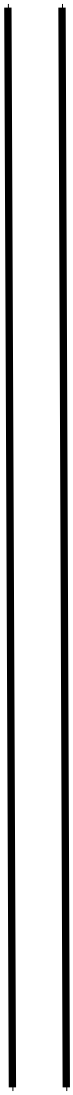
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA



Navegación autónoma para un vehículo sin conductor usando el simulador Webots

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Luis David Torres Trejo

DIRECTOR DE TESIS

Dr. Marco Antonio Negrete Villanueva

Ciudad Universitaria, Cd. Mx., 2022

Agradecimientos

Este trabajo se realizó con el apoyo de la UNAM-DGAPA a través del proyecto PAPIIT TA101222.

Índice general

Índice de figuras

Capítulo 1

Introducción

En las últimas décadas se ha notado un amplío aumento de esfuerzos en investigación y desarrollo de tecnologías para vehículos sin conductor. Pues se cree que los vehículos inteligentes cuentan con el potencial suficiente para mejorar la seguridad, calidad y confort durante viajes en carretera e incluso algunos de ellos ya se encuentran en algunas carreteras del mundo. Muchos de los avances en este sector de la robótica se deben a los constantes cambios en el campo de la informática que, con el paso de los años ha crecido a pasos agigantados, reduciendo costos y desarrollando tecnologías más eficientes.

Empresas dedicadas a tecnologías de la información así como algunas del sector automotriz han abierto sus puertas al descubrimiento, investigación y desarrollo de tecnologías para vehículos autónomos y uso de energías limpias. Google, Tesla, Uber, Toyota, Nvidia, Ford son solo una de las múltiples compañías dedicadas a este sector. Además, diferentes universidades de América, Europa y Asia también han contribuido con múltiples trabajos de investigación relacionados con vehículos autónomos.

Por mencionar uno de los varios objetivos en el desarrollo de vehículos inteligentes se encuentra la disminución de tiempos en tránsito mediante planeación de rutas óptimas y conducción sin distracciones, como consecuencia esto provocará un reducción en emisiones contaminantes y consumo de combustibles fósiles, sin mencionar que algunos vehículos modernos ya cuentan con tecnologías para el uso de energías limpias. También, se prevé que en futuro exista el concepto “movilidad para todos”, es decir, los vehículos autónomos podrán ser un medio de movilidad personal útil y cómodo para personas que por distintas circunstancias tengan limitaciones físicas.

Desde un punto de vista más técnico se debe de entender que un vehículo autónomo es aquél que puede conducir sin la necesidad de un conductor al frente del volante. Sin embargo, no todos los vehículos en la actualidad son de este tipo, la medida en que un vehículo es considerado autónomo varia desde ser operado por un conductor hasta ser completamente independiente, para ello existe una clasificación que define diferentes niveles de autonomía con base en las características de cada vehículo. Por ejemplo, algunos vehículos cuentan

con sistemas de conducción autónoma que permiten detectar carriles y advertir cambios de carril al conductor sin perder el rumbo. Otros, cuentan con sistemas para detección y seguimiento de vehículos que circulan por delante con el fin de mantener una distancia cómoda y segura.

En este sentido, la arquitectura de un sistema de autonomía de vehículos no tripulados suele organizarse en dos partes principales: un sistema de percepción y uno de decisión [26]. El sistema de percepción se divide en subsistemas encargados de tareas específicas como: mapeo de carreteras y obstáculos estáticos, detección y seguimiento de obstáculos en movimiento, detección e identificación de señales de tráfico, entro otras. Mientras que el sistema para toma de decisiones se divide en otros subsistemas para tareas de planificación de rutas y caminos, selección de comportamientos, planificación de movimientos, evasión de obstáculos, etc.

Sin embargo, como toda tecnología tiene inconvenientes, por ejemplo; ¿Qué sucedería si un vehículo autónomo choca contra otro igual?, ¿Cuál de ellos tendría la culpa?, ¿Los humanos ya no deberían de aprender a manejar?, ¿Cómo estandarizar las leyes de tránsito?. Estas y muchas otras cuestiones en combinación con fallos técnicos debido a la naturaleza de los vehículos autónomos son limitaciones que requieren de mucho esfuerzo, desarrollo y trabajo tanto de la parte técnica como del lado legal. Por ello, entre más alto y confiable sea el nivel de autonomía de un vehículo no tripulado será más sencillo resolver estas situaciones en el futuro.

1.1. Motivación

La movilidad es un problema común en estos días debido a la alta densidad de vehículos circulando en las grandes ciudades del mundo, problemas como tráfico, estrés y accidentes de carretera son algunos ejemplos. Según datos del INEGI solo en México en el año 2020 se reportaron más de 75 000 personas involucradas en accidentes de tránsito, de las cuales al menos un 5 % fallecieron en el accidente y el otro 95 % presentó algún tipo de lesión [13]. Uno de los principales objetivos de la conducción autónoma es reducir significativamente el número de defunciones ocasionadas por accidentes de carretera, pues los mismos datos indican que gran parte de los accidentes son ocasionados por errores humanos durante la conducción y en muchas de estas ocasiones son terceras personas las que sufren mayor daño y no el conductor.

Un vehículo inteligente puede ayudar a cambiar está situación mediante la toma de mejores decisiones pues estas serían mejor planeadas con base en el entorno y factores que lo rodean, además, no existiría el factor de distracción como ocurre con los conductores humanos. Todo esto sin mencionar el cumplimiento de reglas viales que en muchos de los casos son omitidas por conductores y derivan en pequeños o grandes percances.

Sin embargo, las investigaciones y desarrollos de tecnología inteligente para vehículos autónomos comúnmente requiere gran cantidad de recursos económicos y humanos. En general, el hardware necesario para instrumentar un vehículo

sin conductor es costoso y difícil de desarrollar, además los ambientes para pruebas de navegación del vehículo deben ser controlados y los más parecidos a un entorno real para obtener resultados y conclusiones significativas. Por ello, en gran medida el desarrollo de estas tecnologías es realizado por grandes empresas involucradas en tecnologías de la información y automotrices, ya que cuentan con los recursos necesarios para innovar.

Otros trabajos relacionados como los hechos por universidades de diferentes latitudes del mundo son desarrollados a través de simuladores. Los simuladores dedicados al campo de la robótica resultan ser una buena alternativa para solucionar el problema de ausencia de recursos físicos, pues mediante un simulador se pueden desarrollar sistemas de control, visión artificial y navegación autónoma para vehículos no tripulados. Se puede contar con el hardware necesario (simulado) para instrumentar un vehículo, además de recursos para crear distintos escenarios de prueba. Claro que al utilizar simuladores se omiten muchos factores presentes en un entorno urbano real, pero sin duda juegan un papel importante en el desarrollo de nuevas tecnologías para vehículos autónomos.

1.2. Planteamiento del problema

Con el propósito de contribuir en la investigación de tecnologías para vehículos autónomos utilizando simuladores es necesario un ambiente simulado con características suficientes para diseñar y desarrollar algoritmos de control, visión artificial y navegación autónoma para automóviles inteligentes. En este sentido, es requerido por el vehículo autónomo diferentes sistemas que le permitan percibir el ambiente. Dentro de estos sistemas se encuentran: un sistema de visión artificial capaz de reconocer carriles, un sistema encargado de detectar y seguir obstáculos en movimiento.

También, se requiere de un conjunto de comportamientos que ayuden a simular el acto de conducir, estos comportamientos deben ser capaces de realizar seguimiento de carriles, seguimiento de vehículos en situaciones de tránsito vehicular y rebase de otros vehículos. Este conjunto de comportamientos en combinación con un sistema de decisión deben dar como resultado una experiencia cercana a la navegación autónoma real en ambientes controlados.

1.3. Hipótesis

El planteamiento del problema y el desarrollo del mismo tiene de base las siguientes hipótesis:

- Los simuladores son una buena opción durante el desarrollo de tecnologías para vehículos autónomos.
- La transformada Hough y el detector de bordes de Canny son lo suficientemente poderosos para lograr el reconocimiento de carriles en imágenes RGB.

- El simulador Webots cuenta con las características necesarias para desarrollar sistemas de visión artificial, control y navegación autónoma para vehículos sin conductor.
- Las máquinas de estados finitos son apropiadas para el desarrollo de sistemas robóticos basados en el paradigma reactivo.
- La navegación autónoma de un vehículo no tripulado en un ambiente controlado se puede conseguir mediante la implementación de comportamientos para seguimiento de carriles, rebase y mantener distancia.

1.4. Objetivos

Con base en el planteamiento del problema y las hipótesis anteriormente mencionadas se esperan poder alcanzar los siguientes objetivos al final de este trabajo:

- Diseñar y modelar un ambiente vial urbano con las condiciones necesarias para realizar pruebas de navegación autónoma con el simulador Webots.
- Utilizar las herramientas de detector de bordes de Canny y transformada Hough a fin de desarrollar un algoritmo para detección de carril a partir de imágenes RGB.
- Diseñar y desarrollar un sistema de control para seguimiento de carril.
- Establecer e implementar comportamientos para seguimiento de carriles, rebase de obstáculos y mantener distancia.
- Integrar todos los módulos desarrollados empleando la plataforma ROS y el simulador Webots.
- Realizar pruebas de navegación sin obstáculos y pruebas de navegación con obstáculos tanto estáticos como en movimiento.

1.5. Descripción del documento

Este trabajo cuenta con diferentes capítulos que se adentran en el mundo de la conducción autónoma. Para comenzar, en el capítulo 2 se presentan antecedentes históricos y definiciones de conceptos básicos que serán utilizados a lo largo del proyecto como: conceptos de vehículos autónomos, simuladores, visión artificial, entre otros. Además, se mencionan algunos trabajos relacionados con vehículos sin conductor. Enseguida, el capítulo 3 está dedicado al simulador Webots, el cual se utilizó para modelar escenarios y realizar pruebas de navegación autónoma, se describen la características básicas del simulador y se explica como utilizar algunas de ellas. Al final de este capítulo se presentan contribuciones de este trabajo que influyeron en la categoría AutoModelCar del TMR-2022.

El capítulo 4 trata acerca de la detección de carriles a partir de imágenes RGB, se explican teóricamente las herramientas matemáticas de transformada Hough y detector de bordes de Canny. Además, se realiza una implementación de cómo pueden ser utilizadas estas herramientas para detectar carriles en una imagen. Este capítulo concluye con el diseño e implementación de un par de leyes de control para el movimiento lateral y longitudinal del vehículo con base en su modelo cinemático. A continuación, el capítulo 5 se centra en el proceso de detección de obstáculos, se explica como utilizar un algoritmo de agrupación para identificar grupos, además se da a conocer el concepto de filtro de Kalman y su utilidad en aplicaciones de seguimiento de objetos, esto se ejemplifica con el diseño e implementación de un filtro de Kalman para estimar posición y velocidad de vehículos. El capítulo 5 también da una explicación del por qué es necesario desarrollar un algoritmo de empatado durante el proceso de seguimiento de objetos.

Posteriormente, en el capítulo 6 se describen brevemente los principales paradigmas de la robótica y se menciona qué paradigma es utilizado por el robot (vehículo autónomo) de este trabajo. Después, se definen y describen los comportamientos que debe realizar el robot en distintas situaciones, igualmente se expone un sistema de decisión para la elección de comportamientos con base en observaciones del ambiente. Por su parte, el capítulo 7 es destinado a la presentación de resultados obtenidos por sistema de navegación bajo pruebas de conducción autónoma sin obstáculos y conducción autónoma con obstáculos estáticos y con movimiento.

El capítulo 8 es el término del proyecto, se exponen las conclusiones finales del trabajo teniendo en cuenta las hipótesis y objetivos planteados al principio del mismo. Finalmente, se propone el trabajo futuro por hacer en la búsqueda de mejores resultados en conducción autónoma.

Capítulo 2

Antecedentes

En este capítulo se hace una revisión de los conceptos básicos y fundamentales que serán utilizados a lo largo de este trabajo. Primero, en la sección 2.1, se define el concepto de vehículo sin conductor y se responde el ¿Por qué? es necesario su desarrollo, se toman en cuenta referencias históricas en su evolución y se revisan los conceptos necesarios para que un vehículo sea considerado como autónomo. Además, se describen los niveles de autonomía que puede alcanzar un vehículo que es autónomo en base a las operaciones tácticas y operativas desempeñadas, finalmente se analiza el nivel de autonomía que puede lograr este trabajo. En la sección 2.2, se ofrece una justificación a las preguntas ¿Por qué? y ¿Para qué? usar simuladores, en las subsecciones se dar a conocer conceptos relacionados con simuladores y ejemplos de simuladores dedicados al desarrollo de robótica. Las secciones 2.3, 2.4 y 2.5 están dedicadas a definir conceptos teóricos relacionados con el procesamiento digital de imágenes, aprendizaje no supervisado y máquinas de estados finitos respectivamente. En la sección 2.6 se abordan tecnologías y trabajos relacionados con este proyecto.

2.1. Vehículos sin conductor

Vehículo autónomo, automóvil sin conductor, automóvil con conducción automática o vehículo guiado automatizado son algunos de los nombres que hacen referencia al concepto de conducción autónoma. De manera general, un vehículo autónomo se define como cualquier vehículo de pasajeros que es capaz de conducirse por si mismo. Sin embargo, se requiere de un mecanismo de control bien estructurado para llevar a cabo la tarea de conducción autónoma [27].

Existen registros acerca de la historia de los automóviles autónomos donde se indica un comienzo en los Estados Unidos en la primera mitad del siglo XX, al rededor de los años 1920 debido al constante y fuerte aumento de accidentes de tránsito,se estimaron al menos 200 mil muertes accidentales provocadas por transportes motorizados en aquella época, donde el mayor número de estas defunciones fueron peatones y no conductores. El error humano fue concebido

como la principal causa de dichos accidentes. Sin embargo, el diseño e infraestructura de los autos de la época también fueron factores críticos que influyeron directamente en la forma y gravedad de los percances viales, fue entonces que surgió la idea de sustituir a humanos propensos a errores con tecnología que permitiera un mejor desempeño.

De esta manera inició el desarrollo de nuevas tecnologías que permitieran un acercamiento a la conducción autónoma. Algunos desarrollos previos tomaron mayor fuerza. Lawrence B. Sperry usó el primer giroscopio estabilizador de avión en junio de 1914, este es considerado hasta la actualidad como el primer piloto automático. Otro de los adelantos tecnológicos de la época fueron las ondas de radio, esta nueva ciencia de radio-guía se vio comprometida con el control remoto de mecanismos en movimiento a través de ondas de radio. Esta tecnología fue desarrollada, entre otros, por el ejercito de los estados unidos, con el objetivo de controlar misiles, barcos y aviones a distancia. Para el año de 1921 estos trabajos pioneros vieron como resultado el primer vehículo sin conductor. Ingenieros del *Radio Air Service* presentaron al público un automóvil de 2.5 metros de largo que fue controlado por medio de ondas de radio desde un camión del ejército que conducía 30 metros por detrás del vehículo. En el estricto sentido no se trataba de un vehículo con conducción autónoma, sino de uno controlado a distancia con el conductor fuera del auto. Una situación que resulta digna de mención es que la historia y los fundamentos de los autos sin conductor esta fuertemente relacionada con el ejército y los avances impulsados por conflictos bélicos.

En la actualidad y de acuerdo con datos de la OMS en promedio suceden 1.24 millones de defunciones en el mundo relacionadas con accidente viales por año[18]. Una gran parte de estos accidentes son ocasionados por fallos humanos debido a que la cantidad de asistencia que requiere un conductor o usuario es muy demandante, pues se enfrenta a actividades exhaustivas como tráfico intermitente, tramos largos en carreteras, estar bajo influencia de medicamentos o algún otro agente y cansancio excesivo, son algunos factores que al final terminan con cualquier placer de conducir. En cualquiera de estos casos, la capacidad de un vehículo autónomo abre nuevas oportunidades a una mejor movilidad y optimización en el flujo de tráfico.

2.1.1. Aspectos necesarios para un vehículo autónomo

Uno de los principales retos de los vehículos autónomos es disminuir el número de accidentes vehiculares, pues ya se han mencionado pruebas evidentes para afirmar que la mayoría de los accidentes automovilísticos son ocasionados con mayor frecuencia por errores humanos. Además, prometen numerosas mejoras para el tráfico vehicular, aumento en la capacidad de las carreteras y mejoras en el flujo de tráfico debido a tiempos de respuesta más rápidos, también se estima menor uso de combustible y disminución de emisiones contaminantes como consecuencia de una conducción más previsora pero sobre todo supervisada por agentes inteligentes.[20]

Una plataforma robusta de un vehículo autónomo es esencial para lograr la operación del mismo en condiciones reales y ambientes urbanos. Principalmente

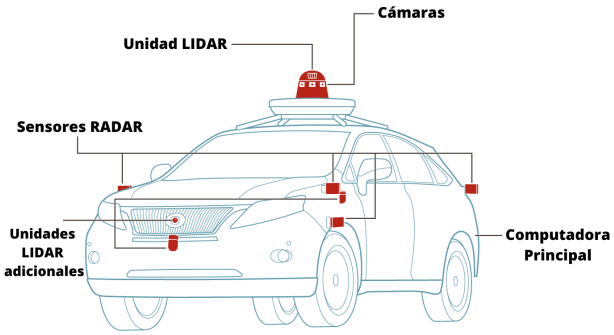


Figura 2.1: Hardware dedicado a vehículos autónomos.
Nota. Adaptada de *Software Engineering Daily*, 2018 [3].

esta plataforma está compuesta de:

1. **Hardware**: Representado por toda la arquitectura física para reconocer el ambiente y actuadores capaces de interpretar información de los sensores, la figura 2.1 ilustra esto. Un hardware dedicado a vehículos autónomos consta de tres bloques esenciales.
   - Sensores: Para percibir el entorno y sus movimientos, es común el uso de cámaras estéreo, RGB y RGB-D, dispositivos para realizar mediciones de rango como sensores RADAR y LIDAR, GPS para información de odometría.
   - Actuadores: Permiten realizar acciones de control en el vehículo
   - Ordenadores: Para procesar entradas y generar salidas.
2. **Software**: Son todos aquellos módulos lógicos que hacen uso de la información de sensores, se encargan de procesarla mediante algoritmos específicos [19]. Entre las principales tareas que realiza el software de un vehículo autónomo destacan.
   - Mapeo del ambiente
   - Localización
   - Planificación de rutas
   - Reconocimiento y percepción del ambiente
   - Modelado de leyes de control

2.1.2. Niveles de autonomía

Conducir un vehículo requiere ejecutar gran variedad de acciones y toma de decisiones ya sea si el vehículo se encuentra en movimiento constante o bien,

estático en una situación de tráfico. El acto de conducir se puede clasificar en tres diferentes tipos de esfuerzo que realiza el conductor: estratégico, táctico y operativo[22]. Los niveles de autonomía de un vehículo motorizado sin conductor se refieren al nivel de ayuda que provee el sistema de automatización con el que está instrumentado. Es decir, el nivel de autonomía es otorgado según el grado de asistencia que se ofrece en la tarea de conducción dinámica.

El término Tarea de Conducción Dinámica o DDT (*Dynamic Driving Task*) por sus siglas en inglés es el concepto que se atribuye a la suma de esfuerzos o funciones tácticas y operativas ejecutadas en tiempo real necesarias para un vehículo en carretera. Otras subtareas como la selección de destinos, programación de viajes y puntos de referencia están excluidas del concepto de DDT porque pertenecen al grupo de funciones estratégicas. Los esfuerzos operativos y tácticos son complementarios, en conjunto resultan como la acción de operar un vehículo[15]. De acuerdo con la clasificación de Michon en 1985, el esfuerzo estratégico implica la planificación del viaje, decidir cuándo y dónde ir, cómo viajar, además de elegir la mejor ruta. El esfuerzo táctico incluye técnicas de maniobra durante el trayecto del viaje, esto incluye la toma de decisiones de cómo y cuándo se puede rebasar a otro vehículo, cambiar de carril, mantener velocidad adecuada, revisión de retrovisores laterales y superiores. La última categoría es el esfuerzo operativo que hace referencia a aquellas acciones consideradas innatas o constantes, como es realizar pequeñas correcciones del ángulo de dirección con el volante para mantener el vehículo estable, frenar y acelerar para evitar obstáculos repentinos o eventos peligrosos en el camino[22].

