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

Carlos Fernando Arribas Valdés



UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería



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

Trabajo de graduación presentado por Carlos Fernando Arribas Valdés
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería



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

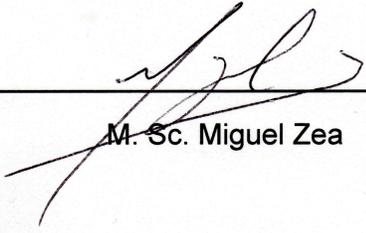
Trabajo de graduación presentado por Carlos Fernando Arribas Valdés
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

Vo.Bo.:

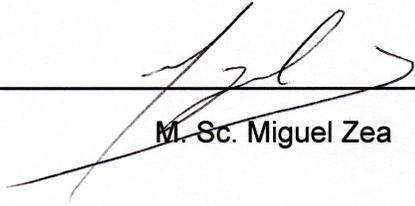
(f)



M. Sc. Miguel Zea

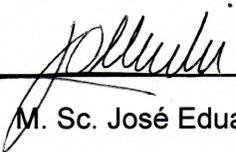
Tribunal Examinador:

(f)



M. Sc. Miguel Zea

(f)



M. Sc. José Eduardo Morales

(f)



Dr. Luis Alberto Rivera Estrada

Fecha de aprobación: Guatemala, 13 de enero de 2024

En este trabajo le agradezco a todas las personas que aportaron a su realización de diversas formas. A mis padres Fernando Arribas Menes y Araceli Valdés del Busto les agradezco su apoyo brindado durante los últimos cinco años, desde el apoyo económico que me han brindado a lo largo de la realización de la carrera como el apoyo emocional que me han dado en los tiempo difíciles que he sufrido a lo largo de la carrera. A mis compañeros y amigos de la universidad que me han apoyado cuando lo he necesitado en diversos tramos de la carrera y con los que he compartido grandes momentos que recordare en la posteridad. A los catedráticos de la universidad por aportar sus conocimiento y apoyar siempre que existieran dificultades a lo largo de sus cursos impartidos. Por último, un especial agradecimiento a mi asesor de tesis Miguel Zea por apoyarme y guiarme en la realización de este trabajo de graduación.

Prefacio	III
Lista de figuras	VII
Lista de cuadros	VIII
Resumen	IX
Abstract	X
1. Introducción	1
2. Antecedentes	2
2.1. Estudio del módulo “OpenMV Cam H7” para aplicaciones de visión artificial .	2
2.2. Desarrollo de un algoritmo de trayectoria para un robot seguidor de línea mediante visión e inteligencia artificial	3
2.3. Robot móvil autónomo para la siembra de semillas en el campo	3
2.4. Control method of three-wheel intelligent tracking logistics car based on OpenMV4	4
3. Justificación	6
4. Objetivos	7
4.1. Objetivo general	7
4.2. Objetivos específicos	7
5. Alcance	8
6. Marco teórico	9
6.1. OpenMV Cam H7	9
6.2. Pololu 3pi+	10
6.2.1. ESP32 WROOM 32D	11
6.3. Vehículos autónomos	12
6.4. Cinemática de vehículos	15

6.5.	Control longitudinal	19
6.6.	Control lateral	21
6.6.1.	Pure Pursuit	22
6.6.2.	Stanley	23
6.6.3.	MPC	24
7.	Desarrollo de un simulador para control longitudinal y lateral	25
7.1.	Metodología	25
7.2.	Selección de algoritmos de control	25
7.3.	Entorno de simulación de manejo en Matlab	27
7.3.1.	Creación y descripción del entorno	27
7.3.2.	Pruebas de simulación	29
7.4.	Adaptación del algoritmo de control lateral	31
8.	Implementación de los algoritmos de control en los Pololu 3pi+	35
8.1.	Metodología	35
8.2.	Validación de algoritmos de control en los Pololu 3pi+	35
8.2.1.	Archivos y funciones de control de ecosistema Robotat	35
8.2.2.	Creación del programa de control de los Pololu 3pi+	37
8.2.3.	Resultados del control empleando el ecosistema Robotat	38
9.	Implementación de algoritmos de visión de computadora	42
9.1.	Metodología	42
9.2.	Validación del algoritmo de seguimiento de línea en los Pololu 3pi+	42
9.2.1.	Creación del algoritmo de control de seguimiento de línea	42
9.2.2.	Resultados del seguimiento de línea en el ecosistema Robotat	45
9.3.	Validación de los algoritmos de detección de señales en los Pololu 3pi+	47
9.3.1.	Creación del algoritmo de control para la reacción a señales	47
9.3.2.	Resultados de la detección de señales en el ecosistema Robotat	52
10.	Pruebas preliminares de protocolos de comunicación	55
10.1.	Protocolo SPI	56
10.1.1.	SPI en el Esp32	56
10.1.2.	SPI en la OpenMV cam H7	57
10.2.	Protocolo I2C	57
10.2.1.	I2C en el Esp32	58
10.2.2.	I2C en la OpenMV cam H7	58
11.	Conclusiones	60
12.	Recomendaciones	61
13.	Bibliografía	62
14.	Anexos	64
14.1.	Repositorio de Github	64
14.2.	Enlaces a videos demostrativos y explicativos	64
15.	Glosario	65

Lista de figuras

1.	Robot seguidor de linea [1].	3
2.	Robot sembrador [2].	4
3.	Montacargas autónomo [3].	5
4.	Cámara OpenMV H7 [3].	10
5.	Robot móvil con ruedas Pololu 3pi+ [8].	11
6.	Mapa de características. [10].	14
7.	Mapa de ocupación [10].	14
8.	Mapa detallado [10].	15
9.	Rueda rodando en una superficie sin deslizamiento [11].	16
10.	Diagrama cinemático de una bicicleta tomado de la rueda trasera [10].	17
11.	Diagrama cinemático de una bicicleta tomado del centro de gravedad. [10].	19
12.	Sistema de control en lazo cerrado [10].	20
13.	Diagrama de control Pure Pursuit [10].	22
14.	Diagrama de control Stanley [10].	23
15.	Trayectoria D* controlada por PID.	30
16.	Trayectoria cuadrada controlada por PID.	30
17.	Trayectoria circular controlada por PID.	31
18.	Trayectoria DSD controlada por PID.	31
19.	Trayectoria D* controlada por Stanley-PID.	33
20.	Trayectoria cuadrada controlada por Stanley-PID.	33
21.	Trayectoria circular controlada por Stanley-PID.	34
22.	Trayectoria DSD controlada por Stanley-PID.	34
23.	Pololu 3pi+ con un marcador del sistema de cámaras Optitrack.	36
24.	Trayectoria D* realizado por el Pololu 3pi+.	39
25.	Trayectoria cuadrada realizado por el Pololu 3pi+.	39
26.	Trayectoria circular realizado por el Pololu 3pi+.	40
27.	Trayectoria DSD realizado por el Pololu 3pi+.	40
28.	Efecto de la tolerancia en el resultado del control.	41
29.	Funcionamiento del algoritmo de detección de línea dentro del IDE de OpenMV.	43
30.	Pololu 3pi+ montado con la OpenMV cam H7 para el seguimiento de líneas.	45

31.	Pista de prueba para el seguimiento de línea.	46
32.	Trayectorias realizadas por medio de seguimiento de línea a distintas velocidades.	46
33.	Señal de alto.	47
34.	Señal de peatón.	48
35.	Señal escolar.	48
36.	Semáforo.	49
37.	Diagrama de máquinas de estados finitos del funcionamiento de control de reacción a la detección de elementos de tránsito.	51
38.	Pololu 3pi+ montado con la OpenMV cam H7 para la detección de señales.	52
39.	Cambio de la posición durante la rutina de reacción a señal de alto.	53
40.	Cambio de la posición durante la rutina de reacción a señal de peatón/escolar.	54

Lista de cuadros

1.	Efecto de la modificación de las ganancias en PID.	21
2.	Constantes de los controladores PID y AE del entorno de simulación	29
3.	Constantes del controlador adaptado	32
4.	Constantes de los controladores utilizadas en el entorno Robotat.	38
5.	Constantes de los controladores utilizadas en el entorno Robotat.	41
6.	Constantes del PID utilizados para visual servoing.	45

El objetivo de este proyecto fue la creación de un vehículo autónomo a escala empleando los agentes robóticos Pololu 3pi+ como base y la infraestructura disponible en el Robotat de la Universidad del Valle de Guatemala. Para lograrlo, se investigaron los algoritmos de control Pure Pursuit y Stanley, los cuales son algoritmos de control empleados en la industria para así desarrollar un entorno de simulación por medio de Matlab para verificar el funcionamiento de los algoritmos de control. Posteriormente se implementaron los algoritmos de control en los Pololu 3pi+ empleando las herramientas disponibles en el entorno de Robotat para validar los algoritmos de control por medio de cuatro trayectorias definidas previamente, las cuales fueron un círculo, un cuadrado, una generada por medio del método D* y una generada por medio de la toolbox DSD. En donde el agente robótico fue capaz de seguir las trayectorias de forma satisfactoria desde una perspectiva cualitativa. Finalmente se implementaron los algoritmos de control por medio del ESP 32 y algoritmos de visión de computadora por medio del módulo OpenMV cam H7 para la identificación de señales de tránsito y el seguimiento de carriles.

The objective of this project was to create a scaled-down autonomous vehicle using Pololu 3pi+ robotic agents as a foundation and leveraging the infrastructure available at the Robotat of the Universidad del Valle de Guatemala. To achieve this, control algorithms Pure Pursuit and Stanley, commonly employed in the industry, were researched. Subsequently, a simulation environment was developed using Matlab to verify the functionality of these control algorithms. Next, the control algorithms were implemented on the Pololu 3pi+ agents using the tools available in the Robotat environment, and they were validated by conducting tests on four predefined trajectories. These trajectories included a circle, a square, one generated using the D* method, and another generated using the DSD toolbox. The robotic agent was able to successfully follow the trajectories from a qualitative perspective. Finally, the control algorithms were implemented using an ESP 32, and computer vision algorithms were incorporated using the OpenMV Cam H7 module for traffic signal identification and lane tracking.

Los vehículos autónomos son una de las tecnologías que más relevancia ha tomado en los últimos años alrededor del mundo. Pero debido a los altos costos y dificultades técnicas el desarrollo en Latinoamérica se ha visto ralentizado por la falta de financiamiento. Para el desarrollo de esta tecnología se ha empleado el uso de agentes robóticos como sustituto a los vehículos particulares, ejemplo de esto son los vehículos autónomos desarrollados en la Universidad Politécnica Salesiana [1] y [2] y el desarrollado en el Wuhan University of Technology [3].

Este trabajo de graduación se centra en el diseño y control de un vehículo autónomo a escala de bajo costo económico el cual pueda ser utilizado como base para futuras investigaciones dentro del ecosistema de Robotat en la Universidad del Valle de Guatemala. Para este vehículo se emplearon los Pololu 3pi+ como vehículo a escala y el módulo OpenMV cam H7 como sensor óptico. Se tiene como objetivos la investigación y desarrollo de algoritmos de control típicamente usados en la industria y su posterior integración en los agentes robóticos Pololu 3pi+ por medio de la placa de desarrollo ESP32. Una vez desarrollados e implementados los algoritmos de control se realiza una interconexión con el módulo OpenMV cam H7 para implementar algoritmos de visión de computadora que complementen a los algoritmos de control.

Esto se logrará por medio de un proceso de diseño de tres fases, la primera es la investigación y desarrollo de algoritmos de control empleados en vehículos autónomos a escala real que se muestra en el capítulo 7. Esto con el objetivo de seguir las tendencias en el desarrollo de plataformas de vehículos autónomos. Luego en el capítulo 8 se validan e implementan los algoritmos de control en los agentes robóticos para verificar que el trabajo realizado previamente pueda ser aplicado. Esto se realizó por medio de pruebas previas empleando el entorno de Robotat y el sistema de cámaras Optitrack para verificar los resultados de control. Una vez verificados los resultados de control se implementaron y validaron los algoritmos por medio de un Esp32 que controlara directamente los Pololu 3pi+. Por último, en el capítulo 9 se implementaron los algoritmos de visión de computadoras utilizando el sensor óptico de OpenMV cam H7 junto al Esp32 para validar su funcionamiento de forma independiente.

Los vehículos autónomos son una de las tecnologías que más se han desarrollado en los últimos años teniendo sus orígenes en los 80 y 90 en donde el primer vehículo autónomo se desarrolló en Alemania de la mano de Ernst Dickmans por parte de la Universidad de Bundesweher en Múnich durante 1995 [4]. Desde ese momento diversas instituciones educativas y empresas de automovilismo han aportado al desarrollo de vehículos autónomos tripulados, siendo otro ejemplo de la utilización de vehículos autónomos el Rover Perseverance de la NASA el cual ha ayudado en la exploración de la superficie de Marte desde 2020 hasta su “muerte” en el año 2021[5].

El alto costo del hardware y la dificultad de desarrollar software funcional para el manejo autónomo ha creado una alta barrera de entrada a los individuos e instituciones que quieran desarrollar plataformas autónomas. Por suerte en los últimos años empresas de hardware han desarrollado plataformas compatibles con vehículos autónomos como son el caso de los Pololu 3Pi+ que son pequeños robots móviles con ruedas programables y la OpenMV cam H7 la cual es una placa con un microcontrolador la cual es capaz de procesar imágenes a tiempo real de una forma sencilla, haciendo posible crear un vehículo autónomo a escala de bajo costo programada en Micro Python.

2.1. Estudio del módulo “OpenMV Cam H7” para aplicaciones de visión artificial

En [6] se muestra que, para aplicaciones de Machine Learning, el entrenamiento de esta cámara se realiza por medio de un tipo de archivo llamado “*cascade*”, el cual contiene toda la información requerida sobre el objeto a identificar, siendo estos una gran cantidad de imágenes positivas y negativas. Estos positivos son una serie de imágenes que muestran el objeto con distintos fondos para que el algoritmo de detección de la cámara puede detectarlo mientras que los negativos son imágenes sin ninguna correlación con el objeto para que esta aprenda a diferenciar el objeto del resto del entorno. Este procedimiento se emplea por medio

del software Cascade Trainer GUI diseñado para las herramientas de OpenCV, dando un archivo .xml el cual es compatible con Python más no con el Micro Python utilizado en la Cam H7. Para solucionar este problema se utiliza Python 2.7 en Linux para poder convertir el archivo obtenido en un formato .cascade compatible en el IDE de OpenMV.

2.2. Desarrollo de un algoritmo de trayectoria para un robot seguidor de línea mediante visión e inteligencia artificial

En [1] se presenta la creación del robot mostrado en la Figura 1, equipado con la OpenMV Cam H7 el cual fue capaz de seguir trayectorias definidas por medio del reconocimiento por imágenes realizado en el año 2021. Esto fue posible mediante la utilización de redes neuronales, las cuales son una rama de la inteligencia artificial que simula el cerebro humano con sus neuronas y conexiones. La red neuronal se entrenó por medio del software Edge Impulse al ingresar referencias al programa. Al utilizar la red neuronal junto con la cámara se logró seguir la trayectoria en el cien por ciento de las ocasiones, ya que la red neuronal toma capturas por medio de la cámara y compara con lo aprendido previamente para indicarle al robot si debe de ir adelante, izquierda, derecha o un giro con radio para cualquier dirección.

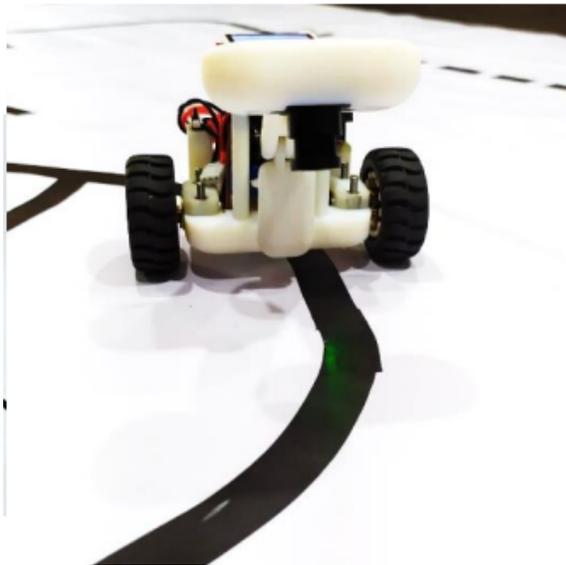


Figura 1: Robot seguidor de línea [1].

2.3. Robot móvil autónomo para la siembra de semillas en el campo

Este proyecto muestra la creación de un vehículo autónomo el cual tiene como objetivo la siembra de semillas en las áreas rurales de Ecuador utilizando una Raspberry Pico y la OpenMV Cam H7 como puede observarse en la Figura 2. Este funciona por medio de una red neuronal convolucional la cual fue entrenada por medio de la arquitectura Mobile Net

la cual se configuro para la detección de surcos en la tierra, el entrenamiento se realizó por medio de la aplicación Edge Impulse por medio de imágenes positivas y negativas. Gracias a este entrenamiento se logró que el robot fuese capaz de detectar las zonas en el campo en donde se debían de colocar semillas por medio de un sistema de dispensación con un mecanismo de dosificación controlado por la Raspberry Pico [2].



Figura 2: Robot sembrador [2].

2.4. Control method of three-wheel intelligent tracking logistics car based on OpenMV4

En [3] se puede observar un montacargas autónomo como se muestra en la Figura 3 diseñado por la Wuhan University of Technology, la cual se equipó con un chip STM32 F407 ZGT6 para el control de movimiento y una cámara OpenMV4 para la detección de códigos QR colocados en los objetos. Este fue capaz de posicionarse independientemente y realizar la búsqueda de productos dentro del almacén para poder movilizarlos a las zonas deseadas. Al detectar el código QR el vehículo clasifica los elementos en el programa y comienza el posicionamiento para poder tomar la mercancía y colocarla en 3 distintos niveles dentro del montacargas. Una vez terminada esta tarea por medio de un sensor de escala de gris el montacargas sigue una trayectoria hasta llegar a la zona de descarga y dejar la mercancía en el lugar especificado.



Figura 3: Montacargas autónomo [3].

En la actualidad los vehículos autónomos son una de las tecnologías de mayor crecimiento en los países desarrollados gracias a empresas como Tesla y Google. A pesar de su creciente investigación y desarrollo, en Latinoamérica no han tenido un alto impacto debido a la dificultad técnica que representa en programación, control y diseño a lo que hay que sumarle el costo de desarrollo. Por estas razones la realización de un vehículo autónomo a escala es una gran oportunidad para validar algoritmos de visión y control avanzados de vehículos autónomos reales, aportando a investigaciones futuras que se realizarán dentro de la Universidad del Valle de Guatemala, permitiendo desarrollar esta tecnología en escalas mayores.

4.1. Objetivo general

Emplear los agentes Pololu 3Pi+ dentro del ecosistema Robotat para implementar y evaluar algoritmos de visión y control para vehículos autónomos a escala.

4.2. Objetivos específicos

- Especificar el escalamiento a emplear para que los resultados con los agentes robóticos se apliquen a vehículos de escala real.
- Validar algoritmos de visión de computadora para vehículos autónomos en el módulo OpenMV Cam H7.
- Implementar y validar algoritmos de control para vehículos autónomos utilizando los agentes Pololu 3pi+.

