecuencias relevante, sin que hubiera pérdida de información.

8.6.1. Filtro pasa bajas *leaky integrator*

Un filtro pasa bajas bajo la arquitectura de un *leaky integrator*, es la variación de un filtro integrador pero que permite un decaimiento progresivo de su respuesta. Esto permite que la señal de la salida no crezca indefinidamente y ayudó en el caso que la señal de entrada tuviera distintos valores en distintos momentos del tiempo, ya que la salida sería la superposición de las respuestas de todas estas entradas ajustadas por un factor de decaimiento debido al periodo de muestreo posible de la señal de entrada.

$$H_{lp}(s) = \frac{1}{\tau s + 1}. \tag{39}$$

En la función de transferencia del *leaky integrator* (39), el término τ permite el decaimiento progresivo de la respuesta del filtro. Este valor debía de ser entre 0 y 1 para garantizar la estabilidad del filtro y su convergencia. Para valores de τ más cercanos a 1, el filtro se comportaría más como un integrador puro, mientras que para valores más cercanos a 0, la fuga o el decaimiento serían mayores.

En el dominio de la transformada Z, la función de transferencia (40) permitió obtener la ecuación de diferencias del filtro. Con esta ecuación de diferencias se pudieron determinar las variables necesarias para implementar en código este filtro.

$$H_{lp}(Z) = \frac{1}{1 - \lambda Z^{-1}}. \tag{40}$$

Utilizando un valor de λ de 0.5, la ecuación de diferencia (41 mostró que la salida del filtro dependía tanto de la entrada, es decir de la medición actual obtenida del sistema de captura de movimiento, como de la salida calculada una iteración pasada. En este caso, la fuga del filtro afectaba únicamente al cálculo de la iteración anterior y no a los datos medidos directamente por el sistema de captura de movimiento.

$$y[n] = 0.5y[n-1] + x[n]. \tag{41}$$

Desde el punto de vista de frecuencia, esto quiere decir que los datos confiables del sistema de captura de movimiento se incorporan directamente a la señal de salida cada vez que hay una medición nueva. Por otro lado, los cálculos anteriores de las iteraciones pasadas, se van fugando conforme pasan las iteraciones, tal y como se espera de un filtro de esta clase. De nuevo, se puede apreciar como al tener un valor de λ igual a 1, el filtro se comportaría como un integrador puro, sumando completamente los cálculos de la iteración pasada a la entrada nueva.

8.6.2. Filtro pasa altas complementario

Para el filtro pasa altas, se calculó la función de transferencia en base a la del filtro pasa bajas, haciendo la resta de la función pasa bajas de la unidad. La función de transferencia obtenida se muestra en (42).

$$H_{hp}(s) = \frac{\tau s}{\tau s + 1}. \tag{42}$$

Desde el punto de vista de frecuencia, este filtro efectivamente representa a un filtro pasa altas, ya que para valores de frecuencia bajas en los que s tiende a cero, se atenúan las ganancias de las salidas mientras que para valores de frecuencias altas en donde s tiene valores grandes, las ganancias son muy cercanas a 1.

En el dominio de la transformada Z, la función de transferencia (43) permitió obtener la ecuación de diferencias del filtro. Con esta ecuación de diferencias, se pudieron determinar las variables necesarias para implementar en código este filtro.

$$H_{hp}(s) = \frac{-\lambda Z^{-1}}{1 - \lambda Z}. \tag{43}$$

Nuevamente con un valor de λ de 0.5, la ecuación de diferencias (44) que representa la filtro complementario mostró que las variables que determinaron la salida del filtro fueron la entrada en de la iteración pasada, al igual que la salida del filtro de la iteración pasada.

$$y[n] = 0.5y[n-1] - 0.5x[n-1]. \tag{44}$$

En el caso de este filtro, al ser complementario al filtro pasa bajas, se pudo apreciar que las entradas del filtro, es decir las lecturas de odometría con una mayor frecuencia, no tienen un efecto completo en la salida del filtro. En este caso, al ser restadas y atenuadas en un factor igual al de fuga, se aseguraba que la poca duración de fiabilidad de los datos de entrada no afectara en su totalidad a la salida. Por otro lado, al tomar en cuenta la entrada de la iteración pasada y no la entrada actual, el filtro podía capturar las variaciones rápidas teniendo información sobre el cambio entre ambos datos entre iteraciones. En el caso de la

salida de la iteración pasada, esta almacenaba información relevante de los datos anteriores pero tampoco en su totalidad, sino atenuando su aporte a la salida en el mismo factor. Esta combinación permitió obtener el comportamiento esperado para el filtro que se utilizaría para la fuente de datos rápida pero no tan fiable.

8.6.3. Resultados de estimación de pose utilizando un filtro complementario y dos fuentes de datos

Como se puede observar en la Figura 32, con las dos fuentes de información disponibles para la pose del vehículo, se implementaron los filtros complementarios descritos en las secciones anteriores para determinar la pose actual del Pololu sin importar la disponibilidad de datos de cualquiera de las dos fuentes. Esto se realizó con el propósito de tener información disponible sobre la pose del robot en las zonas muertas de la plataforma del Robotat y así poder seguir corriendo el controlador diseñado.

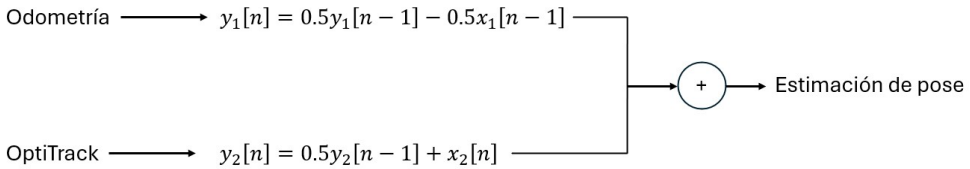

Figura 32: Implementación de filtro complementario para estimación de pose del vehículo sobre la plataforma del Robotat

Para poder evaluar el rendimiento del filtro complementario, se colocó el vehículo en cuatro posiciones aleatorias dentro de la plataforma del Robotat y se tomó la pose exacta del mismo utilizando el sistema de captura de movimiento y la pose estimada con el filtro complementario. Se compararon las poses obtenidas para las cuatro posiciones aleatorias y se calculó el porcentaje de error de las mismas.