Ejemplos de funciones operativas y tácticas son:

1. Operativas
   - Control de movimiento lateral por medio de la dirección (*Steering*).
   - Control de movimiento longitudinal a través de aceleración y desaceleración.
2. Tácticas
   - Planificación de maniobras.
   - Evasión de obstáculos y seguimiento de rutas.
3. Operativas y Tácticas
   - Supervisión del ambiente de conducción: detección, reconocimiento y clasificación de objetos.
   - Ejecución de respuesta ante eventos y objetos (OEDR).

La Sociedad de Ingenieros Automotrices (SAE) ha desarrollado un estándar que define la escala de niveles de autonomía. Esta escala determina 6 niveles de autonomía para conducción automática, desde el nivel 0 (Sin Conducción Automática) hasta el nivel 5 (Conducción Automática Completa)[15]. En breve, se enuncian y describen los 6 niveles propuestos por la Sociedad de Ingenieros Automotrices (SAE).

- **Nivel 0: *No Driving Automation***

  El nivel 0 implica que el conductor es completamente responsable del control de vehículo, es decir, se encarga de las funciones tácticas y operativas. Aunque los vehículos de nivel 0 cuenten con características de seguridad que ayuden a los conductores como: advertencias de colisión y punto ciego, frenado automático de emergencia e incluso cámaras de retroceso, se clasifican dentro del nivel 0 porque ninguna de estas características actúan durante un periodo prolongado.

- **Nivel 1: *Driver Assistance***

  En el nivel 1 el sistema de conducción automatizado (ADS) toma control del vehículo en situaciones muy específicas. Tal es el caso de control de movimiento lateral y longitudinal, pero no ambas simultáneamente. Esto quiere decir que se mantiene la expectativa de que el conductor realice el resto de la DDT. Un ejemplo es el sistema ACC (*Adaptative Cruise Speed*) que controla la aceleración y el frenado, generalmente en conducciones largas sobre carretera.

- **Nivel 2: *Partial Driving Automation***

  Este nivel de autonomía presenta mejoras en el reconocimiento del entorno y permite al vehículo realizar acciones más complejas. En el caso de control longitudinal para aceleración y frenado, para control lateral ayuda en la corrección de dirección.

  De manera general, el nivel 2 provee ayuda por parte del sistema de conducción automatizado (ADS) para cumplir un porcentaje de la tarea de ejecución de respuesta ante eventos y objetos para completar acciones de control en movimiento lateral y longitudinal con asistencia limitada. Existen eventos ante los cuales el sistema no es capaz de responder y por lo tanto, el conductor debe de supervisar el desempeño de la asistencia proporcionada con el propósito de completar la conducción dinámica (DDT).

- **Nivel 3: *Conditional Driving Automation***

  El nivel 3 es considerado el punto de entrada a la conducción autónoma, pues en situaciones muy específicas con cierto tipo de carreteras y condiciones climáticas adecuadas el conductor comienza a desconectarse del acto de conducir.

  El desempeño por parte del sistema de conducción automatizado es considerablemente mejor, pues toma el control de todas las acciones que permiten completar la tarea de conducción dinámica (DDT). En este nivel aún se mantiene la expectativa de que el conductor este listo ante las solicitudes que pueda requerir el ADS para retomar el control del vehículo en cualquier momento. Un ADS de este nivel permite completar un conducción dinámica en condiciones específicas, por ejemplo en autopistas con velocidad moderada y manejo de frenos. En situaciones de tráfico intermitente el vehículo envía alertas al conductor para que retome el control.

- **Nivel 4:** ***High Driving Automation***

  En este nivel el sistema de conducción automatizado es completamente capaz de vigilar el entorno de conducción y ejecutar funciones tácticas y operativas. El vehículo también cuenta con la capacidad de alertar al conductor si es que los límites operativos se sobrepasan, en caso de no encontrar respuesta por parte del conductor el vehículo debe asegurarse automáticamente. En otras palabras, un usuario dentro de un vehículo adaptado con características de nivel 4 es un pasajero que no necesita responder ante fallas del sistema.

  El cumplimiento de condiciones de riesgo mínimo en adición con la capacidad de recuperación automática resultan ser la principal diferencia que ofrece el ADS de nivel 4 frente a uno de nivel 3. Vehículos adaptados con condiciones de nivel 4 son capaces de seguir una ruta predeterminada dentro de regiones geográficas limitadas; por ejemplo: circuitos cerrados, campus escolar o una base militar.

- **Nivel 5:** ***Full Driving Automation***

  Los vehículos adaptados a un nivel 5 de autonomía son completamente independientes, es decir, el usuario no necesita supervisar el sistema de conducción automatizado. Esto significa que el ADS puede controlar el vehículo dentro de cualquier carretera en cualquier lugar del mundo bajo las mismas condiciones en las que un humano lo pueda hacer.

  Esto quiere decir que no existen restricciones climáticas, geográficas o de horario donde pueda operar. Sin embargo, esto no significa que un vehículo del nivel 5 pueda actuar en situaciones donde resulte imposible la tarea de conducción dinámica incluso para los humanos; por ejemplo en tormentas de nieve, derrumbes o inundaciones. En tales casos el ADS debe de lograr una condición de riesgo mínimo, deteniéndose o esperando a que las condiciones cambien. Un vehículo en este nivel puede ser programado con un punto de inicio y un punto de destino siendo capaz de cubrir el trayecto completo en vías públicas independientemente de las condiciones de tráfico, carretera y o clima.

A partir del nivel 0 y hasta el nivel 3 se entiende que el conductor tiene mayor grado de responsabilidad sobre el manejo del vehículo. El conductor debe de supervisar constantemente las funciones de apoyo con las que cuente el vehículo, según el nivel. Sin embargo, debe de mantenerse atento para frenar, acelerar y dirigir según sea necesario. Un sistema de conducción automatizado (ADS) de estos niveles cuenta con características que permiten proporcionar asistencia y advertencias momentáneas en el caso del nivel 0 por medio de frenado de emergencia automático (AEB) y advertencia de punto ciego. En los niveles 2 y 3 existen características que ayudan en acciones de control lateral y longitudinal del vehículo como: sistemas ACC y centrado de carril [15].

Los niveles 3, 4 y 5 brindan un mayor grado de asistencia y en consecuencia producen una disminución en las tareas que debe desempeñar el conductor. Para

el caso de los niveles 3 y 4 la conducción autónoma es limitada y restringida, el usuario solo retoma el control del vehículo si el sistema de conducción lo solicita. El nivel 5 de autonomía permite navegación de manera completa en cualquier lugar y sin restricciones, el ADS de este sistema esta conformado por una combinación y mejora de cada uno de los componentes usados en niveles inferiores[15].

2.1.3. Nivel de autonomía esperado en este trabajo

En este trabajo se pretende alcanzar un nivel 4 de autonomía, con el fin de lograr este nivel se planea instrumentar un vehículo(simulado en 3D) con sensores que le permitan percibir el ambiente de conducción. LIDAR, cámaras RGB y GPS ayudarán en la estimación de objetos para resolver acciones de rebase y seguimiento de carriles.

También se buscan desarrollar las leyes de control correspondientes para el desempeño de tareas operativas y tácticas de conducción dinámica (DDT). Es considerado como nivel 4 por las características que posee el sistema de conducción automatizada (ADS), además de que la intervención humana es casi nula, pues al ser un ambiente simulado no se cuentan con estas características. Funciones estratégicas quedan fuera del alcance de este trabajo.

2.1.4. Sensores

- **LIDAR**: *Light Detection and Ranging* por sus siglas en inglés es una tecnología óptica utilizada para detección de objetos, es decir, permite medir distancia y otras propiedades de un objeto objetivo que es iluminado por un haz de luz, a menudo se hace por medio de pulsos láser. Luz ultravioleta, visible o infrarroja son ejemplos de luz que puede usar un sensor LIDAR. Objetos no metálicos, rocas, compuestos químicos, sólidos, nubes e incluso moléculas individuales son ejemplos de objetos que puede detectar este sensor [27]. El principio fundamental de el funcionamiento del LIDAR es emitir un pulso de luz hacia el objetivo y por consecuencia activar un circuito interno de temporización. Internamente se calcula el tiempo que le toma al láser llegar al objetivo y regresar al receptor desde el objetivo, este proceso obtiene la distancia a la que se encuentra el objeto objetivo. Sin embargo, la información que se presenta al usuario puede no estar en un rango real debido a diferentes factores externos, por ejemplo; el ruido inherente del ambiente.

- **Cámaras RGB**: Uno de los sensores más complejos utilizados en la robótica son las cámaras digitales. Las cámaras digitales son capaces de capturar y almacenar imágenes digitalmente. De este modo, una cámara RGB es capaz de medir la capacidad de luz dentro de un espectro visible, es decir, el mismo espectro que pueden ver los ojos humanos. Mediante un cámara RGB se pueden captar e interpretar la gama de colores que percibimos los humanos. A diferencia de las cámaras en escala de grises,

una cámara RGB agrega una capa de pintura sobre la máscara de píxeles, entonces cada píxel solo registra la intensidad de una determinada componente de color [1].

- **Cámaras Estéreo**: Una cámara estéreo es un tipo de cámara digital que intenta simular la visión humana y cuenta con la capacidad de capturar imágenes tridimensionales. Una cámara estéreo posee dos o más lentes con un sensor de imagen separado para cada lente, la distancia entre lentes es aproximadamente la distancia promedio en los ojos humanos. Su funcionamiento se basa en tomar una imagen por cada sensor en el mismo instante, luego mezclarlas y obtener como resultado un imagen 3D. El uso de cámaras estéreo es amplio en el sector tecnológico con diferentes campos de aplicación como: vehículos autónomos, astronomía, medicina, topografía, entre otros.

- **Cámaras RGB-D**: Las cámaras digitales convencionales producen imágenes como una matriz bidimensional. Por lo general, cada píxel tiene valores asociados al rojo, verde y azul, o RGB. Cada atributo va desde 0 hasta 255, por lo que el color negro es una combinación $(0, 0, 0)$ y un rojo brillante puro sería $(255, 0, 0)$. Juntos miles o millones de píxeles crean este tipo de imágenes. Por otro lado, una cámara de profundidad, tiene píxeles con un valor numérico diferente asociado a ellos, este número representa la distancia o profundidad desde la cámara. Existe diferentes métodos para calcular la profundidad, este se elige según la aplicación en turno. Algunas cámaras de profundidad combinan sistemas RGB más un sistema de profundidad y por lo tanto generan píxeles con los cuatro valores mencionados, o RGBD [14].

- **Sonares**: El propósito de los sensores sonares es emitir una señal acústica corta con una frecuencia ultrasónica en un rango de $[50, 250]$ kHz, luego, calcula el tiempo desde la emisión hasta que el eco vuelve al sensor. Este tiempo es proporcional al doble de la distancia del objeto más cercano. Si no se recibe ninguna señal de vuelta en un lapso de tiempo, entonces ningún objeto es detectado en la distancia correspondiente [1]. Durante décadas, los robots móviles han sido instrumentados con diferentes sensores para navegación, debido a esto, los sensores para medir distancias se encuentran entre los más importantes en robótica. A pesar de lo poderoso que son los sonares en detección de objetos, también poseen desventajas, por ejemplo; interferencias y reflejos. La interferencia ocurre cuando se operan varios sensores de sonar a la vez, por otro lado, los reflejos pueden implicar que un obstáculo parezca estar más lejos de lo que en realidad está.

- **GPS**: El sistema de posicionamiento global o GPS de sus siglas en inglés es un sistema de navegación que usa satélites, un receptor y algoritmos para sincronizar información de ubicación, velocidad y tiempo. Se compone de tres componentes diferentes, conocidos como segmentos (satélites,

control terrestre, equipo de usuario) y trabajan conjuntamente para proporcionar información de ubicación. El conjunto de satélites utilizado por los GPS consta de una constelación con 24 satélites repartidos en 6 planos orbitales alrededor del planeta Tierra. Solo son requeridos tres satélites para producir una ubicación en la superficie terrestre, sin embargo, a menudo se usa un cuarto satélite para validar información de los otros tres. Además, el cuarto satélite permite calcular la altitud de un dispositivo [9].

Actualmente el uso de GPS está presente en la industria de localización, navegación, seguimiento de objetos, creación de mapas del mundo, entretenimiento, vehículos autónomos, seguridad y comunicaciones móviles.

2.2. Simuladores

El desarrollo de nuevas tecnologías y agentes inteligentes se encuentra en constante crecimiento a nivel mundial y los vehículos autónomos no son la excepción, debido a la creencia de que los vehículos autónomos pueden ayudar e incluso mejorar tareas de conducción dinámica, comodidad y calidad en viajes. Sistemas inteligentes como ACC, LKA, CA, DN ofrecen mayor eficiencia y confianza en condiciones normales de manejo, en situaciones de tráfico mejoran el rendimiento de combustible y mantienen un ambiente seguro.

El creciente interés en el área de vehículos inteligentes requiere la innovación de más y mejores componentes que instrumenten a un vehículo que pretende ser autónomo. Una gran variedad de sensores, redes de sensores, tecnologías de comunicación, poder de cómputo y controladores requieren de constantes mejoras y actualizaciones en hardware y software[6]. En este caso los simuladores son un gran aliado en el diseño y testeo de estos componentes.

2.2.1. ¿Por qué usar simuladores?

El desarrollo sistemas para agentes inteligentes en vehículos autónomos es un proceso complejo que involucra diferentes componentes tecnológicos, en específico electrónicos y mecánicos que en conjunto resultan ser difíciles y costosos de conseguir en caso de no contar con los recursos necesarios. Esta es la principal razón para considerar el uso de simuladores durante el desarrollo de sistemas dedicados a vehículos autónomos. Los simuladores permiten realizar desarrollos de muy bajo costo en escenarios similares al mundo real. Otra razón de mucho peso es la seguridad y robustez ante fallos, pues al usar un ambiente simulado se tiene mayor margen de maniobra ante situaciones catastróficas que en un ambiente real podrían resultar perjudiciales.

La simulación juega un papel fundamental en la evaluación y validación del desarrollo de sistemas inteligentes. Además, visto desde otro punto de vista los simuladores poseen características que permiten observación de diferentes conductas en situaciones viales para la formulación de nuevas políticas que permitan mejorar la movilidad. Convirtiéndose entonces en una herramienta para investigar escenarios futuros[6].

2.2.2. ¿Para qué usar simuladores?

Como se mencionó anteriormente, cumplir con las condiciones y elementos necesarios para el desarrollo de sistemas inteligentes en vehículos autónomos está lleno de riesgos, es a largo plazo, costoso y complejo. Es por ello que el uso de simuladores se vuelve esencial en este desarrollo.

Los simuladores en este caso ayudan a identificar diferentes aspectos para:

- Conceptualizar la idea principal con base de nociones teóricos y modelos matemáticos.
- Implementar de ideas con tecnologías que permitan describir el comportamiento esperado.
- Desarrollar sistemas completos y creación de prototipos.
- Desarrollar productos finales con base en prototipos, sin dejar de lado el mantenimiento y actualizaciones.

En el caso específico de vehículos sin conductor se buscan simulaciones que permitan diseñar entornos semejantes a condiciones de conducción vehicular en el mundo real. Estos simuladores distinguen diferentes tipos de simulación:

- **Simulación de Tráfico:** Pretende una simulación de todos los sistemas relacionados con el tráfico (control y gestión de tráfico), su propósito es estudiar a los actores involucrados en una red de tráfico (vehículos, conductores, sistemas de tráfico) además de impactos directos en la eficiencia y seguridad del tráfico.
- **Simulación de aplicaciones y vehículos:** Es una simulación con mayor detalle de tráfico e involucra más unidades vehiculares en una red de tráfico. El objetivo de esta simulación es el desarrollo de aplicaciones y evaluación técnica.
- **Simulación de comunicación:** El desarrollo de aplicaciones y evaluación del rendimiento de comunicación entre aplicaciones es el propósito de esta simulación. La simulación de comunicación se puede presentar en diferentes niveles según las necesidades y capacidad de computo disponibles.
- **Simulación de conducción:** Quizás la más importante de las simulaciones es la representación de un conductor, algunas de las funciones no son completamente inteligentes sino que resulta en una cooperación entre conductor y sistema. El objetivo de esta simulación es evaluar el comportamiento de un conductor y el efecto que tiene sobre alguna función específica.

Cada ambiente de simulación tiene sus propias necesidades para modelar y describir la dinámica de los actores presentes. Para simular funciones inteligentes en vehículos se requiere de modelos que describan la dinámica del vehículo,

sensores, actuadores, conductores, rendimiento de comunicaciones y unidades de carretera. Según las necesidades, en algunos casos los modelos pueden ser reemplazados por simuladores dedicados, como el caso de telecomunicaciones a gran escala donde son manejadas por medio de modelos estadísticos[6].

Claramente se puede afirmar que la coherencia entre cada una de las fases de desarrollo de sistemas inteligentes en vehículos es vital con el fin de salvaguardar la portabilidad de resultados. De esta manera se obtiene un marco de trabajo que permite una simulación con diferentes niveles de detalle desde modelos de controladores hasta modelos detallados de sensores y actuadores que al final resultan en procedimientos avanzados de prueba en cada paso de simulación.

2.2.3. Motores de físicas, ODE y otros

Un motor de físicas es un software con la capacidad de realizar simulaciones de comportamientos físicos como: cinemática, dinámica de cuerpos rígidos y blandos, elasticidad, movimiento de fluidos, colisiones. Realizan las operaciones correspondientes para estos comportamientos físicos. Además, manejan los comportamientos de manera independiente del entorno, es decir, reusan código para diferentes situaciones por medio de parámetros. Existe un clasificación para los motores de físicas de acuerdo con la capacidad de procesamiento que requieran: simulación en tiempo real y los de simulación de alta precisión. Las simulaciones de alta precisión requieren de mucha capacidad de computo y generalmente son utilizados con fines científicos, por otro lado las simulaciones en tiempo real se centran más en el campo de los videojuegos.

ODE (*Open Dynamics Engine*) es una biblioteca de gráficos profesional desarrollada en C y C++ y de alto rendimiento para simular dinámicas de cuerpos rígidos. Posee funciones estables, maduras, fáciles de usar y también añade detección de colisiones con fricción. ODE es útil en simulaciones de vehículos, objetos en entornos de realidad virtual y criaturas virtuales. Este motor prioriza la velocidad y estabilidad dejando de lado la precisión física, esto lo hace robusto y confiable pero también poco preciso respecto a otros motores en el mercado. Sin embargo, cuenta con la capacidad de personalizar el entorno incluso durante la simulación [28].

Actualmente es utilizado en la industria de videojuegos de ordenador, creación de modelos 3D y herramientas de simulación. Otros motores de físicas importantes para simulaciones en tiempo real y de alta precisión son:

- **Tiempo real:** PhysX, Vortex, Havoc, SOFA, Box2D.
- **Alta precisión:** VisSim

2.2.4. Bibliotecas para gráficos, OGRE, OpenGL y otros

Una biblioteca de gráficos es un conjunto de programas enfocados a representar gráficos de computadora en un monitor. Puramente esto se puede realizar ejecutando software directamente en el CPU, o siendo acelerado a través de hardware como en el caso de las GPU. Al emplear este tipo de funciones, un

programa ensambla una imagen y posteriormente la envía al monitor. Esto libera al programador de implementar funciones de bajo nivel y concentrarse en especificaciones más concretas de la aplicación. Las bibliotecas de gráficos son utilizadas principalmente en el sector de videojuegos y simulaciones.

1. **OpenGL:** OpenGL es una biblioteca de gráficos para renderizado. En el momento de renderizar OpenGL solo ve una esfera formada por triángulos y un estado de representación. La forma de usar OpenGL es dibujar todo lo que necesita dibujar, luego mostrar una imagen a través de un comando de intercambio de búfer que es independiente de la plataforma. Para actualizar una imagen, se deben dibujar de nuevo todos los componentes, incluso si solo se necesita actualizar una parte del imagen. Para animar objetos en la pantalla es necesario crear un bucle que borre y vuelva a dibujar constantemente en la pantalla. Una de las ventajas de OpenGL es ser multiplataforma y permite a los desarrolladores crear aplicaciones de software gráfico atractivas y de alto rendimiento, principalmente en mercados como CAD, entretenimiento, desarrollo de videojuegos, medicina, realidad virtual, fabricación e investigación científica [25].

2. **OGRE:** *Object-Oriented Graphics Rendering Engine* por sus siglas en inglés es un motor de renderizado 3D de software libre, orientado a escenas y escrito en C++. Las bibliotecas de OGRE evitan la dificultad de utilizar capas inferiores de bibliotecas gráficas como OpenGl y Direct3D, pues provee una interfaz basada en objetos del mundo y otras clases de alto nivel. Específicamente fue diseñado para que a los desarrolladores les resulte más sencillo e intuitivo la producción de aplicaciones gráficas 3D. OGRE actualmente es multiplataforma, y su principal propósito es dar una solución general al renderizado de gráficos con soporte para físicas y audio [24].