Este trabajo se basó en el desarrollo de un vehículo autónomo a escala empleando un agente robótico como base y un sensor óptico para la detección del ambiente. El objetivo principal fue la implementación y validación de algoritmos de visión y control para vehículos autónomos a escala. El proyecto se centró en tres secciones principales: la primera es el escalamiento y validación de los algoritmos de control usados en vehículos autónomos de escala real, luego se implementaron los algoritmos en el agente robótico para validar su funcionamiento, por último, se implementaron algoritmos de visión de computadora en los agentes robóticos para validar su funcionamiento individual.

En este proyecto se tomaron en consideración el control Pure Pursuit y Stanley como algoritmos de control, ya que estos son dos algoritmos diseñados para vehículos autónomos a escala utilizados en la industria. En los algoritmos de visión se hicieron uso de los diseñados por el estudiante Gabriel Fong en su proyecto “Implementación de infraestructura a escala para la evaluación de algoritmos por visión de computadora para vehículos autónomos” [7] los cuales incluyen dos programas: el primero es un algoritmo de detección de las líneas de los carriles y el segundo es un algoritmo de detección de semáforos y de señales de tránsito. Para la validación de algoritmos de visión de computadora se emplean pruebas independientes de los elementos de tránsito y seguimiento de línea, debido a que el objetivo es la validación del funcionamiento de los algoritmos y no su integración. Por lo que se realizaron pruebas individuales de cada elemento de tránsito y pruebas en un segmento corto de carretera para la tarea de seguimiento de línea. Esto último debido a las limitaciones del sistema de cámaras Optitrack disponible en el ecosistema de Robotat, el cual presenta puntos ciegos en donde las cámaras no son capaces de seguir al agente robótico y recabar la información requerida para la correcta validación del algoritmo de visión de computadora.

El alcance de este trabajo se vio limitado debido al tiempo de uso del laboratorio CIT-116 en donde se encuentra el Robotat, ya que este al ser un laboratorio de uso general para clases de la universidad limita el horario disponible para trabajar en el mismo. En adición a esto los agentes robóticos y sensores ópticos serán los Pololu 3pi+ y la OpenMV cam H7 respectivamente, ya que estos son los materiales disponibles en el Robotat. Por último, los segmentos de carreteras y elementos de tránsito empleados para validar los algoritmos de visión de computadoras fueron diseñados y fabricados por el estudiante Gabriel Fong.

6.1. OpenMV Cam H7

La cámara desarrollada por OpenMV, como se observa en la Figura 4, permite la implementación de algoritmos de visión de forma sencilla gracias a su módulo de visión artificial. Sus características principales son:

- Sensor de imagen OmniVision OV7727 de 1/3 pulg.
- Procesador ARM Cortex-M7 de 32 bits con un reloj de 216 MHz.
- Puerto microSD.
- Conexión a la computadora por medio de MicroUSB.
- Protocolos de comunicación:
 - UART.
 - I2C.
 - SPI.
 - CAN.
- Compatibilidad con el OpenMV IDE para ser programado en el lenguaje MicroPython.
- Bibliotecas de visión predefinidas:
 - Diferenciación de fotogramas.
 - Seguimiento de color.
 - Seguimiento de marcadores.
 - Detección de rostros.
 - Seguimiento de ojos.
 - Detección de personas.

- Flujo óptico.
- Decodificación de códigos de barras.
- Reconocimientos de etiquetas QR.
- Detección de líneas, círculos y rectángulos.
- Toma de fotos y captura de vídeos.

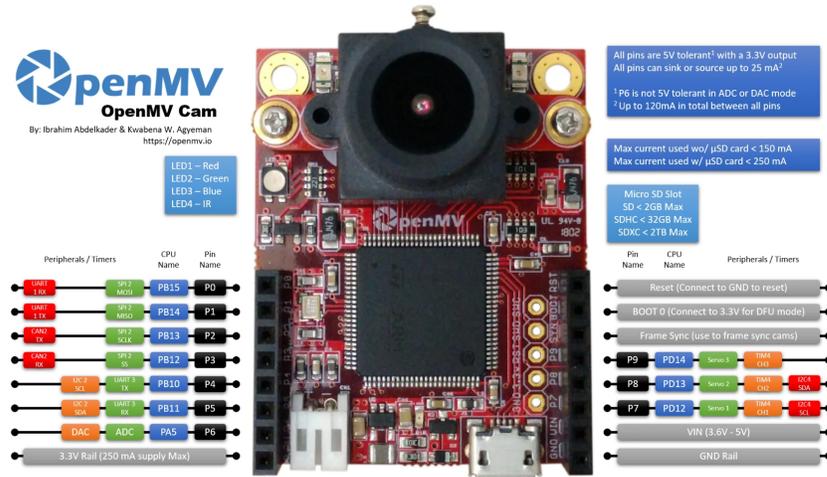


Figura 4: Cámara OpenMV H7 [3].

6.2. Pololu 3pi+

Los Pololu 3pi+, mostrados en la Figura 5, son robots móviles con ruedas desarrollados por Pololu Robotics and Electronics. Estos son de bajo bajo costo y de fácil programación gracias a su compatibilidad con el IDE de Arduino, sus características principales son[8]:

- Diámetro del robot: 9.8 cm.
- Altura: 3.9 cm.
- Peso: 83 g.
- Diámetro de las ruedas: 3.2 cm.
- Velocidad máxima: 1.5 m/s.
- Batería: 4 baterías AAA ubicadas en la parte inferior.
- IMU (Unidad de medición inercial): acelerómetro de 3 ejes, magnetómetro y giroscopio.
- Sensores de choque frontal en el parachoques y de seguimiento de línea.

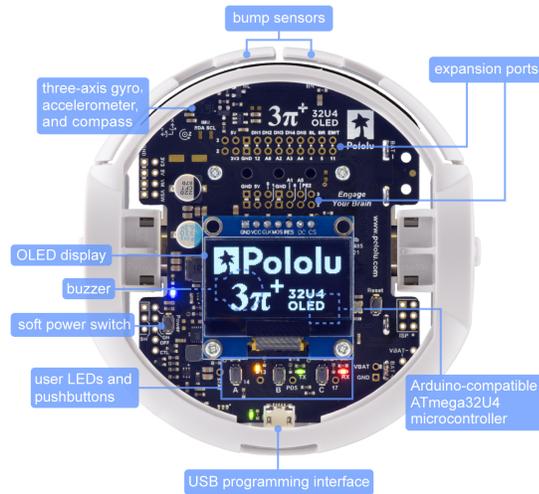


Figura 5: Robot móvil con ruedas Pololu 3pi+ [8].

6.2.1. ESP32 WROOM 32D

El ESP32 WROOM 32D y 32U es un módulo de desarrollo basado en el microcontrolador ESP32 de Espressif Systems, ofreciendo una amplia gama de funcionalidades y flexibilidad gracias a su compatibilidad con el IDE de Arduino. Algunas de sus características son [9]:

- Microcontrolador ESP32-D0WD el cual contiene un doble núcleo Xtensa LX6 de 32 bits con velocidades de entre 80 a 240 MHz.
- Conectividad inalámbrica por medio de un módulo WiFi con protocolo 802.11 b/g/n.
- Cristal integrado de 40 MHz.
- Memoria flash de 4 mb para código y datos.
- 512 kb de memoria SRAM.
- Múltiples protocolos de comunicación como:
 - UART.
 - SPI.
 - I2C.
 - I2S.
 - CAN.
- 38 pines de GPIO con funciones como PWM, ADC, I/O, etc.
- Alimentación con voltajes de entre 2.2 a 3.6V.
- Puede ser programado por medio del entorno ESP-IDF de Espressif Systems o por medio del IDE de Arduino.

6.3. Vehículos autónomos

Un vehículo autónomo se define como aquel vehículo con sistemas informáticos que es capaz de imitar las capacidades de conducción humanas en cierto grado. Estos, para su correcto funcionamiento, deben de cumplir con una serie de tareas de manejo, las cuales le van a permitir al vehículo cumplir con sus funciones de la forma más segura y efectiva posible. Estas tareas comienzan con la percepción o, en otras palabras, la recolección de datos necesaria para planificar los movimientos del vehículo como lo son el movimiento del propio vehículo, el entorno en el que se encuentra como las señales, peatones, las baquetas y líneas de división que indican los límites de las calles, etc. Luego, el vehículo debe de planificar el movimiento por medio de información previa como los son mapas y los datos recabados al momento en las tareas de percepción. Por último se debe de controlar el vehículo para que este cumpla con la ruta indicada en la tarea de control, lo cual se logra por medio de dos tipos de control. El primero es el control de la velocidad o también llamado control longitudinal, el cual regula qué tanto frena o acelera el vehículo. El segundo es el control del volante o control lateral, el cual se encargará de la dirección del vehículo. Ambos controles se logran por medio de sistemas de control clásicos y modernos y son diseñados bajo un marco de diseño llamado ODD, el cual significa dominio de diseño operativo y regula el entorno bajo el cual el vehículo puede funcionar. Este toma en cuenta elementos como las calles, clima o la velocidad de operación del vehículo[10].

En la actualidad existen 5 niveles de autonomía para vehículos autónomos los cuales son[10]:

- Al primer nivel se le denomina como asistencia de manejo, el cual sólo es capaz de asistir en el control longitudinal o en el lateral, no al mismo tiempo. En este nivel el piloto realiza la mayor parte de las tareas de control del vehículo mientras que este le facilita el cumplimiento de una de las dos tareas principales. El ejemplo más común de este nivel es el control de cruce, el cual permite al conductor mantener una velocidad constante sin necesidad de presionar el pedal de aceleración y freno.
- Al segundo se le denomina como automatización de conducción parcial y mezcla ambos controles ayudando al piloto al regular la velocidad como a realizar correcciones en la dirección del vehículo.
- El tercer nivel, llamado automatización de conducción condicional, suma todo lo anterior y le agrega la función de OEDR (detección de objetos y eventos) en donde el vehículo podrá reaccionar a elementos y eventos repentinos que sucedan durante el viaje como, por ejemplo, el frenado de emergencia si un vehículo al frente frena repentinamente.
- El cuarto nivel, llamado conducción de alta automatización, es el primero que permite al usuario concentrarse en otras tareas en lugar de manejar aunque en ocasiones puede solicitar al piloto tomar el volante para realizar maniobras. Este incluye un ODD limitado a situaciones idóneas en donde no se requiera maniobrar en lugares poco habitables para un vehículo u operar durante situaciones en donde los datos de los sensores se puedan ver afectados debido a las características de los mismos.
- El último nivel denominado como automatización de conducción completa es el que

permite al usuario completo enfoque en otras tareas que no sean manejar gracias a su ODD, el cual toma en consideración climas adversos, baja calidad de las calles a utilizar y velocidades más altas de operación.

Para realizar las mediciones de la percepción correctamente se emplea el uso de diversos sensores alrededor del vehículo dependiendo del ODD para el que se este diseñando, ya que en la actualidad existen múltiples configuraciones de un mismo tipo de sensor los cuales varían sus características como el alcance o condiciones de uso. Las dos categorías principales de sensores son los exteroceptivos y los propioceptivos. La primera conforma aquellos sensores que capturan datos externos al vehículo, mientras que los propioceptivos son aquellos que capturan datos internos del vehículo. Algunos de los sensores más comunes son[10]:

- Exteroceptivos:
 - Cámara.
 - Stereo: combinación de dos cámaras entrelazando sus campos de visión.
 - LIDAR: sensor de alcance y detección de luz.
 - RADAR: detección de radio y alcance.
 - Ultrasónico.
- Propioceptivos:
 - GNSS: sistema de navegación global satelital.
 - IMU: unidad interna de medición.
 - Odómetro de rueda: sensor de velocidad de ruedas.

La generación de trayectorias es un proceso complejo el cual toma en consideración decisiones a largo, corto y plazo inmediato. El primero sirve para generar la trayectoria del punto de partida al deseado, la planeación a corto plazo por otro lado se apoya en el punto actual que se esta navegando de la ruta generada y toma decisiones como por ejemplo ¿Se puede cruzar al otro carril? o ¿Se puede incorporar en esta intersección?. Por último, las decisiones inmediatas se refieren a decisiones como, por ejemplo, si se debe acelerar o frenar o si el vehículo es capaz de mantenerse en el carril en la situación actual del vehículo. Para lograr estas estas decisiones de forma correcta se hace uso de mapas y datos recopilados por los sensores, algunos de estos mapas son[10]:

- Mapa de características (Figura 6):
 - Utilizado en decisiones a corto plazo.
 - Brindado por los datos recolectados constantemente por el LIDAR.

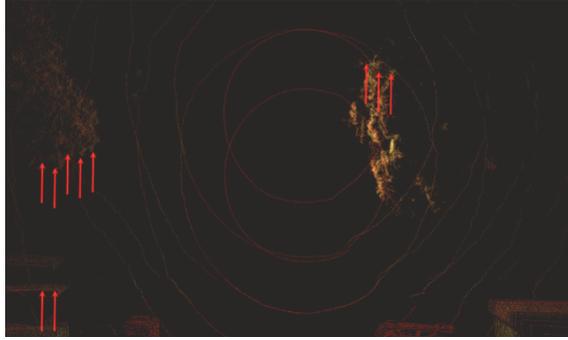


Figura 6: Mapa de características. [10].

- Mapa de cuadrícula de ocupación (Figura 7):
 - Existen dos variantes principales, el binario y el probabilístico. El primero le asigna valores de 1 y 0 en función de si está ocupado el espacio o libre mientras que el probabilístico le asigna un valor entre 0 y 1 dependiendo de qué tan probable es que el espacio esté ocupado.
 - Se utiliza para generar trayectorias.

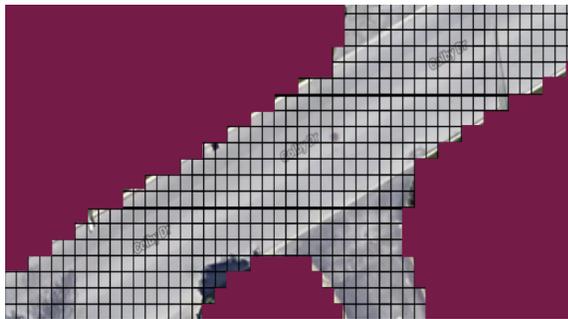


Figura 7: Mapa de ocupación [10].

- Mapa detallado de las calles (Figura 8):
 - Es un mapa que muestra de forma detallada los elementos de una calle y se utiliza para generar las trayectorias a seguir por el vehículo.
 - Se generan de 3 formas:
 - Online: generado completamente por computadora con datos conocidos del entorno.
 - Offline: poco utilizado debido a su gran consumo de recursos para poder generarlo.
 - Creado Offline y actualizado Online: este se crea con vehículos especiales como podrían ser los automóviles de Google los cuales recolectan datos detallados de las calles y actualizan en línea los mapas con los datos recolectados de forma manual.

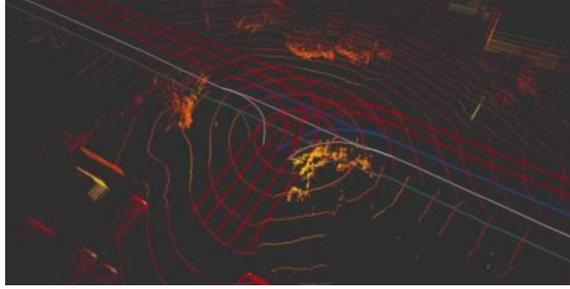


Figura 8: Mapa detallado [10].

- En el caso de las decisiones inmediatas se hace uso de las cámaras y sensores como los ultrasónicos o radares para poder verificar en tiempo real las condiciones del vehículo.

Por último se encuentran las tareas de control, las cuales traducen los datos de las dos tareas previas para que el vehículo pueda cumplir con sus funciones. Para esto se emplea una arquitectura de control de vehículos autónomos, la cual va integrar las tareas de manejo del vehículo para que puedan funcionar en conjunto. La típica arquitectura de control de vehículos autónomos se divide en 4 secciones principales[10]:

- La primera es la percepción de la calle y sus alrededores (el ambiente), la cual es capturada por medio de los diversos sensores y genera referencias de entrada que se podrá utilizar en el control.
- La segunda se divide en dos partes, la generación de trayectorias y de un perfil de velocidad definido comúnmente como perfil de manejo. Ambos se generan por medio del proceso de planificación de movimiento siendo que para el control longitudinal se emplea el punto objetivo al que se desea llegar, la aceleración y desaceleración del sistema. Mientras que para el control lateral se emplea el error en la posición con respecto a la trayectoria y el error en el ángulo de dirección del vehículo.
- La tercera fase de la arquitectura de control son los controladores longitudinal y lateral, los cuales tienen el propósito de seguir el plan de la forma más precisa posible y minimizar el error entre la ruta y velocidad actual con las deseadas.
- La última parte de la arquitectura de control son los actuadores que toman las señales de los controladores y las transfieren a la vida real.

6.4. Cinemática de vehículos

La cinemática/dinámica de un vehículo autónomo es de suma importancia para el funcionamiento del mismo, ya que su correcta obtención permite crear un vehículo con mejor control, mayor seguridad y un rendimiento más eficiente a la hora de realizar sus funciones. Esto debido a que se podrá diseñar el vehículo de una forma más precisa, ya que las simulaciones y algoritmos de control que se emplearán podrán acercarse más a la realidad física del vehículo. En el caso de los Pololu 3pi+, debido a sus características pueden aproximarse

como un tipo especial de robots móviles con ruedas llamadas unicyclos. “El robot móvil con ruedas más simple es aquel con una sola rueda rodante vertical, o monociclo” [11].

Para comenzar con la obtención de la cinemática del unicyclo se define r como el radio de la rueda. Se definirá la configuración de esta rueda como $q = (\phi, x, y, \theta)$, en donde (x, y) es el punto de contacto, ϕ es el ángulo de dirección y θ es el ángulo de rotación como se observa en la Figura 9. Por lo tanto, las ecuaciones de la cinemática de movimiento son (1) [11].

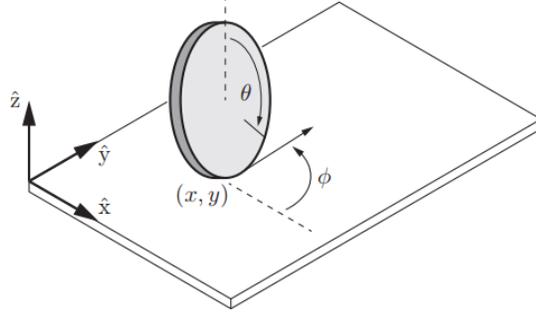


Figura 9: Rueda rodando en una superficie sin deslizamiento [11].

$$\dot{q} = \begin{bmatrix} \dot{\Phi} \\ \dot{x} \\ \dot{y} \\ \dot{\Theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ r \cos \Phi & 0 \\ r \sin \Phi & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = G(q)u = g_1(q)u_1 + g_2(q)u_2. \quad (1)$$

En esta ecuación, $u = (u_1, u_2)$ son las entradas del sistema, siendo u_1 la velocidad de la rueda y u_2 la velocidad de giro. Estos controles están sujetos a restricciones teniendo cada uno un valor máximo y mínimo a los que pueden llegar. Las funciones con valores vectoriales $g_i(q) \in \mathbb{R}^4$ son las columnas de la matriz $G(q)$, y se les llama campos de vectores tangentes o control sobre q asociado con los controles $u_i = 1$. Pero, debido a que el ángulo de rotación de la rueda θ no es de importancia, se puede quitar la cuarta fila de (1) para obtener la ecuación (2) [11].

$$\dot{q} = \begin{bmatrix} \dot{\Phi} \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ r \cos \Phi & 0 \\ r \sin \Phi & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \quad (2)$$

Debido a que el robot no posee un radio r ya que no esta formado por una llanta el modelo se puede aproximar a la ecuación (3). Esta deja de tomar en consideración el radio de la llanta del unicyclo y modela al actor como un punto en donde θ es el ángulo ϕ de la ecuación (2) [12].