Posición 1	**Robotat**	**Filtro complementario**	**% Error**
x (m)	1.25	1.39	11.20
y (m)	0.79	0.88	11.39
Theta (rad)	-1.73	-1.92	11.11

Cuadro 2: Comparación entre pose real y pose estimada utilizando filtros complementarios en posición aleatoria 1

Posición 2	**Robotat**	**Filtro complementario**	**% Error**
x (m)	1.28	1.42	10.94
y (m)	0.37	0.42	13.51
Theta (rad)	-2.50	-2.77	11.14

Cuadro 3: Comparación entre pose real y pose estimada utilizando filtros complementarios en posición aleatoria 2

Posición 3	Robotat	Filtro complementario	% Error
x (m)	-1.22	-1.36	11.48
y (m)	0.75	0.84	12.00
Theta (rad)	4.04	4.49	11.11

Cuadro 4: Comparación entre pose real y pose estimada utilizando filtros complementarios en posición aleatoria 3

Posición 4	Robotat	Filtro complementario	% Error
x (m)	-1.27	-1.42	11.81
y (m)	-1.57	-1.75	11.46
Theta (rad)	-5.80	-6.45	11.12

Cuadro 5: Comparación entre pose real y pose estimada utilizando filtros complementarios en posición aleatoria 4

Como se puede observar en los Cuadros 2, 3, 4 y 5, el porcentaje de error promedio de los datos de la estimación de la pose obtenidos con el filtro complementario es de 11.52 %. Tomando en cuenta la escala utilizada para el diseño de la plataforma, este porcentaje de error equivale a un error promedio de 49.52 centímetros o el equivalente a 2.7 carriles en cuanto a la posición x y y del vehículo. En cuanto a la orientación del vehículo, este porcentaje de error equivale a un error promedio de 41.47 grados. Estos errores son significativos para el desempeño del controlador, lo que haría que el vehículo no pueda transitar dentro de los carriles designados, ni dirigirse al punto deseado.

8.6.4. Análisis de las limitaciones encontradas en el uso del filtro complementario para fusión de sensores

El alto porcentaje de error obtenido en la estimación de la pose mediante el filtro complementario se debió a múltiples factores relacionados con las características de las fuentes de datos, las limitaciones del sistema de captura de movimiento y la superficie sobre la cual se realizaron las pruebas. Estos factores hicieron que los errores presentados fueran altos en cuanto a la escala utilizada y que no permitieran que la solución propuesta fuera adecuada para asegurar el funcionamiento correcto del sistema de control propuesto en todo momento.

El sistema de odometría proporcionaba datos rápidamente, pero estos solo eran confiables a corto plazo debido a la acumulación de error conforme el robot recorría distancias mayores. Por otro lado, el sistema de captura de movimiento, a pesar de que ofrecía datos con mayor precisión, operaba con una frecuencia de muestreo más baja. Esta diferencia en las tasas de muestreo generó problemas al intentar fusionar ambas fuentes de información. El filtro complementario, a pesar de estar diseñado para trabajar con todo el espectro de frecuencias, al procesar datos con distintas frecuencias de muestreo no lograba compensar adecuadamente la diferencia temporal entre la medición de los datos de ambas fuentes. Esto también incrementó los errores en la estimación de la pose.

Otro factor que influyó en el porcentaje de error alto fueron las zonas muertas donde los marcadores no fueron detectados correctamente o donde no se capturaron datos en absoluto.

En estas zonas muertas, el sistema de estimación de pose dependía exclusivamente de la odometría, lo cual incrementó los errores debido al error presente en la fuente de datos que se propagaba rápidamente conforme el vehículo avanzaba. Como se había mencionado, la odometría acumulaba errores de forma no lineal, lo que afectaba especialmente a las estimaciones de largo plazo que no eran corregidas por el sistema de captura de movimiento.

Por otro lado, las calles por donde se realizaron las pruebas fueron impresas en una lona vinílica flexible, por lo que siempre presentaba arrugas y perturbaciones en su superficie. Estas irregularidades afectaron directamente a los *encoders* del vehículo, ya que la distancia medida no correspondía exactamente con la distancia real recorrida por el robot. Las variaciones y cambios en la superficie generaron pequeñas variaciones en la velocidad de las ruedas, lo que incrementó el error en la medición de la distancia recorrida. Dado que el filtro complementario depende en parte de los datos de la odometría para su funcionamiento, este error adicional causado por la superficie irregular aumentó los porcentajes de error obtenidos en la estimación de la pose.

Los errores relacionados con la orientación del robot se acumularon de manera progresiva durante las pruebas. Pequeños errores en la estimación de la orientación inicial resultaron en desviaciones significativas a lo largo del recorrido del robot. El filtro complementario no fue capaz de corregir completamente las desviaciones angulares cuando los datos de odometría eran los únicos disponibles, ya que el sistema de captura de movimiento no brindaba una actualización suficiente en tiempo real.

8.7. Fusión de sensores por medio de filtro de Kalman extendido

Para poder eliminar las limitantes del filtro complementario en la aplicación de la estimación de la pose del vehículo, se optó por trabajar con el FKE. A diferencia del filtro complementario, que opera de manera relativamente sencilla al combinar dos señales de entrada (una de respuesta rápida pero propensa al error acumulativo, y otra más lenta pero precisa), el FKE es capaz de gestionar las dos fuentes de datos y su fusión, minimizando el error cuadrático medio mediante un proceso de predicción y corrección iterativo.

En el caso de los vehículos autónomos, la pose del vehículo está descrita por ecuaciones de movimiento no lineales debido a las componentes trigonométricas en las ecuaciones de la velocidad y la orientación. Estas no linealidades son difíciles de manejar con métodos simples como el filtro complementario, que asume que las señales de entrada pueden combinarse mediante métodos lineales. El FKE extiende el filtro de Kalman clásico para manejar sistemas no lineales mediante la linealización local del sistema en cada iteración, esto permite procesar las ecuaciones de movimiento del vehículo de manera más precisa para mejora la estimación de la pose.