2.2.5. Gazebo y sus características

Gazebo es un software 3D de código abierto para simulación de robots con modelado de diferentes componentes y entornos, se incluyen texturas de gran calidad, iluminación y sombras. También, ofrece amplía variedad de sensores y actuadores típicos de robótica así como una simulación eficiente de poblaciones de robots en entornos complejos de interiores y exteriores. Gazebo integra OpenGL para gráficos de alta calidad e interfaces gráficas programáticas convenientes y utiliza un motor de físicas robusto, en este caso ODE.

Características destacadas de Gazebo:

- Modelos de Robots predefinidos y opciones para crear estilos propios.
- Sensores y actuadores con la característica de añadir ruido y obtener simulaciones más reales.
- Control e introspección de las simulaciones a través de línea de comandos.

- Gráficos 3D avanzados y acceso a múltiples motores de físicas de alto rendimiento como: ODE, Bullet, Simbody y DART.
- Simulación en la nube.
- Comunidad ampliamente activa.

2.2.6. Webots y sus características

Webots es un software de escritorio de código abierto utilizado en simulación de robots. Provee un completo entorno para modelado, programación y simulación de robots. Es ampliamente usado en campos de educación e investigación además de ambientes industriales. Webots permite crear una gran variedad de simulaciones para diferentes enfoques como: robots de dos ruedas, brazos industriales, robots modulares, vehículos autónomos y aeroespaciales e incluso drones voladores [33].

Los aspectos que hacen de Webots un simulador flexible y robusto son:

- Motor de físicas ODE y OpenGL como motor de renderizado.
- Es miltiplataforma y disponible para sistemas operativos Linux, Windows y macOS.
- Programación de robots en diferentes lenguajes de programación: C, C++, Python, Java, MATLAB e integración con la plataforma ROS.
- Robusto, determinista y bien documentado.
- Amplía variedad de robots, sensores, actuadores, objetos y materiales.

2.3. Conceptos básicos de visión artificial

2.3.1. Modelo de cámara *pinhole*

El modelo de cámara *pinhole* permite describir la relación matemática que existe entre las coordenadas de un punto con tres dimensiones y su proyección en el plano de imagen, la apertura de la cámara es descrita por un punto infinitesimal-mente pequeño y no utiliza ningún tipo de lente para enfocar luz [7].

Una cámara *pinhole* es la más simple, no tiene lente y solo cuenta con una y muy pequeña apertura, de ahí deriva su nombre, *pinhole*. Esta cámara puede ser vista como una caja a prueba de luz con un pequeño orificio en alguno de sus lados, siendo este orificio la única entrada de luz. En cuanto la luz de una imagen atraviese el orificio se forma una imagen invertida respecto a la original en el lado opuesto de la caja, ver 2.2. A menudo, el modelo *pinhole* es utilizado para describir el cómo una cámara representa un escena en tres dimensiones.

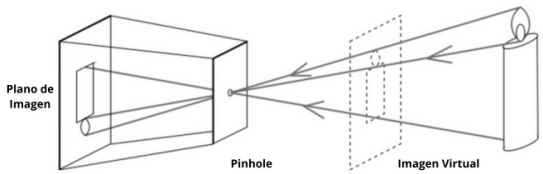


Figura 2.2: Modelo de cámara *pinhole*.
Nota. Adaptada de *Computer Vision Group*, 2010 [16].

2.3.2. Espacios de Color

La idea de estandarizar el concepto de color surge debido a que los nombres de los colores son insuficientes para toda la gama de combinaciones posibles, otro inconveniente es que un porcentaje de la población solo conoce algunos nombres de colores, y en muchas ocasiones asocian un solo nombre a una gran variedad de un color en específico.

Se puede entender que los espacios de color o también llamados modelos de color son formas que indican la manera en que un color está definido, permiten aprovechar y analizar toda la información presente dentro de una imagen. Las leyes de Grassman significan que las mezclas de luces de colores se mezclan linealmente, es decir, son una aproximación precisa y lineal[7].

Cuando los colores se mezclan linealmente se puede construir un algoritmo sencillo para determinar qué pesos de los primarios se deben de usar para conseguir el color base. Dado que la coincidencia del color es lineal, la combinación de los primarios resulta en una suma ponderada de fuentes de longitud de onda única que es afectada por los pesos específicos de coincidencia. Para cualquier conjunto de primarios (P1, P2, P3) se puede obtener un conjunto de funciones de combinación de colores mediante experimentación[7]. La Comisión Internacional de la Iluminación CIE (*Commission Internationale d' eclairage*) ha estandarizado una variedad de sistemas para espacios de color lineales y no lineales.

- **Espacios de Color Lineales:**
  - **CIE XYZ:** Creado por la CIE en 1931, las funciones de combinación de color fueron diseñadas para que el resultado obtenido sea positivo en todas partes, es decir, las coordenadas de luz real fueran siempre positivas.
  - **RGB:** Es un espacio que utiliza formalmente primarios de una sola onda (R = 645.16nm, G = 526.32nm , B = 444.44nm). Los colores en este espacio de color se representan mediante un cubo unitario, donde los bordes del cubo son los pesos específicos de las componentes R, G y B.
  - ***Opponent Color*:** Es un espacio derivado de RGB que busca tres sistemas de color debido a que existe evidencia acerca de que los

primates responden a tres sistemas de color. El más alto responde a la intensidad (Comparaciones claras y obscuras), el segundo compara entre amarillo y azul, el sistema final compara en entre rojo y azul.

- **Espacios de Color No Lineales:** Un espacio lineal no necesariamente codifica propiedades comunes e importantes en aplicaciones reales. Se excluyen propiedades como tonalidad, saturación y brillo que hacen referencia al cambio de pasar de rojo a verde, de rojo a rosa y de negro a blanco respectivamente. Otra dificultad de los espacios de color lineales es dejar de lado las intuiciones humanas sobre la topología de colores, donde los tonos forman un círculo (Rojo, Naranja, Amarillo, Verde, Cyan, Azul, Morado, Rojo). Esto significa que ninguna coordenada de color en un espacio lineal puede modelar la propiedad de tono[7].

  El espacio HSV(Hue, Saturation, Value), matíz, saturación y valor por sus siglas en inglés es un espacio que basado en RGB y usa una función no lineal para representar las propiedades de tonalidad, saturación y brillo.

2.3.3. Imágenes RGB y su representación en memoria

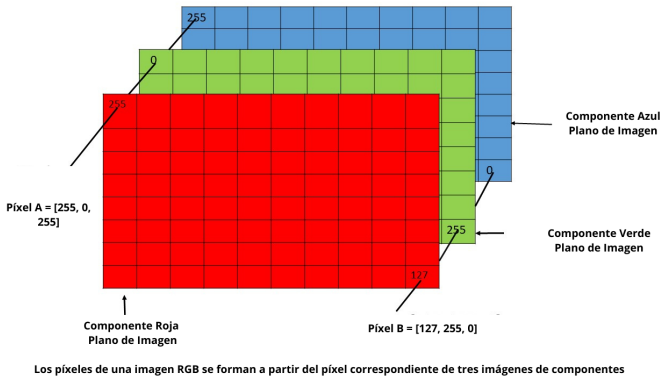


Figura 2.3: Representación de una Imagen en el espacio RGB.
Nota. Adaptada de *RGB image representation*, 2018 [8].

Una imagen RGB se puede expresar como tres imágenes diferentes para cada una de las principales componentes RGB (Rojo, Verde, Azul) apiladas una sobre otra, cuando se introducen en un monitor del tipo RGB se combinan para producir una sola imagen de color compuesta. La cantidad de píxeles utilizados para representar cada píxel dentro del modelo RGB se denomina profundidad de píxel. Tomando como ejemplo una imagen RGB en la que cada una de las imágenes componentes es de 8-bits en cada píxel (cada píxel es una tripleta de valores [RGB]) tiene una profundidad de píxel de 24 bits, es decir, 3 planos multiplicados por el número de bits (8 bits en este caso), ver 2.3.

En la práctica, es común escuchar el término de imagen a todo color para referirse a una imagen con profundidad de 24 bits. El número total de colores posibles en una imagen RGB con una profundidad de 24 bits es de $(2^8)^3 = 16777216$, es decir, más de 16 millones de combinaciones posibles. De manera que, en cada uno de los ejes (R, G, B) se puede seleccionar un punto en el rango [0, 255], donde el color negro se encuentra en [0, 0, 0] mientras que el blanco se encuentra en [255, 255, 255][11], la figura 2.4 muestra esto.

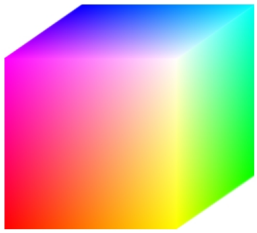


Figura 2.4: Cubo de color RGB con profundidad de 24-bits.
Nota. Tomada de *How to produce RGB cube matrix in python?*, 2014 [29].

2.4. Aprendizaje no supervisado

La agrupación de datos en grupos o *clusters* en inglés es una herramienta de la minería de datos con diversos campos de aplicación como biología, seguridad, inteligencia artificial, robótica e incluso búsqueda web. El análisis de grupos es el proceso de distribuir un conjunto de objetos en subconjuntos. Cada subconjunto es conocido como *cluster*, de tal manera que los objetos resultantes dentro de un subconjunto son similares entre sí pero diferentes a los objetos contenidos en otros. Las similitudes y diferencias de los objetos se evalúan en función de las propiedades de los objetos, a menudo, implican medidas de distancia [12].

2.4.1. Agrupamiento (*clustering*)

El agrupamiento es un tipo de aprendizaje no supervisado porque a diferencia del aprendizaje supervisado los datos de entrada no están etiquetados. El agrupamiento es una forma de aprendizaje por observación contrario al aprendizaje mediante ejemplos en el método supervisado. Uno de los retos más grandes del agrupamiento es adaptarse a diferentes tipos y cantidades de datos. Los requerimientos típicos para agrupación en *clusters* dentro de minería de datos son:

- **Escalabilidad:** Necesaria para que algoritmos puedan funcionar de forma aceptable en pequeños y grandes bases de datos.
- **Habilidad para tratar con diferentes tipos de atributos:** Muchos algoritmos son diseñados para datos numéricos. Recientemente más y más

aplicaciones en el campo de la informática requieren de técnicas de agrupamiento para tipos de datos más complejos como grafos, imágenes y documentos.

- **Descubrimiento de *clusters* en forma arbitraria:** Es importante desarrollar algoritmos que puedan detectar *clusters* de forma estándar, diferentes algoritmos determinan *clusters* en base a mediciones de distancia Euclidiana o distancia de Manhattan y centran sus esfuerzos en formar grupos esféricos con tamaño y densidad similares. Sin embargo, un grupo puede tener cualquier forma.
- **Habilidad para tratar con datos ruidosos:** Muchos conjuntos de datos reales pueden contener valores atípicos, erróneos o desconocidos. Lecturas de sensores, por ejemplo, a menudo son ruidosas y pueden producir valores incorrectos. Los algoritmos de agrupamiento tienden a ser sensibles al ruido de estas lecturas y pueden generar mala calidad en resultados finales. Por lo tanto, se requiere de métodos robustos ante el ruido.
- **Capacidad para agrupar conjuntos de datos con dimensiones grandes:** Muchos algoritmos de agrupación generan resultados rápidos en conjuntos de datos dimensionalmente pequeños, caso contrarío con lo que sucede con conjuntos más amplíos donde el desempeño de los algoritmos se ve comprometido.
- **Interpretabilidad y usabilidad:** Es importante estudiar cual será el campo de aplicación del agrupamiento para realizar una buena elección en el algoritmo a implementar y al final del proceso se obtengan resultados entendibles, usables y con valor para la aplicación.
- **Medida de similitud:** Las medidas de similitud juegan un papel importante en el diseño de métodos de agrupamiento. Los métodos basados en distancia a menudo pueden aprovechar técnicas de optimización en el desarrollo, mientras que métodos basados en densidad o continuidad por lo general pueden encontrar grupos con formas arbitrarias.

Por lo general, se utiliza el aprendizaje no supervisado para descubrir automáticamente diferentes tipos clases dentro de un conjunto de datos, esta es una clara ventaja del aprendizaje no supervisado frente a otro tipo de aprendizaje porque un *cluster* de objetos puede ser entendido como una clase implícita, pues los elementos de un *cluster* son similares entre si pero diferentes respecto a objetos de otros *clusters* [12].

2.5. Máquinas de estados finitos

2.5.1. Definición

Las máquinas de estados finitos o FSM (*Finite State Machine*) por sus siglas en inglés son el corazón en la mayoría de los diseños digitales. La idea principal

de una FSM es almacenar un conjunto de diferentes estados únicos y realizar transición entre ellos con base en los valores de entrada y el estado actual de la máquina. Una máquina de estados finitos puede ser de dos tipos: Moore y Mealy [34]. La primer forma específica que, la salida de la máquina de estados depende puramente de las variables de estado. Por otro lado, en una máquina Mealy la salida puede depender de las variables de estado actuales y valores de entrada a la máquina. El método para describir una FSM es utilizar un diagrama de transición de estados. Este diagrama muestra los estados, salidas y condiciones de transición.

Una alternativa al uso de máquinas de estados finitos es utilizar una técnica llamada Máquina de Estados Algorítmicos (ASM) o carta ASM, esta técnica es útil porque permite separar la ruta de datos y control en un diagrama simple. Una carta ASM es similar a un diagrama de flujo, con acciones y decisiones que pueden ser combinatorias y secuenciales. La diferencia entre FSM Y ASM es la manera gráfica en que se expresan, la forma gráfica de una carta ASM permite definir explícitamente estados, decisiones y resultados, con apariencia similar a un diagrama de flujo. Esto hace que el algoritmo sea más entendible que un diagrama de transición de estados. Una carta ASM tiene tres bloques principales, en 2.5 se ilustran.

- **Estado:** Es representado por una caja rectangular, contiene el valor del estado actual, como un nombre o valor numérico que representa la variable de estado. También puede contener el valor de las salidas según el tipo de máquina de estado (Mealy o Moore).

- **Decisión:** El bloque de decisión indica diferentes caminos condicionales dependiendo del valor de las variables de entrada. Este bloque es representado en forma de diamante.

- **Salida:** Una salida puede ser condicional o incondicional. Donde; una salida es condicional cuando es precedida por un diamante de decisión (Mealy), por el contrario una salida condicional está presente en un estado sin decisión previa (Moore).

2.5.2. Formas de implementarlas

Tanto las máquinas de estados finitos (FSM) y las máquinas de estados algorítmicos pueden ser implementadas en diferentes lenguajes de programación, en específico mediante el paradigma de programación imperativa, es decir, la implementación de la máquina debe consistir en una secuencia claramente definida de instrucciones. Las instrucciones se encadenan una detrás de otra para determinar el comportamiento en cada momento y lograr el resultado deseado. Algunos lenguajes que implementan este paradigma son: Python, C, C++, Ensambladores, Ruby, Pascal.

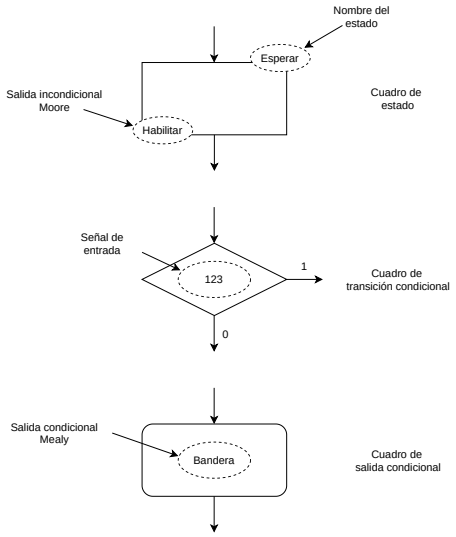


Figura 2.5: Bloques de una carta ASM.

2.6. Trabajo relacionado

2.6.1. Los vehículos Tesla y de Google

- **Tesla:** La compañía Tesla fue fundada en 2003 por un grupo de ingenieros, su objetivo inicial era demostrar que es posible conducir con energía eléctrica y que los vehículos eléctricos pueden ser más rápidos y divertidos de manejar que los vehículos a gasolina. Además, la empresa se enfoca en desarrollar tecnologías que sumen a la conducción autónoma. Tal es el caso de su tecnología Autopilot.

  El sistema Autopilot incluye un sofisticado conjunto de cámaras delanteras, traseras, laterales, amplias y estrechas. En adición con una base de red neuronal profunda, el vehículo deconstruye el entorno para incluir mayor nivel de confiabilidad. En la parte de procesamiento, Autopilot añade una capa de hardware con alta velocidad de procesamiento para ejecutar la red neuronal, que es la base de entrenamiento y desarrollo del Autopilot.

  Este sistema permite navegar en condiciones viales reales y realizar acciones tediosas durante la conducción como: maniobrar, acelerar y frenar automáticamente dentro del carril. Con su característica Autosteer un Tesla puede navegar en calles estrechas y complejas, con Smart Summon el vehículo puede conducir en entornos y espacios de estacionamiento para efectuar un estacionamiento completamente autónomo. Actualmente las funciones de este sistema requieren una supervisión activa del conductor para no dejar que el vehículo sea completamente autónomo. El uso sin supervisión de un conductor depende de que se logre una confiabilidad

superior a las habilidades de conductores humanos, además de demostrarlo con millones de kilómetros recorridos, así como diferentes aprobaciones legales [31].

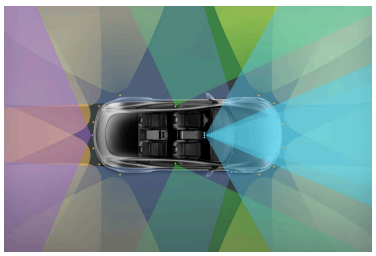
Tesla no solo produce vehículos totalmente eléctricos y tecnología autónoma, también fabrica productos de almacenamiento y generación de energías limpias que pueden ampliarse de manera ilimitada.

- **Google:** Anteriormente conocido como "Proyecto de vehículo autónomo de Google" en 2009, Waymo es una empresa de tecnología dedicada a vehículos autónomos, su misión es lograr que pasajeros y objetos lleguen de forma fácil y segura a su destino. Su tecnología busca mejorar el acceso a la movilidad, mientras intenta salvaguardas vidas que se podrían haber perdido en accidentes vehiculares.

  Waymo enfoca sus esfuerzos en el desarrollo de Waymo Driver, su tecnología de conducción autónoma y se divide en dos partes: hardware y software. La parte de hardware incluye un paquete de instrumentación con sistemas lidar, cámaras, radares y una importante plataforma de procesamiento de inteligencia artificial, en conjunto forman una vista de 360° del ambiente. El software recopila la información de todos los sensores para saber ¿Dónde está?, ¿Qué existe a su alrededor? y ¿Qué debe hacer?. El hardware y software trabaja en conjunto para crear una imagen completa del mundo que rodea al vehículo y navegar de forma segura por las calles [32].

  La flota de vehículos de Waymo incluye al Toyota Prius, vehículos deportivos Lexus, un vehículo prototipo (Firefly) y camionetas híbridas Chrysler, ver 2.6a.

(a) Vehículo Waymo.
Nota. Tomada de *Waymo*, 2022 [32].



(b) Vehículo Tesla con Autopilot.
Nota. Tomada de *Autopilot-Tesla*, 2022 [31].

Figura 2.6: Vehículos Waymo y Tesla.

2.6.2. La categoría AutoModelCar del TMR

El Torneo Mexicano de Robótica (TMR) es la competencia de robótica más importante en México. Tiene como principal objetivo motivar e impulsar la investigación y desarrollo de la robótica en busca de un desarrollo integral de nivel internacional. El TMR incluye diferentes categorías de competición con diferentes enfoques [4].

La categoría AutoModelCar busca incentivar el desarrollo de algoritmos de percepción, planificación y control para vehículos autónomos. Los vehículos no tripulados que participan en esta categoría son modelos a escala 1:10 instrumentados con sensores y actuadores similares a los que tendría un vehículo autónomo en la vida real. Sin embargo, también existe una categoría donde se utilizan simuladores para efectos del vehículo, sensores y actuadores [5]. Esta categoría permite el desarrollo de los mismos algoritmos, además de abrir puertas a la participación a distancia, la figura 2.7 es una representación de un vehículo a escala utilizado en el TMR en su modalidad presencial.