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}. \quad (3)$$

Con base en esta cinemática se puede modelar la arquitectura del robot diferencial o *Differential-Drive Robot* en inglés. Esta es la arquitectura más básica de los robots móviles con ruedas. Estos robots están compuestas por dos llantas de radio r colocadas a una distancia l del centro del robot. Debido a esto no existe una velocidad lineal o angular, en cambio se utilizan dos velocidades angulares ω_l y ω_r que representan la velocidad de las dos llantas del robot. Debido a las diferencias entre esta arquitectura con respecto a la cinemática del uniclo existen las ecuaciones (4) y (5) que representan la velocidad lineal y angular del robot. Al despejar para obtener las velocidades de las llantas se obtienen las ecuaciones (6) y (7) que permiten convertir ambas velocidades a las velocidades requeridas por el robot diferencial para funcionar [12].

$$v = \frac{r(\omega_r + \omega_l)}{2}, \quad (4)$$

$$\omega = \frac{r(\omega_r - \omega_l)}{2l}, \quad (5)$$

$$\omega_r = \frac{v + l\omega}{r}, \quad (6)$$

$$\omega_l = \frac{v - l\omega}{r}. \quad (7)$$

En el caso de vehículos a escala real un modelo cinemático que describe de forma sencilla la cinemática de un vehículo de cuatro llantas es el modelo de bicicleta. Este toma las llantas delanteras y traseras y las une en un par colocados en el centro de un vehículo, como se observa en la Figura 10, en donde la llanta trasera permanece recta en todas las situaciones y es la llanta delantera la que rota para girar el vehículo alrededor del radio de giro [10].

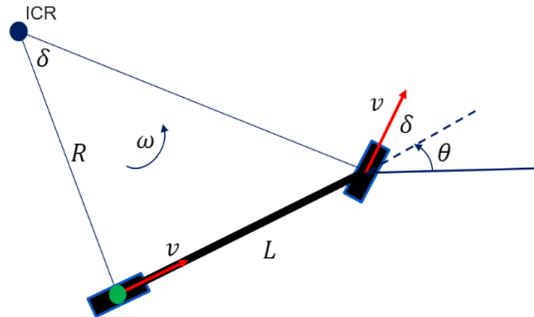


Figura 10: Diagrama cinemático de una bicicleta tomado de la rueda trasera [10].

Dependiendo del punto seleccionado la cinemática de la bicicleta tendrá variaciones dependiendo del punto de referencia tomado para obtener la cinemática. Se comenzará por la rueda trasera tomando como referencia la Figura 10, en donde L es la distancia entre ruedas y θ es el ángulo de dirección del vehículo con respecto a la horizontal. Las entradas del sistema serán distintas a las del monociclo ya que se depende de la velocidad v y de el ángulo de giro de las llantas δ . La velocidad de cada rueda apunta a la misma dirección que esta, lo cual es una suposición llamada la condición de deslizamiento de la nariz. Debido a esta condición se tiene que la velocidad angular está dada por (8), en donde R es el radio del punto de construcción de la cinemática hacia el ICR o centro instantáneo de rotación. Luego se obtiene el ángulo de rotación por medio de (9) y al reemplazar R en (8) se obtiene la ecuación (10) que describe la velocidad angular del vehículo, que junto con las ecuaciones (11) y (12) se obtiene la dinámica del sistema tomando como referencia la rueda trasera [10].

$$\dot{\theta} = \omega = \frac{v}{R}, \quad (8)$$

$$\tan \delta = \frac{L}{R}, \quad (9)$$

$$\dot{\theta} = \omega = \frac{v}{R} = \frac{v \tan \delta}{L}, \quad (10)$$

$$\dot{x}_r = v \cos \theta, \quad (11)$$

$$\dot{y}_r = v \sin \theta. \quad (12)$$

Si ahora se cambia el punto de referencia hacia la rueda delantera se tendrán ecuaciones similares pero se tomará en cuenta el ángulo de dirección de la llanta δ para las velocidades \dot{x}_f y \dot{y}_f . Como se puede observar en (13), (14) y (15) las ecuaciones que describen la cinemática de la bicicleta son similares a las descritas anteriormente [10].

$$\dot{\theta} = \frac{v \sin \delta}{L}, \quad (13)$$

$$\dot{x}_f = v \cos (\theta + \delta), \quad (14)$$

$$\dot{y}_f = v \sin (\theta + \delta). \quad (15)$$

Por último se tomará como referencia el centro de gravedad del sistema, como se muestra en la Figura 11, en donde a diferencia a los modelos anteriores se agrega un ángulo β llamado ángulo de deslizamiento, ya que el centro de gravedad no girará al mismo ritmo que las ruedas delanteras. Las ecuaciones (16), (17) y (18) se muestran las ecuaciones cinemáticas que describen el movimiento de la bicicleta desde el centro de gravedad [10].

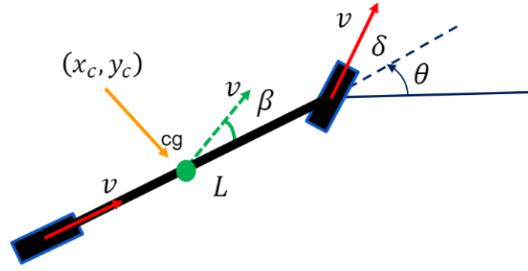


Figura 11: Diagrama cinemático de una bicicleta tomado del centro de gravedad. [10].

$$\dot{\theta} = \frac{v \cos \beta \tan \delta}{L}, \quad (16)$$

$$\dot{x}_c = v \cos(\theta + \beta), \quad (17)$$

$$\dot{y}_c = v \sin(\theta + \beta). \quad (18)$$

En caso que las condiciones de uso del vehículo no sean ideales, despreciando elementos como la resistencia del aire o el deslizamiento de las llantas, se debe de emplear la cinemática junto a la dinámica del sistema. La dinámica del sistema tomará en consideración variables como la inercia del vehículo, la fuerza que realizan las llantas, la fricción con el suelo o la resistencia aerodinámica. Esto tendrá un efecto en el sistema de control del vehículo ya que la respuesta del vehículo será distinta que utilizando la cinemática del sistema. Debido a las dimensiones mayores del mismo se deberá frenar progresivamente ya que será imposible frenar de forma repentina. Para la dinámica, esta deberá tomar en consideración dos situaciones, la primera es la longitudinal la cual va a indicar la aceleración y frenado del vehículo. La segunda es el control lateral el cual ayudará a prevenir que las fuerzas que actúan sobre el vehículo provoquen deslizamiento durante las curvas [10].

6.5. Control longitudinal

El control longitudinal de un vehículo autónomo implica el control de la velocidad a lo largo de una trayectoria definida. Esto se puede lograr de forma efectiva con métodos de control clásicos como lo es el PID. El controlador proporcional-integral-derivativo, o PID por sus siglas, es uno de los métodos de control más utilizados en la actualidad. Este se diseña conforme a la ecuación (19) la cual posee tres términos que dependen del error entre la referencia y la salida. El PID posee tres constantes K conocidas como las ganancias de control proporcional K_p , integral K_i y derivativa K_d , las cuales se varían para obtener el resultado deseado para el sistema [13].

$$u(t) = K_p e(t) + K_i \int e(t) dt + K_d \dot{e}(t). \quad (19)$$

En la Figura 12 se observa el típico diagrama de control en lazo cerrado en donde $G(s)$ se refiere a la planta a controlar, en este caso la velocidad del sistema, $G_c(s)$ son las ganancias del controlador y $H(s)$ es la salida medida por medio de sensores. Aunque en ocasiones se puede omitir el uso de sensores, es indispensable cerrar el lazo en sistemas de control. En la ecuación (20) se observa el valor $U(s)$ que entra a la planta en donde el error, que es la resta entre la referencia con la retroalimentación sensada en la salida, se multiplica por las ganancias del controlador. Este valor entra a la planta dando así una salida $Y(s)$ como se observa en (21) [13].

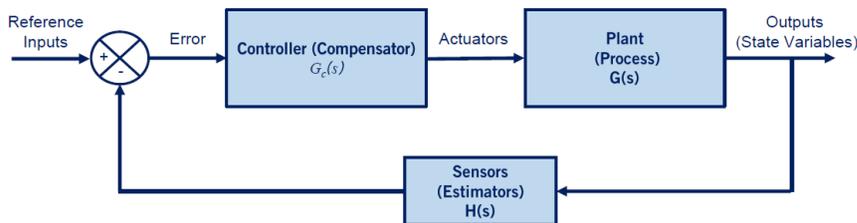


Figura 12: Sistema de control en lazo cerrado [10].

$$U(s) = G_c(s)E(s), \quad (20)$$

$$Y(s) = G(s)U(s). \quad (21)$$

Para diseñar el controlador de forma correcta se emplea el Cuadro 1, el cual muestra los efectos de aumentar o disminuir cada una de las ganancias. Una combinación de las tres variables suele dar la mejor respuesta pero en ocasiones se puede obviar la ganancia integral o derivativa ya que no siempre aportan de forma significativa al control. Para diseñar el PID se toma en cuenta la cinemática o dinámica del sistema para conocer así la respuesta de la planta a controlar, una vez conociendo esto se debe comenzar a modificar las ganancias del PID en función de 4 parámetros principales [13]:

- Tiempo de subida t_r .
- Sobre-oscilamiento (*overshoot*) M_p .
- Tiempo de asentamiento t_s .
- Error en estado estacionario e_{ss} .

Es recomendable trabajar el controlador en el dominio de la frecuencia como se muestra en la ecuación (22) ya que esto permitirá implementar de forma más sencilla el controlador en programación. Además que por medio de herramientas como Matlab y su PID Tuner se podrá visualizar los cambios que el controlador tiene en su planta.

$$U(s) = \frac{K_d s^2 + K_p s + K_i}{s} E(s). \quad (22)$$

Respuesta en lazo cerrado	Tiempo de subida	Sobre oscilación	Tiempo de asentamiento	Error en estado estacionario
Aumento K_p	Disminuye	Aumenta	Cambio pequeño	Disminuye
Aumento K_i	Disminuye	Aumenta	Aumenta	Elimina
Aumento K_d	Cambio pequeño	Disminuye	Disminuye	Cambio pequeño

Cuadro 1: Efecto de la modificación de las ganancias en PID.

Una variante al controlador PID es el PID con acercamiento exponencial, este controlador esta diseñado para tareas en donde se requiera un acercamiento no lineal al punto de equilibrio. Este variante del control PID mejora la respuesta en cambios bruscos de las condiciones del sistema y disminuye la no convergencia en estas situaciones. En la ecuación (23) se puede observar el controlador por acercamiento exponencial en donde e_p es el error de posición, v_o es la velocidad lineal máxima que puede alcanzar el actor y α es un coeficiente de ajuste [13].

$$v = -\frac{v_o(1 - e^{-\alpha e_p^2})}{e_p} e_p. \quad (23)$$

6.6. Control lateral

Para el control lateral la salida objetivo será el ángulo de dirección del vehículo. Para este controlador existen dos grupos principales de controladores que se pueden utilizar, los geométricos y los dinámicos. Los geométricos hacen uso de la cinemática del sistema por lo que son más sencillos de implementar, pero si en las condiciones de uso del vehículo se tienen en consideración variables externas como la fricción o fuerzas externas el control puede llegar a fallar. El segundo grupo de controladores tienen una mayor dificultad de emplearse pero son más fiables en condiciones de uso inciertas. Los principales controladores de cada grupo son [10]:

- Geométricos
 - Pure Pursuit (carrot following).
 - Stanley.
- Dinámicos
 - MPC (Control de modelo predictivo).

6.6.1. Pure Pursuit

El Pure Pursuit es un controlador que tomará la trayectoria a seguir y, por medio de la diferencia entre el ángulo de la trayectoria y el ángulo de dirección, ajustará el ángulo de rotación de las llantas para que el vehículo se dirija donde la trayectoria le indica. Para vehículos a escala real se usa la cinemática de la bicicleta tomada desde la llanta trasera, como se observa en la Figura 13, en donde el ángulo α es el que indica la dirección a la que se debe dirigir el vehículo y l_d es la distancia entre el punto en el que se encuentra actualmente el vehículo y la posición deseada en la trayectoria. Si se aplica la ley de senos se obtiene que la ecuación que describe la curvatura es (24), utilizando esta curvatura se obtiene el ángulo de rotación de la llanta δ dando como resultado la ecuación (25).

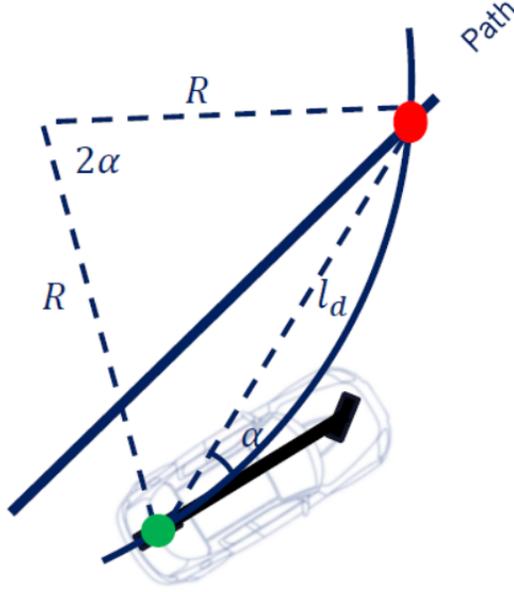


Figura 13: Diagrama de control Pure Pursuit [10].

$$k = \frac{1}{R} = \frac{2 \sin \alpha}{l_d}, \quad (24)$$

$$\delta = \tan^{-1}(kL) = \tan^{-1}\left(\frac{2L \sin \alpha}{l_d}\right). \quad (25)$$

Sabiendo que la relación del error de posición total es (26), se reemplazo en la ecuación (24) obtendrá como resultado una curvatura que sólo depende de l_d como se observa en (27). Se desea que la curvatura dependa de la velocidad, por lo que se tiene la relación (28) en donde v_f es la velocidad hacia el frente que el vehículo está experimentando y K_{pp} es una ganancia dada por el controlador Pure Pursuit, dando como resultado final la ecuación (29) la cual dará como resultado el ángulo de giro de la llanta dependiendo de la trayectoria a seguir [10].

$$\sin \alpha = \frac{e}{l_d}, \quad (26)$$

$$k = \frac{2}{l_d^2} e, \quad (27)$$

$$l_d = K_{pp} v_f, \quad (28)$$

$$\delta = \tan^{-1} \left(\frac{2L \sin \alpha}{K_{pp} v_f} \right). \quad (29)$$

6.6.2. Stanley

Este controlador, en comparación con el controlador Pure Pursuit, usa la cinemática de la bicicleta tomando como referencia la rueda delantera. Como se puede observar en la Figura 14 el cambiar la referencia a la llanta delantera cambia el ángulo de error a ψ , ya que ahora el ángulo de rotación de la llanta δ estará tomado bajo la misma línea de referencia. Este ángulo de rotación estará dado por la ecuación (30), en donde se emplea una ganancia k y se deberá de acotar el ángulo δ para no exceder los límites. Al incluir el error ψ en la ecuación se obtiene como resultado (31) [10].

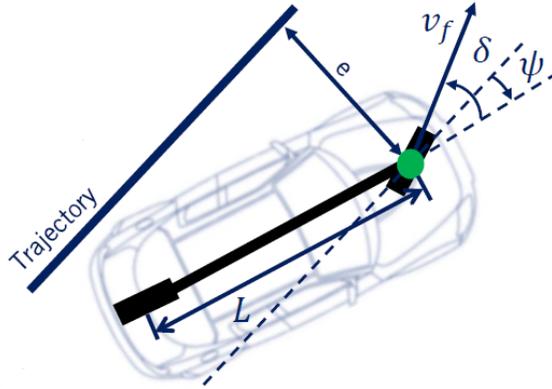


Figura 14: Diagrama de control Stanley [10].

$$\delta = \tan^{-1} \left(\frac{ke(t)}{v_f(t)} \right), \quad (30)$$

$$\delta = \psi(t) + \tan^{-1} \left(\frac{ke(t)}{v_f(t)} \right). \quad (31)$$

6.6.3. MPC

El MPC, o control de modelo predictivo, es un método de control avanzado utilizado en sistemas dinámicos para el control de vehículos autónomos. Este en comparación a controles como el PID utiliza modelos matemáticos y de optimización para tomar decisiones de control, ya que como su nombre lo dice es capaz de predecir el comportamiento del sistema y actuar en consecuencia[10].

Este método de control es cada vez más usado gracias al aumento de las capacidades del hardware de computación, ya que este realiza una gran cantidad de operaciones matemáticas para poder lograr su cometido. En comparación con los controladores Pure Pursuit y Stanley su desempeño y rango de operación es mayor gracias a sus capacidades predictivas. Por lo que mientras mayores especificaciones se coloquen en el ODD del vehículo y más se acerque este a la escala real se recomienda explorar este método de control [10].

Desarrollo de un simulador para control longitudinal y lateral

7.1. Metodología

Para el diseño de los algoritmos de control se comenzó con la investigación de los diversos algoritmos que se emplean comúnmente en vehículos autónomos. Una vez se tuvo conocimiento de los distintos algoritmos se categorizaron los algoritmos en las dos categorías de control requeridas: el control de velocidad o longitudinal y el control de dirección o lateral. Luego de organizar los algoritmos se realizó una comparativa entre las distintas opciones para seleccionar la opción que mejor se adecue al proyecto.

Ya teniendo los algoritmos seleccionados se prosiguió a diseñar un entorno de simulación de manejo en Matlab para comprobar el funcionamiento de los algoritmos en distintas trayectorias. Una vez diseñado el entorno se realizaron cuatro pruebas empleando algoritmos de control comúnmente utilizados para robots móviles con ruedas para verificar el funcionamiento del simulador. Al tener el simulador funcional se prosiguió a realizar una adaptación del control lateral seleccionado, esto ya que los algoritmos de control lateral típicamente utilizados en vehículos autónomos están diseñados para emplear cinemáticas distintas a la que modela a los Pololu 3pi+. Finalizado la adaptación del algoritmo se realizaron pruebas con las mismas trayectorias utilizadas previamente para así realizar una comparativa con los resultados previamente obtenidos y así verificar si los algoritmos de control son aplicables a los Pololu 3pi+.