Por otro lado, el método incorpora explícitamente el modelado del ruido tanto en el proceso como en las mediciones. El filtro asume que el sistema tiene dos fuentes principales de incertidumbre: el ruido del proceso, que es el error que se introduce cuando el vehículo se desplaza y las predicciones no son perfectas, y el ruido de la medición, que refleja la incertidumbre de los sensores de la odometría y el sistema de captura de movimiento. El

filtro de Kalman modela ambos tipos de ruido mediante las matrices de covarianza Q_w y Q_v. Esto representa una ventaja sobre el filtro complementario que no tiene una forma explícita de modelar ni corregir el ruido acumulado, teniendo un efecto directo en la estimación de la pose.

El modelado de las matrices de covarianza para el sistema del vehículo se puede observar en las ecuaciones (45) y (46). En este caso, los errores en las mediciones provenían directamente de las mediciones de los *encoders*, modelados como las varianzas de dichos datos $\sigma_{\delta\rho}$(47) y $\sigma_{\delta\theta}$(48).

$$\mathbf{Q_w} = \begin{pmatrix} \sigma^2_{\delta\rho} & 0 \\ 0 & \sigma^2_{\delta\theta} \end{pmatrix}, \tag{45}$$

$$\mathbf{Q_v} = \begin{pmatrix} \sigma^2_{\delta x} & 0 & 0 \\ 0 & \sigma^2_{\delta y} & 0 \\ 0 & 0 & \sigma^2_{\delta\theta}, \end{pmatrix} \tag{46}$$

$$\delta_\rho = \frac{2\pi r\,(\Delta\text{tick}/N)\,(L+R)}{2}, \tag{47}$$

$$\delta_\rho = \frac{2\pi r\,(\Delta\text{tick}/N)\,(L-R)}{\ell}. \tag{48}$$

El filtro de Kalman también ajusta la importancia y efecto de las predicciones del sistema frente a las mediciones mediante la ganancia de Kalman. Esta cantidad se re calcula en cada iteración. Si las mediciones son precisas, como sucede con el sistema de captura de movimiento cuando no hay zonas muertas, el filtro confiaría más en los datos medidos que en la predicción. Por otro lado, si las mediciones tienen mucha incertidumbre como cuando la odometría proporciona datos con ruido debido a las irregularidades en la superficie de la lona o se está pasando por zonas muertas, el filtro de Kalman confiaría más en el modelo de predicción.

Debido a que las variables de estado del sistema del vehículo eran x, y y θ, el vector de estado del sistema se pudo modelar como (49).

$$\xi = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}. \tag{49}$$

Por otro lado la predicción se modeló como (50), basado en el modelo del vehículo y sus tres variables de estado.

$$\hat{\boldsymbol{\xi}}_{k|k-1} = \begin{bmatrix} \hat{x}_{k|k-1} \\ \hat{y}_{k|k-1} \\ \hat{\theta}_{k|k-1} \end{bmatrix} = \begin{bmatrix} \hat{x}_{k-1|k-1} + \delta_\rho \cos\left(\hat{\theta}_{k-1|k-1}\right) \\ \hat{y}_{k-1|k-1} + \delta_\rho \sin\left(\hat{\theta}_{k-1|k-1}\right) \\ \hat{\theta}_{k-1|k-1} + \delta_\theta \end{bmatrix}. \tag{50}$$

Dado el modelo de predicción mostrado en (21) y (22) presentados en el marco teórico y las variables de estado, las matrices A y F están dadas por (51) y (52).

$$\boldsymbol{A} = \frac{\partial \mathcal{F}}{\partial \xi} = \begin{bmatrix} 1 & 0 & -\delta_\rho \sin\left(\hat{\theta}_{k-1|k-1}\right) \\ 0 & 1 & \delta_\rho \cos\left(\hat{\theta}_{k-1|k-1}\right) \\ 0 & 0 & 1 \end{bmatrix}, \tag{51}$$

$$\boldsymbol{F} = \frac{\partial \mathcal{F}}{\partial w} = \begin{bmatrix} \cos\left(\hat{\theta}_{k-1|k-1}\right) & 0 \\ \sin\left(\hat{\theta}_{k-1|k-1}\right) & 0 \\ 0 & 1 \end{bmatrix}. \tag{52}$$

Dado el modelo presentado en (23) y (24) y las variables de estado, el vector Z_k se modeló como (53).

$$\boldsymbol{Z_k} = \begin{bmatrix} x_{OT} \\ y_{OT} \\ \theta_{OT} \end{bmatrix} - \begin{bmatrix} \hat{x}_{k|k-1} \\ \hat{y}_{k|k-1} \\ \hat{\theta}_{k|k-1} \end{bmatrix}, \tag{53}$$

en donde los subíndices $_{OT}$ indican que son mediciones tomadas del sistema de captura de movimiento.

Teniendo estas ecuaciones definidas, se procedió a desarrollarlas en Matlab. Debido a que son ecuaciones matriciales, desarrollarlas en C como ecuaciones algebraicas a mano no hubiera sido un acercamiento eficiente para la aplicación. Por eso se convirtieron las ecuaciones matriciales en funciones dentro de Matlab, utilizando el comando matlabFunction. Este comando permite convertir una expresión simbólica en una función que se puede ejecutar para calcular el resultado de la expresión definida.

Al haber obtenido las funciones en Matlab tanto de la predicción del estado y la varianza como de la corrección del estado y la varianza, se procedió a utilizar el Toolbox llamado Coder de Matlab. Matlab Coder tradujo las funciones de Matlab a código en C optimizado, incluyendo tanto las funciones principales como las subrutinas asociadas. Esto permitió desarrollar las ecuaciones matriciales del filtro de Kalman presentadas en cuatro funciones principales. Estas cuatro funciones fueron:

1. Predicción del estado.
2. Predicción de la varianza.
3. Corrección del estado.
4. Corrección de la varianza.

Estas cuatro funciones fueron recopiladas en un mismo archivo llamado kalman.c. De igual manera, las 4 funciones necesitaban su archivo *header*, por lo que se recopilaron los 4 *headers* de las funciones en un mismo archivo llamado kalman.h. Estas funciones fueron

utilizadas dentro de las funcionalidades previamente descritas que se desarrollaron en el ESP32. La predicción tanto del estado como de la varianza, se ejecutó luego de leer los datos de odometría recibidos de los *encoders*. La corrección tanto del estado como de la varianza, se ejecutó luego de obtener la pose exacta del vehículo por medio del sistema de captura de movimiento. Con estas variables *a posteriori* del estado, se calcularon los errores tanto de posición como de orientación para el controlador previamente diseñado.