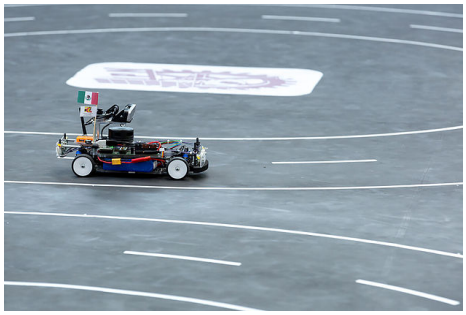


Figura 2.7: Categoría AutoModelCar del TMR.
Nota. Tomada de *TMR-AutoModelCar*, 2022 [5].

2.6.3. El equipo FUB

El trabajo desarrollado por la Universidad Libre de Berlín incluye hardware y software. Cuenta con modelos de vehículos a escala 1:10 y el hardware añade: sensores LIDAR, cámaras RGB con diferentes rangos de visión, puertos USB y adaptadores de red. El software de estos modelos se encuentra sobre la plataforma ROS, este software implementa algoritmos que permiten realizar diferentes tareas de navegación como: detección de líneas de carril, detección de señales de tráfico, control de movimientos, entre otros. Para obtener información más detallada sobre este proyecto visitar [1].

En este capítulo se presentaron en los conceptos teóricos básicos que serán necesarios a lo largo de este trabajo. Se dio a conocer el término de vehículo autónomo y las características necesarias que requiere para ser considerado como

[1] https://github.com/AutoModelCar/AutoModelCarWiki/wiki

tal, con base en ello se indicó el nivel de autonomía que es pretendido alcanzar al finalizar este proyecto.

En los capítulos siguientes se abordaran de manera específica y detallada muchas de las definiciones aquí mencionadas, también, se expondrán implementaciones de las mismas en aplicaciones para vehículos sin conductor.

Capítulo 3

Simulación con Webots

Webots es el simulador utilizado en el resto de este trabajo para mostrar el funcionamiento de los diferentes algoritmos desarrollados, este capítulo expone sus cualidades. En la sección 3.1, se describen las principales características que posee, los lenguajes y plataformas compatibles con el software. También, se definen conceptos esenciales para el desarrollo con el simulador como: mundo, controladores y controladores supervisores. La sección 3.2 explica la forma de integrar el formato *Open Street Maps* con Webots en la búsqueda de crear escenarios de prueba más realistas. Por otro lado, la sección 3.3 presenta las necesidades requeridas en este proyecto, se listan las características del ambiente simulado y del vehículo de pruebas, a continuación, se muestran los modelos finales del vehículo y escenario de pruebas. La sección 3.4 da a conocer las contribuciones de este trabajo en el TMR-AutoModelCar 2022.

3.1. Conceptos básicos del simulador Webots

Webots es un software multiplataforma de código abierto enfocado a simulación de robots móviles, el ambiente de Webots provee un entorno de desarrollo completo que facilita la programación, modelado y simulación de robots de manera profesional. También, le permite al usuario crear entornos virtuales 3D con propiedades físicas importantes en el comportamiento de robots, tal es el caso de propiedades como: masa, coeficiente de fricción, articulaciones, entre otros[33].

En el simulador se pueden encontrar objetos pasivos y activos también llamados robots móviles, estos pueden ser equipados con una gran variedad de sensores y actuadores que contiene el propio simulador, donde destacan: sensores de distancia, sensores láser, motores, cámaras, ruedas, gps, emisores y receptores. Además ofrece al usuario la facilidad de programar de manera individual cada robot creado con el fin de lograr el comportamiento deseado para cada uno. La figura 3.1 ilustra la interfaz de usuario de Webots.

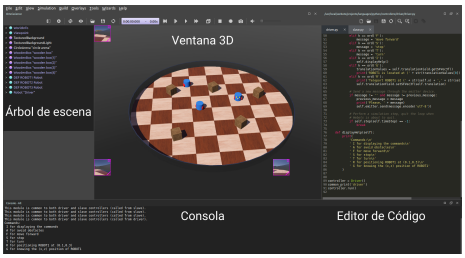


Figura 3.1: Interfaz de Usuario Webots.

3.1.1. Lenguajes y Sistemas Operativos

Al ser un software de código abierto Webots puede ser utilizado en diferentes plataformas de escritorio como: Windows, Linux y macOS. Para cada una de estas plataformas se encuentra disponible la documentación correspondiente, además existe una comunidad muy activa en cada una de sus variantes. Sin embargo, las distribuciones Linux son las que se mayor ayuda proveen, esto debido a la amplia compatibilidad entre Webots y las diferentes herramientas en el desarrollo de robots móviles. Por otra parte, Webots cuenta con soporte para diferentes lenguajes de programación en la creación de programas controladores, entre los lenguajes soportados se encuentran lenguajes compilados como: C, C++ y Java, también permite el uso de lenguajes interpretados como: Python y MATLAB[33].

3.1.2. Controladores, supervisores y mundos

Para crear una simulación en Webots son necesarios dos elementos principales: un mundo y uno o varios controladores.

- **Mundo:** En Webots un mundo es el conjunto de elementos que unido al robot forman una escena 3D o abstracción del mundo real con el fin de simular un ambiente real. Un mundo específica la descripción de cada objeto mediante propiedades como: posición, geometría, orientación, apariencia y propiedades físicas (en caso de ser necesarias). Cada objeto definido puede contener a su vez otro objeto diferente[33]. En la figura 3.2, se observa un mundo de ejemplo creado en Webots.

- **Controlador:** Es un programa que permite ejecutar acciones de control para un robot específico en un mundo específico. Dichos controladores pueden ser desarrollados en los diferentes lenguajes de programación soportados(C, C++, Java, Python, MATLAB). Para cada uno de estos lenguajes se provee la documentación necesaria. Una característica importante de los controladores es que un controlador puede ser utilizado por diferentes robots pero un robot solo puede usar un controlador. Al iniciar una simulación Webots lanza los controladores previamente asignados en cada robot, es decir, para cada controlador existe un proceso independiente[33].

- **Controlador Supervisor:** Este tipo de controladores son una variante especial de los controladores tradicionales, la principal diferencia entre ellos es que los controladores supervisores tienen acceso a operaciones privilegiadas. Por ejemplo; el control de simulación para mover el robot a una posición aleatoria, realizar una captura de vídeo de la simulación, entre otras. Los supervisores también pueden ser escritos en los lenguajes soportados por Webots[33].

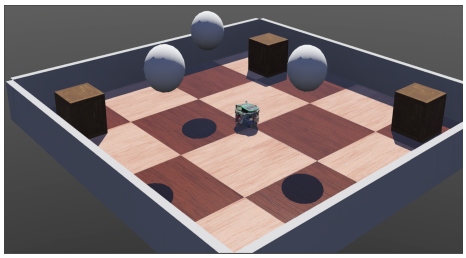


Figura 3.2: Ejemplo de mundo en Webots.

3.2. El formato *Open Street Maps*

3.2.1. Características y especificaciones

El formato *Open Street Maps* es un proyecto de software libre, global y cooperativo para la creación de mapas en base a información geográfica capturada por dispositivos GPS y ortofotografías, al ser un proyecto libre se cuenta con grandes cantidades de información y representación de mapas de casi cualquier sitio del mundo. Con base en el formato *OSM* Webots es capaz de transformar la información del mapa en un ambiente 3D donde los principales objetos son las carreteras, conexiones viales y edificios. Además, mediante la integración de nodos específicos como: vehículos, señales de tránsito, obstáculos y mobiliario urbano se obtiene un ambiente de simulación (mundo 3D) con mayor detalle para el desarrollo de pruebas en conducción autónoma.

3.2.2. Como usarlo en Webots

Con el fin de implementar tareas de vehículos autónomos se requiere de un ambiente de simulación que cuente con las características necesarias y que sean lo más parecidas a un ambiente del mundo real. Para lidiar con este aspecto Webots ofrece una herramienta muy útil, la cual es generar modelos 3D a partir de una sección del mundo real tomada del formato *Open Street Maps*. La manera de integrar OSM con Webots es muy sencilla debido a que desde la instalación inicial del simulador se incluyen las herramientas necesarias. En breve se enuncian los pasos a seguir para crear un escenario 3D inspirado en un mapa real.

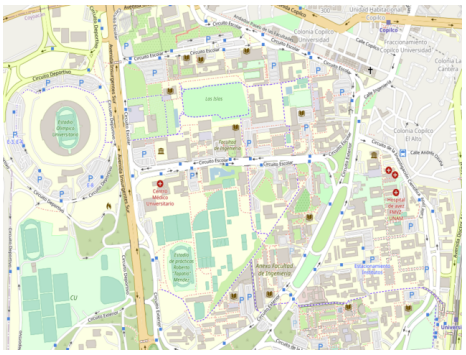


Figura 3.3: Sección de mapa obtenida de *Open Street Maps*.

1. Crear un directorio donde se alojará el proyecto.
2. Obtener y descargar el mapa en formato OSM desde la página oficial de *OpenStreetMap* 3.3 para generar el mundo de Webots.
3. Generar el mundo de Webots a partir del mapa obtenido. Webots cuenta con script "importer.py" que recibe como parámetro un archivo con extensión ".osm" y retorna como salida un archivo con extensión ".wbt". Es necesario ejecutar el script "importer.py" asignando como entrada el mapa previamente obtenido. Si la operación es exitosa se debe localizar un archivo ".wbt" en el directorio creado al inicio. Desde Webots se puede abrir el archivo para su visualización.
4. Entrar a Webots y editar las carreteras, caminos y nombres de los mismos con el fin de eliminar cualquier *bug* producido por "importer.py" debido a la complejidad del archivo ".osm". En esta etapa se pueden agregar más objetos (autos, árboles, señales, edificios, obstáculos) según sean requeridos. La figura 3.4 muestra el resultado esperado de este proceso.

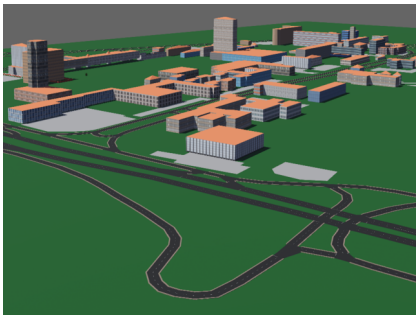


Figura 3.4: Mundo de Webots generado a partir de una sección de mapa en formato ".osm".

3.3. El ambiente de simulación

En la búsqueda de modelar comportamientos de vehículos autónomos reales como: detección y seguimiento de carriles, seguimiento de vehículos, detección y evasión de obstáculos tanto estáticos como dinámicos, es de vital importancia tener un escenario los más parecido posible a un ambiente vial cotidiano. Un ambiente vehicular tradicional es aquel que cuenta con automóviles en movimiento y estáticos, señales de tránsito, semáforos, carriles con dirección y espacios definidos. Además de contar con construcciones como edificios, casas, puentes e incluso personas y animales. Cada uno de estos elementos en conjunto forman un ecosistema vial como cualquier otro presente en localidades urbanas.

En este trabajo se pretende modelar un escenario 3D con el simulador Webots que integre las características suficientes para realizar ejercicios de prueba de diferentes comportamientos como:

- Detección de Carril
- Seguimiento de Carril
- Detección de Obstáculos
- Evasión de Obstáculos
- Seguimiento de vehículos
- Acciones de rebase

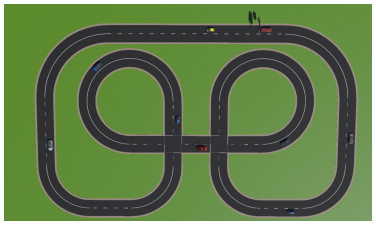
Además de modelar un escenario con los elementos correspondientes se necesita de un vehículo (modelado en 3D) instrumentado con sensores y actuadores suficientes para cumplir tales comportamientos, ejemplos de sensores para instrumentación son:

- Cámaras estéreo, RGB, RGD-D
- LIDAR
- RADAR
- GPS
- Giroscopio

Teniendo en cuenta las características descritas anteriormente se decide crear un escenario que satisfaga las necesidades solicitadas. Dentro de estos elementos se encuentra la carretera como un circuito cerrado. Al ser un circuito cerrado se tienen curvas y rectas, estas son pruebas muy comunes para cualquier vehículo ya sea tripulado o autónomo. La distribución de la carretera se compone de dos carriles en un solo sentido, esta distribución de espacio permite modelar el movimiento de otros vehículos, cada carril cuenta con líneas que delimitan el espacio asignado. Especialmente estas líneas ayudan a completar las tareas de

Figura 3.5: Carriles de la carretera.

detección y seguimiento de carril pues funcionan como guía para el vehículo. La figura 3.5 ejemplifica los carriles de la carretera modelada. Con la adición de otros vehículos dentro de la carretera se busca realizar detección y evasión de obstáculos, seguimiento y rebase de vehículos. El conjunto de estos objetos más otros elementos como: bordes en la carretera, áreas verdes e iluminación forman el escenario final donde se realizarán pruebas de conducción autónoma. Algunos elementos como: personas, mobiliario urbano, edificios, puentes, señales de tránsito y viales no se agregan al escenario ya que no resultan necesarias para este trabajo. El escenario final es el mostrado en 3.6a.



(a) Escenario de pruebas de navegación.

(b) Vehículo de pruebas instrumentado con sensores.

Figura 3.6: Escenario y vehículo de pruebas.

En lo que respecta al vehículo de pruebas Webots incluye modelos de vehículos populares, estos modelos tienen características muy generales como, luces, llantas, tipo de motor, retrovisores, color y otros elementos comunes de un vehículo. Lo importante a destacar es que cada vehículo cuenta con secciones predefinidas para instrumentar el vehículo con los sensores necesarios. Para los propósitos de este proyecto se utiliza un vehículo 'X5' de 'BMW' en color rojo. A este vehículo se le añaden los siguientes sensores:

- Cámara RGB, colocada en la parte frontal del vehículo.
- Lidar, colocado en la parte superior del vehículo.
- GPS, colocado en la parte frontal y superior del vehículo.

El vehículo para pruebas de navegación instrumentado con los sensores y actuadores necesarios se encuentra ilustrado en la figura 3.6b.

Es es importante mencionar que el vehículo de pruebas 'BMW X5' es manipulado a través de un controlador, este se encarga de recibir información de cada sensor. Además, el mismo controlador envía señales para el control de velocidad, ángulo de dirección entre otros. En los próximos capítulos se ofrece una descripción con mayor detalle de cada uno de los nodos utilizados. En cuanto a los vehículos en movimiento dentro de la carretera se emplea un controlador supervisor para manipular la posición y velocidad de los vehículos.

3.4. Contribuciones en el TMR

El ambiente de simulación y vehículo de pruebas instrumentado fueron de ayuda en el desarrollo de la plataforma de uso para las pruebas consideradas en el TMR- AutoModelCar 2022 en su modalidad virtual. El uso de este simulador permitió generar los ambientes de simulación para las pruebas de:

- Navegación autónoma sin obstáculos.
- Navegación autónoma con obstáculos estáticos.
- Navegación autónoma con obstáculos dinámicos.
- Estacionamiento autónomo.

Con ayuda del desarrollo de controladores supervisores se logró el control de posición y velocidad de los diferentes autos que actuaron como obstáculo en cada prueba. El desarrollo del controlador principal del vehículo de pruebas permitió el control de movimiento del mismo en base a los sensores integrados.

Al termino de este capítulo se entienden los conceptos básicos, ventajas y desventajas del simulador Webots. Además, se conoce el enfoque que se le dará al entorno de simulación en la búsqueda de implementar y probar algoritmos de navegación autónoma. Los escenarios modelados así como el vehículo de pruebas instrumentado en este capítulo son parte esencial en los capítulos siguientes, pues permitirán evaluar el desempeño de los algoritmos desarrollados y el comportamiento del vehículo en las diferentes situaciones que se le presenten.

Capítulo 4

Seguimiento de carriles

Este capítulo se enfoca en hacer un primer reconocimiento del ambiente, se buscan identificar los carriles de la carretera, en especial, el carril donde se encuentra el vehículo autónomo. Al inicio, la sección 4.1 realiza una introducción para el proceso de seguimiento de carriles. A continuación, en 4.2 se define el concepto de espacio de color y algunas de las variantes más utilizadas, en la subsección 4.2.1 se muestra una forma de procesar una imagen RGB con la plataforma ROS. Enseguida, las secciones 4.3 y 4.4 tratan de manera teórica los conceptos de detección de bordes de Canny y transformada Hough. Sus correspondientes subsecciones 4.3.2 y 4.4.2 son ejemplos de como implementar estas herramientas matemáticas en código de programación para aplicaciones de vehículos autónomos. Posteriormente, la sección 4.5 es una muestra del modelo cinemático del vehículo, siguiendo este modelo la sección 4.6 diseña las leyes de control correspondientes para el movimiento lateral y longitudinal del vehículo. Finalmente, se presentan los resultados de estas implementaciones.

Cabe mencionar que todas las implementaciones en código de algoritmos y pseudocódigos propuestos en este capítulo son desarrollados con el lenguaje de programación Python.

4.1. Introducción al seguimiento de carriles

El primer paso para cumplir con la tarea de navegación autónoma es el seguimiento de carriles, en un ambiente vial urbano los carriles son el principal indicador para mantener una ruta. Bajo este escenario, es fundamental contar con un instrumento que permita simular en medida de lo posible la visión humana, el sensor más cercano a este efecto es una cámara, con ella se puede conocer una parte del ambiente donde navegará el vehículo. Tal como se describió en la sección 3.3 del capítulo 3, el vehículo autónomo modelado está instrumentado con una cámara digital en la parte superior para tener una imagen RGB frontal del ambiente. La imagen obtenida de la cámara es la principal fuente de información para supervisar el frente de la escena, está información (imagen)

es procesada con el fin de identificar el carril donde debe conducir el vehículo autónomo y en base a ello calcular las leyes de control que permitan manipular las funciones operativas (movimiento lateral y longitudinal) del vehículo para mantenerse dentro del carril correspondiente en líneas curvas y rectas de la carretera. Para que el vehículo autónomo sea capaz de navegar bajo estas condiciones es recomendable separar el proceso en tareas más específicas, en concreto dos: Detección de carril y Seguimiento de Carril.

El reconocimiento del ambiente hace referencia a la detección del carril en el que debe mantenerse el vehículo, para realizar esta tarea se llevan a cabo los siguientes pasos:

1. Obtener la información (imagen) de la cámara del vehículo.
2. Determinar características de la imagen que ayuden en la detección del carril.
3. Identificar las líneas que correspondan a los bordes del carril.

La acción de seguimiento de carril se pretende conseguir mediante:

1. Diseño e implementación de un control para velocidad y dirección del vehículo.
2. Garantizar la seguridad del vehículo en todo momento.

En las siguientes secciones de este capítulo se describen conceptos teórico matemáticos necesarios para el procesamiento digital de imágenes en la búsqueda de detección de carriles en la escena, además se ejemplifican los casos prácticos de cada una de las herramientas empleadas.

4.2. El espacio de color RGB

Dentro del procesamiento de imágenes es de suma importancia el uso de color por dos principales razones. En primer lugar, el color es un descriptor muy poderoso para la identificación y extracción de objetos de una escena. Por otra parte, los humanos podemos distinguir entre cientos de tonos de color pero solo un par de docenas en tonos de grises. Así, el procesamiento de imágenes a color se puede dividir en dos categorías: Procesamiento de Pseudo-color y Procesamiento de color completo. Para el procesamiento por Pseudo-color se emplea la asignación de color a una determinada intensidad de escala de grises mientras que en procesamiento de color completo las imágenes se obtienen con ayuda de algún sensor a todo color(cámara digital, escáner a color) [11].

Un espacio de color es una forma para indicar como está definido un color, el propósito de un espacio de color, modelo de color o sistema de color como también es conocido consiste en facilitar la especificación de colores de una manera estándar. Es decir, es una especificación de un sistema de coordenadas y un sub-espacio dentro del sistema, de tal modo que cada color está representado por un solo punto contenido dentro del sub-espacio. En la práctica de

procesamiento de imágenes los modelos de color más utilizados son el modelo RGB (Red, Green, Blue) para monitores de color, cámaras de vídeo; los modelos CMY (Cyan, Magenta, Yellow) y CMYK (Cyan, Magenta, Yellow, Black) son muy usados dentro de la industria de impresión de color. El modelos HSI (Hue, Saturation, Intensity) es el modelo que más se asemeja a la forma en que los humanos interpretamos el color, este modelo tiene la ventaja de poder desacoplar el color y el gris(escala de grises de una imagen) y por lo tanto es bastante común en el uso de técnicas de escala de grises[11].