7.2. Selección de algoritmos de control

Debido a la variedad de algoritmos de control longitudinal y lateral que se toman en consideración se debe de realizar una comparativa para utilizar aquel que se adapte de

mejor forma a las necesidades del proyecto. Como consideraciones iniciales se sabe que estos algoritmos se emplearán en un Pololu 3pi+, el cual se basa en la cinemática del unicycle debido a su configuración de dos llantas. Adicionalmente se tomó como referencia un vehículo sedán para obtener los parámetros que se emplean en la cinemática de la bicicleta. Este tipo de vehículo tiene en promedio un largo entre llantas $l = 2.677m$ que se obtuvo del documento *Promedio_largo_llantas.xls* que toma el largo entre llantas de diversos vehículos sedan comerciales y un ángulo de giro máximo de unos treinta y cinco grados.

1. Control longitudinal

Para el control de la velocidad del vehículo se tomaron en consideración el control por medio de PID y por medio de el acercamiento exponencial. Ambos algoritmos de control son capaces de funcionar bajo la cinemática del unicycle y de la bicicleta sin inconvenientes pero estos poseen diferencias en diseño y funcionamiento a considerar.

El PID es un algoritmo de control con una gran versatilidad ya que puede usarse en diversas aplicaciones, pero debido a esto el ajuste de las constantes debe ser preciso ya que de lo contrario no funcionará correctamente. Este ajuste preciso provoca que el controlador sea susceptible a cambios en las condiciones del sistema, ya que si estas cambian significativamente el control puede llegar a dejar de funcionar.

Por otro lado el acercamiento exponencial es más específico en las aplicaciones en donde se puede emplear. Esto debido a que este supone que el elemento a control se debe de acercar de forma exponencial. Pero, debido a esta característica, su diseño es más sencillo que su contra parte y es menos susceptible a cambios en las condiciones del sistema.

Teniendo en consideración que las condiciones de uso son ideales se puede emplear ambos algoritmos de control. Pero considerando que se puede replicar este proyecto en distintas escalas y condiciones se elige como algoritmo de control longitudinal el acercamiento exponencial, ya que este presenta mejores cualidades para adaptarse a cambios repentinos de condiciones que un PID además de tener un diseño más simple que el antes mencionado.

2. Control lateral

En el caso del control lateral se tomaron como opciones el control de Stanley y el Pure Pursuit. Ambos controladores cumplen con el objetivo de control pero a diferencia de los algoritmos expuestos en el control longitudinal estos poseen diferencias de diseño entre sí.

El control Stanley fue diseñado para funcionar bajo la cinemática de la bicicleta con referencia en la llanta delantera. Este es capaz de seguir con gran precisión las trayectorias pero requiere de un mayor ajuste para trabajar correctamente. Adicionalmente requiere de una gran cantidad de información para trabajar en óptimas condiciones. Una ventaja de este algoritmo es que debido a que esta compuesto en dos fragmentos, como se observa en (31), puede emplearse sólo una de estas.

Por otro lado, el control Pure Pursuit se diseñó para trabajar bajo la cinemática de la bicicleta pero con referencia en la llanta trasera. Este sólo esta compuesto por un componente por lo que se debe de emplear el control completo. Su diseño es más simple que el de Stanley y requiere menos información para funcionar. Pero debido a esto posee una menor precisión a la hora de seguir la trayectoria.

Tomando en cuenta estas consideraciones el control de Stanley es el elegido para ser utilizado en este proyecto. Esto por dos razones importantes, la primera es que el control de Stanley puede ser utilizado si una trayectoria ya que como se observa en (31) esta estructurado por dos secciones independientes de control mientras que Pure Pursuit (29) depende totalmente de que exista una trayectoria. Por lo tanto con Stanley se pueden realizar pruebas empleando únicamente algoritmos de visión de computadora. La segunda razón es la diferencia en las cinemáticas de ambos controladores, ya que a pesar de que ambos utilizan la bicicleta como cinemática de referencia Stanley se basa en la llanta delantera y Pure Pursuit en la trasera. Esto es importante ya que el algoritmo de control se debe emplear en un Pololu 3pi+ que se basa en la cinemática del uniclo, siendo la cinemática que utiliza Stanley la más cercana. Esto se debe a que se puede modelar la llanta delantera como si fuese un Pololu 3pi+.

7.3. Entorno de simulación de manejo en Matlab

7.3.1. Creación y descripción del entorno

Para la creación del entorno de simulación de manejo se decidió utilizar el software de Matlab, ya que este es un lenguaje de alto nivel con altas capacidades de procesamiento, el cual es capaz de realizar cálculos matemáticos complejos, manejo de matrices, creación de gráficas y posee una variedad de Toolboxes que añaden una variedad de funciones al programa. Como base se utilizaron dos programas, el primero es un programa base de simulación de trayectorias desarrollado en la Universidad del Valle de Guatemala y el segundo es un código de procesamiento de trayectorias diseñado por *Mathworks* [14].

Para la generación de la trayectorias a seguir se utilizaron tres métodos, la primera es la generación de figuras geométricas en Matlab, la segunda fue por medio del método de planeación D* y por último empleando una Toolbox disponible en Matlab llamada *Driving Scenario Designer* (DSD). Esta Toolbox permite generar y simular trayectorias para distintos actores como vehículos, bicicletas o peatones y realizar una simulación que muestra el movimiento que tendrá el actor a lo largo de la trayectoria. Esta herramienta de Matlab no se puede usar como un entorno de simulación capaz de validar algoritmos de control ya que no se pueden modificar los métodos ni parámetros de control a utilizar, por lo que no se sabe el método de control que se emplea para realizar la simulación.

Para generar una trayectoria en DSD primero se debe de colocar una calle con el botón *Add Road* y presionando el botón izquierdo del ratón en la cuadrícula blanca se coloca en el área de trabajo. Una vez finalizada la colocación de la calle se presiona *Enter* para finalizar la edición. Una vez se tiene la calle se debe de colocar el actor por medio del botón *Add Actor*, el cual desplegará una lista con los distintos actores en donde se selecciona el actor deseado y presionando en el área de trabajo se coloca en la calle. Por último, para generar la trayectoria se tiene dos formas, la primera es presionando el botón derecho del ratón sobre el actor y seleccionando *Add Forward Waypoints* o *Add Reverse Waypoints* y seguir el mismo proceso que al colocar la calle. La segunda forma es presionando los comandos *Ctrl + F* para una trayectoria hacia adelante y *Ctrl + R* para la trayectoria hacia atrás. Una vez teniendo toda la trayectoria generada se presiona en *Save* para generar un archivo *.mat*

con la trayectoria.

Una vez teniendo la trayectoria generada se creó un archivo de Matlab llamado *Drive_Simulator.m*. El cual se encuentra en el repositorio 14.1 dentro de la carpeta *Entorno_de_simulacion*. Este archivo se divide en 3 bloques principales de código: la inicialización de datos y procesamiento de la trayectoria, aplicación del algoritmo de control y generación de la simulación:

1. Inicialización de datos

En este bloque del código se preparan las condiciones requeridas para iniciar la simulación ya que se definen parámetros como el tiempo, período de muestreo, constantes de los controladores, largo del vehículo, condiciones iniciales, etc. Se define la cinemática a utilizar por el actor y se carga la trayectoria. El programa posee cuatro trayectorias guardadas en *.mat* para cargar al cambiar el valor de la variable *selectraj*, estas son una trayectoria generada por el método de planificación D^* , un cuadrado, un círculo y una trayectoria generada por medio de DSD.

Debido que no se puede utilizar directamente la trayectoria generada por DSD, se emplea una serie de operaciones matemáticas para transformar una pequeña cantidad de puntos, la cual depende de la cantidad de puntos colocados a la hora de generar la trayectoria, a una trayectoria completa a seguir. Primero se calcula la distancia entre cada par de puntos (x, y) utilizando la función *squareform* que devuelve una matriz $M \times M$ en donde M es el número de puntos en la trayectoria original. Este resultado se guarda en una matriz columna llamada *distancesteps* donde se tendrá la distancia entre cada punto. Una vez teniendo estas distancias se realizan dos operaciones de suma, la primera es una suma simple para obtener la distancia total de la trayectoria y la segunda es una suma acumulativa, la cual da el valor desde el punto inicial al que esta cada punto.

Al tener estas distancias se realiza una cuadrícula de la distancia por medio de la función *linspace* para así generar un vector fila de tamaño N , donde este será el número a iteraciones a realizar por el simulador. Al tener esta cuadrícula se procede a realizar dos interpolaciones en una dimensión empleando la función *interp1*, obteniendo así las trayectorias en X y en Y para el actuador. Por último, para tener un mejor resultado se emplea la función *smooth* para realizar un suavizado entre puntos para obtener una trayectoria con mejor cohesión entre puntos.

2. Aplicación de algoritmo de control

Para aplicar los algoritmos de control a la trayectoria se emplea un ciclo *for* con duración de $N-1$ iteraciones. Dentro se estará calculando constantemente el error de posición restando el punto de la trayectoria de la posición actual dada por la cinemática. Para el error de orientación se calcula la tangente del error de posición utilizando *atan2(Ey, Ex)* que da la orientación deseada en radianes y se resta el ángulo θ obtenido de la cinemática. Una vez calculados los errores se aplican los algoritmos de control empleando los errores previamente calculados. Estos errores cambiarán mediante un *if* al inicio del ciclo *for*, el cual cambiará en cada iteración el punto objetivo.

Para actualizar el estado de la trayectoria controlada se emplea una discretización por medio del método de Runge-Kutta cuyo resultado es una matriz *xi* con la trayectoria reali-

zada por el actuador. Se guarda la trayectoria controlada en una variable XI y las variables de estado en U .

3. Generación de simulación

Por último, se generan dos gráficas, la primera es una gráfica que muestra al actor aplicando el control a la trayectoria y la segunda muestra los cambios de las variables de estado a lo largo del tiempo. Para generar la simulación primero se crea una figura y se extraen las variables del sistema que fueron discretizadas previamente en XI . Luego, se grafica la trayectoria deseada por medio de un *plot* y la real se grafica y se guarda en una variable *trajplot*. Una vez generadas las trayectorias se crea el actor por medio de una matriz llamada BV , a este actor se le aplica una matriz de rotación para que pueda rotar a medida que se mueva en la simulación y se guarda en IV . Por último, para generar el actor en la gráfica se emplea un *fill* el cual sera guardado en una variable *bodyplot*.

Una vez realizados los pasos previos se crea un ciclo *for* que va de $2 : N - 1$ para así poder animar la trayectoria completa y ver el comportamiento en tiempo real. En este ciclo se generara el actor y la trayectoria paso por paso a partir de las variables *trajplot* y *bodyplot* generadas previamente, ya que estas variables poseen la información requerida sobre el comportamiento del actor.

7.3.2. Pruebas de simulación

Como actor del simulador se emplea un Pololu 3pi+ bajo la cinemática del unicycle, el cual es controlado por medio de acercamiento exponencial (AE) longitudinalmente y por un PID lateralmente. Se realizaron pruebas con cuatro trayectorias distintas en donde el actor se colocó en $(0,0)$ como punto inicial. Las constantes utilizadas para cada prueba están listadas en el Cuadro 2.

Constante	Valor
KpO	11
KiO	0.001
KdO	2
Vo (D*)	17.5
α (D*)	3
Vo (Cuadrada)	27.5
α (Cuadrada)	6
Vo (Circular)	8
α (Circular)	0.5
Vo (DSD)	65
α (DSD)	6

Cuadro 2: Constantes de los controladores PID y AE del entorno de simulación

Como se puede observar en las Figuras 15, 16, 17 y 18, se presentan las cuatro trayectorias descritas previamente en donde cada figura presenta la trayectoria a controlar en color azul y la trayectoria realizada por el actor en color naranja. En todos los casos el actor fue capaz de seguir de forma satisfactoria las trayectorias. En el caso de la trayectoria D* en la Figura 15,

la circular en la Figura 17 y la generada por el DSD en la Figura 18 se obtuvieron los mejores resultados, ya que el actor fue capaz de seguir la trayectoria sin ningún error significativo.

Por otro lado, la trayectoria cuadrada en la Figura 16 presentó diferencias en las esquinas del cuadrado. Este error no es significativo ya que el actor cumple el objetivo de seguir la trayectoria y a su vez es un comportamiento esperado ya que un vehículo real no es capaz de realizar cambios de dirección tan cerrados como las esquinas de un cuadrado.

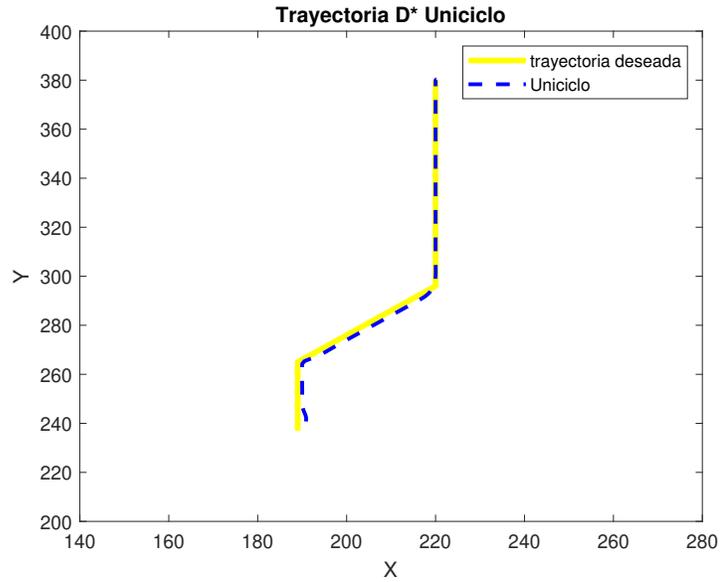


Figura 15: Trayectoria D* controlada por PID.

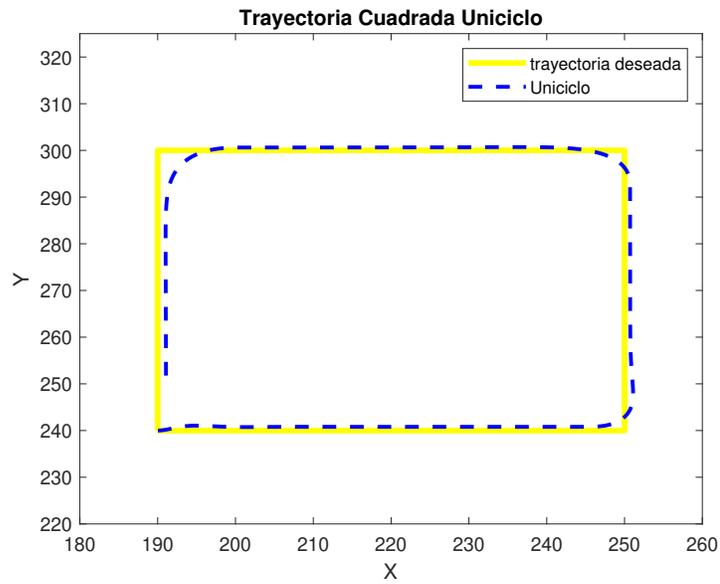


Figura 16: Trayectoria cuadrada controlada por PID.

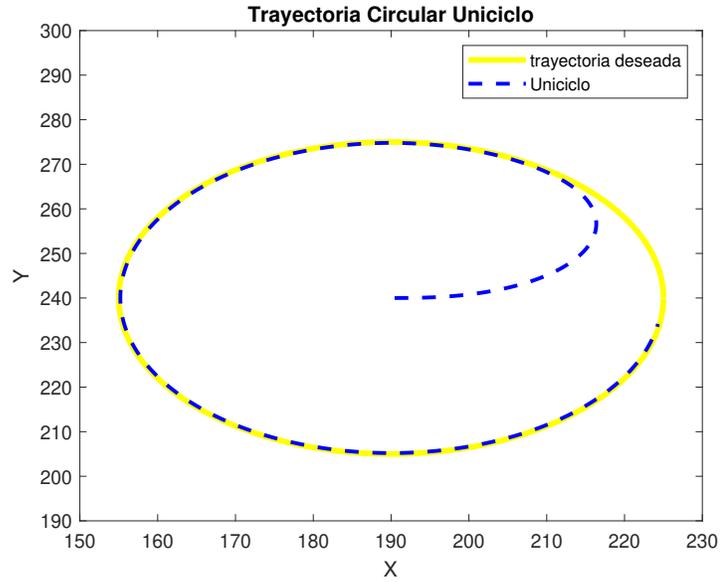


Figura 17: Trayectoria circular controlada por PID.

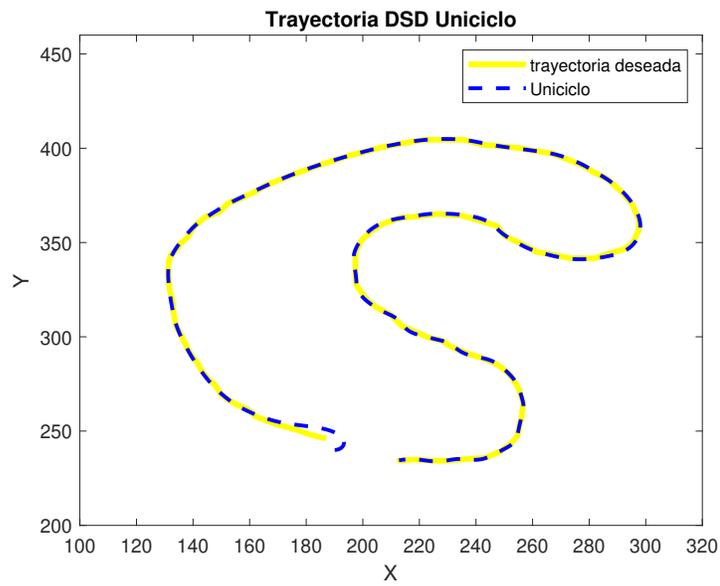


Figura 18: Trayectoria DSD controlada por PID.

7.4. Adaptación del algoritmo de control lateral

Debido que el algoritmo de control de Stanley no está diseñado bajo la misma cinemática en la cual se basan los Pololu 3pi+ no se puede emplear directamente. Esta diferencia radica en los métodos de orientación de ambas cinemáticas más que en la diferencia de sus variables de estado, ya que una bicicleta se orienta gracias a un ángulo δ que representa el ángulo de

dirección de la llanta delantera, mientras que un Pololu 3pi+ se orienta por medio de una velocidad angular.

Para solucionar este problema se decidió emplear una segunda capa de control por medio de un PID. Esto agrega complejidad de diseño y aumenta el consumo computacional requerido para controlar la orientación del los Pololu 3pi+. A pesar de esto fue la opción más eficiente de adaptar el controlador de Stanley, ya que es uno de los métodos de control más utilizados en robots móviles con ruedas y la forma más sencilla de transformar el ángulo que da como resultado Stanley en una velocidad angular.