8.7.1. Resultados de estimación de pose utilizando el FKE y dos fuentes de datos

Para poder evaluar el rendimiento del FKE, se colocó el vehículo en cuatro posiciones aleatorias dentro de la plataforma del Robotat y se tomó la pose exacta del mismo utilizando el sistema de captura de movimiento y la pose estimada con el filtro complementario. Se compararon las poses obtenidas para las cuatro posiciones aleatorias y se calculó el porcentaje de error de las mismas.

Posición 1	**Robotat**	**Filtro Kalman**	**% Error**
x (m)	1.7293	1.7294	0.01
y (m)	1.9860	1.9859	0.01
Theta (rad)	2.2298	2.2299	0.00

Cuadro 6: Comparación entre pose real y pose estimada utilizando filtro Kalman en posición aleatoria 1

Posición 2	**Robotat**	**Filtro Kalman**	**% Error**
x (m)	-1.0996	-1.0996	0.00
y (m)	1.3181	1.3181	0.00
Theta (rad)	-2.4790	-2.4792	0.01

Cuadro 7: Comparación entre pose real y pose estimada utilizando filtro Kalman en posición aleatoria 2

Posición 3	**Robotat**	**Filtro Kalman**	**% Error**
x (m)	-1.0251	-1.0250	0.01
y (m)	-1.4093	-1.4093	0.00
Theta (rad)	0.1890	0.1889	0.05

Cuadro 8: Comparación entre pose real y pose estimada utilizando filtro Kalman en posición aleatoria 3

Posición 4	Robotat	Filtro Kalman	% Error
x (m)	1.3808	1.3808	0.00
y (m)	-1.6156	-1.6157	0.01
Theta (rad)	2.6395	2.6394	0.00

Cuadro 9: Comparación entre pose real y pose estimada utilizando filtro Kalman en posición aleatoria 4

Como se puede observar en los Cuadros 6, 7, 8 y 9, los porcentajes de error obtenidos con este método son sumamente bajos. En este caso, el error promedio es del 0.01 %, lo que bajo la escala utilizada para la infraestructura diseñada equivale a un error promedio de 4.3 centímetros de error o el equivalente a 0.03 carriles. En cuanto a la orientación, este porcentaje de error equivale a un error promedio de 3.6 grados. Con estos valores, sí se puede asegurar un buen desempeño del controlador, haciendo que el vehículo pueda transitar dentro de los carriles designados y dirigirse al punto deseado.

8.7.2. Análisis de los resultados y las limitaciones encontradas en el uso del filtro de Kalman para fusión de sensores

Los resultados obtenidos con el filtro de Kalman extendido mostraron que este método tuvo mejoras significativas respecto al filtro complementario, particularmente en la estimación de la pose del vehículo. Los porcentajes de error obtenidos con el filtro de Kalman extendido fueron extremadamente bajos, con un error promedio de 0.01 % en comparación con los altos errores promedio 11.52 % obtenidos con el filtro complementario.

El FKE permitió gestionar de manera más efectiva la incertidumbre inherente en las mediciones provenientes de la odometría y el sistema de captura de movimiento. A diferencia del filtro complementario que asume que las señales pueden combinarse de manera lineal, el FKE ajustó las predicciones del sistema a medida que se disponía de nuevas mediciones para cada una de las fuentes de datos, independientemente de su periodo de muestreo.

Un aspecto importante para los bajos porcentajes de error fue la capacidad del filtro de Kalman para modelar explícitamente el ruido en el sistema mediante las matrices de covarianza Q_v y Q_w, lo que no fue posible en el filtro complementario. Estas matrices representaban el ruido de proceso y de medición, respectivamente. El filtro ajustaba la confianza en las mediciones en función de la calidad de los datos. Por ejemplo, cuando las mediciones del sistema de captura de movimiento eran precisas, el filtro priorizaba estos datos sobre las predicciones. Por otro lado, en zonas donde las mediciones presentaban incertidumbre o zonas muertas, se confiaba más en las predicciones del modelo.

A pesar de los buenos resultados, se identificaron algunas limitaciones. El FKE requiere una correcta inicialización de las matrices de covarianza y una buena estimación inicial del estado para garantizar un desempeño óptimo. Para esto fue necesario obtener una cantidad significativa de mediciones tanto de los *encoders* como del sistema de captura de movimiento, para poder calcular la varianza real de ambos sistemas y así inicializarlos correctamente. Cualquier error en esta fase puede causar inestabilidad en el sistema o una estimación incorrecta de la pose. Además, el filtro es computacionalmente más costoso que el filtro

complementario, lo que puede ser un desafío en sistemas con recursos limitados, como el ESP32 del Pololu en el cual se implementó. Posiblemente, para otros microcontroladores que permitan calcular ecuaciones matriciales de una manera más efectiva computacionalmente, este problema se pueda erradicar.

En este capítulo se documentó la implementación del controlador PID en tiempo real en la plataforma Pololu 3Pi+, utilizando el módulo ESP32 para la comunicación y la integración de los componentes necesarios para el control del sistema. Se desarrollaron las funcionalidades del controlador en el microcontrolador y se llevaron a cabo pruebas experimentales que permitieron evaluar su desempeño en condiciones reales. Los resultados confirmaron la capacidad del controlador para seguir trayectorias predefinidas a bajas velocidades, aunque evidenciaron limitaciones a velocidades más altas, principalmente debido a la latencia en la comunicación y las características dinámicas de la plataforma.

CAPÍTULO 9

Validación del controlador en la infraestructura diseñada

Con los resultados del rendimiento del filtro de Kalman con errores muy cercanos a cero, se procedió con la validación del controlador implementado en el ESP32 del Pololu en la infraestructura del Robotat con la lona de la infraestructura diseñada.