El modelo de color RGB se basa en un sistema cartesiano de 3 coordenadas y el sub-espacio de interés es un cubo unitario. Es decir, se indica que todos los valores R, G y B se encuentran en un rango de [0, 1]. Dentro del espacio RGB, cada color aparece en su componente espectral primaria(R = 645.16nm, G = 526.32nm , B = 444.44nm)[19]. El color negro se encuentra en el origen del cubo mientras que el blanco está situado en el vértice más lejano, los valores primarios (RGB) se sitúan en tres esquinas del cubo y los colores secundarios aparecen en las tres esquinas restantes. Los diferentes colores sobre o dentro del cubo están definidos por vectores que comienzan desde el origen. Además, la escala de grises (puntos donde los valores RGB son iguales) se extiende de negro a blanco a través de la línea que une estos dos vértices. La figura 4.1 muestra la representación del espacio de color RGB.

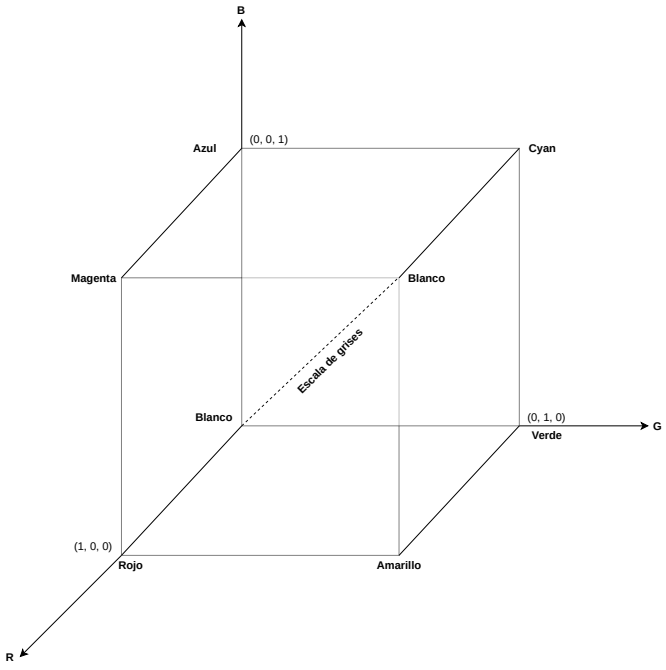
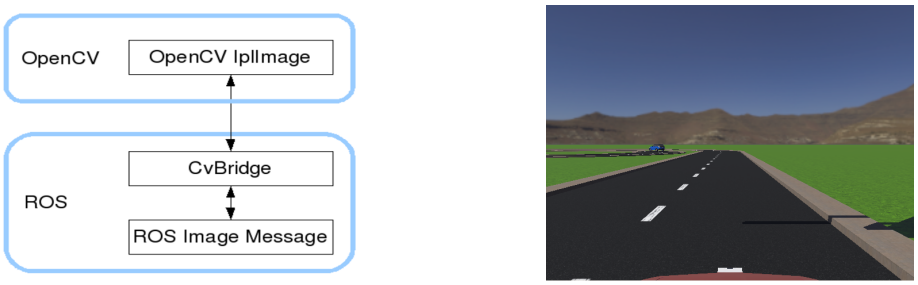


Figura 4.1: Espacio RGB en coordenadas cartesianas.



(a) Mensaje *Image* de ROS a imagen de OpenCV.

(b) Imagen de la cámara en formato de OpenCV.

Figura 4.2: Resultado de transformar un mensaje *Image* de ROS a una imagen de OpenCV.

Nota. (a) Tomada de *Stechschulte*, 2020 [30].

4.2.1. Ejemplo para obtener la imagen de la cámara

La cámara del vehículo provee una imagen en formato RGBA de 8-bits con dimensiones de "640x480" píxeles. A fin de obtener la imagen son necesarias las funciones especificas del nodo "Camera" del simulador Webots, además se hace uso de la plataforma ROS para el manejo de tópicos entre diferentes nodos, en específico se utiliza un mensaje del tipo *Image* de ROS con la intención de publicar un tópico que contenga la información de la cámara.

Para comenzar el procesamiento de la imagen se realiza una conversión de el mensaje *Image* de ROS a una imagen tipo CV2 de OpenCV. Esto porque el mensaje *Image* de ROS contiene la información de imagen en forma de lista donde el tamaño corresponde a el número de píxeles de la imagen (ancho por alto), en este caso el tamaño de la lista es de 640x480 = 307200. Cada uno de los elementos de la lista representa un píxel de la imagen en formato ARGB de 32 bits.

OpenCV es una biblioteca de código abierto para manipulación y procesamiento de imágenes, en adición con la alta compatibilidad entre la plataforma ROS y OpenCV son las principales razones para usar OpenCV dentro de este proyecto pues las funciones que ofrece son suficientes para cada uno de los pasos en el diseño de un algoritmo de detección carriles. Dentro de la plataforma ROS se encuentra un paquete llamado "cv_bridge" el cual permite la conversión bilateral entre mensajes del tipo *Image* de ROS e imágenes en formato OpenCV. La imagen 4.2a ejemplifica la conversión entre estos dos tipos de mensajes.

Referente a la implementación de código, la principal fuente información es proporcionada en la documentación oficial de OpenCV, disponible en [1]. A continuación se muestra el código necesario para el tratado de un mensaje *Image* de ROS y la conversión a imagen de OpenCV. En 4.2b se ilustra el resultado de la imagen obtenida.

[1] https://opencv.org/

```
from cv_bridge import CvBridge

bridge = CvBridge()
cv2_img = bridge.imgmsg_to_cv2(msg, 'bgr8')
```

4.3. Detección de bordes

La detección de bordes es una herramienta en el procesamiento de imágenes diseñada para detectar píxeles de borde. Los píxeles de borde son píxeles donde la intensidad de una imagen cambia abruptamente y los bordes son conjuntos de píxeles entre bordes conectados. Es decir, la detección de bordes se utiliza para segmentar imágenes en función de cambios bruscos de intensidad.

Dentro de este contexto existen diferentes formas de modelar un borde, una muy utilizada clasificación es usar como criterio la intensidad. Este escenario contempla: *Step Edge* o borde de paso, como aquél que se presenta cuando existe una transición entre dos niveles de intensidad en distancia de un píxel. En la práctica es muy común encontrar imágenes distorsionadas por ruido o por algún filtro añadido previamente, de manera coloquial se entiende que la imagen se ve borrosa. En tales situaciones se pueden modelar bordes mediante una rampa de intensidad *Ramp Edge*, donde la pendiente de la rampa es inversamente proporcional al grado de desenfoque, ya que no se cuenta con un solo punto de borde, cada punto dentro de la rampa es un punto borde aceptable, de esta manera un segmento de borde es un conjunto de puntos situados en la rampa. Como tercer tipo de borde está *Roof Edge* o borde de techo, los bordes de techo son modelos de líneas dentro de una región, donde el ancho del borde es determinado por medio del grosor y nitidez de la línea, es normal encontrar bordes de techo en digitalización de dibujos e imágenes satélites[11]. La figura 4.3 ejemplifica los tres tipos de bordes descritos.

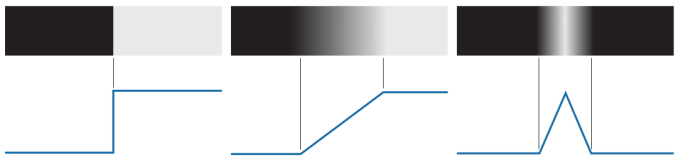


Figura 4.3: Tipos de Bordes.
Nota. Tomada de *Gonzalez*, 2009 [11].

Los bordes mencionados anteriormente son los más comunes y utilizados por algoritmos de detección de bordes, debido a que son comportamientos muy frecuentes en imágenes. En general, un algoritmo de detección de bordes consta de tres pasos esenciales:

1. Suavizar la imagen para reducir ruido.

2. Detectar puntos borde (Puntos potenciales para ser candidatos a puntos de borde).

3. Localizar bordes (Selección de puntos candidatos que pertenecen a un borde).

4.3.1. Detector de bordes de Canny

Desarrollado en 1986 por John F. Canny, es un algoritmo para detección de bordes con mayor complejidad que los algoritmos tradicionales. Sin embargo, el rendimiento de este también resulta ser superior. El enfoque de este algoritmo se basa en tres principales objetivos:

1. Tasa de error baja. Se deben localizar todos los bordes y no deben existir respuestas ilegítimas.

2. Los puntos borde deben ser bien ubicados. Los bordes detectados deben estar lo más cercano posible a los bordes verdaderos. Es decir, debe existir distancia mínima entre ellos.

3. Respuesta única de punto de borde. El detector debe devolver solo un punto borde por cada punto de borde verdadero.

El funcionamiento para detección de bordes mediante el algoritmo de Canny se fundamenta en el uso del vector gradiente debido a que este vector cuenta con la conocida propiedad de apuntar en la dirección de máxima tasa de cambio [11]. El vector gradiente de una imagen se define en la ecuación (4.1):

$$\nabla f(x,y) \equiv grad[f(x,y)] \equiv \begin{bmatrix} g_x(x,y) \\ \\ g_y(x,y) \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} \tag{4.1}$$

Esta herramienta permite determinar la magnitud y dirección de un borde, donde la magnitud $M(x,y)$ es el valor de la tasa de cambio en la dirección del gradiente y está compuesta por su norma vectorial euclidiana.

$$M(x,y) = \nabla \| f(x,y) \| = \sqrt{g_x^2(x,y) + g_y^2(x,y)} \tag{4.2}$$

La dirección del vector gradiente en un punto (x,y) está dada por:

$$\alpha(x,y) = \tan^{-1} \left[\frac{g_y(x,y)}{g_x(x,y)} \right] \tag{4.3}$$

El algoritmo para detección de bordes de Canny consta de los siguientes pasos.

1. Aplicar filtro gaussiano a la imagen de entrada para disminuir ruido.

2. Calcular magnitud y ángulo del vector gradiente.

3. Suprimir puntos no máximos.

4. Detectar y vincular bordes mediante aplicación de doble umbralización.

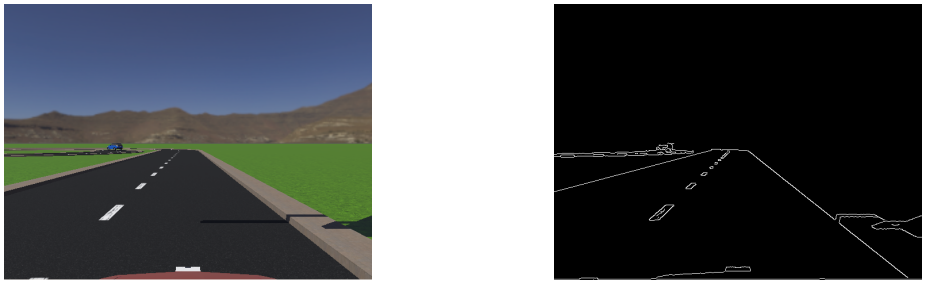
4.3.2. Ejemplo para detección de bordes con el detector de Canny

El algoritmo de bordes de Canny es la herramienta seleccionada para lograr el proceso de extraer los bordes que representan los carriles de la carretera, en específico las líneas que conforman los bordes del carril donde se encuentra el auto de pruebas. OpenCV provee una función para utilizar el algoritmo de Canny, en breve se presenta la sección de código donde se hace uso de la función "Canny()" de OpenCV.

```
1 import cv2
2
3 def canny(image):
4
5     gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
6     blur = cv2.GaussianBlur(gray, (5, 5), 0)
7     canny = cv2.Canny(blur, 50, 150)
8     return canny
```

El código anterior expone la forma de usar la función "Canny()" de OpenCV para detección de bordes. En la primer sentencia (línea 5) se realiza una conversión entre espacios de color, en específico de una imagen RGB a una imagen en escala de grises, el segundo paso (línea 6) tiene aplica un filtro Gaussiano a la imagen en escala de grises para reducir ruido. Finalmente en la línea 7 se aplica en concreto la función 'cv2.Canny()' para obtener una imagen en blanco y negro que contiene los bordes detectados por "Canny()".

La imagen 4.4, ofrece una comparación entre la imagen original 4.4a obtenida de la cámara y la imagen de bordes alcanzada por el algoritmo de Canny 4.4b. El resultado de está operación es muy importante para el siguiente paso en la detección los carriles.



(a) Imagen original. (b) Imagen de Bordes.

Figura 4.4: Comparación en entre la imagen original y la imagen de bordes.

4.4. Transformada Hough

Buscar formas dentro de una imagen digital es una tarea común en el procesamiento de imágenes, en muchas de las ocasiones se buscan ajustar un conjunto de puntos en una determinada estructura, ya sean líneas rectas, círculos o alguna otra geometría. La transformada de Hough es una técnica propuesta por Paul Hough en 1962 y ampliamente utilizada dentro del campo de visión computacional para detección de líneas rectas y curvas en imágenes a color y en escala de grises.

El algoritmo busca identificar formas básicas en una imagen haciendo uso de los parámetros que describen la geometría deseada. Dentro de este concepto la transformada de Hough funciona a través de un sistema de votación, es decir, se identifica una forma específica si se obtiene un número suficiente de votos. Mediante esta técnica es posible encontrar cualquier figura que pueda ser expresada matemáticamente, el caso más sencillo es la detección de líneas rectas [7].

4.4.1. Detección de líneas rectas mediante transformada Hough

Dentro del espacio de imagen una línea puede ser expresada en un sistema cartesiano mediante los parámetros (m, b) con la ecuación:

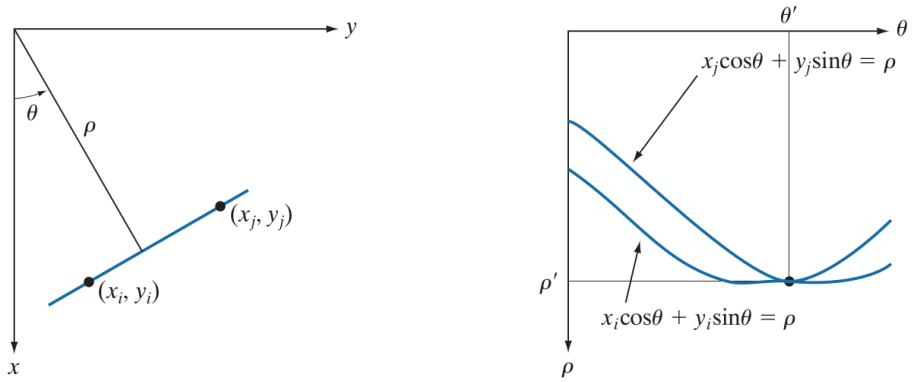
$$y = mx + b \tag{4.4}$$

mientras que en un sistema polar con los parámetros (ρ, θ) como:

$$\rho = x \cos\theta + y \sin\theta \tag{4.5}$$

El principio fundamental del algoritmo de Hough para detección de líneas rectas consiste en trazar n cantidad de líneas rectas para cada punto de la imagen y mediante el sistema de votación determinar que puntos forman una línea. Para la transformada de Hough resulta más sencillo trabajar en un sistema polar con los parámetros (ρ, θ), debido a que para cualquier punto de la imagen existen una infinidad de líneas que pasan a través de él, en el caso de líneas verticales los parámetros (m, b) están indefinidos, en un sistema polar no existe este problema pues para líneas verticales se tiene $\theta = 90°$ y para líneas horizontales $\theta = 0°$. De esta manera se evita cualquier tipo de discontinuidad. Además, en la forma polar si la línea corresponde a un borde, el ángulo θ es la dirección del gradiente [11].

Como se mencionó anteriormente en cada punto de la imagen se trazan n líneas, teniendo en cuenta esto es evidente que al menos una de las n líneas atraviesa dos o más puntos de la imagen, es decir, los puntos sobre la recta tienen la propiedad de ser colineales, este es el sistema de votación que utiliza el algoritmo de Hough, pues entre más puntos estén contenidos dentro de una recta más votos tendrá, como consecuencia la recta con el mayor número de votos será considerada como una línea recta identificada dentro de la imagen.



(a) Parametrización de una recta en el plano xy.

(b) Familia de Curvas en el espacio de Hough.

Figura 4.5: Interpretación geométrica de los espacios cartesiano y de Hough. *Nota.* Tomadas de *Gonzalez*, 2009 [11].

Tomado como ejemplo la imagen 4.5a se tienen dos puntos (x_i, y_i) y (x_j, y_j) en el espacio de imagen contenidos en el plano xy, la recta azul L representa una recta en forma polar con parámetros (ρ, θ) que pasa a través de los puntos (x_i, y_i) y (x_j, y_j). Esta línea L en el espacio cartesiano corresponde a un punto P_h en el espacio de Hough con coordenadas (ρ', θ') en 4.5b. Un punto $P(x_i, y_i)$ en el espacio cartesiano corresponde a una curva C_i en el espacio de Hough, donde C_i es representada por los parámetros (ρ_i, θ_i) de todas las rectas L_i que pasan por el punto $P(x_i, y_i)$ en el espacio cartesiano calculados mediante la ecuación (4.5), la imagen 4.5b muestra la familia de curvas C que pasan a través del punto P_h en el espacio de Hough.

Finalmente el método de votación del algoritmo de Hough consiste en encontrar las curvas C_i dentro del espacio de Hough que pasan a través del punto $P(x_i, y_i)$ del espacio cartesiano. Los puntos P_h, es decir la intersección (ρ', θ') en el espacio de Hough por donde pasen más curvas C_i serán las rectas resultantes en el espacio cartesiano.

4.4.2. Ejemplo para detección de líneas rectas con transformada Hough

Para continuar con el proceso de detección de carril es necesario identificar las líneas que los conforman, ver 4.4a. La manera más sencilla de encontrar un carril dentro de una imagen es delimitar el espacio donde se encuentra. Un carril está contenido por dos líneas rectas paralelas que pueden ser continuas, discontinuas e incluso líneas rectas dobles. El problema que se presenta en este caso es la detección de líneas rectas y el enfoque de la transformada de Hough para líneas rectas es justo la herramienta que permite resolver este problema.

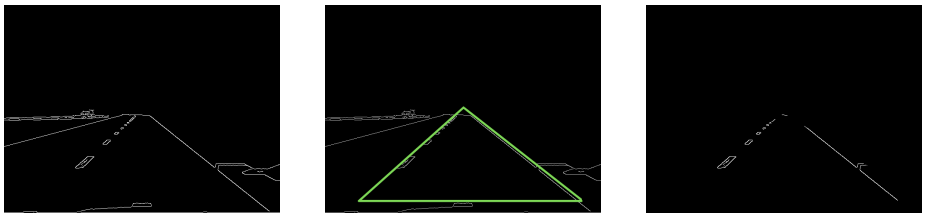
Los párrafos siguientes describen una forma en que se puede usar la trans-

formada de Hough para detección de líneas rectas e identificar las líneas que forman un carril.

Es importante mencionar que la transformada de Hough para detección de líneas rectas produce un mejor resultado con una imagen que presente solo la información de los bordes y en una escala de grises o bien en blanco y negro, este requisito justifica el hecho de haber detectado los bordes de la imagen de la cámara del vehículo de pruebas con el algoritmo de Canny, pues con está imagen se obtendrán las líneas que forman el carril donde se encuentra el vehículo. Sin embargo, no todos los bordes de la imagen son necesarios, por ello es necesario delimitar solo el área de interés donde se encuentra el carril, una vez determinada el área de interés es más fácil y eficiente obtener las líneas pertenecientes al carril.

El proceso de detección de carril mediante líneas rectas requiere definir el área donde está situado el carril de interés, este procedimiento se describe a continuación:

1. Obtener una imagen con los bordes de la imagen original, ver 4.6a.
2. Delimitar el perímetro donde se encuentra el carril a través de una geometría que pueda ser descrita mediante líneas rectas. Un triángulo es la geometría que mejor se adecua en este caso particular ver, 4.6b.
3. Descartar los bordes que se encuentren fuera de la región de interés para solo contar con los bordes que corresponden a la geometría del carril, ver 4.6c.



(a) Imagen de bordes. (b) Triángulo de interés. (c) Bordes de interés.

Figura 4.6: Proceso para delimitar la región de interés de el carril.