Para esta segunda capa de control el PID tomó como métrica de error el ángulo dado por el controlador de Stanley. Debido a las características de los PID este ángulo puede ser convertido a diversas unidades como torque, fuerza, corriente, voltaje y velocidad angular, por lo que el resultado del PID sí puede ser empleado bajo la cinemática del unicycle y en los Pololu 3pi+. Para verificar si el algoritmo de control fue efectivamente adaptado se empleó el entorno de simulación utilizando las mismas trayectorias que se muestran en la sección 7.3.2. Las constantes empleadas por los controladores laterales se muestran en el Cuadro 3. Para el control longitudinal se emplean las constantes que se muestran en el Cuadro 2.

Constante	Valor
K (Stanley)	0.5
P (PID Lateral)	11
I (PID Lateral)	0.001
D (PID Lateral)	2

Cuadro 3: Constantes del controlador adaptado

Como se puede observar en las Figuras 19, 20, 21 y 22 se empleó el control de doble capa en las trayectorias. En el caso de las trayectorias: cuadrada, circular y generada por medio de D^* , los resultados no tuvieron diferencias significativas con los resultados obtenidos utilizando un control lateral por medio de PID. Por otro lado la trayectoria generada por medio del DSD tuvo diferencias perceptibles con respecto al control con PID. A pesar de estas diferencias los resultados del control de doble capa fueron satisfactorios, ya que el error presentado no es significativo y el controlador cumple con el objetivo principal de seguir una trayectoria definida.

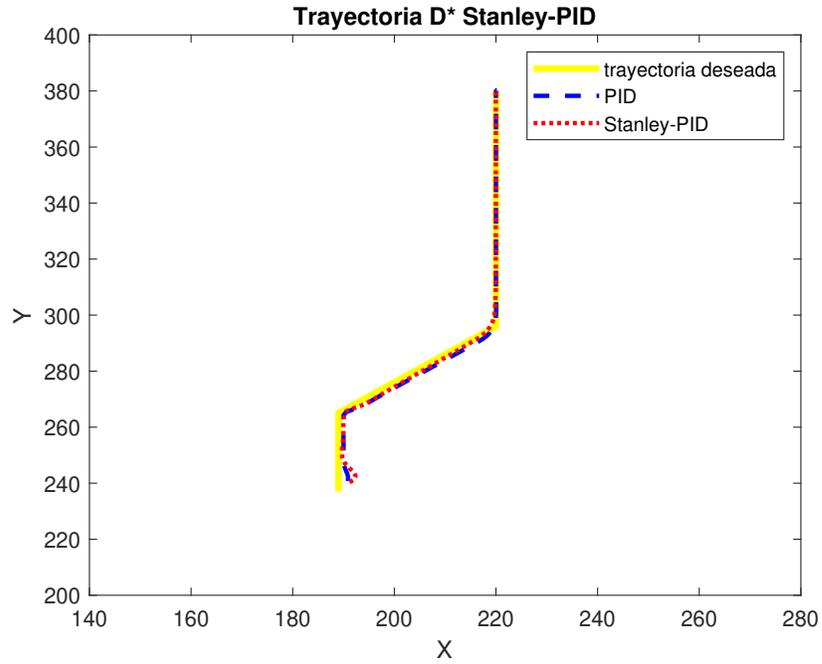


Figura 19: Trayectoria D* controlada por Stanley-PID.

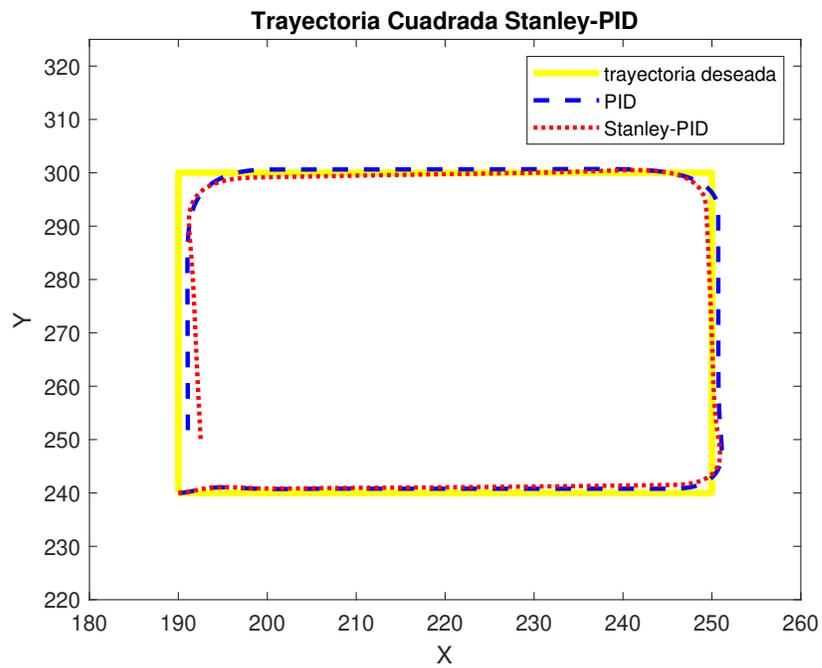


Figura 20: Trayectoria cuadrada controlada por Stanley-PID.

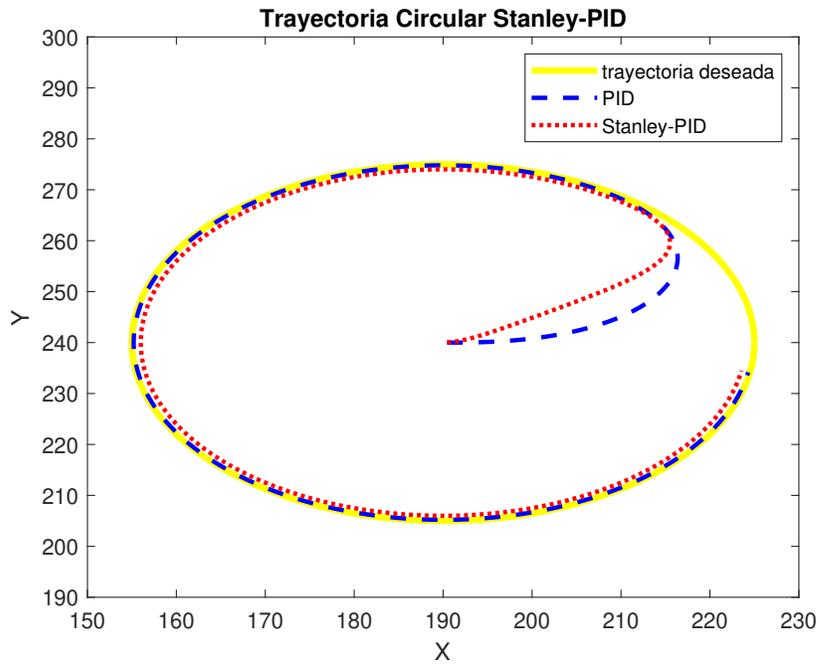


Figura 21: Trayectoria circular controlada por Stanley-PID.

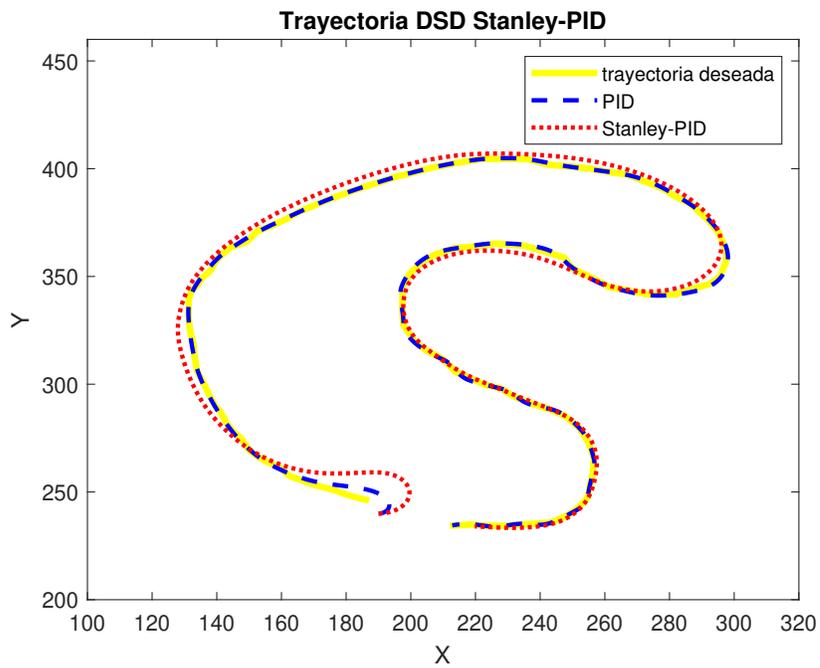


Figura 22: Trayectoria DSD controlada por Stanley-PID.

Implementación de los algoritmos de control en los Pololu 3pi+

8.1. Metodología

Para implementar los algoritmos de control diseñados previamente se creó un programa en Matlab empleando las funciones disponibles en el entorno de Robotat para obtener la pose y controlar al Pololu 3pi+. Una vez diseñado el programa se utilizan cuatro tipos de trayectorias distintas para ver el funcionamiento del controlador en distintas situaciones. Se emplearon una trayectoria cuadrada, circular, generada por D* y generada por el DSD. Los resultados de estas trayectorias son guardados en un archivo *.mat* para ser comparados con los resultados obtenidos en el Capítulo 7 y así validar el funcionamiento de los controladores.

8.2. Validación de algoritmos de control en los Pololu 3pi+

8.2.1. Archivos y funciones de control de ecosistema Robotat

Para validar los algoritmos de control en los Pololu 3pi+ se empleó el entorno de Robotat [15]. Este entorno está compuesto por dos elementos principales, el primero es un sistema de seis cámaras OptiTrack. Este sistema de cámaras es capaz, por medio de marcadores ópticos, detectar en tiempo real la Pose del marcador indicado. Cada marcador tiene una configuración única que permite al sistema de cámaras diferenciar entre todos los marcadores dentro del ecosistema. En la Figura 23 se puede observar uno de estos marcadores montado sobre el Pololu 3pi+, lo que permite obtener la pose del robot de forma sencilla. El segundo elemento es una red WiFi a la cual se puede acceder para obtener los datos del sistema de cámaras. Esto por medio de funciones existentes del entorno de Robotat, las cuales están diseñadas para ser utilizadas en Matlab.



Figura 23: Pololu 3pi+ con un marcador del sistema de cámaras Optitrack.

1. `robotat_connect` y `robotat_disconnect`

La función de `robotat_connect` es una función empleada para conectarse a la red de Robotat. Este utiliza como argumento de entrada la dirección IP de la red la cual es 192.168.50.200 y se guarda en un objeto que se utiliza en otras funciones. Por otro lado la segunda función sirve para desconectarse de la red cuando se deje de utilizar para evitar sobre saturación en la red, esta utiliza como argumento el objeto creado con la función anterior.

2. `robotat_get_pose`

Esta función da como resultado la Pose del marcador solicitado con respecto al sistema de cámaras OptiTrack. Se utilizan tres elementos como argumento de la función, el primero es el objeto generado con `robotat_connect`, luego se indica el número de marcador del que se requiere la pose y por último la forma en la que se desea el resultado. Este resultado puede ser dado en diversos formatos como XYZ , ZYX , cuaternion (*quat*), etc al solicitarlo como argumento de lo contrario dará cuaternion como resultado.

3. `robotat_3pi_connect` y `robotat_3pi_disconnect`

La función `robotat_3pi_connect` permite la conexión por medio de Matlab hacia un Pololu 3pi+ por medio del ESP32 utilizado como intermediario de control. Esta utiliza como argumento el número de agente del robot el cual es un número único que le indica a cual ESP32 comunicarse. El resultado se guarda en un objeto que será utilizado en otras funciones. Por otro lado la función de desconexión funciona de forma similar a la del Robotat ya que toma como argumento de entrada el objeto creado previamente.

4. `robotat_3pi_set_wheel_velocities`

Esta permite enviar las velocidades de la llanta derecha e izquierda de los Pololu 3pi+ en revoluciones por minuto. Sus argumentos son el objeto obtenido por medio de `robotat_3pi_connect`, la velocidad del motor izquierdo y la del motor derecho.

5. `robotat_3pi_force_stop`

Su funcionamiento es similar al de la función anterior, ya que esta le indica al Pololu 3pi+ que debe detenerse de inmediato, pero a diferencia de la función de velocidad esta solo hace uso del objeto generado por la función de conexión como argumento.

8.2.2. Creación del programa de control de los Pololu 3pi+

Empleando las funciones disponibles para el entorno de Robotat se creó un programa llamado `Robot_Trajectory_Control.m` para emplear los algoritmos de control en el Pololu 3pi+. El cual se encuentra en el repositorio 14.1 dentro de la carpeta `Pruebas_Robotat`. Este programa está dividido en dos secciones principales, la primera es la inicialización de variables y selección de trayectoria. La segunda es la aplicación del control y muestra de los resultados obtenidos por el agente robótico.

1. Inicialización y trayectoria

En este bloque se realizan todas las tareas iniciales requeridas para aplicar los algoritmos de control al Pololu 3pi+. Primero se conecta al agente robótico a la red de Robotat para poder acceder a la información que brinda el sistema de cámaras OptiTrack y obtener la posición inicial del actor. Luego se inicializan las condiciones iniciales requeridas para el funcionamiento como lo son las constantes de los controladores, el radio de las llantas, la distancia de la llanta al centro del robot y la tolerancia de error del controlador.

Para la selección de trayectoria se cambia el valor de `seltraj` entre uno y cuatro para seleccionar tres trayectorias pre-generadas guardadas en archivos `.mat`, las cuales son una trayectoria cuadrada, una circular, una generada por el DSD, y una generada por medio de la planeación D* empleando la posición inicial y un punto de meta arbitrario dentro del mapa. Este mapa posee un tamaño de 380×480 píxeles de tamaño en donde el punto (190, 240) representa el centro del mapa.

2. Aplicación del control

En el segundo bloque se aplican los algoritmos de control empleando las funciones disponibles en el ecosistema de Robotat. Se comienza con un ciclo `while` del cual no se saldrá hasta que se termine la trayectoria. Dentro del ciclo se toman los datos de la pose del robot para así calcular el error de posición y orientación con respecto de la trayectoria y guardar la pose en una variable llamada `real_traj`, la cual es una matriz que guardara la trayectoria para poder analizarla más tarde. Una vez calculados los errores se aplican los controladores longitudinales y laterales y se calcula la velocidad de cada llanta para ser enviada a los Pololu 3pi+. Para verificar el progreso de la trayectoria se emplean dos condicionales `if`, la primera por medio de la tolerancia verifica la cercanía al siguiente punto y si el error de posición es menor a la tolerancia se pasa a tomar como referencia el siguiente punto. El segundo

if verifica si se ha llegado al último punto y de ser ese el caso le indica al robot que debe detenerse. Para observar el resultado del controlador se gráfica la trayectoria deseada y la obtenida en dos figuras para ser guardadas en un archivo *.mat* para su posterior utilización.

8.2.3. Resultados del control empleando el ecosistema Robotat

Las pruebas se realizaron empleando los controladores seleccionados cuyas constantes están listadas en el Cuadro 4. Como parámetros en el programa se utilizó una tolerancia de 0.1 un radio de llanta $r = 1.6cm$ y un largo de llanta al centro de $l = 4.5cm$. Para poder realizar una comparativa con los resultados del simulador se utilizaron las mismas cuatro trayectorias empleadas previamente.

Constante	Valor
K (Stanley)	0.5
P (PID Lateral)	11
I (PID Lateral)	0.001
D (PID Lateral)	2
V_0 (Acercamiento Exponencial)	200
α (Acercamiento Exponencial)	6
Tolerancia	0.1

Cuadro 4: Constantes de los controladores utilizadas en el entorno Robotat.

Como se puede observar en las Figuras 24, 25, 26 y 27 se muestra la comparativa entre la trayectoria a seguir, la realizada en el simulador por el actor empleando los algoritmos de control seleccionados para el Pololu 3pi+ y la trayectoria realizada por el Pololu 3pi+ captada por el sistema de cámaras OptiTrack. En el caso de las trayectorias circular y generada por DSD se obtuvieron los resultados esperados ya que el robot fue capaz de seguir la trayectoria de la misma forma que en el simulador, obteniendo un valor de RMSE de 0.0792 y de 0.0925. Los cuales indican que la diferencia entre la trayectoria teórica y realizada no fue significativa, ya que un valor RMSE cercano a cero indica que la trayectoria fue seguida de manera precisa y consistente. Por otro lado la trayectoria D* tuvo diferencias no significativas con respecto a lo esperado, ya que realizó las curvas ligeramente más abiertas que en el simulador, sin embargo el valor RMSE fue de 0.067, lo cual indica que no se tuvo errores significativos en el seguimiento de la trayectoria..

En el caso de la trayectoria cuadrada es donde se obtuvieron los peores resultados, ya que como se observa en la Figura 25, a pesar que el robot logró seguir la trayectoria tuvo diferencias significativas con respecto a lo esperado. Estos errores se dieron en las esquinas del cuadrado, donde el robot realizó curvas con mayor amplitud que en el simulador. A pesar de estos errores el robot siguió la trayectoria con un valor RMSE de 0.048, el cual es el menor valor entre las cuatro trayectorias. Pero como se aprecia en la Figura 25 la trayectoria realizada tiene errores apreciables de forma visual, este valor de RMSE se explica ya que es la diferencia media entre la trayectoria teórica y la real elevada al cuadrado, por lo que al ser el error concentrado en las esquinas del cuadrado este se ve opacado por el resto de la trayectoria.

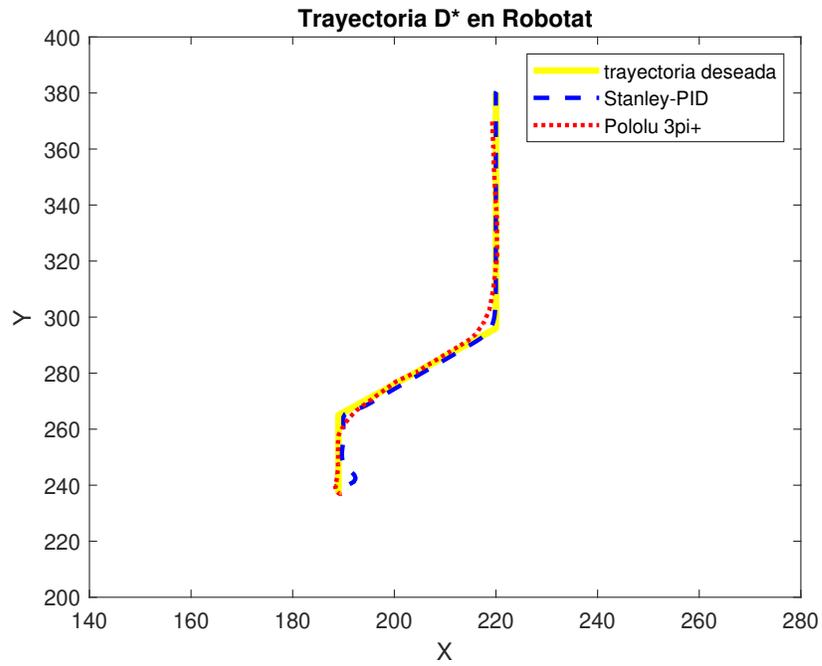


Figura 24: Trayectoria D* realizado por el Pololu 3pi+.