9.0.1. Mejoras en el sistema de captura de movimiento para reducir el efecto de las zonas muertas

Para ayudar a mejorar el rendimiento en las zonas muertas, se procedió a agregar un marcador al cuerpo que reconoce el sistema de captura de movimiento, teniendo un total de 4 marcadores en el cuerpo del Pololu, como se puede observar en la Figura 33. Esto permitió que se redujera aún más el efecto de no poder leer datos con el sistema de captura de movimiento en estas zonas, ya que al tener 4 puntos de referencia para identificar al cuerpo, se ayudó a que fuera más probable que se cumpliera la condición de tener visibilidad de por lo menos 3 puntos de referencia para que el servidor del Robotat pudiera determinar la pose del Pololu. Estos fueron los datos utilizados para la corrección en el filtro de Kalman, mejorando así el rendimiento general del controlador en todas las zonas de la plataforma.

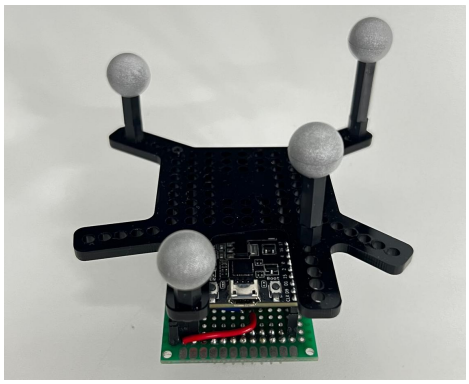

Figura 33: 4 marcadores para ayudar al sistema de captura de movimiento en las zonas muertas de la plataforma del Robotat

9.0.2. Resultados de correr las rutas presentadas en la simulación inicial sobre la plataforma del Robotat con el controlador en tiempo real implementado en el Pololu

Para poder comparar la ruta planificada con la ruta ejecutada por el controlador implementado, se fueron almacenando los datos de posición real enviados al servidor de Python. Con estos datos se pudo recrear la trayectoria real ejecutada por el controlador.

Para poder recibir los datos de los puntos que conforman la trayectoria, se tomó una distancia entre puntos de 10 cm para seccionarla. Con la trayectoria seccionada, se almacenaron los puntos en un archivo .csv el cual se envió por medio de la comunicación TCP implementada. Los datos enviados fueron únicamente los puntos que conforman la meta a alcanzar por el controlador, siento estos xg y yg. Teniendo una velocidad lineal máxima seleccionada para probar el rendimiento del control con las mejoras realizadas y sabiendo la distancia entre los puntos, se calculó el intervalo de tiempo para el envío del punto siguiente en la lista de la trayectoria. La velocidad seleccionada fue de 0.7 m/s, por lo que teniendo la distancia entre puntos de 10 cm, se calculó el intervalo de tiempo para el envío de los puntos cada 142 mili segundos.

Para la primera ruta, se obtuvo el resultado que se observa en la Figura 34. En color rojo punteado, se muestra la ruta ideal planificada y en color azul continuo la trayectoria ejecutada por el Pololu. Se puede apreciar que las curvas cerradas son un reto aún para el controlador ya que no las ejecuta a la perfección, pero si es capaz de mantenerse dentro del carril con una velocidad lineal alta. En comparación con la misma ruta simulada tomando en cuenta la alta latencia sin el controlador en tiempo real, se puede afirmar que existe una diferencia significativa en el rendimiento del agente. Las partes rectas de la trayectoria no presentan mayor dificultad para el controlador, ya que estas son ejecutadas sin desviarse significativamente de la trayectoria original planificada.

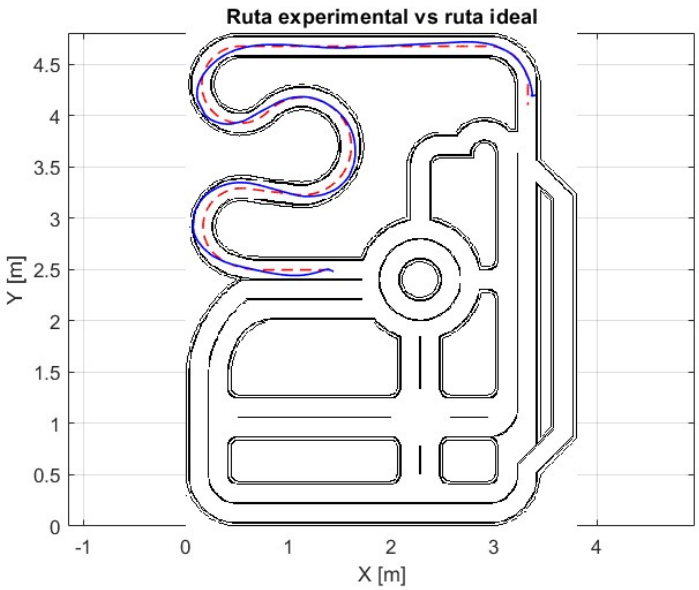

Figura 34: Comparación entre primera ruta ideal planificada y la ruta ejecutada por el agente mediante el controlador en tiempo real implementado

En cuanto a la segunda ruta experimental de la Figura 35, se puede apreciar nuevamente que las curvas cerradas a alta velocidad todavía presentan complicaciones para el rendimiento del controlador pero, en general, el agente es capaz de mantenerse dentro del carril y ejecutar en su totalidad la ruta planificada, teniendo una velocidad lineal alta constante. Nuevamente, este rendimiento y esta ejecución no hubieran sido logradas con el controlador corriendo en Matlab y enviando las velocidades al agente, debido a la latencia en la comunicación y la distancia recorrida por el agente entre cada ciclo del controlador debido a la velocidad lineal alta.

Este mismo comportamiento se observó en la Figura 36 de la tercera ruta experimental ejecutada. En este caso, las curvas cerradas casi no se encuentran en la trayectoria sino esta está compuesta en su mayoría por tramos rectos, por lo que se puede apreciar un mejor rendimiento y una mejor ejecución por parte del agente.