Previamente se realizó la detección de bordes en la imagen. Ahora es turno de delimitar la zona del carril, la implementación en código para definir el área de interés es la siguiente:

```
1  import cv2
2  import numpy as np
3
4  def region_of_interest(image):
5      triangle = np.array( [ (0, 450), (640, 450), (320, 250) ])
6      mask = np.zeros_like(image)
7      cv2.fillPoly(mask, triangle, 255)
8      cropped_img = cv2.bitwise_and(image, mask)
9
10     return cropped_img
11
12 canny_img = canny(lane_img)
13 cropped_img = region_of_interest(canny_img)
```

Donde la línea 12 contiene la imagen de bordes obtenidos de la imagen original. A partir de la línea 4 se define la función "region_of_interest()" para calcular la región de interés, esta función recibe como parámetro la imagen de bordes. En la línea 5 se forma el perímetro del triángulo que contiene los bordes necesarios para identificar el carril, por último la línea 8 secciona la imagen de bordes y solo permanecen los bordes que forman el carril de interés. En 4.6c se ilustra el resultado de la ejecución de este desarrollo.

La tarea más importante en la detección de las líneas que forman el carril es el uso de la transformada Hough para líneas rectas, en breve se enuncian los pasos para obtener las líneas resultantes de el carril.

1. Aplicar transformada de Hough a la imagen que contiene los bordes de la región de interés.

2. Identificar las líneas resultantes con el propósito de saber a que borde del carril pertenecen (izquierdo o derecho).

   - Para obtener el conjunto de líneas pertenecientes a cada borde se utiliza la pendiente de las rectas detectadas con Hough como diferenciador. En el caso rectas pertenecientes al borde izquierdo del carril $m > 0$, es decir, pendiente positiva y para el borde derecho se tienen rectas con $m < 0$, pendiente negativa. Líneas horizontales son despreciadas.

3. Obtener un promedio para cada conjunto de líneas debido a que el carril de una carretera cuenta con solo dos líneas rectas (izquierda y derecha), por esta razón es necesario obtener una sola línea del conjunto de rectas pertenecientes a cada grupo que se ajuste bien en cada borde del carril.

4. Ilustrar el resultado final de las dos líneas rectas con coordenadas cartesianas para comprobar que la detección de carril es correcta.

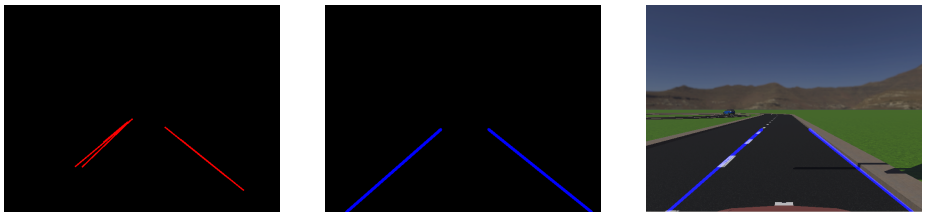
La línea 1 en la siguiente porción de código es la forma de usar la función que provee OpenCV para la transformada de Hough, donde su primer parámetro es la imagen de bordes pertenecientes a la región del carril que se obtuvo previamente, los parámetros restantes son constantes que requiere la función, estos valores son recomendados en la documentación de OpenCV. Como efecto de esta operación se extraen las líneas rectas detectadas, estas líneas se ilustran en la figura 4.7a.

```
1 lines = cv2.HoughLinesP(
2             cropped_img,
3             2,
4             np.pi/180,
5             80,
6             minLineLength=40,
7             maxLineGap=50
8         )
```

Como lo indican los pasos descritos en está sección se deben de filtrar las líneas detectadas en dos grupos (izquierdas y derechas) para poder calcular un promedio por grupo y generar una sola línea en cada borde del carril. La función que realiza este proceso es la siguiente:

```
1 def avg_slope_intercept(image, lines):
2     if angle < -0.3 or angle > 0.3:
3         if slope < 0:
4             right_fit.append((slope, intercept))
5         else:
6             left_fit.append((slope, intercept))
7 
8     left_line_avg = np.average(left_fit, axis=0)
9     right_line_avg = np.average(right_fit, axis=0)
```

Donde la distribución de líneas detectadas en cada grupo se inicia en la línea 2, en primer lugar se desprecian aquellas líneas con ángulo de inclinación θ en rango de [-0.3, 0.3] radianes porque estás líneas representan aquellas que son horizontales o están muy cerca de ser horizontales y pueden provocar imprecisiones en el resultado final. A partir de la línea 3 se inicia la condición para dividir en grupos de rectas izquierdas o derechas con la pendiente como diferenciador, $m < 0$ para líneas rectas del borde derecho y $m > 0$ para líneas rectas del borde izquierdo. Finalmente las líneas 8 y 9 contienen el resultado de calcular el promedio de las líneas del borde derecho e izquierdo respectivamente y como consecuencia tener una sola línea por borde tal como se muestra en 4.7b. Para ejemplificar el resultado final de la detección de carriles se genera la imagen 4.7c, donde se muestra la imagen original, las líneas del carril detectadas son colocadas en color azul, esta imagen permite demostrar que la detección del carril funciona correctamente, pues las líneas rectas detectadas se encuentran sobrepuestas en el carril real.



(a) Líneas detectadas. (b) Líneas promedio. (c) Carril detectado.

Figura 4.7: Líneas del carril detectadas por medio de Transformada Hough.

El resultado del proceso de detección de carril finaliza con la obtención de las líneas que conforman los bordes del carril en coordenadas cartesianas. Estas líneas servirán como modelo en el cálculo de las leyes de control para la tarea de seguimiento de carril donde se involucra velocidad y ángulo de dirección.

4.5. Modelo cinemático del vehículo

El modelo cinemático es una descripción del movimiento realizado por el Robot en función de su geometría. La cinemática del Robot puede utilizarse como modelo matemático de partida en el diseño de un controlador, para simular el comportamiento cinemático, o para establecer ecuaciones en cálculos de odometría [21].

Para el caso del modelo cinemático del vehículo autónomo se considera el llamado modelo de vía única o *Single-Track-Model* en inglés. En el modelo de única vía, el vehículo cuenta con dos ruedas que están unidas a través de un eslabón que está comprometido a moverse en un plano. Se parte del supuesto que las ruedas pueden girar libremente sobre sus ejes de rotación y que las ruedas no patinan sobre el punto de contacto con el suelo. Además, para modelar la dirección (*steering*) la rueda delantera cuenta con un grado adicional de libertad que le permite girar sobre un eje normal al plano de movimiento. Con estas dos características se refleja la experiencia que tienen los conductores cuando el vehículo no realiza movimiento lateral sin previamente haber avanzado [26].

Considerando la figura 4.8, donde p_r y p_f son los puntos de contacto con el suelo para los neumáticos trasero y delantero respectivamente, θ es el ángulo que describe la dirección hacia donde mira el vehículo y δ es el ángulo de dirección para la rueda delantera. Todo esto en un sistema de coordenadas inercial con vectores base $(\hat{e}_x, \hat{e}_y)$. Para satisfacer el supuesto de no deslizamiento, el movimiento de los puntos p_r y p_f debe de ser colineal respecto a la orientación de la rueda. Esta restricción para la llanta trasera se expresa como:

$$(\dot{p}_r \cdot \hat{e}_y)\cos(\theta) - (\dot{p}_r \cdot \hat{e}_x)\sin(\theta) = 0 \tag{4.6}$$

y para la llanta delantera:

$$(\dot{p}_f \cdot \hat{e}_y)\cos(\theta + \delta) - (\dot{p}_f \cdot \hat{e}_x)\sin(\theta + \delta) = 0 \tag{4.7}$$

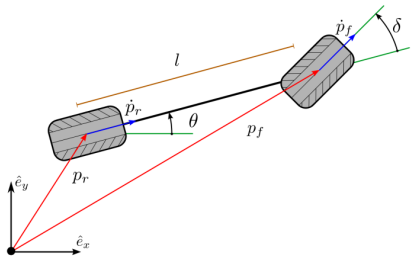


Figura 4.8: Modelo cinemático de vía única (*Single-Track Model*).
Nota. Tomada de *Paden*, 2016 [26].

Por lo general estas expresiones se reordenan en términos del movimiento en cada punto de los vectores base $(\hat{e}_x, \hat{e}_y)$. El movimiento de la llanta trasera a lo largo de $\hat{e}_x$ es $x_r := p_r \cdot \hat{e}_x$. Para la dirección $\hat{e}_y$, $y_r := p_r \cdot \hat{e}_y$. La velocidad de movimiento es la magnitud de $\dot{p}_r$ con el signo correcto para indicar conducción hacia adelante o en reversa.

En términos de cantidades escalares x_r, y_r y θ, la restricción diferencial es:

$$\begin{aligned} \dot{x}_r &= v_r \cos(\theta) \\ \dot{y}_r &= v_r \sin(\theta) \\ \dot{\theta}_r &= \frac{v_r}{l} \tan(\theta) \end{aligned} \tag{4.8}$$

Alternativamente se puede escribir la misma restricción diferencial en términos del movimiento de la rueda delantera.

$$\begin{aligned} \dot{x}_f &= v_f \cos(\theta + \delta) \\ \dot{y}_f &= v_f \sin(\theta + \delta) \\ \dot{\theta}_f &= \frac{v_f}{l} \sin(\delta) \end{aligned} \tag{4.9}$$

pero ahora se utiliza la velocidad de la rueda delantera, esta velocidad está relacionada con la velocidad de la llanta trasera por:

$$\frac{v_r}{v_f} = \cos(\delta) \tag{4.10}$$

Los problemas de control de este modelo pueden implicar el seleccionar un ángulo de dirección δ dentro de los rangos mecánicos aceptables por el vehículo, $\delta \in [\delta_{min}, \delta_{max}]$, y mantener la velocidad de avance en un rango estable, $v_r \in [v_{min}, v_{max}]$. Otro inconveniente importante de este modelo es que permite cambios instantáneos en el ángulo de dirección y por ende provocar cambios bruscos en la dirección.

La continuidad del ángulo de dirección para la rueda delantera se pude imponer añadiendo una velocidad más en la ecuación (4.9), donde el ángulo de

dirección integra una velocidad constante. Entonces, la ecuación (4.9) se reescribe como:

$$\begin{aligned} \dot{x}_f &= v_f \cos(\theta + \delta) \\ \dot{y}_f &= v_f \sin(\theta + \delta) \\ \dot{\theta}_f &= \frac{v_f}{l} \sin(\delta) \\ \dot{\delta} &= v_\delta \end{aligned} \tag{4.11}$$

Ahora, además de limitar el ángulo de dirección también se puede limitar la velocidad en la dirección para la rueda del frente, $v_\delta \in [\dot{\delta}_{min}, \dot{\delta}_{max}]$. Este mismo problema se puede presentar con la velocidad del automóvil descrita por v_r y se puede abordar de la misma manera. El único inconveniente de esta técnica es que el aumento en la dimensión del modelo podría complicar más los problemas de planificación y control del movimiento.

Más adelante se considerará este modelo cinemático para el diseño de las leyes de control que influyan en el movimiento lateral y longitudinal del vehículo autónomo. En concreto, la relación que existe con el movimiento de la rueda delantera y el rango del ángulo de dirección (*steering*).

4.6. Leyes de control

Un controlador es un componente que permite procesar información de sensores y generar salidas específicas. La forma en que un controlador opera sobre una señal se conoce como algoritmo de control o ley de control. El objetivo de una ley de control es ajustar el valor de una señal respecto a un valor conocido, el cual puede provenir de la lectura de un sensor o bien ser el resultado de una operación previa. La diferencia entre valor conocido y valor medido se conoce como error. Una ley de control actúa sobre el error a través de algún mecanismo y manipula el proceso para provocar la salida esperada. Es decir, se pretende que la ley de control pueda compensar efectos de los elementos dinámicos del sistema [10].

En este proyecto se busca calcular un error con base en las líneas observadas de los bordes del carril en la sección 4.4.2 y las líneas reales de los bordes del carril, este error permitirá diseñar un ley de control para el movimiento lateral del vehículo con base en el modelo cinemático. Mediante esta ley de control el vehículo debe ser capaz de controlar el ángulo de dirección (*steering*) en un rango estable para mantenerse dentro del carril en líneas rectas, curvas a la derecha y curvas hacia la izquierda.

4.6.1. Diseño e implementación de una ley de control para movimiento lateral del vehículo

En la búsqueda de mantener el vehículo autónomo alineado y dentro de su carril se deben diseñar leyes de control que faciliten el proceso de seguimiento

de carril, el objetivo de las leyes de control es manipular el error entre líneas observadas y líneas reales del carril para posteriormente calcular y establecer valores numéricos para las operaciones tácticas del vehículo, es decir, velocidad para movimiento longitudinal y ángulo de dirección (*steering*) en las llantas delanteras como se vio en el modelo cinemático de la sección 4.5, esto a fin de generar movimiento lateral.

Como resultado de la detección de carriles en la sección 4.4.2, se obtuvieron dos líneas rectas que corresponden a los bordes del carril original en coordenadas cartesianas. Este par de líneas se aprovechan para calcular el ángulo de dirección del vehículo. Con el propósito de mantener el vehículo alineado dentro de el carril es necesario conocer el ángulo con respecto a la horizontal que existe desde el centro del coche hacia cada una de las líneas observadas, este ángulo θ en cada línea indica si el vehículo se encuentra alineado o desalineado con respecto a las líneas del carril. Bajo este enfoque resulta buena opción trazar una línea desde el centro del vehículo hacia cada borde del carril (izquierdo y derecho) y así conocer la orientación del vehículo en todo momento.

Considerar la imagen 4.9, en ella se muestran dos líneas en color azul desde el centro del vehículo hacia cada uno de los bordes del carril con parámetros distancia y ángulo (ρ, θ) respectivamente. La imagen permite especificar la geometría de cada una de las líneas. A continuación se describe un algoritmo general para cuantificar los parámetros (ρ, θ) de estas líneas imaginarias tomando como referencia la imagen 4.9.

1. Formar dos líneas rectas desde el centro del vehículo con coordenadas $C(320, 480)$ hacia los bordes izquierdo y derecho del carril con coordenadas (x_{pl}, y_{pl}) y (x_{pr}, y_{pr}) respectivamente. Donde, (x_{px}, y_{px}) es un punto promedio sobre el borde del carril.

2. Calcular los parámetros (ρ, θ) de cada línea con el teorema de Pitágoras:

$$\rho = \sqrt{a^2 + b^2} \qquad \theta = \tan^{-1}\left(\frac{b}{a}\right) \tag{4.12}$$

   - Para la línea del borde izquierdo:

$$a_l = 320 - x_{pl} \qquad b_l = 480 - y_{pl} \tag{4.13}$$

   - Para la línea del borde derecho:

$$a_r = x_{pr} - 320 \qquad b_r = 480 - y_{pr} \tag{4.14}$$

3. Ordenar los parámetros de cada línea en forma polar como:

   - Línea izquierda $[\rho_l, \theta_l]$
   - Línea derecha $[\rho_r, \theta_r]$

Una vez que se cuenta con los parámetros de cada una de las rectas desde el centro del vehículo a cada borde del carril es más sencillo calcular las leyes de control. El seguimiento de carril de este trabajo considera cuatro situaciones diferentes en las que podría estar el vehículo.

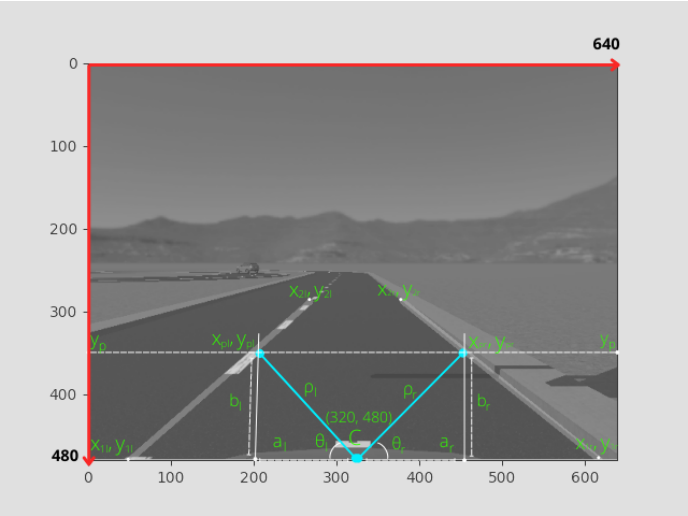


Figura 4.9: Geometría de líneas guía del vehículo.

1. Ambas líneas (izquierda y derecha) de los bordes del carril son detectadas indican que el vehículo se encuentra en línea recta.

2. Solo la línea del borde izquierdo es detectada supone que el vehículo se encuentra en una curva hacía la derecha.

3. Solo la línea del borde derecho es detectada supone que el vehículo se encuentra en una curva hacía la izquierda.

4. Ningún borde es detectado indica que el vehículo está fuera de un carril y debe detenerse.

Estas cuatro situaciones son suficientes para que el vehículo pueda determinar la línea con la cual debe guiarse a fin de mantenerse dentro del carril mientras avanza. El seguimiento de carril es logrado usando una ley de control que combina la distancia y el ángulo con respecto al borde del carril. Considerar la figura 4.10, donde las líneas de color verde representan las líneas deseadas si el vehículo se encuentra alineado con los bordes del carril, estas líneas pertenecen a los parámetros (ρ_{ld}, θ_{ld}), (ρ_{rd}, θ_{rd}) para los bordes izquierdo y derecho del carril respectivamente. Líneas en color cyan corresponden a las líneas actualmente observadas de cada borde (ρ_l, θ_l) y (ρ_r, θ_r).

La ley de control para movimiento longitudinal del vehículo o velocidad se considera constante en todo momento.

$$v = C \tag{4.15}$$

La ley de control en el caso del movimiento lateral o ángulo de dirección es determinado mediante:

$$\delta = K_\rho e_\rho + K_\theta e_\theta \tag{4.16}$$

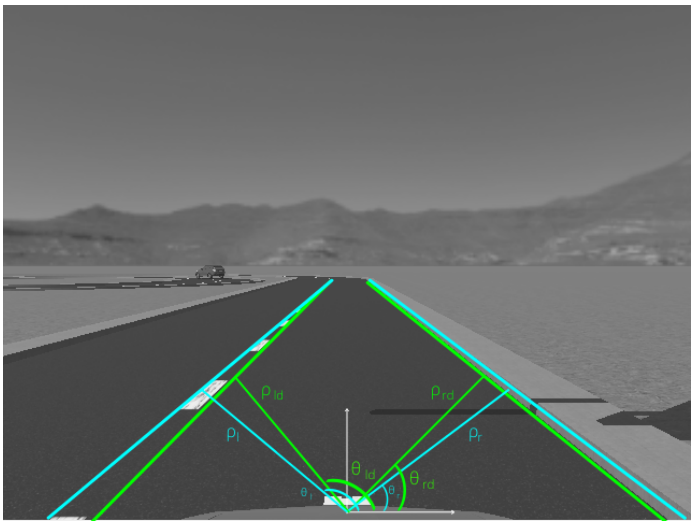


Figura 4.10: Comparación entre líneas deseadas (verde) y líneas detectadas (cyan).

Donde e_ρ y e_θ son errores de distancia y ángulo respectivamente, calculados con las ecuaciones:

$$e_\rho = \rho_d - \rho_o \tag{4.17}$$

$$e_\theta = \theta_d - \theta_o \tag{4.18}$$

Los subíndices d y o en las ecuaciones (4.17)-(4.18) son etiquetas para "deseado" y "observado" respectivamente.

Como se mencionó anteriormente dentro del seguimiento del carril se consideran tres casos, las ecuaciones (4.16)-(4.17)-(4.18) funcionan bien en los casos donde existen curvas, es decir, cuando solo se detecten bordes del lado izquierdo o bordes del lado derecho. Sin embargo, para seguir el carril en línea recta resulta mejor trabajar con un error promedio de distancia y ángulo en la ecuación (4.16) pues ambos bordes del carril se encuentran presentes. Por lo tanto, en este caso las ecuaciones (4.17)-(4.18) se reescriben de la forma.

$$e_{\rho_p} = \frac{\rho_{l_d} - \rho_{l_o} + \rho_{r_d} - \rho_{r_o}}{2} \tag{4.19}$$

$$e_{\theta_p} = \frac{\theta_{l_d} - \theta_{l_o} + \theta_{r_d} - \theta_{r_o}}{2} \tag{4.20}$$

Nuevamente los subíndices d y o son "deseado" y "observado". Mientras que p es "promedio", r y l son *right* y *left* respectivamente.