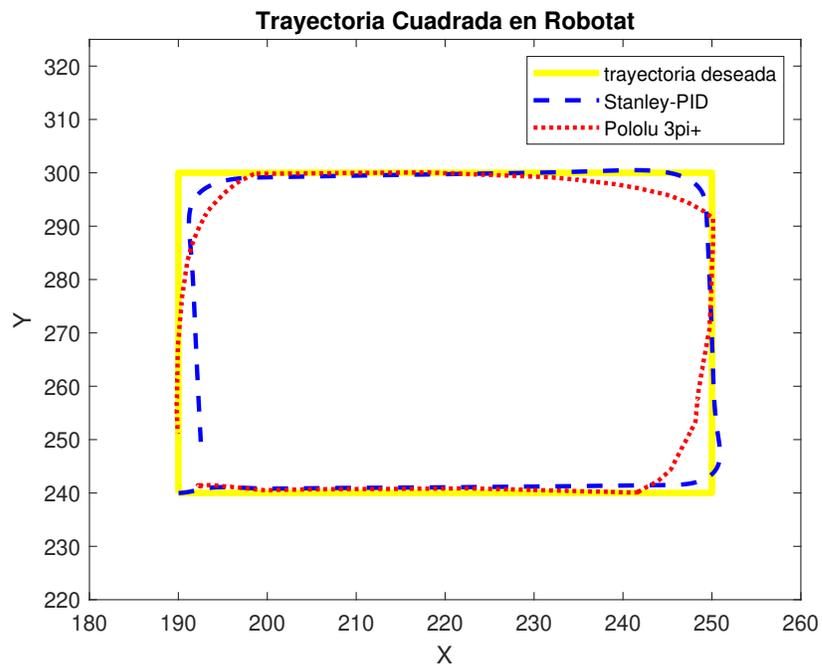


Figura 25: Trayectoria cuadrada realizado por el Pololu 3pi+.

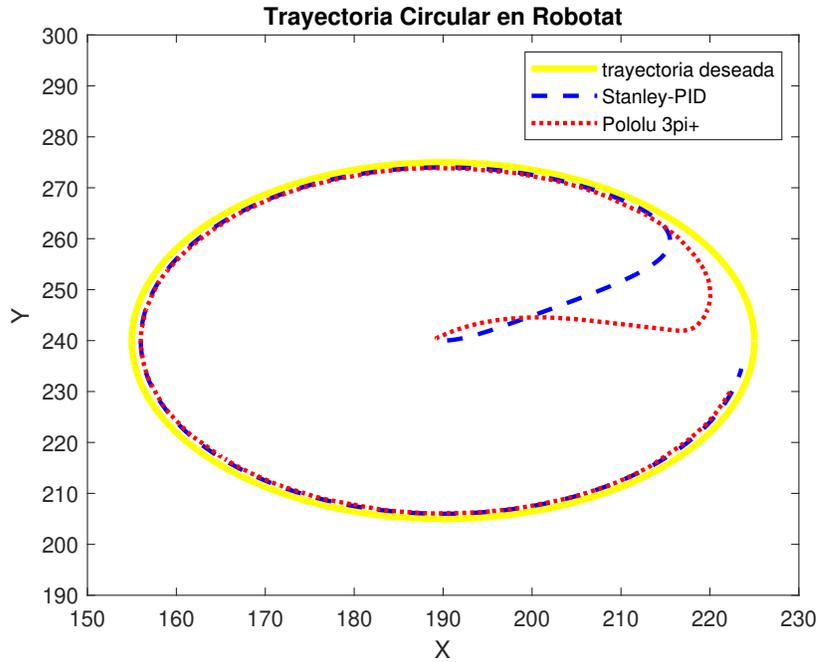


Figura 26: Trayectoria circular realizado por el Pololu 3pi+.

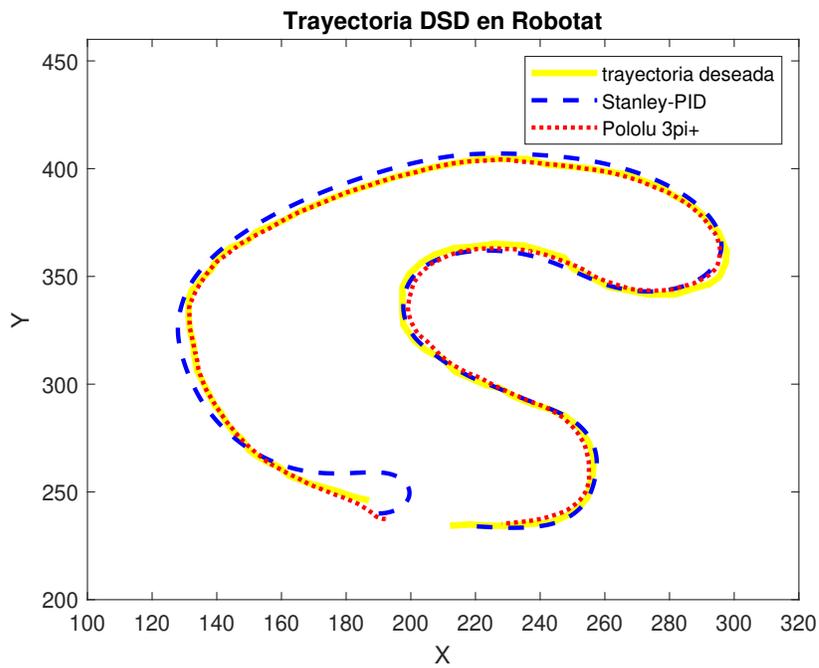


Figura 27: Trayectoria DSD realizado por el Pololu 3pi+.

Debido a los resultados obtenidos en el seguimiento de la trayectoria cuadrada se decidió realizar una comparativa al variar la tolerancia como se muestra en el Cuadro 5. Se realizaron cuatro recorridos con tolerancias distintas en donde a medida que esta fuera más estricta

se aumenta la velocidad para que el robot sea capaz de moverse, ya que con una tolerancia menor el robot se acerca más a los puntos antes de cambiar de objetivo reduciendo el error y por consiguiente la velocidad. Como se puede observar en la Figura 28, para tolerancias mayores a 0.1, el error en el seguimiento de la trayectoria es significativo en todo el tramo del recorrido. En cambio con las trayectorias con tolerancias menores a 0.1 se obtuvieron resultados satisfactorios.

Constante	Valor
V_0 (tolerancia = 0.4)	80
V_0 (tolerancia = 0.2)	120
V_0 (tolerancia = 0.1)	200
V_0 (tolerancia = 0.05)	400
α	6

Cuadro 5: Constantes de los controladores utilizadas en el entorno Robotat.

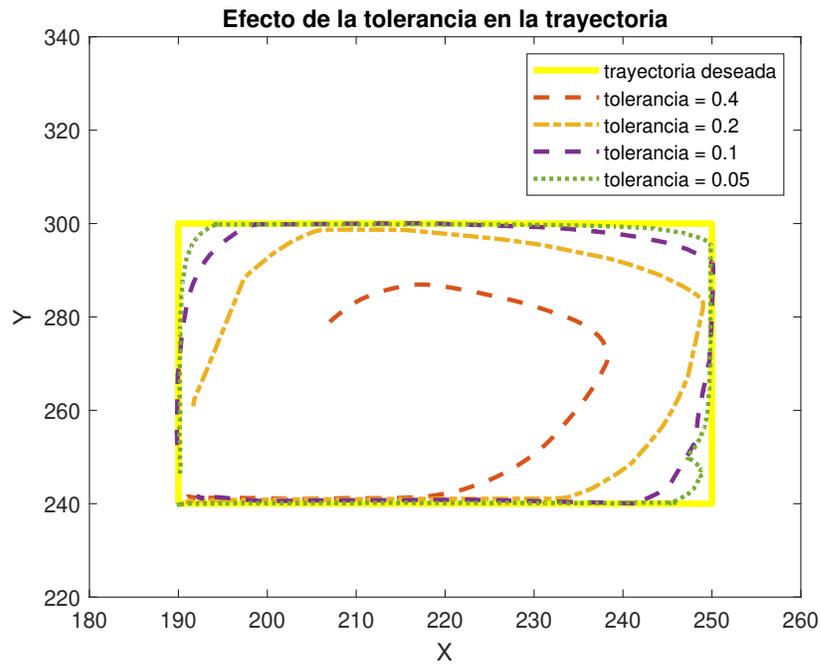


Figura 28: Efecto de la tolerancia en el resultado del control.

Implementación de algoritmos de visión de computadora

9.1. Metodología

Para la implementación de algoritmos de visión de computadora se hará uso de la OpenMV Cam H7, en este capítulo se utilizarán dos algoritmos de visión de computadora que se encuentran en el trabajo “Implementación de infraestructura a escala para la evaluación de algoritmos por visión de computador para vehículos autónomos” [7], el primero es uno capaz de detectar las líneas de una calle y el segundo es capaz de detectar elementos de tránsito como señales de alto, de peatón o los estados de un semáforo. Para el control del Pololu 3pi+ se utilizó un programa disponible en el ecosistema de Robotat y se modificó para adecuarse a las necesidades de cada algoritmo de visión de computadora. Una vez creados los programas se realizaron dos pruebas independientes, la primera es una aplicación de seguimiento de línea empleando una pista que simule las calles de una ciudad. La segunda es la detección y reacción de los distintos elementos de tránsito que el algoritmo de visión de computadora es capaz de detectar.

9.2. Validación del algoritmo de seguimiento de línea en los Pololu 3pi+

9.2.1. Creación del algoritmo de control de seguimiento de línea

En esta sección se emplearon los algoritmos de visión de computadora creados en el trabajo “Implementación de infraestructura a escala para la evaluación de algoritmos por visión de computador para vehículos autónomos” [7] realizado en la Universidad del Valle de Guatemala. El programa de detección de líneas usado se encuentra en el repositorio 14.1 dentro

de la carpeta *Deteccion_linea* con el nombre *DETECCION_LINEA_DE_COLOR.py*. Este programa se encuentra optimizado para la detección de líneas con tonalidad amarilla que se encuentren en un fondo oscuro, emulando de esta forma una calle pavimentada con líneas amarillas. El programa regresa dos valores, el primero se denomina como ρ y es el valor en píxeles en donde se encuentra la línea en la visión de la cámara en donde la zona más a la izquierda posee un valor de 0 y a la derecha posee un valor de 160. El segundo valor es θ y representa el ángulo de inclinación de la línea. En la Figura 29 se observa el funcionamiento del algoritmo de detección de línea dentro del OpenMV IDE.



Figura 29: Funcionamiento del algoritmo de detección de línea dentro del IDE de OpenMV.

Para el control del Pololu 3pi+ se empleó un programa diseñado en la Universidad del Valle de Guatemala para la plataforma de desarrollo ESP32, el cual ya posee la programación para enviar las velocidades de las llantas del robot desde el ESP32 hasta el Pololu 3pi+. Este programa tiene el nombre de *Pololu_Drive_Control.ino* y se encuentra en el repositorio 14.1 en la carpeta *Deteccion_linea\Pololu_Drive_Control*. Para la modificación del programa se utilizó la IDE de Arduino para la programación del mismo gracias a la gran cantidad de librerías y funcionalidades integradas para el ESP32.

En este programa se empleó la metodología RTOS para el manejo de tareas, ya que este permite realizar la ejecución de forma eficiente y predecible. Debido que al asignarle prioridades a las tareas se puede asegurar que una tarea se ejecute sin importar la situación creando un así un comportamiento deseado.

1. Declaración de librerías y variables

En este bloque del código se declararon las librerías a utilizar, se definieron las variables, como el tamaño de la llanta, distancia al centro, pines seriales, etc., que se van a utilizar en el resto del código y se creó un puerto serial por medio de software utilizando una de las librerías declaradas. Las librerías usadas para este código fueron:

- TinyCBOR: esta es una representación ligera de CBOR en C y diseñada para sistemas embebidos con capacidades limitadas. Este permite representar datos binarios de una

manera estructurada, eficiente y compacta.

- `SoftwareSerial`: esta librería permite al usuario por medio de dos pines del ESP32 crear por software los pines de transmisión y recepción de comunicación serial.
- `ArduinoJSON`: esta permite la creación y procesamiento de formatos JSON dentro del microcontrolador.

2. Tarea de envío de velocidades al Pololu 3pi+

Esta tarea tiene el nombre de `encode_send_wheel_speeds_tasks` y se encarga de enviar las velocidades de cada llanta al Pololu por medio del segundo puerto UART que posee el ESP32 y tiene la prioridad más alta dentro del programa. Esto se realiza utilizando la librería `tinyCbor`, la cual como se mencionó anteriormente codifica la información indicada en un formato ligero. Se utiliza el módulo `Encoder` el cual permite transformar datos en estructura de memoria en la representación binaria de CBOR. Primero se inicializa el módulo con la función `init` y se crea un arreglo por medio de la función `create_array` con el tamaño de datos deseado. Una vez creado el arreglo se codifican las velocidades de las llantas por medio de la función `encode_float` y se cierra el contenedor por medio de la función `close_container`, lo cual finaliza el proceso de codificación.

3. Tarea de Visual Servoing

En este bloque de código se encuentra la tarea `visual_servoing_task`, en la cual se encuentra todo aquello relacionado con el control del Pololu empleando la OpenMV cam H7 y se le fue asignada la segunda prioridad más alta dentro del programa. Se divide en dos partes principales, la primera es el área de comunicación serial que se encuentra dentro de una condicional `if` cuyo argumento es `mySerial.available()` el cual verifica si hay algún valor dentro del puerto serial. Dentro de este `if` se recibe un JSON por medio de una cadena con la función `mySerial.readStringUntil('\n')` y se procesa por medio de la función `deserializeJson(doc, jsonStr)`. La primera función lee una cadena de caracteres hasta encontrar un salto de línea mientras que la segunda procesa el JSON y lo guarda en una variable capaz de guardar información en formato JSON. Una vez guardado el JSON se extraen los componentes individuales y se guardan en variables globales declaradas en el primer bloque. Por último, se limpia el documento con la función `doc.clear()` para evitar que este se sature y se imprime por medio de UART para verificar si la información fue enviada correctamente.

La segunda parte se refiere al control del Pololu 3pi+ por medio de visual servoing por medio de un controlador proporcional para el controlador lateral y una velocidad constante para el longitudinal. Una vez se aplica el control se aplica una transformación para dividir la velocidad lineal y angular en las velocidades individuales de cada llanta. Por último, se aplica una conversión a las velocidades de las llantas para pasar de $\frac{rad}{s}$ a *RPM*.

4. Configuración de librerías y creación de tareas

El último bloque de código consiste en la creación de una función de configuración sin valor de retorno llamada `setup`. En esta se configuran los parámetros iniciales de tres puertos seriales usados en el código, se inicializa la librería `tinyCbor` y se crean las tareas mencionadas en el bloque dos y tres del programa.

9.2.2. Resultados del seguimiento de línea en el ecosistema Robotat

Para el control del Pololu 3pi+ que se observa en la Figura 30 se empleó un controlador PID como método de control y se tomó como parámetro de control la variable ρ . El control tomo como objetivo un valor de $\rho = 150$ el cual le va a indicar al vehículo que debe permanecer con la línea a su derecha. Como constantes para el controlador se utilizaron las listadas en el Cuadro 6 en donde se realizaron pruebas a distintas velocidades para evaluar el funcionamiento de los algoritmos de visión de computadora.

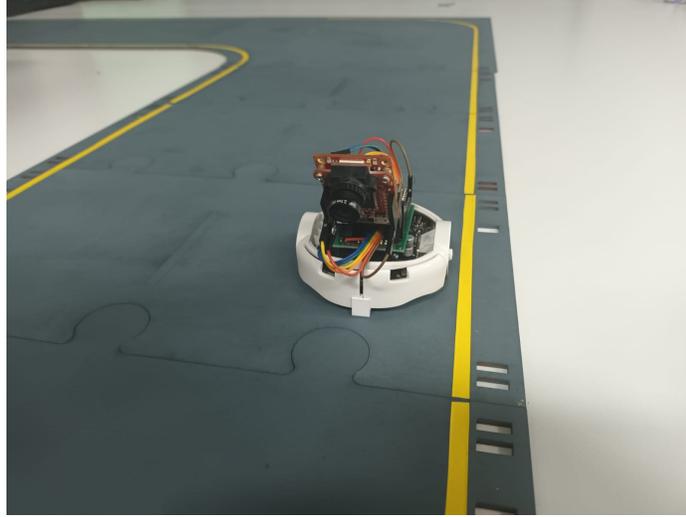


Figura 30: Pololu 3pi+ montado con la OpenMV cam H7 para el seguimiento de líneas.

Velocidad $\frac{rad}{s}$	Kp	Ki	Kd
10	0.02	0.002	0.0005
20	0.025	0.002	0.0005
30	0.025	0.002	0.0005
40	0.03	0.002	0.0005

Cuadro 6: Constantes del PID utilizados para visual servoing.

En la Figura 31 se observa el recorrido que el vehículo deberá realizar, el cual es un segmento de carretera en forma de L con el que se podrá verificar el funcionamiento de los algoritmos, ya que posee los dos elementos de importancia para una aplicación de seguimiento de línea. Una recta y una curva a noventa grados. Como se observa en la Figura 32 y en el vídeo Demostración de la aplicación de seguimiento de línea [1] el Pololu fue capaz de seguir la línea de forma satisfactoria. En los casos de velocidad 30 y 40 el Pololu obtuvo los mejores resultados con valores RMSE de 0.1794 y 0.1841 respectivamente. Mientras que para las velocidades de 10 y 20 el Pololu presenta perturbaciones en algunos segmentos del trayecto. Esto se debe a la baja velocidad a la que trabaja el Pololu en estos casos, ya que al ser estas tan bajas el controlador provoca que el Pololu cambie de la zona de operación a la zona muerta. Lo que provoca un aumento repentino de la velocidad provocando las

perturbaciones que se observan en la Figura 32. A pesar de las perturbaciones se obtuvieron valores RMSE de 0.2058 y 0.2383 respectivamente.



Figura 31: Pista de prueba para el seguimiento de línea.

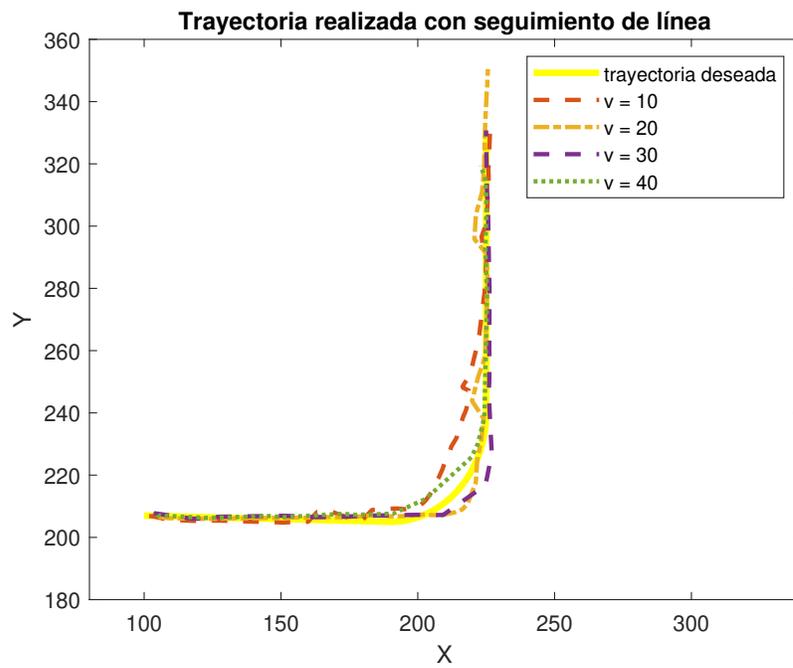


Figura 32: Trayectorias realizadas por medio de seguimiento de línea a distintas velocidades.