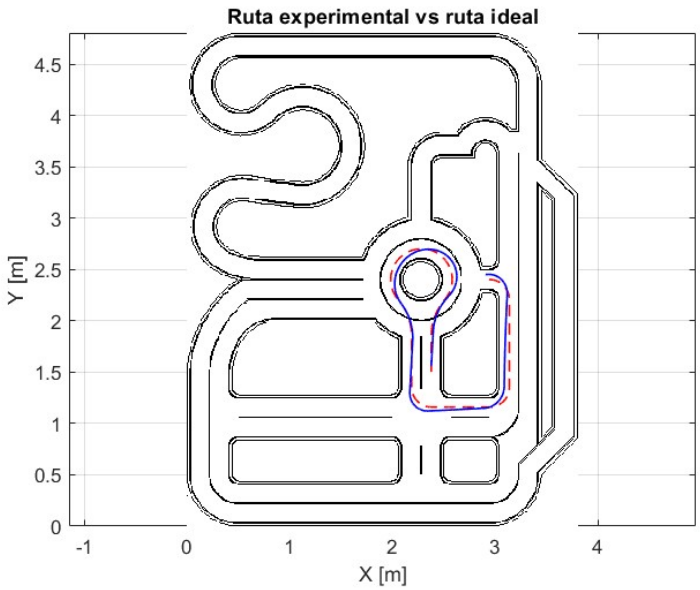

Figura 35: Comparación entre segunda ruta ideal planificada y la ruta ejecutada por el agente mediante el controlador en tiempo real implementado

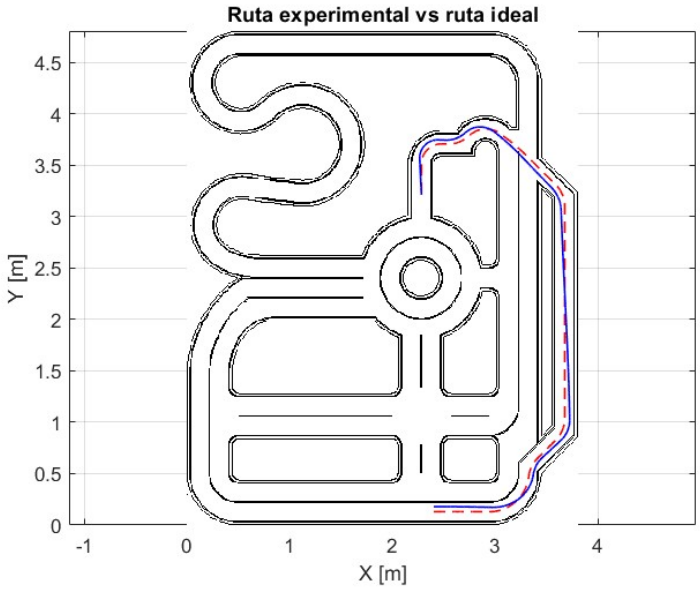

Figura 36: Comparación entre tercera ruta ideal planificada y la ruta ejecutada por el agente mediante el controlador en tiempo real implementado

En cuanto a los errores presentados por las rutas experimentales, se puede observar en las Figuras 37, 38 y 39 que para todos los errores en x de las tres trayectorias, no se obtiene un valor mayor a 0.05 m, es decir no mayor a 5 cm. Por otro lado, únicamente para la segunda trayectoria, la del redondel, se obtiene un error en y ligeramente mayor a los 5 cm, en el momento en el que el controlador intenta adaptarse a la curva cerrada continua. Luego, para el resto de la trayectoria no se vuelve a superar un valor de error de 5 cm. Para las otras

dos trayectorias, este mismo comportamiento de no superar los valores de 5 cm se cumple para los errores en y.

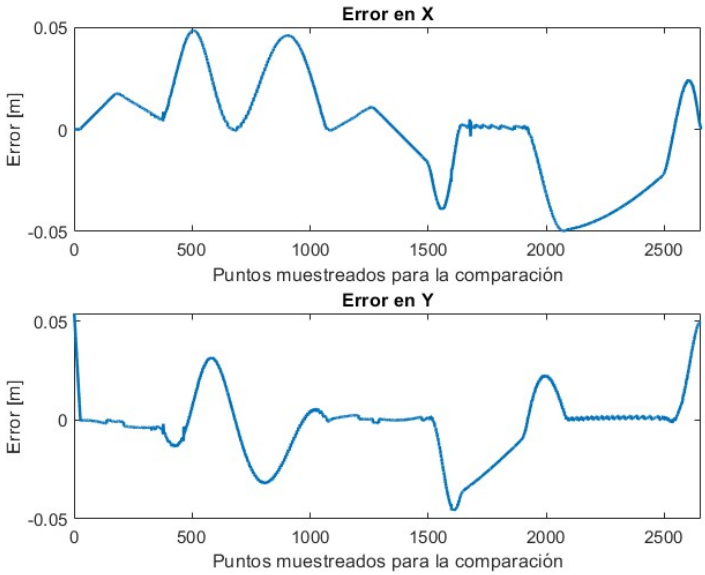

Figura 37: Errores de ejecución de controlador con primera trayectoria calculada y velocidad máxima de 0.7 m/s

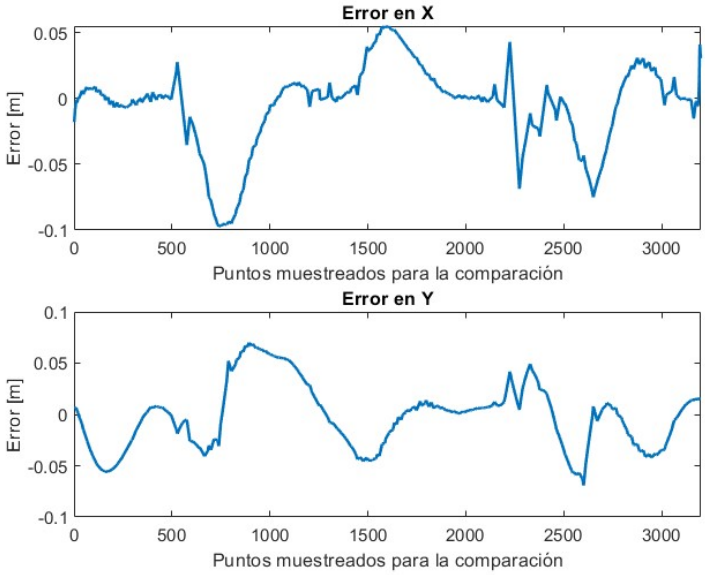

Figura 38: Errores de ejecución de controlador con segunda trayectoria calculada y velocidad máxima de 0.7 m/s