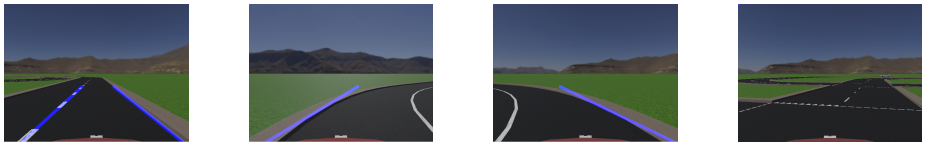
Entonces la ecuación (4.16) queda de la forma:

$$\delta = K_\rho e_{\rho_p} + K_\theta e_{\theta_p} \tag{4.21}$$

En cualquiera de los casos anteriores las constantes K_ρ y K_θ son constantes de sintonización que permiten ajustar el error de distancia y ángulo para precisar el ángulo de dirección que mantendrá el vehículo en cada caso. Estas constantes son obtenidas de manera experimental porque al ser un ambiente simulado no se pueden tener medidas reales de estos parámetros. Una manera fácil de obtener los valores numéricos esperados para $(\rho_{l_d}, \theta_{l_d})$ y $(\rho_{r_d}, \theta_{r_d})$ es obtenerlos desde el principio de la simulación con el supuesto de que el vehículo se encuentra bien centrado dentro del carril. Cabe resaltar que se pretenden giros suaves del vehículo para mantener un comportamiento lo más natural y parecido en comparación como lo realizan conductores al frente del volante, bajo estas condiciones el ángulo de dirección disminuye o aumenta en pasos de 1°.

Con el cálculo de estas ecuaciones, el ángulo de dirección del vehículo siempre se mantiene en un rango seguro para que no salga del carril, mientras que la velocidad es constante en todo momento. Con los cuatro casos anteriores en cuanto a detección de las líneas que conforman el carril se determina que el vehículo puede manejar en situaciones reales guiándose con los bordes del carril. Además, con el objetivo de mantener la integridad del vehículo en líneas curvas se considera una velocidad menor respecto a líneas rectas donde la velocidad es mayor. Sin embargo, en un caso extremo donde el vehículo no sea capaz de detectar un carril por diferentes situaciones la mejor opción es dejar de avanzar, es decir, se considera velocidad = 0 y ángulo de dirección = 0.

Finalmente y con el propósito de demostrar que las operaciones de detección y seguimiento de carril corresponden al resultado esperado se presenta 4.11 como conjunto de imágenes que ilustran cada uno de los escenarios esperados.



(a) Ambos bordes

(b) Solo borde izquierdo

(c) Solo borde derecho

(d) Sin bordes detectados

Figura 4.11: Casos considerados según bordes detectados en el seguimiento de carril.

En 4.11a ambos bordes son detectados e indica movimiento en línea recta, las figuras 4.11b y 4.11c corresponden a los casos de curva hacia la derecha y curva hacia la izquierda respectivamente. En curvas a la izquierda debe de prevalecer un ángulo de dirección negativo y en el caso de curvas a la derecha el ángulo tiende a ser positivo. 4.11d representa una situación donde ningún borde de carril es detectado y la mejor opción es frenar el movimiento. Con este conjunto de imágenes se comprueba el correcto funcionamiento de la detección de carril y seguimiento de carril con las leyes de control diseñadas.

Este capítulo concluye con la implementación de las herramientas matemáticas de detector de bordes de Canny en una imagen y transformada Hough para detección de líneas rectas. Se demostró que se puede desarrollar un sistema de visión artificial que permita el reconocimiento de carriles en la escena a partir de imágenes RGB. Además, se diseñó e implementó un control para establecer operaciones tácticas de movimiento longitudinal y lateral que se ven involucradas en el seguimiento de carril. Con ello, el vehículo fue capaz de recorrer en su totalidad el circuito propuesto en la sección 3.3, sin obstáculos.

Sin embargo, en esta sección solo se contemplan situaciones controladas donde no existe ningún otro vehículo a parte del autónomo, lo cual es muy poco probable en el mundo real. De esta manera, el único inconveniente que se puede presentar es que el vehículo deje de detectar carriles para guiar su trayecto. En los siguientes capítulos se aspira a añadir más características al ambiente simulado que sumen en dificultad para la navegación autónoma del vehículo, como es el caso de nuevos vehículos para la detección obstáculos, evasión y seguimiento de vehículos.

Capítulo 5

Detección de obstáculos

Este capítulo busca añadir mayor detalle a la caracterización del ambiente, en específico detectar objetos potencialmente peligrosos para el vehículo autónomo. Primero, en la sección 5.1, se exponen aspectos básicos acerca de como un conductor puede detectar objetos en el camino. La sección, 5.2 es una descripción de las categorías y características de los algoritmos de agrupación, mientras que la subsección, 5.2.1 es un ejemplo de como procesar una nube de puntos con ROS. Enseguida, la sección 5.3 y subsección 5.3.1 se enfocan en la definición teórica e implementación en código del algoritmo *k-means* respectivamente. A continuación, la sección 5.4 habla acerca de la importancia de implementar un algoritmo de empatado. Finalmente, en 5.5 se expone teóricamente el concepto de filtro de Kalman y 5.5.1 su versión extendida, además, la subsección 5.5.2, es un ejemplo de diseño e implementación de un filtro de Kalman para seguimiento de objetos.

Es importante mencionar que la implementación de códigos de este capítulo resulta en una combinación de lenguajes de programación C++ y Python, esta decisión se justifica y explica en las secciones correspondientes. Al finalizar este capítulo se muestran los resultados de las implementaciones aquí desarrolladas y en adición con la labor del seguimiento de carriles previamente diseñada, el vehículo autónomo no tripulado contará con un sistema más robusto que le permitirá tener una mejor percepción del ambiente.

5.1. Introducción a la detección de obstáculos

Los humanos pueden evitar obstáculos en el ambiente vehicular mientras manejan haciendo uso de los sentidos de la vista y oído. Con ellos, pueden percibir el ambiente y conocer si algún vehículo u obstáculo se encuentra al frente, por detrás o en los laterales. Es decir, conocen aunque con poca exactitud la posición de los obstáculos. Además, mediante la posición determinan que tan rápido avanzar para no chocar al frente, cuando comenzar a frenar en caso necesario e incluso fijar el ángulo de dirección en un rango estable para no

chocar a los costados. Sin embargo, un vehículo autónomo no cuenta con un sistema que le permita realizar estas tareas de manera natural pues solo está instrumentado con sensores que le permiten conocer el ambiente. Con ayuda de algoritmos e instrumentos matemáticos y en combinación con información de sensores es posible simular estos comportamientos innatos para el ser humano.

El objetivo de este capítulo es contar con instrumentos que permitan determinar una situación donde intervenga la evasión de obstáculos a partir de dos tareas principales. En la sección anterior se mencionó la nula presencia de vehículos dentro del circuito de pruebas, ya es bien sabido que un ambiente vial cotidiano existen diversos objetos que podrían provocar colisiones. En específico, se buscan evitar accidentes del vehículo autónomo con otros vehículos dentro del circuito. Antes de definir una situación de evasión de objetos se requieren dos fases:

- Detección de objetos
- Seguimiento de objetos

El objetivo de la detección es encontrar aquellos objetos potencialmente peligrosos para provocar cualquier tipo de accidente con el vehículo no tripulado, un escenario sencillo para navegación autónoma es usar las líneas del carril como guía dentro de un circuito cerrado. Sin embargo, en la búsqueda de situaciones más reales es importante considerar vehículos u otro tipo de objetos con diferentes tamaños, colores y obviamente posiciones distintas. La detección de objetos debe clasificar aquellos objetos que se encuentren dentro de la carretera pero en especial que impliquen un problema en la navegación del vehículo autónomo.

Una herramienta tecnológica en robótica que ayuda a ubicar objetos son los sensores láser. Como se mencionó en la subsección 2.1.4 un sensor LIDAR otorga puntos pertenecientes a una superficie. Este conjunto de puntos también conocido como nube de puntos no provee la información necesaria para decidir que objetos resultan un riesgo para el vehículo, por esta razón es necesario agrupar la nube de puntos en conjuntos que describan un objeto potencialmente peligroso y posteriormente determinar su posición. Las técnicas de agrupación tienen la capacidad de cumplir esta labor además, pueden trabajar con conjuntos grandes en tamaño para agruparlos en pequeños grupos de acuerdo a sus características. Dentro de la gran variedad de algoritmos de agrupación se encuentra el algoritmo *k-means* que se caracteriza por agrupar bancos de datos en base a la cercanía (distancia) que existe entre ellos.

Por otro lado, se entiende que los vehículos pueden desplazarse principalmente de manera vertical y en menor medida horizontalmente o por alguna razón pueden permanecer estáticos. Es evidente que los vehículos estáticos tienen velocidad $v = 0$, mientras que la posición puede ser cualquier punto en el espacio pero constante. Sin embargo, en el caso contrario cuando la velocidad $v \neq 0$, la posición es distinta en cada momento y es proporcional a la velocidad que presente el vehículo. Este comportamiento implica la necesidad de predecir de alguna manera la velocidad y posición de vehículos estáticos y en movimiento. El seguimiento de vehículos tiene el objetivo de cumplir esta meta con ayuda

de un instrumento matemático conocido como filtro de Kalman, el cual permite predecir estados futuros de un sistema dinámico.

Cada una de estas fases tienen desarrollos distintos, los cuales son:

- Detección de objetos
  1. Obtener la nube de puntos otorgada por el sensor LIDAR.
  2. Agrupar la nube de puntos.
  3. Obtener la posición de cada objeto detectado.
- Seguimiento de objetos
  1. Diseñar un sistema dinámico que involucre posición y velocidad.
  2. Estimar posiciones y velocidades futuras de cada objeto.

La combinación de detectar y seguir obstáculos es esencial para iniciar con la tarea específica de evasión de obstáculos.

5.2. Algoritmos de agrupamiento

Existe una gran variedad de algoritmos de agrupación y por ello, resulta complicado hacer una clasificación exacta de estos métodos porque algunas categorías podrían superponerse con otras, de modo que, un algoritmo puede poseer características de una o varías categorías. No obstante, es de mucha utilidad crear una organización relativa para los métodos de agrupamiento existentes. Los principales algoritmos de agrupación pueden ser clasificados en las siguientes categorías [12].

1. **Métodos de división:** Centran su funcionamiento en el calculo de distancias por ejemplo, distancia de *Manhattan* o distancia Euclidiana (5.1). Dado un conjunto de n objetos, un método de división construye k divisiones. Cada división representa un *cluster* o grupo con $k \leq n$. Este tipo de algoritmos agrupan los datos en k *clusters* y cada *cluster* contiene al menos un objeto.

   Sea k el número de divisiones a construir, el método de división crea una división inicial. Después usa una técnica de reubicación iterativa que busca mejorar el agrupamiento moviendo los objetos de un grupo a otro. El criterio general para asignar objetos en un grupo es que tan cerca o relacionados están los objetos de un grupo, mientras que objetos en diferentes *clusters* se mantienen lejanos o son muy diferentes. *k-means* y *k-medoids* son ejemplos de esta técnica de agrupación. Estos métodos funcionan bien para agrupar conjuntos con formas esféricas en conglomerados de tamaño pequeño o mediano. Para formar grupos con formas complejas y con conjuntos de mayor tamaño, es necesario mejorar o buscar otros métodos.

2. **Métodos jerárquicos:** Un método jerárquico se caracteriza por crear una descomposición jerárquica del conjunto de datos inicial en objetos de datos. Los métodos jerárquicos se pueden clasificar según la estrategia que implementan, ya sea aglomerativa o divisiva. También conocido como enfoque de abajo hacia arriba, el método de aglomeración comienza formando un grupo por cada objeto para después fusionar los objetos o grupos cercanos entre sí, hasta que todos los grupos forman uno solo o bien se cumple una condición de paro. El enfoque divisivo, igualmente llamado enfoque de arriba hacia abajo, inicia con todos los objetos dentro de un grupo. Luego, un grupo se divide en grupos más pequeños por cada iteración hasta que cada objeto está en un grupo o una condición de paro se cumpla.

   Los métodos jerárquicos cuentan con una condición de rigidez porque una vez que un paso de combinación o división es realizado no se puede anular. Sin embargo, esta desventaja es aprovechada por provocar menores costos de cálculo al no tener preocupación por un número de diferentes combinaciones. Una desventaja más de estas técnicas radica en que no pueden corregir decisiones erróneas.

3. **Métodos basados en densidad:** Uno de las principales inconvenientes de los métodos de agrupación por división es que solo funcionan bien para encontrar formas esféricas mientras que estructuras arbitrarías resulta una tarea difícil. Otros métodos de agrupación han sido desarrollados bajo la noción de densidad (número de objetos o puntos de datos). Su idea general consiste en incrementar un determinado grupo siempre y cuando la densidad de una vecindad exceda un umbral. Es decir, para cada punto contenido en un grupo, la vecindad debe de contener un mínimo de puntos.

   Este tipo de métodos son útiles para reducir ruido en valores atípicos provocados por lecturas de sensores además de descubrir grupos con formas arbitrarias.

4. **Métodos basados en cuadrículas:** Los métodos basados en cuadrículas se pueden integrar con distintos métodos de agrupación, como los métodos basados en densidad y los métodos jerárquicos, a menudo resulta eficiente el uso de cuadrículas en muchas aplicaciones de la minería de datos. El agrupamiento por cuadrículas cuantifica el espacio de un objeto en un número finito de celdas que forman una estructura de cuadrícula, todas las operaciones de agrupación son realizadas dentro de la cuadrícula, es decir, el espacio cuantificado. La principal ventaja de este método es el bajo coste computacional y rápido tiempo de procesamiento, pues por lo general es independiente del número de datos a agrupar y solo depende de la cantidad de celdas en cada dimensión dentro del espacio cuantificado.

$$d_E(P_1, P_2) = \sqrt{(x_1 - x_2)^2 - (y_1 - y_2)^2} \tag{5.1}$$

Método	Características Principales
De división	Basado en distancia Efectivo en conjuntos pequeños y medianos Utiliza la media o medoides para representar el centro de un *cluster*
Jerárquico	Puede incorporar otras técnicas El agrupamiento es una descomposición jerárquica No corrige después de una combinación o división
Basado en densidad	Encuentran formas arbitrarias Filtran valores atípicos Cada punto debe tener un mínimo de puntos dentro de su vecindad
Basado en cuadrículas	Una cuadrícula es la estructura de datos Es más rápido que otros métodos

Cuadro 5.1: Clasificaciones de algoritmos de agrupamiento y sus principales características.

Algunos de los algoritmos de agrupación integran ideas de distintos métodos, por lo que en ocasiones es difícil colocar un algoritmo dentro de una categoría específica. Además, algunas aplicaciones llegan a presentar criterios que requieren de más de un método de agrupación. El cuadro 5.1 muestra las principales características las clasificaciones antes descritas. En este trabajo se utilizará un método de agrupación por división, ver sección 5.3.

5.2.1. Ejemplo para obtener una nube de puntos con *ROS*

El primer paso en la detección de objetos es contar con un instrumento que permita caracterizar el ambiente que rodea al vehículo de pruebas. Webots tiene la facilidad de simular diferentes sensores que instrumentan el vehículo de forma independiente, de manera similar a la cámara existe un nodo específico en el simulador para seleccionar un sensor LIDAR de un amplío catálogo que incluye algunas de las marcas y modelos más comerciales como: Velodyne, SICK, Hokuyo y Robotis. Sin embargo, Webots también ofrece la posibilidad de caracterizar un nodo LIDAR con las características únicas y necesarias para propósitos específicos. Está flexibilidad es una gran ventaja porque permite especificar los parámetros requeridos que mejor se ajusten a la aplicación en turno.

De este modo, los principales parámetros requeridos del sensor LIDAR a usar son:

- **Campo de visión horizontal:** $2\pi = 360°$
- **Resolución horizontal:** 1024 puntos por capa

- **Campo de visión vertical:** 0.3°, ángulo entre primera y última capa
- **Número de capas:** 16
- **Rango máximo:** 30[m]

Otra razón para usar un sensor LIDAR convencional y con parámetros específicos es qué se tiene un mayor control sobre el número de puntos otorgados por el sensor, pues esta característica es de vital importancia para el desempeño de un algoritmo de clusterización.

De manera similar a la imagen de la cámara descrito en el capítulo 4, pero con las diferencias necesarias se pretende obtener la información del sensor LIDAR, en específico la nube de puntos. El proceso para obtener la nube de puntos del sensor requiere un mensaje tipo *PointCloud2*, las parámetros más importantes de este mensaje incluyen: alto y ancho de la nube de puntos, tamaño de una fila y de cada punto en bytes además de la propia nube de puntos en cada momento, etc. Cada uno de los parámetros requeridos por el mensaje *PointCloud2* contienen los valores recomendados en las documentaciones de Webots y *ROS*, el parámetro más importante de este mensaje es *data* y debe de contener una estructura del tipo *bytearray* por recomendación de la propia documentación, ya que usar una lista convencional tiene la desventaja de ser más lento entre más puntos contenga la nube de puntos. El controlador principal del vehículo de pruebas desarrollado en Python propone el siguiente código para este efecto.

```
# INIT LIDAR
lidar = Lidar('lidar')
lidar.enable(TIME_STEP)
lidar.enablePointCloud()

# POINT CLOUD2 MESSAGE
msg_point_cloud = PointCloud2()
msg_point_cloud.header.stamp = rospy.Time.now()
msg_point_cloud.header.frame_id = 'lidar_link'
msg_point_cloud.height = 1
msg_point_cloud.width = lidar.getNumberOfPoints()
msg_point_cloud.point_step = 20
msg_point_cloud.row_step = 20 * lidar.getNumberOfPoints()
msg_point_cloud.is_dense = False
msg_point_cloud.fields = [
    PointField(name = 'x', offset = 0, datatype = PointField.
    FLOAT32, count = 1),
    PointField(name = 'y', offset = 4, datatype = PointField.
    FLOAT32, count = 1),
    PointField(name = 'z', offset = 8, datatype = PointField.
    FLOAT32, count = 1),
]
msg_point_cloud.is_bigendian = False
msg_point_cloud.data = lidar.getPointCloud(data_type='buffer')
```

Con el mensaje *PointCloud2* se define un tópico que publica el contenido del mensaje, posteriormente se debe agrupar la nube puntos con un algoritmo

de agrupación, también se usa el mensaje para tener una visualización de la nube de puntos a través del visualizador RViz de *ROS*. Este paquete tiene la particularidad de recibir múltiples mensajes de sensores disponibles en *ROS* y realizar visualizaciones 3D de estos.

La imagen 5.1a muestra la nube de puntos vista desde el simulador. A manera de respaldar el resultado del código anterior se presenta la misma nube de puntos pero en el visualizador RViz, ver 5.1b.

(a) Nube de puntos en Webots.

(b) Nube de puntos en Rviz.

Figura 5.1: Nube de puntos en Webots y Rviz.

5.3. El algoritmo K-medias

La versión más simple y fundamental para el análisis de grupos es la división, organiza objetos de un conjunto en varios grupos o *clusters*. Para mantener la especificación del problema, se entiende que el número de *clusters* iniciales se proporciona con conocimiento previo ya que este parámetro es el punto de partida en cualquier método de agrupación por división.

Sea D un conjunto de datos, con n objetos, y k, el número de grupos por formar, un algoritmo de agrupación por división ordena los objetos en k particiones con $k \leq n$, donde cada partición representa un grupo. Los *clusters* se forman a partir de una función de disimilitud basada en distancia como criterio, objetos contenidos en un *cluster* son similares entre sí pero diferentes a los objetos de otros *clusters*. Dado un conjunto D con n objetos en el espacio euclidiano. Los métodos de división distribuyen los n datos de D en k grupos $C_1, C_2, C_3, ..., C_k$. Se utiliza una función objetivo para evaluar la partición de la división. Es decir, la función objetivo busca una alta similitud dentro de un grupo y una baja similitud con grupos externos.

El algoritmo *k-means* es una técnica de división basada en centroide, dicho de otra manera utiliza el centroide c para representar un grupo C. Teóricamente, el centroide de un *cluster* es su punto central. La definición de centroide puede ser de distintas maneras, como la media o medoide de los puntos u objetos asignados al grupo.