9.3. Validación de los algoritmos de detección de señales en los Pololu 3pi+

9.3.1. Creación del algoritmo de control para la reacción a señales

Para la detección de señales de tránsito y semáforos se emplearon los algoritmos diseñados en el trabajo “Implementación de infraestructura a escala para la evaluación de algoritmos por visión de computador para vehículos autónomos” [7]. Estos permiten por medio de algoritmos de visión de computadora detectar:

- Señales de alto.
- Señales de peatones/escolares.
- Los estados de un semáforo.

Esto se logra por medio de TensorFlow Lite, al cual se le asigna un *label* que al finalizar el entrenamiento devuelve los pesos de las características de cada señal. En las Figuras 33, 34, 35 y 36 se muestran los elementos de tránsito a detectar por el algoritmo de visión de computadora. Este método da como resultado un archivo tipo *tflite* el cual contiene la información resultante del entrenamiento de la cámara. Estos archivos junto con el programa de la cámara con nombre *ei_object_detection.py* se encuentran en el repositorio 14.1 en la carpeta *Deteccion_ señales*.



Figura 33: Señal de alto.



Figura 34: Señal de peatón.



Figura 35: Señal escolar.

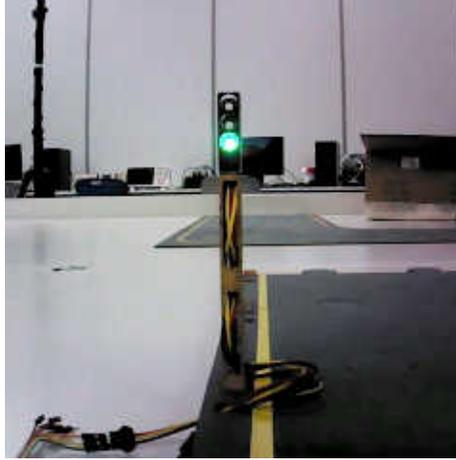


Figura 36: Semáforo.

Para el control del Pololu 3pi+ se empleó el mismo código base utilizado en la sección 9.2.1. Para adaptarlo a las necesidades de los algoritmos de visión de computadora se realizaron modificaciones a la tarea *visual_servoing_task* y se creó y declaró en la función de configuración una nueva tarea con el nombre de *signal_timer_task*. El programa tiene el nombre de *Pololu_Signals_Control.ino* y se encuentra en el repositorio 14.1 dentro de la carpeta *Deteccion_señales\Pololu_Signals_Control*.

1. Tarea de visual servoing

En esta tarea se cambia la forma en la que se recibe el JSON desde la cámara, ya que en esta ocasión se recibe un *string* que indica que elemento de tránsito se está detectando. La cámara es capaz de enviar cinco cadenas de caracteres: ALTO, PEATON, SEV, SEA y SER. Dependiendo de que cadena de carácter fue enviada por medio de condicionales *if* se verifica cual señal fue enviada utilizando como argumento la función *strcmp(street_signal, "señal detectada") == 0*. Luego se elimina el controlador PID que se encuentra en el programa de detección de líneas y se reemplaza por un condicional *if* cuyo argumento es una bandera llamada *noDetection_flag*, dentro se encuentran una serie de *if* que verifican que elemento de tránsito se está detectando. Dependiendo de que elemento se esté detectando se enciende una bandera si esta no está encendida, estas banderas son:

- *stop_flag*.
- *speed_flag*.
- *green_flag*.
- *yellow_flag*.
- *red_falg*.

La bandera *stop_flag* representa a la señal de alto, *speed_flag* representa a la señal escolar y de peatón y las banderas verde, amarilla y roja representan los estados de un semáforo. Adicionalmente existen dos banderas llamadas *stopDelay_flag* y *speedDelay_flag*

las cuales indican el inicio de una espera al haber detectado una señal de alto, de peatón o escolar.

2. Tarea de reacción a señales

Esta tarea contiene las reacciones del Pololu 3pi+ ante la detección de las señales, debido a la importancia del envío de las velocidades al Pololu y el recibimiento de la detección de los elementos de tránsito por parte de la cámara esta tarea tiene la prioridad más baja. En la Figura 36 se muestra un diagrama de máquina de estados finitos que muestra el funcionamiento de las distintas reacciones que el vehículo tendrá ante los elementos de tráfico. Si no se ha detectado ninguna señal el vehículo estará acelerando constantemente para mantenerse en una velocidad de $v = 20 \frac{rad}{s}$ (S0).

La primera rutina está dada por los estados del uno al cuatro y representa la reacción ante las señales de alto. En esta al detectar la señal enciende la bandera (S1) y entra en un condicional *if* que reduce la velocidad hasta llegar a detenerse, en donde se apaga la bandera y se enciende una bandera llamada *stopDelay_flag* (S2). Esta bandera permite el ingreso a otro *if* el cual por medio de un contador detiene al Pololu por un tiempo de cinco segundos (S3). Por último, luego del tiempo de espera se reinicia el contador, se apaga la bandera de delay, se comienza a acelerar hasta alcanzar una velocidad de $v = 20 \frac{rad}{s}$ y se enciende una bandera llamada *noDetection_flag* la cual va a evitar que se pueda detectar una señal de alto, peatón o escolar nuevamente durante cinco segundos para evitar que el Pololu se quede estancando en una señal (S4).

La segunda rutina está dada por los estados del cinco al ocho y representa la reacción a la señal de peatón/escolar. Esta rutina tiene un funcionamiento similar a la de la señal de alto, con la diferencia que en el estado (S5) reduce la velocidad a $v = 10 \frac{rad}{s}$ y se mantiene en ella durante cinco segundos (S7). Para finalmente activar el periodo de no detección y acelerar hasta $v = 20 \frac{rad}{s}$ (S8).

Por último, la rutina de reacción a los estados de un semáforo que consiste en los estados del nueve al trece. En esta rutina no existen tiempos de espera, ya que si se detectó una vez uno de los tres estados de un semáforo debe mantenerse en ese estado hasta detectar uno nuevo. Si se detecta que el semáforo está en verde se mantiene a una velocidad de $v = 20 \frac{rad}{s}$ (S0). Si se detecta que el semáforo está en amarillo se enciende la bandera y se comienza a disminuir la velocidad (S9). Al llegar a $v = 10 \frac{rad}{s}$ se mantiene a esa velocidad y espera a que cambie el estado del semáforo (S10). Si se detecta la señal roja existen dos caminos, si se detectó el semáforo en rojo de inicio se cambia del estado cero al estado once directamente, de lo contrario cambia del estado diez. Cuando detecta el semáforo en rojo enciende la bandera y comienza a reducir la velocidad hasta detenerse por completo (S11) y espera (S12) hasta que el semáforo cambie de estado a verde para volver a acelerar (S13) hasta alcanzar la velocidad de $v = 20 \frac{rad}{s}$ (S0).

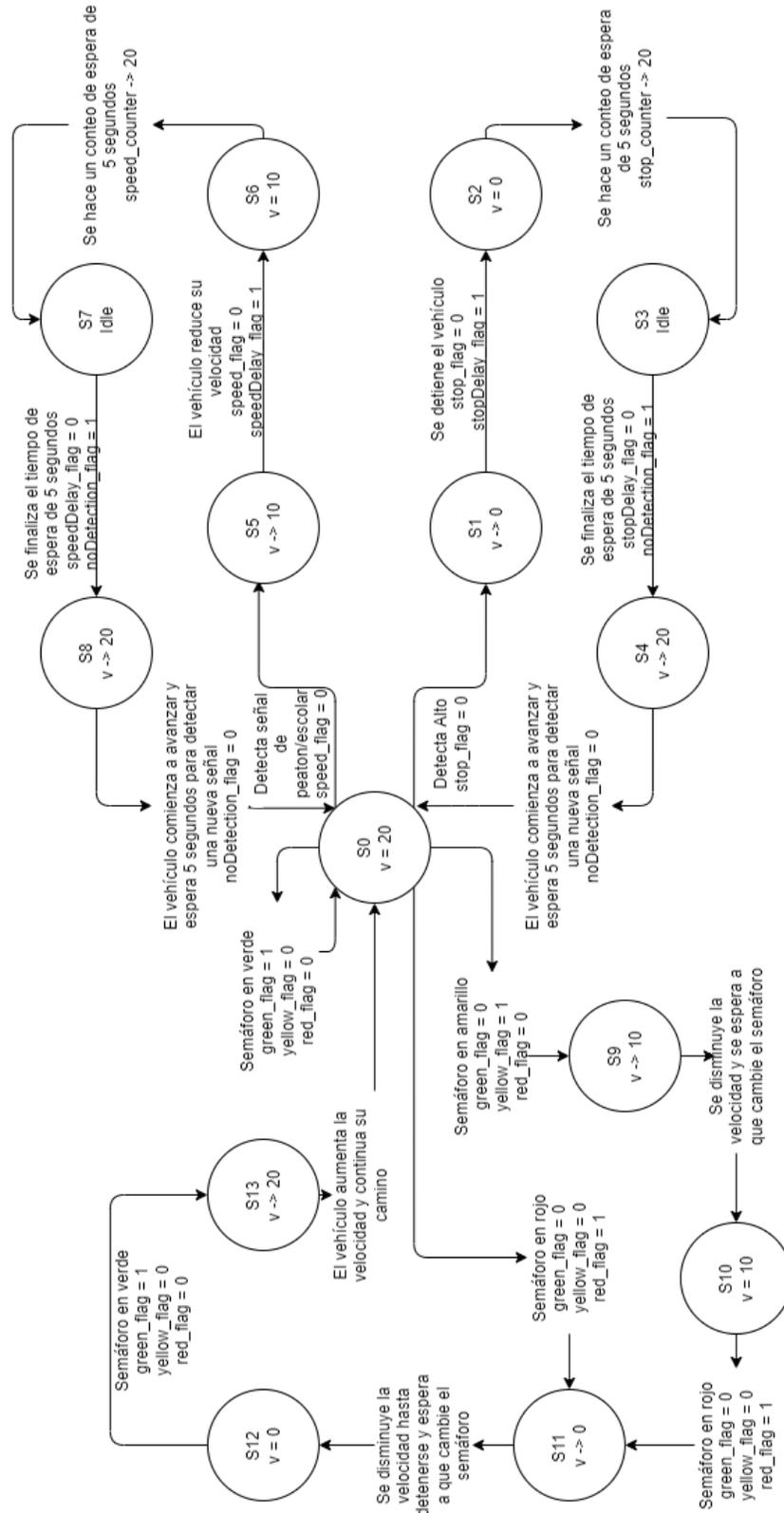


Figura 37: Diagrama de máquinas de estados finitos del funcionamiento de control de reacción a la detección de elementos de tránsito.

9.3.2. Resultados de la detección de señales en el ecosistema Robotat

Para verificar el funcionamiento de los algoritmos de visión de computadora y las funciones de reacción diseñadas para cada elemento de tránsito se empleó un segmento de pista recto colocado paralelo al eje Y del sistema de cámaras Optitrack. El cual contendrá uno de los elementos de tránsito para así verificar si la cámara es capaz de detectar los elementos en movimiento y se las funciones de reacción fueron diseñadas correctamente. Como se puede observar en la Figura 38 se muestra al Pololu con la cámara colocada para que este sea capaz de visualizar los elementos de tránsito sin ningún inconveniente. Para las pruebas de detección de señales no se utilizó ningún controlador lateral o longitudinal y se colocó la velocidad lineal del Pololu como una constante para que este siga la recta sin mayor inconveniente.

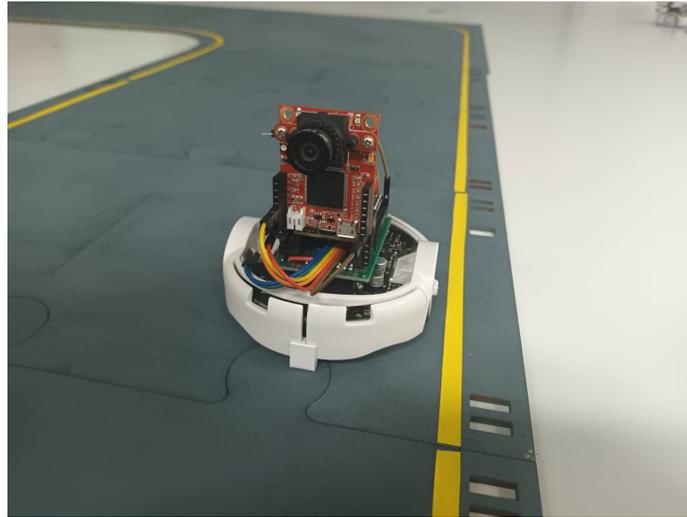


Figura 38: Pololu 3pi+ montado con la OpenMV cam H7 para la detección de señales.

En el video Demostración de la reacción a la señal de alto [2] se muestra el funcionamiento de la rutina de detección de señales de alto. En donde se observa que el Pololu fue capaz de detectar la señal de alto y de ejecutar la rutina según lo esperado. En la Figura 39 se observa una gráfica que muestra el cambio de la posición del Pololu a medida que se recabaron muestras de su posición durante la rutina de reacción a la señal de alto. En esta se pueden observar cuatro segmentos principales de la gráfica, primero es el estado previo a la detección en donde el Pololu viaja a una velocidad constante. Luego la cámara detecta la señal de alto y comienza la rutina disminuyendo la velocidad hasta detenerse. El tercer segmento muestra como el Pololu se detuvo y espero hasta que sucediera el cambio de estado para, por último, comenzar a acelerar nuevamente hasta alcanzar la velocidad objetivo.

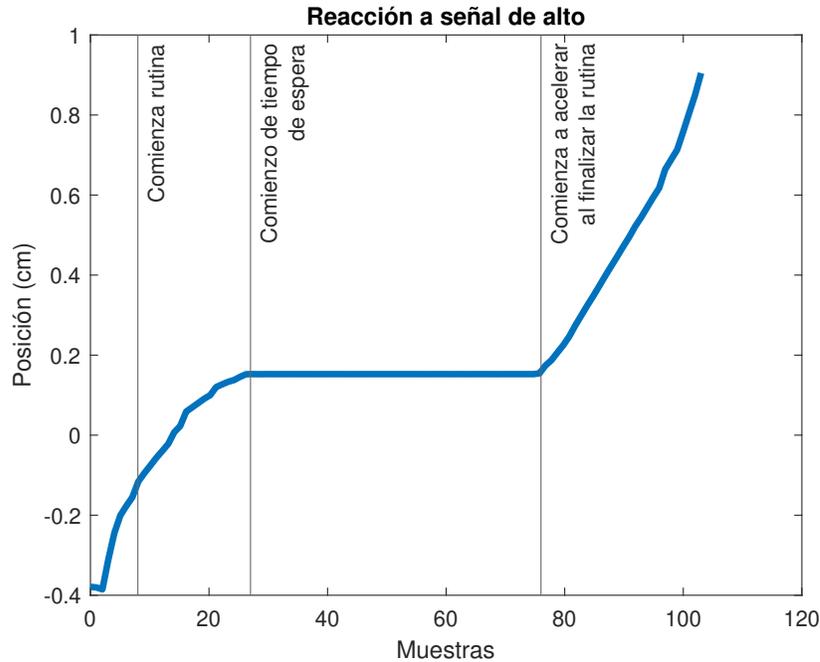


Figura 39: Cambio de la posición durante la rutina de reacción a señal de alto.

Mientras que en el video Demostración de la reacción a la señal de Peatón/Escolar [3] se muestra el funcionamiento de la rutina de reacción a las señales de peatón/escolar. En este se puede observar como el Pololu una vez detectada la señal comienza la reducción de la velocidad hasta reducir su velocidad y mantener la misma durante el tiempo indicado antes de acelerar nuevamente hasta alcanzar la velocidad objetivo. En la Figura 40 se observa una gráfica que muestra el cambio de la posición del Pololu a medida que se recabaron muestras de su posición durante la rutina de reacción a la señal de peatón/escolar. En donde a partir del segundo segmento de la gráfica se observa el comienzo de la rutina de reacción a la señal, ya que se observa una desaceleración en la velocidad del Pololu. Una vez alcanzada la velocidad deseada el Pololu se mantiene en esa velocidad durante el tiempo establecido como se muestra en el tercer segmento para acelerar a la velocidad original una vez el tiempo de espera haya terminado como se observa en el último segmento.

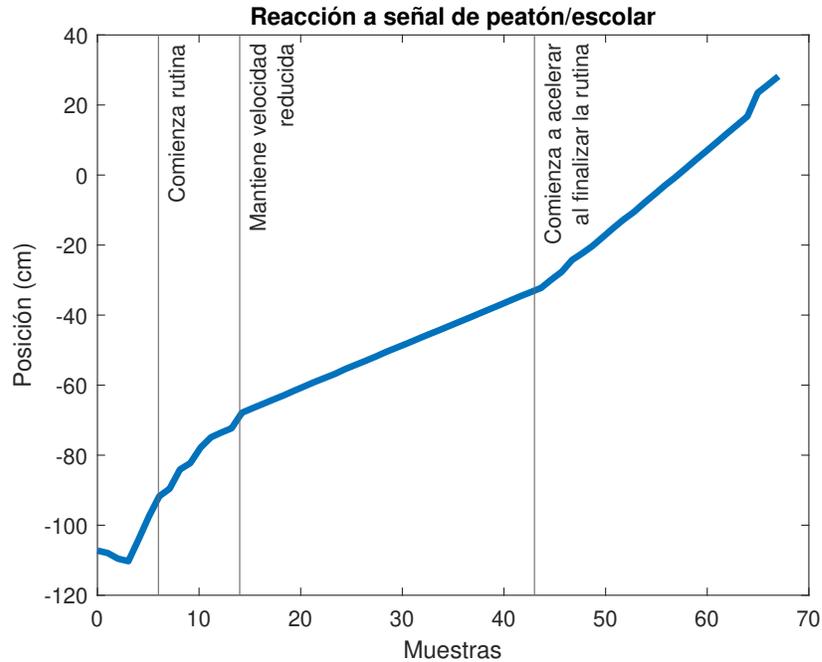


Figura 40: Cambio de la posición durante la rutina de reacción a señal de peatón/escolar.