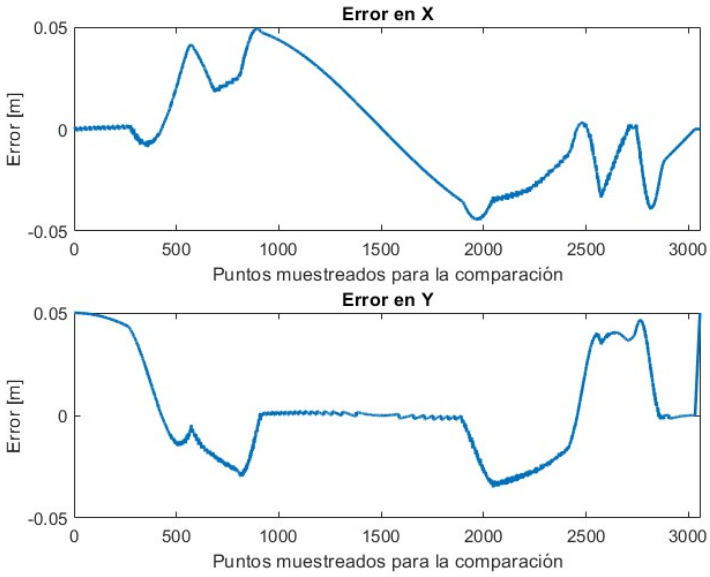

Figura 39: Errores de ejecución de controlador con tercera trayectoria calculada y velocidad máxima de 0.7 m/s

Comparando en el Cuadro 10 los errores obtenidos en el momento de haber simulado las trayectorias con la latencia de comunicación entre Matlab y el Pololu, contra los errores obtenidos en las pruebas de validación experimental, se puede observar que para la misma velocidad lineal máxima el comportamiento del controlador es totalmente distinto. Para el caso del controlador externo ejecutándose en Matlab los errores máximos alcanzaron valores superiores a los 40 cm, mientras que para el controlador ejecutándose en tiempo real como se implementó en el ESP32, los errores máximos no llegan a alcanzar los 10 cm. En cuanto a los errores promedios, para el caso del controlador en Matlab se tienen valores muy cercanos a los 20 cm, pero para el controlador en tiempo real se tienen valores menores a los 5 cm. En el caso del controlador en tiempo real, los errores máximos no llegan a siquiera alcanzar los valores de errores promedio del controlador en Matlab. Esta mejora en el rendimiento del controlador es significativa y justifica la implementación de todas las funcionalidades explicadas en el capítulo anterior.

	Controlador en Matlab con velocidad 0.7 m/s		Controlador en tiempo real con velocidad 0.7 m/s	
Trayectoria	Error máximo (m)	Error promedio (m)	Error máximo (m)	Error promedio (m)
Trayectoria 1	0.423	0.195	0.049	0.026
Trayectoria 2	0.442	0.203	0.082	0.042
Trayectoria 3	0.437	0.201	0.048	0.028

Cuadro 10: Comparación de errores en diferentes trayectorias en simulación y validación experimental

Para futuras mejoras en el rendimiento del sistema de control, una alternativa sería la implementación de un controlador predictivo que permita una planificación anticipada en la ejecución de las trayectorias planificadas. Actualmente, el controlador punto a punto sigue

un enfoque reactivo, donde solo toma en cuenta el próximo punto de la trayectoria sin prever el curso completo que el vehículo debe seguir. Este enfoque limita la capacidad del sistema para anticiparse a cambios en la curvatura o variaciones abruptas en el camino, especialmente en tramos complejos como curvas cerradas y giros pronunciados. Al incorporar un controlador predictivo el sistema podría estimar varios puntos consecutivos de la trayectoria, proporcionando al controlador una visión anticipada de la ruta. Esto permitiría ajustar de forma más eficiente los parámetros de velocidad y dirección, lo cual reduciría la acumulación de errores en las zonas de mayor complejidad. Este acercamiento puede justificarse también, buscando que el controlador tenga un mejor desempeño en zonas complejas, las cuales son representativas de la infraestructura vial guatemalteca.

Un controlador predictivo, como un controlador predictivo basado en modelo (MPC), podría aprovechar la información de la trayectoria completa para optimizar el desempeño en tiempo real, ajustando de forma dinámica la respuesta del agente. Con esta técnica el controlador no solo anticiparía el siguiente punto, sino que calcularía la mejor trayectoria posible dentro de un horizonte de tiempo determinado, minimizando errores y asegurando una transición más suave entre puntos. Implementar un controlador predictivo requeriría ajustes en las funcionalidades implementadas y en el procesamiento de datos en tiempo real, pero el beneficio potencial en términos de precisión haría que el sistema pueda enfrentarse de mejor manera a entornos urbanos complejos, ofreciendo una solución robusta que podría servir como base para futuros desarrollos en el control de vehículos autónomos a escala dentro del ecosistema del Robotat.

En este capítulo se evaluó el desempeño del controlador en la infraestructura diseñada, replicando trayectorias representativas de entornos urbanos y considerando las mejoras implementadas tras los resultados experimentales previos. Los análisis demostraron avances significativos en la precisión y estabilidad del sistema, especialmente en entornos con menos variabilidad dinámica. Sin embargo, se identificaron limitantes relacionadas con la optimización del controlador y la gestión de errores acumulativos en trayectorias con características complejas.