Por último, en el video Demostración de la reacción a los estados de un semáforo [4] se muestra el funcionamiento de la rutina de reacción a los estados del semáforo. En este se observa que el Pololu a pesar de realizar la rutina de la forma esperada no fue capaz de realizarlo en movimiento. Esto fue debido a múltiples factores como el tamaño del semáforo y posición de este dentro de la calle, intensidad de la luz del semáforo e iluminación de la habitación la OpenMV cam H7 no fue capaz de detectar los estados del semáforo con la suficiente velocidad para permitirle al Pololu reaccionar de forma efectiva a los mismos. A pesar de los problemas encontrados se obtuvieron resultados satisfactorios ya que se logró validar el algoritmo de visión de computadora para la detección de semáforos y la rutina de reacción a los estados de un semáforo.

Pruebas preliminares de protocolos de comunicación

Durante las pruebas de reacción a los estados de un semáforo se encontró un problema relacionado a la librería *SoftwareSerial*. Este problema consiste en que el microcontrolador entre en un estado de “pánico”, lo que significa que este presenta un problema crítico que no puede ser manejado y resuelto correctamente. Lo que provoca un reinicio forzado del Esp32 generando problemas en el manejo de las rutinas de reacción, ya que si el microcontrolador se encontraba ejecutando una de estas rutinas se deberá de iniciar nuevamente debido al reinicio.

Este problema se genera debido al uso del puerto serial creado por medio de software, ya que este puerto es manejado y procesado por medio de uno de los dos núcleos que el Esp32 posee. En un inicio se empleó un baud rate (tasa de baudios) de 115200, el cual es uno de los valores más comunes empleados en la comunicación serial. Pero debido a la alta velocidad esto genero un gran estrés en el núcleo encargado de manejar la comunicación serial, el cual tenía a su cargo otras tareas, lo que provocó un fallo crítico en el núcleo y el reinicio del Esp32 como medida de seguridad. Una solución para este problema es el disminuir la tasa de baudios a un valor menor, aunque si el programa a ejecutar es lo suficientemente complejo esta acción no eliminara por completo los fallos críticos en los núcleos. Para el programa de reacción a las señales de tránsito se empleó una tasa de baudio de 9600, la cual se encontró que es la tasa que minimiza en mayor medida los fallos críticos.

Como alternativa a la comunicación serial se encuentran los protocolos SPI (Serial Peripheral Interface) e I2C (Inter-Integrated Circuit). Existe otro protocolo de comunicación disponible en el Esp32 y la OpenMV cam h7 el cual se le conoce como CAN bus, el cual se diseñó para la industria automotriz, pero es utilizado hoy en día en otras aplicaciones, pero este requiere de un integrado externo para poder funcionar. Uno de estos integrados es el MCP2515 el cual funciona por medio del protocolo SPI. Debido a esto se consideran solo los dos protocolos mencionados al inicio como alternativas al protocolo UART.

10.1. Protocolo SPI

Este es un protocolo de comunicación síncrono, lo que significa que utiliza un reloj para sincronizar la comunicación, que emplea el método maestro-esclavo para el manejo de las comunicaciones. Por lo que este protocolo está diseñado con el propósito que un maestro sea capaz de manejar múltiples esclavos a la vez. Pero se puede emplear una comunicación punto a punto similar al protocolo UART al definir un solo esclavo. Este protocolo hace uso de cuatro pines para realizar la comunicación:

- MOSI (Master Out Slave In): Línea de transmisión del maestro a los esclavos.
- MISO (Master In Slave Out): Línea de transmisión de los esclavos al maestro.
- SCK (Serial Clock): Línea del reloj para la sincronización de las transferencias de datos.
- SS/CS (Slave Select/Chip Select): Esta es una línea personal de cada esclavo, por la que se selecciona con que dispositivo se está comunicando.

Al compararlo con el protocolo UART el SPI presenta una mayor velocidad de transferencia, ya que al ser un método síncrono la conexión entre dispositivos está en tiempo real por lo que se pueden comunicar de forma inmediata. Mientras que la comunicación asíncrona se debe esperar a que el otro participante esté disponible para realizar la comunicación. Adicionalmente el SPI es capaz de operar por medio de Full-Duplex o comunicación bidireccional, lo que permite enviar y recibir datos en simultáneo. Pero requiere de una mayor cantidad de pines ya que se utilizan un mínimo de cuatro para realizar la comunicación, debido a su mayor velocidad de transferencia el protocolo SPI las distancias de transmisión son menores ya que se puede ver afectado por el ruido y su implementación es más compleja y tiene menor flexibilidad de uso que UART.

10.1.1. SPI en el Esp32

El Esp32 posee cuatro canales SPI, los primeros dos no pueden ser utilizados por el usuario ya que se utilizan para comunicarse con la memoria flash del dispositivo. Por lo que quedan disponibles los últimos dos canales para realizar la comunicación. Para hacer uso del SPI en el IDE de Arduino se emplea la librería *SPI.h*, la cual posee todas las funciones para inicializar y realizar la transferencia y recepción de información. Para inicializar el protocolo se hace uso de la función *SPI.begin(PIN_SCK, PIN_MISO, PIN_MOSI, PIN_SS)* con la cual por medio de los pines se va a seleccionar que canal SPI se desea utilizar. Adicionalmente se puede configurar diversos parámetros del SPI por medio de la función *SPISettings miConfiguracion(velocidad del reloj, orden de los bits, modo del SPI)*. En esta se pueden configurar tres parámetros del SPI:

- Velocidad del reloj: por medio de la *SPI_CLOCK_NUM* se modifica la velocidad a la que el SPI va a trabajar con valores que van de 2, 4, 8 hasta 128. Este número indica la división que se va a realizar a la frecuencia del reloj y si no se desea reducir la velocidad del reloj no se incluye el número.

- Orden de los bits: indican en qué momento se va a transmitir el bit con mayor significancia, existen dos configuraciones las cuales son: MSBFIRST y LSBFIRST. El primero el bit más significativo se transmite primero y en el segundo se transmite de último.
- Modo del SPI: existen cuatro modos del SPI y se indican por medio de la palabra *SPI_MODEX* en donde *X* representa el modo seleccionado. Estos son:
 - Modo 0: el reloj comienza esta en estado bajo cuando esta inactivo y se capturan los datos en flanco negativo.
 - Modo 1: el reloj comienza esta en estado bajo cuando esta inactivo y se capturan los datos en flanco positivo.
 - Modo 2: el reloj comienza esta en estado alto cuando esta inactivo y se capturan los datos en flanco positivo.
 - Modo 3: el reloj comienza esta en estado alto cuando esta inactivo y se capturan los datos en flanco negativo.

Para comenzar la comunicación primero se hace uso de la función *SPI.beginTransaction(miConfiguracion)*; la cual prepara los puertos para lo comunicación, posteriormente se habilita al esclavo colocando el pin *SS* en bajo. Para enviar información del maestro y recibir información del esclavo se utiliza la siguiente función *dato_recibido = SPI.transfer(dato_enviado)*, en donde la función *transfer* envía información al esclavo, normalmente un comando de relevancia para el esclavo) y el esclavo regresa la información solicitada al maestro. Para finalizar la comunicación se emplea la función *SPI.endTransaction()* y se coloca el pin *SS* como alto para deshabilitar al esclavo.

10.1.2. SPI en la OpenMV cam H7

Para colocar el único canal SPI de la OpenMV cam h7 como esclavo del Esp32 primero se importa la librería del SPI por medio de *from machine import SPI*. Para inicializar y configurar el SPI se utiliza la función *spi = SPI(2, mode=SPI.SLAVE, polarity=0, phase=0)*. Debido a que la cámara es un esclavo se debe esperar a que reciba un comando del maestro para enviar el dato solicitado, esto se logra por medio de *spi.recv(dato)*, una vez recibido el comando para enviar el dato se emplea la función *spi.send(bytearray([my_variable]))* para enviar la información solicitada.

10.2. Protocolo I2C

Al igual que el protocolo SPI el I2C es un protocolo de comunicación síncrono multi-maestros, lo que significa que a diferencia del SPI este puede definir múltiples maestros y esclavos en simultaneo. Esta tiene un mejor funcionamiento en aplicaciones multi-dispositivos, ya que funciona por medio de un bus de datos al cual se conectan todos los dispositivos y permita la transferencia de datos de forma bidireccional. Los pines utilizados en este protocolo son los siguientes:

- SDA (Serial Data Line): Línea de bus de datos para la transferencia bidireccional de datos.
- SCL (Serial CLock Line): Línea del reloj de sincronización para la comunicación.

Este protocolo para identificar el dispositivo hace uso de una dirección única que facilita la identificación de cada dispositivo. Pero debido a que se realiza la comunicación por medio de un bus de datos se tiene una velocidad de transferencia de datos menor al SPI y UART, ya que todos los dispositivos se encuentran conectados a la misma línea de transmisión. Adicionalmente su distancia de transmisión es menor ya que este posee una mayor susceptibilidad al ruido por el uso de una única línea de comunicación.

10.2.1. I2C en el Esp32

El Esp32 posee dos canales de I2C para ser utilizados, para configurar el I2C se emplea la librería *Wire.h*. Este protocolo es más sencillo de implementar que el SPI, ya para inicializar y configurar el canal se emplea la función *Wire.begin()* la cual no permite la configuración de los parámetros individuales del protocolo. Una vez inicializado el canal se comienza con la transmisión, primero se define la dirección del esclavo, un ejemplo de esta dirección puede ser *0x12* y esta dirección se guarda en una variable para ser usada posteriormente. Para transmitir datos se comienza abriendo la transmisión por medio de *Wire.beginTransmission(address)* en donde por medio de la dirección se indica que esclavo debe recibir la información, una vez abierta la transición se envían los datos por medio de *Wire.write(dato)*, en donde en la función de maestros se suele enviar comandos que le indiquen a los esclavo la acción que debe realizar y al finalizar se cierra la transmisión con *Wire.endTransmission*.

Para recibir datos de un esclavo en específico primero se abre la comunicación con la dirección del esclavo y se emplea la función *Wire.requestFrom(slaveAddress, bytes requeridos)*, en donde al esclavo se le solicita una cantidad definida de bytes de información. Una vez solicitada la información se emplean dos funciones para recabar toda la información, la primera es *Wire.available()* en donde mientras exista información siendo transmitida se leerá la información hasta que el esclavo deje de enviar. La segunda es *Wire.read()*, la cual va a leer y guardar en una variable los datos enviados por medio del pin SDA del canal I2C. Una vez finalizada la recepción de datos se cierra la transmisión con el esclavo.

10.2.2. I2C en la OpenMV cam H7

Para configurar el único canal I2C de la OpenMV cam H7 se importa de la librería *pyb* la biblioteca de I2C. Primero se define la dirección del esclavo y se guarda en una variable, esta debe ser la misma que se definió en el maestro. Para inicializar y configurar el canal I2C se emplea la función *i2c.init(I2C.SLAVE, addr = dirección de identificación)*, los parámetros que se pueden configurar en esta función son:

- Configuración como maestro o esclavo por medio del primer argumento.
- Definición de la dirección del esclavo en el segundo argumento.

Una vez inicializado el canal I2C primero se debe esperar a que el maestro solicite información, esto se da al recibir información por medio de la función *data = i2c.recv(número de bytes)*. En donde se recibe una cantidad de bytes del maestro que se define dependiendo de los requerimientos del proyecto, una vez recibido el comando para enviar información se emplea la función *i2c.send(dato)*.

- Se diseñó un controlador lateral empleando el algoritmo de control Stanley, el cual es utilizado para el control de vehículos autónomos. Esto por medio de una segunda capa de control empleando un PID el cual toma como referencia el control de Stanley para controlar al Pololu 3pi+.
- Los algoritmos de control tuvieron un mejor funcionamiento en las trayectorias circular, generada por DSD y generada por D* ya que estas presentaron un valor RMSE máximo de 0.0925 y un mejor seguimiento de las trayectorias desde una perspectiva cualitativa.
- Las trayectorias con cambios de dirección cerrada como la cuadrada presentan problemas para los algoritmos de control como se observa en la Figura 25 ya que el algoritmo de control de Stanley no es capaz de realizar cambios bruscos de dirección. Por lo que al utilizar una tolerancia más estricta entre puntos se minimiza el error generado por esta limitación como se observa en la Figura 28.
- El algoritmo de seguimiento de línea presenta un mejor funcionamiento en velocidades mayores a $20 \frac{rad}{s}$, ya que en estas velocidades presenta un valor RMSE máximo de 0.1841 y presenta una menor cantidad de perturbaciones durante su funcionamiento.
- Los algoritmos de detección de señales de alto y de peatones/escolar lograron detectar las señales de forma satisfactoria y el Pololu 3pi+ fue capaz de reaccionar a las mismas de la forma esperada como se observa en las Figuras 39 y 40.
- Con base a las pruebas realizadas se destaca que, si bien se logró implementar rutinas de reacción a los estados de un semáforo por medio de los algoritmos de detección de elementos de tránsito, se encontró una limitación en la velocidad de detección de los algoritmos de visión de computadora mientras que el Pololu 3pi+ se encontraba en movimiento. A pesar de este desafío se validaron de forma satisfactoria los algoritmos de visión de computadora y las rutinas de reacción a elementos de tránsito como se observa en el video 4.

- Obtener constantes propias para los algoritmos de control ya que las utilizadas pueden no funcionar en otras condiciones.
- Validar los algoritmos de control por medio de trayectorias antes de implementar los algoritmos de visión de computadora para verificar el correcto funcionamiento de los controladores.
- Explorar la implementación del algoritmo Pure Pursuit como control lateral del vehículo.
- Interconectar el algoritmo de detección de línea junto a los algoritmos de detección de señales para realizar una tarea conjunta.
- Explorar los protocolos de comunicación SPI e I2C para la comunicación entre la cámara y el Esp32 en lugar de la librería *SoftwareSerial*, para evitar problemas de rendimiento y errores en la ejecución del programa.
- Se recomienda la utilización de ESP-IDF para la programación del Esp32 en lugar del IDE de Arduino, ya que este permite un mayor control sobre el hardware al poder configurar los pines dependiendo de los requerimientos del proyecto, presenta un mejor rendimiento y eficiencia en el uso de los recursos del Esp32 al estar más optimizado para este microcontrolador. Además, presenta una mayor escalabilidad en la creación de proyectos que el IDE de Arduino.

-
- [1] C. D. Endara y E. J. Maigua, «Desarrollo de un algoritmo de trayectoria para un robot seguidor de línea destreza de competencia mediante visión e inteligencia artificial,» Universidad Politécnica Salesiana, Ecuador, 2021. dirección: <https://dspace.ups.edu.ec/handle/123456789/19880>.
- [2] D. F. Quiroga y B. L. Rubio, «Robot móvil autonomo para la siembra de semillas en el campo,» Universidad Politécnica Salesiana, Ecuador, 2023. dirección: <https://dspace.ups.edu.ec/handle/123456789/24491>.
- [3] Y. Song y Z. Xie, «Control method of three-wheel intelligent tracking logistics car based on OpenMV4,» *J. Phys.: Conf. Ser.*, vol. 2246, n.º 012033, págs. 1-6, 2022. dirección: <https://iopscience.iop.org/article/10.1088/1742-6596/2246/1/012033/meta>.
- [4] M. Maurer, R. Behringer, D. Dickmanns, T. Hildebrandt, F. Thomanek y et al., «VaMoRs-P: an advanced platform for visual autonomous road vehicle guidance,» *SPIE*, 1995. dirección: <https://doi.org/10.1117/12.198974>.
- [5] S. Alcalde, «Los objetivos del Rover Perseverance en Marte,» *National Geographic*, 2023. dirección: https://www.nationalgeographic.com.es/ciencia/objetivos-rover-perseverance-marte_16363#:~:text=El%20robot%20explorador%20Perseverance%20es,la%20superficie%20del%20planeta%20rojo..
- [6] Á. Rayo, «Estudio del módulo “OpenMV Cam H7” para aplicaciones de visión artificial,» Universidad de Alcalá, España, 2021. dirección: <https://ebuah.uah.es/dspace/handle/10017/49493>.
- [7] G. A. Fong, «Implementación de infraestructura a escala para la evaluación de algoritmos por visión de computador para vehículos autónomos,» Tesis de licenciatura sin publicar, Universidad Del Valle de Guatemala, 2023.
- [8] *Pololu 3Pi+ 32U4 User's Guide*, Pololu Robotics y Electronics. dirección: <https://www.pololu.com/docs/0J83/all>.

- [9] *ESP32-WROOM-32D and ESP32-WROOM-32U Datasheet*, Espressif Systems. dirección: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf.
- [10] S. Waslander y J. Kelly, *Specialized program: Self-driving car*. dirección: <https://www.coursera.org/specializations/self-driving-cars>.
- [11] K. Lynch y F. Park, «Modern Robotics: Mechanics, Planning and Control,» en Cambridge University Press, 2017, cap. 13.
- [12] B. Sciliano, L. Sciavicco, L. Villani y G. Oriolo, «Robotics Modelling Planning and Control,» en Springer, 2010, cap. 11.
- [13] K. Lynch y F. Park, «Modern Robotics: Mechanics, Planning and Control,» en Cambridge University Press, 2017, cap. 11.
- [14] D. Barnes y V. Alakshendra, *Vehicle Path Tracking Using Stanley Controller*. dirección: <https://la.mathworks.com/videos/vehicle-path-tracking-using-stanley-controller-1628159837296.html>.
- [15] C. Perafán, «Robotat: un ecosistema robótico de captura de movimiento y comunicación inalámbrica,» Universidad del Valle de Guatemala, Guatemala, 2022. dirección: <https://repositorio.uvg.edu.gt/handle/123456789/4250>.

14.1. Repositorio de Github

Vehiculo_autonomo_Robotat Repositorio de los códigos del proyecto.

14.2. Enlaces a videos demostrativos y explicativos

1. Demostración de la aplicación de seguimiento de línea:
<https://youtu.be/ZgPkCYcZTEw>
2. Demostración de la reacción a la señal de alto:
<https://youtu.be/oAkQ0b3ZN1Q>
3. Demostración de la reacción a la señal de Peatón/Escolar:
<https://youtu.be/xGB65Xrvbdc>
4. Demostración de la reacción a los estados de un semáforo:
<https://youtu.be/z28KK0mDFwk>

- CBOR:** Concise Binary Object Representation por sus siglas en ingles.. 43, 44
- DSD:** Siglas de *Driving Scenario Designer*.. 27, 28, 30, 32, 35, 37, 38, 60
- IDE:** Un IDE o EDI (Entorno de desarrollo Integrado) por sus siglas en español es una herramienta de software que ayuda a los programadores a desarrollar código de software de manera eficiente.. 43, 56, 61
- pose:** Ubicación y orientación de un objeto en un espacio tridimensional.. 35, 36
- RMSE:** El Root Mean Square Error (RMSE) es una medida de la diferencia promedio entre los valores observados y los valores predichos en un conjunto de datos. Es un número que representa la raíz cuadrada de la media de los errores al cuadrado.. 38, 45, 46, 60
- RTOS:** Real Time Operating System por sus siglas en ingles.. 43
- toolbox:** Conjunto de funciones y herramientas especializadas que amplían las capacidades de un software. . 27
- visual servoing:** Es una técnica que utiliza información de retroalimentación extraída de un sensor de visión para controlar el movimiento de un robot.. 44, 45