CAPÍTULO 10

Conclusiones

- Se diseñó y fabricó una infraestructura a escala, incluyendo zonas complejas como el redondel del Obelisco y la Avenida Reforma, permitiendo representar el entorno urbano de la Ciudad de Guatemala.
- El controlador PID con acercamiento exponencial demostró ser efectivo en simulación con velocidades lineales cercanas a 0.2 m/s, debido al tiempo que requiere la comunicación con el servidor de captura de movimiento, el cálculo del controlador en Matlab y el envío de las velocidades al Pololu 3Pi+.
- Se implementaron con éxito todas las funciones necesarias para migrar el controlador de Matlab al ESP32. Estas funciones incluyen la obtención de la pose del vehículo, la conversión de cuaterniones a ángulos de Euler, la fusión de sensores y la comunicación con múltiples servidores, permitiendo ejecutar el controlador en tiempo real sin depender de Matlab.
- El filtro de Kalman extendido presentó un error promedio del 0.02 % en comparación con el 11.52 % del filtro complementario, demostrando mayor precisión en la estimación de la pose del vehículo al combinar dos fuentes de datos con distinto periodo de muestreo, distinta exactitud de los datos proporcionados y distinto error y ruido en las mediciones.
- La validación del controlador punto a punto mostró un buen desempeño respaldado por errores que no superaron los 5 cm en las tres trayectorias ejecutadas, pero también mostró limitaciones en tramos con curvas cerradas y giros pronunciados. Estas limitaciones se deben a al naturaleza del controlador, el cual solo conoce el punto siguiente y no la trayectoria completa.

CAPÍTULO 11

Recomendaciones

- Se recomienda implementar una comunicación basada en UDP entre el servidor del Robotat y el ESP32 para la obtención de la pose del vehículo con el sistema de captura de movimiento, para reducir el tiempo del cálculo de la salida del controlador y que este corra en tiempo real.
- Para futuras investigaciones se recomienda explorar la implementación de un controlador predictivo basado en modelo (MPC) que permita al sistema anticiparse a varias posiciones futuras en la trayectoria. Esta mejora proporcionaría al controlador una visión más amplia de la ruta, permitiendo ajustar los parámetros de dirección y velocidad de forma más eficaz en tramos de alta complejidad, como las curvas cerradas. La implementación del MPC podría, no solo mejorar la precisión y estabilidad del vehículo en la infraestructura a escala, sino también servir de base para adaptaciones en entornos urbanos reales, ampliando el alcance y aplicabilidad del sistema desarrollado.

CAPÍTULO 12

Bibliografía

[1] C. Arribas, "Pruebas a escala de algoritmos básicos de visión de computadora y control para vehículos autónomos a escala," pág. 76, 2023.

[2] G. Fong, "Implementación de infraestructura a escala para la evaluación de algoritmos por visión de computador para vehículos autónomos," pág. 80, 2023.

[3] J. Hincapié, "Desarrollo de un sistema para la navegación autónoma de ambientes urbanos tipo intersección y su evaluación en la plataforma Duckietown," pág. 100, 2021.

[4] B. Vincke, S. Rodriguez y P. Aubert, "An Open-Source Scale Model Platform for Teaching Autonomous Vehicle Technologies," pág. 20, 2021.

[5] B. I. M. Consuegra, "Indicadores de accidentes de tránsito.," pág. 39, 2022.

[6] T. Duvall, "A new look at autonomous-vehicle infrastructure," pág. 6, 2019.

[7] Y. Wang, "Longitudinal and lateral control ofautonomous vehicles in multi-vehicle drivingenvironments," pág. 12, 2020.

[8] "SAE Levels of Driving Automation Refined for Clarity and International Audience." (), dirección: `https://www.sae.org/blog/sae-j3016-update`.

[9] D. Hernández, "Diseño, construcción y modelo dinámico de un robot móvil de tracción diferencial aplicado al seguimiento de trayectorias," pág. 7, 2017.

[10] M. Zea, "MT3005 - Lecture 11," 2024.

[11] Á. Valera, "Plataformas de Bajo Coste para la Realización de Trabajos Prácticos de Mecatrónica y Robótica," pág. 15, 2014.

[12] L. Ríos, "Modelo matemático para un robot móvil," pág. 7, 2008.

[13] M. Zea, "MT3005 - Lecture 12," 2024.

[14] "Ciinemática de Robots Móviles." (2020), dirección: `http://oramosp.epizy.com/teaching/201/fund-robotica/clases/8_Cinematica_Robots_Moviles.pdf?i=1`.

[15] "3pi+ 32U4 OLED Robot." (), dirección: `https://www.pololu.com/category/280/3pi-plus-32u4-oled-robot`.

[16] C. Salazar, "Sistemas de captura de movimiento," 2023.

[17] "PrimeX41." (), dirección: https://optitrack.com/cameras/primex-41/.

[18] "Redes Neuronales Convolucionales." (), dirección: https://fineproxy.org/wiki/convolutional-neural-networks-cnn/.

[19] "Introducción a Support Vector Machine (SVM)." (), dirección: https://la.mathworks.com/discovery/support-vector-machine.html.

[20] "Introducción a Support Vector Machine (SVM)." (), dirección: https://la.mathworks.com/help/fusion/ug/extended-kalman-filters.html.

[21] M. Zea, "MT3005 - Lecture 13," 2024.

[22] V. P. Álvarez, "El Filtro de Kalman," pág. 49, 2019.

[23] M. C. Munuera, "Filtro de Kalman y sus aplicaciones," pág. 66, 2018.

[24] R. M. Murray, "Optimization-Based Control," pág. 111, 2009.

[25] E. Tacconi, "Controladores Basados en Estrategias PID," pág. 54, 2005.

[26] M. Marhaban, "Review of visual odometry: types, approaches, challenges, and applications," pág. 26, 2016.

[27] D. Navarro, "Mejoras en la localización odométrica de un robot diferencial mediante la corrección de errores sistemáticos.," pág. 6, 2007.

[28] R. D., "Rotation Quaternions, and How to Use Them," 2015.

[29] MuniGuate, "Memoria de labores 2018 Dirección de Movilidad Urbana," Municipalida de Guatemala, Informe, 2018.

[30] SCP, "Memoria de labores 2022," Municipalida de Santa Catarina Pinula, Informe, 2022.

[31] "Historia de la Avenida Reforma en Guatemala." (), dirección: https://aprende.guatemala.com/cultura-guatemalteca/patrimonios/historia-avenida-reforma-guatemala/.

[32] P. Bormann, "Concise Binary Object Representation (CBOR)," pág. 54, 2013.

CAPÍTULO 13

Anexos

13.1. Repositorio de código de controlador y funcionalidades desarrolladas para el ESP32

El siguiente enlace dirige a un repositorio en GitHub con el código completo desarrollado para la implementación del controlador y las funcionalidades presentadas en los capítulos 7 y 8.

https://github.com/mon20054/Robotat-Pololu3Pi-Control