La diferencia entre un objeto $p_i \in C_i$ y su centroide c_i se mide como $d(p_i, c_i)$,

donde d es la distancia euclidiana entre dos puntos en $[x, y]$. La calidad del *cluster* C_i se puede medir como la variación del grupo, que es la suma del error cuadrático entre todos los objetos p_i en C_i y el centroide c_i, como:

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} d(p_i, c_i)^2 \tag{5.2}$$

donde E es la suma del error cuadrático de todos los puntos en el conjunto de datos, p_i es un punto en el espacio que representa un objeto; y c_i es el centroide del *cluster* C_i. Dicho de otra manera, la distancia desde un punto hacia el centro de su grupo se eleva al cuadrado y las distancias se suman. El propósito de esta función objetivo es hacer que los k *clusters* resultantes sean lo más separados y compactos posibles, *k-means* establece el centroide de un *cluster* como el valor medio de los puntos que contiene.

Al comenzar, *k-means* selecciona aleatoriamente k objetos de D, cada uno de estos representa inicialmente un centro o media de *cluster*. Todos y cada uno de los objetos p_i restantes se reasignan al grupo que más sean similares en función de la distancia euclidiana que existe entre el punto p_i y la media de grupo c_i. En otras palabras, cada objeto p_i se une al grupo C_i más cercano y c_i representa la media de todos los objetos que contiene C_i. Después, el algoritmo mejora iterativamente la variación dentro del grupo calculando una nueva media de los objetos asignados al *cluster* en la iteración anterior. En breve, todos los objetos se reasignan utilizando el valor medio de cada *cluster* como el nuevo centroide c_i de C_i. El proceso iterativo se repite hasta que la asignación es estable, es decir, los grupos formados en la ronda actual son los mismos que se formaron en la ronda anterior. Dicho de otro modo, la diferencia entre c_i anterior y c_i actual es mínima. El algoritmo 1 describe este proceso.

Sin embargo, no se garantiza que *k-means* converja a un óptimo global y en muchos de los casos termina en un óptimo local. Por otro lado, el resultado de la agrupación puede depender de la selección aleatoria inicial de centroides. En la práctica, para obtener mejores resultados es común ejecutar el algoritmo n veces con diferentes centroides iniciales. La complejidad de *k-means* es de $O(nkt)$; donde n es total de objetos, k es el número de *clusters* a formar y t el número de iteraciones. Por lo tanto, este método es relativamente escalable para grandes conjuntos de datos.

El método *k-means* no resulta adecuado para descubrir formas arbitrarias. Además, es sensible al ruido y a datos atípicos pues pueden influir sustancialmente en el valor medio. Existen diferentes variantes del método *k-means* que pueden diferir en la selección de los k *clusters* iniciales, el cálculo de la disimilitud y estrategias para calcular la media de los *clusters*.

5.3.1. Ejemplo de implementación del algoritmo K-medias

El segundo paso en la detección de objetos es implementar un algoritmo de agrupación, como se mencionó anteriormente el algoritmo *k-means* es el elegido para agrupar la nube de puntos del sensor LIDAR y lograr la detección de

Algoritmo 1 Algoritmo *K-means*.

Entrada: k : Número de *clusters*
D : Conjunto de datos con n objetos
Salida: k centroides
Obtener centroides iniciales $C = \{c_0, c_1, ..., c_k\}$ con k objetos de D
para todo p en D **hacer**
Calcular $d(p_i, c_i) = \sqrt{(p_{i_x} - c_{i_x})^2 + (p_{i_y} - c_{i_y})^2 (p_{i_z} - c_{i_z})^2}$
Añadir p_i al *cluster* c_i más cercano
fin para
Obtener nuevos centroides $C_n = \{\bar{c_0}, \bar{c_1}, ..., \bar{c_k}\}$
Calcular $\Sigma d(C, C_n)$.
mientras $\Sigma d(C, C_n) > tolerancia$ **hacer**
Actualizar $C \leftarrow C_n$
Reagrupar p_i en c_i
Recalcular $C_n = \{\bar{c_0}, \bar{c_1}, ..., \bar{c_k}\}$
Recalcular $\Sigma d(C, C_n)$
fin mientras
devolver k centroides

objetos, en específico vehículos. Debido a que el sensor LIDAR está situado en la parte superior del vehículo, ver 3.6b, es importante delimitar el área donde existan puntos de interés en la búsqueda de objetos potencialmente peligrosos. En breve se enuncia el proceso para agrupar la nube de puntos.

1. Eliminar puntos que pertenezcan al suelo del escenario.

2. Definir el número de *clusters* iniciales que serán los encargados de identificar obstáculos.

3. Aplicar *k-means* al conjunto de puntos previamente filtrado.

En un primer intento por implementar el algoritmo *k-means* se usó Python como lenguaje de programación donde se logró el objetivo de obtener los centroides que indican la posición de cada objeto contenido en la escena. Sin embargo, se presentaron algunos inconvenientes en los resultados finales, por ejemplo:

- La posición de los centroides no es constante con la posición de los objetos. Es decir, la posición de los centroides varía en cada momento aún cuando los objetos permanecen estáticos.

- La velocidad del algoritmo incrementa conforme crece el tamaño de la nube de puntos.

Para lidiar con estos inconvenientes se decide hacer una nueva implementación del algoritmo *k-means* pero ahora en lenguaje de programación C++ con el fin de tener más control en cada una de las operaciones que involucra el algoritmo además de disminuir el tiempo de procesamiento del mismo.

Para completar el primer requisito del agrupamiento es necesario filtrar la nube de puntos y eliminar puntos pertenecientes al suelo. Es difícil tener una medición exacta que permita delimitar la altura que existe desde el suelo hasta la posición del sensor LIDAR. Debido a esto mediante experimentación se llega al valor de $-1.5[m]$ en el eje y que representa la altura en el sistema coordenado, este valor en y es un umbral que delimita puntos que forman el suelo y puntos que no son parte de él. Es decir, todos los puntos $y < -1.5$ son despreciados, además se eliminan puntos con valor inf o NaN de la nube de puntos para evitar posibles errores en operaciones posteriores del algoritmo *k-means*.

En breve se ejemplifica la implementación en código, donde las líneas 2 y 3 muestran las condiciones para eliminar puntos inf y suprimir puntos perteneciente al suelo respectivamente.

```
1 void objectDetectCallback(const sensor_msgs::PointCloud2::
      ConstPtr& msg){
2     if( (isinf(x) or isinf(y) or isinf(z)) != true){
3         if( y > -1.5 ){
4             std::vector<double> point = {x, y, z};
5             point_cloud.push_back(point);
6         }
7     }
8 }
```

Definir el número de *clusters* considerados para el algoritmo *k-means* es una ardua tarea en esta aplicación, pues el propósito general es detectar autos y en un ambiente vial existen múltiples situaciones con un número diferente de vehículos en cada situación, como consecuencia de este hecho no se puede definir a priori la cantidad de grupos iniciales que mejor se ajuste a cada una de las situaciones, porque existen una infinidad de casos. Aunque el método del codo o *elbow-method* del algoritmo *k-means* es una herramienta de utilidad que funciona en otro tipo aplicaciones pero, para este caso en particular resulta costoso en tiempo de procesamiento debido a que la nube de puntos cambia mientras el vehículo se desplaza vertical u horizontalmente.

Sin embargo, uno de los propósitos de este trabajo es probar cada uno de los algoritmos bajo un ambiente supervisado y controlado, tal y como se hizo con el circuito de prueba para detección y seguimiento de carril donde el vehículo de pruebas está dentro de un circuito cerrado también se añade un número invariable de vehículos con el objetivo de realizar pruebas en la detección de objetos de manera controlada. Siguiendo este enfoque se pretende añadir un máximo de 5 diferentes vehículos con distintas posiciones en el circuito de pruebas, por lo tanto el número mínimo de *clusters* iniciales debe de ser ≥ 5.

Con los dos pasos realizados previamente se puede comenzar la implementación del algoritmo *k-means* para agrupar la nube de puntos, como se mencionó anteriormente la implementación en código de este algoritmo es en lenguaje C++ por razones ya explicadas, la implementación corresponde directamente al algoritmo 1 y se tienen en cuenta las siguientes consideraciones.

1. La agrupación de puntos se define a través de la cercanía que existe entre ellos.
2. El número y posición de los *clusters* iniciales es definido antes de iniciar el agrupamiento.
3. La cercanía entre puntos es calculada con distancia euclidiana.
4. Si un grupo no contiene puntos entonces es ignorado.
5. Si dos centroides se encuentran a distancia mínima se obtiene el promedio de ambos que resulta en uno solo.
6. La condición de paro del algoritmo es el cambio mínimo de posición de los centroides o cuando se exceda un máximo de intentos.

```
1  /* KMEANS FUNCTION */
2  std::vector<std::vector<double>> kmeans(std::vector<std::
       vector<double>> point_cloud){
3
4      new_centroids = calulate_centroids(point_cloud,
       initial_centroids);
5      total_distance = compare_centroids(new_centroids,
       initial_centroids);
6
7      while (total_distance > tol && attemps < max_attemps){
8          std::vector<std::vector<double>> centroids(
       new_centroids);
9          new_centroids = calulate_centroids(point_cloud,
       centroids);
10         total_distance = compare_centroids(new_centroids,
       centroids);
11         attemps += 1;
12     }
13     return new_centroids;
14 }
```

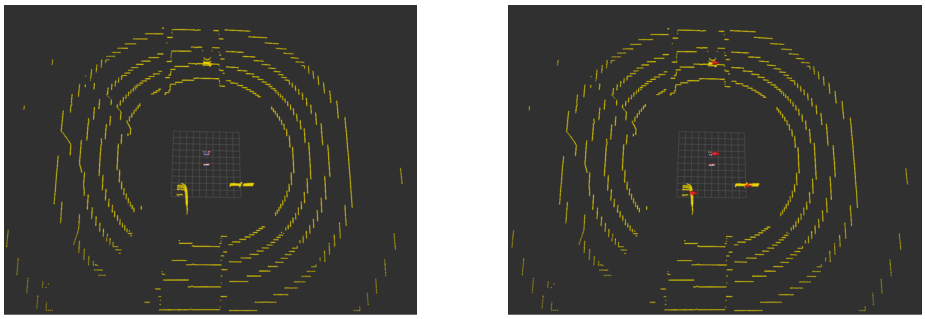
La función *kmeans* del código anterior recibe como parámetro una nube puntos y el valor de retorno es el conjunto de centroides calculados, es decir, la posición media de cada uno de los grupos formados por el algoritmo.

Las líneas más importantes del código anterior son explicadas a continuación:

- La línea 4 efectúa el agrupamiento para cada punto de la nube y calcula la media de cada centroide por medio de la función *calculate_centroids()*. Donde el primer parámetro es la nube de puntos y el segundo parámetro corresponde al conjunto de centroides iniciales previamente establecidos.

- La línea 5 realiza el calculo de distancia euclidiana total entre centroides actuales e iniciales.
- A partir de la línea 7 se inicia un ciclo para repetir el agrupamiento hasta que se cumpla una de las condiciones de paro.
- La línea 8 realiza una copia los nuevos centroides para efectuar el siguiente calculo con el valor actualizado. Mientras que 9 y 10 realizan agrupamiento y calculo de distancia entre centroides respectivamente.

Al finalizar este proceso se obtiene como resultado un conjunto de centroides que corresponden a la posición media de los grupos formados por el algoritmo de agrupación. La posición de cada centroide se expresa en coordenadas cartesianas en un espacio de tres dimensiones (x, y, z). En conclusión, al finalizar el proceso de agrupación se adquieren las posiciones medias (centroides) de los objetos detectados en la nube de puntos. En 5.2b se ilustra este resultado con el visualizador RViz, donde los puntos amarillos forman la nube de puntos y los centroides correspondientes a cada objeto en color rojo. Si se observa 5.2 con detenimiento se puede verificar que las posiciones de los centroides se encuentran en un punto medio de cada objeto sin tomar en consideración puntos pertenecientes al piso del escenario. La tarea de detección de objetos concluye



(a) Antes de aplicar *k-means*.

(b) Después de aplicar *k-means*.

Figura 5.2: Nube de puntos antes y después de agrupar mediante *k-means*.

al obtener la posición de cada centroide que representa un objeto, este resultado es fundamental para iniciar el proceso de seguimiento de obstáculos.

5.4. Empatado

Retomando los centroides calculados en 5.3.1 con el algoritmo de agrupación *k-means* que representan los objetos detectados como la posición media de los puntos de un *cluster* se entiende que conforme el vehículo avanza la nube de puntos cambia en tamaño y forma, como consecuencia el número de centroides varía en cada instante. Este comportamiento presenta la necesidad de saber con

certeza que centroide es en cada momento, es decir, se requiere definir si un objeto ha sido previamente detectado con *k-means* y solo ha variado mínimamente su posición. En cambio, si un nuevo objeto es detectado debe ser considerado en la toma de decisiones para evitar colisiones. Dicho de otra forma, el problema por resolver es identificar todos y cada uno de los potenciales obstáculos pues es diferente detectar que identificar objetos. Una vez conocida la posición de los objetos es necesario saber si en cada desplazamiento del vehículo un centroide es el mismo que el anterior o se trata de uno nuevo.

Para empatar dos conjuntos con tamaño n de centroides actual y previo C_a y C_p respectivamente se propone, calcular la distancia euclidiana $d(C_a, C_p)$ entre cada par.

$$d(c_a, c_p) = \sqrt{(c_{a_x} - c_{p_x})^2 + (c_{a_y} - c_{p_y})^2 + (c_{a_z} - c_{p_z})^2} \tag{5.3}$$

Al analizar el resultado de esta operación se obtienen dos posibles escenarios:

$$d(c_a, c_p) = \begin{cases} \text{Son el mismo si,} & d(c_a, c_p) \leq u \\ \\ \text{Son diferentes si,} & d(c_a, c_p) \geq u \end{cases} \tag{5.4}$$

Para el primer caso cuando la distancia $d(c_a, c_p)$ es menor que un umbral u significa que ambos centroides c_p y c_a representan al mismo objeto, por otro lado cuando la distancia $d(c_a, c_p)$ es mayor que el mismo umbral u, implica que c_a y c_p son dos objetos diferentes. El umbral u es considerado como un valor promedio de las dimensiones de un vehículo pequeño. El algoritmo 2 describe el empatado de centroides.

Algoritmo 2 Algoritmo de Empatado

Entrada: Conjunto de centroides previos C_p.
Cojunto de centroides actuales C_a.
Salida: Conjunto de centroides empatados.
para todo c_a **hacer**
Obtener el centroide c más cercano a c_p
si $d(c, c_a) \geq u$ **entonces**
c_a es un nuevo centroide
Agregar c_a al conjunto de centroides empatados.
si no
c y c_a son el mismo centroide.
fin si
fin para
devolver Conjunto de centroides empatados.

5.5. El filtro de Kalman

El filtro de Kalman fue desarrollado por Rudolf E. Kalman en 1960, en principio fue conocido como una solución recursiva al problema de filtrado lineal con datos discretos. En la actualidad es uno de los algoritmos de estimación más importantes y comunes porque produce estimaciones de variables ocultas basadas en mediciones inciertas e inexactas.

En general, el filtro de Kalman proporciona un método recursivo para estimar el estado futuro de un sistema dinámico en presencia de ruido. El objetivo del filtro es obtener la mejor estimación de un estado *"x"* en el *"k-ésimo"* paso de tiempo dada una estimación previa en combinación con entradas conocidas. Este filtro sirve para estimar estados futuros del sistema cuando se tiene ruido que se adiciona al modelo (ruido de proceso) y a las señales medidas (ruido de medición), además, considera que existe un modelo que relaciona los estados del sistema con las mediciones realizadas (provenientes de los sensores). El proceso de estimación tiene dos dificultades que deben ser superadas. Primero, existe la presencia de vectores de ruido desconocidos e inmedibles. Por lo tanto, una de las tareas es filtrar las perturbaciones no deseadas. La segunda dificultad es que el estado en general no se puede observar directamente desde las salidas previas. Esto significa que la estimación del estado debe reconstruirse en base a la historia temporal de señales y parámetros conocidos [2].

El proceso de estimación mediante el filtro de Kalman es un proceso iterativo y que consta de dos etapas: predicción y actualización:

1. **Predicción:** En esta etapa el filtro predice el estado, incertidumbre y error siguiente de acuerdo a un modelo del sistema, sin tomar en cuenta el ruido de proceso.

2. **Actualización:** Este paso utiliza la información del sensor o sensores utilizados para corregir las estimaciones mediante la diferencia entre valores medidos y valores predichos, posteriormente se calcula una ganancia de Kalman que actualizará los nuevos valores de estado, incertidumbre y error. La salida del paso de actualización sirve de retroalimentación al paso de predicción, este proceso continúa hasta que la diferencia entre valor predicho y valor medido tiende a cero.

La figura 5.3 ilustra estos dos pasos.

5.5.1. El filtro de Kalman extendido

En esencia el filtro de Kalman fue diseñado para estimación de sistemas lineales. Sin embargo, en aplicaciones reales, la mayoría de los sistemas son no lineales, por esta razón, el filtro de Kalman extendido representa una mejora del filtro de Kalman original. El EKF (Extended Kalman Filter) por sus siglas en inglés tiene la facilidad de predecir el estado futuro de un sistema no lineal aplicando métodos de linealización como la matriz jacobiana.

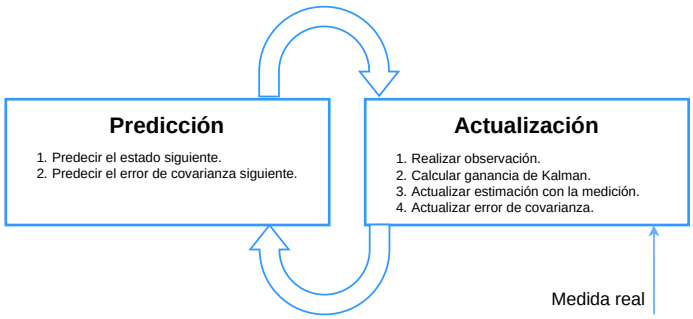


Figura 5.3: Etapas del filtro de Kalman.

El algoritmo para el filtro de Kalman extendido es descrito con los mismos pasos de predicción y actualización como en el caso de sistemas lineales. Con la diferencia de que en cada estimación de estado, se realiza una linealización del sistema.

5.5.2. Estimación de posición y velocidad con el filtro de Kalman extendido

El EKF es utilizado en aplicaciones de seguimiento, navegación, control de vehículos y guía. En el caso de un vehículo autónomo se puede usar el filtro para predecir el siguiente movimiento que realizarán los vehículos detectados al frente o los costados de él. El proceso se logra en función de los datos que recibe el vehículo autónomo, específicamente el conjunto de centroides previamente calculados y posteriormente empatados.

Para este trabajo se busca un modelo que permita predecir posición p y velocidad v de los objetos detectados. En pocas palabras, se busca hacer un seguimiento de objetos (vehículos).

Considerar un entorno 2D como en la imagen 5.4, el vehículo autónomo (en color gris) se encuentra en el origen del sistema de referencia (x, y), uno de los posibles vehículos (en color amarillo) detectados previamente, tiene posición (p_x, p_y) y velocidad (v_x, v_y). Dado que se conoce la posición del vehículo detectado se puede obtener la velocidad del mismo como la derivada de la posición p respecto al tiempo t.

$$v = \frac{dp}{dt}$$

Mientras que la aceleración es la derivada de la velocidad, en este trabajo se considera velocidad v constante del vehículo detectado en todo momento, por lo tanto, se tiene aceleración $a = 0$

$$a = \frac{dv}{dt} = 0$$

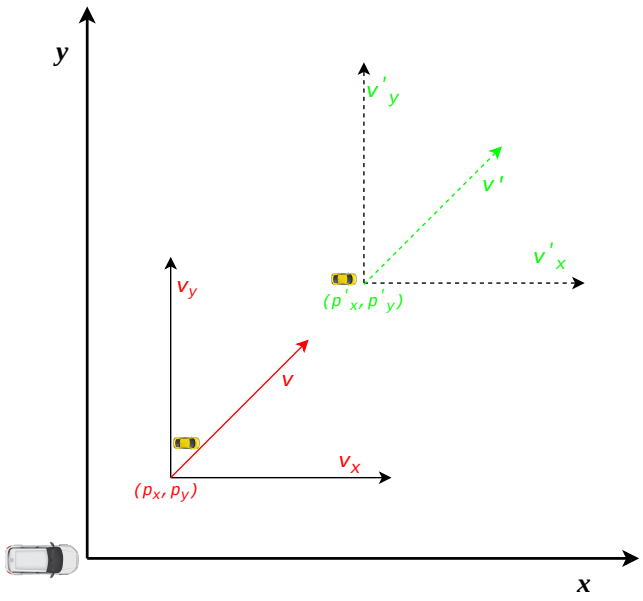


Figura 5.4: Posición y velocidad de un vehículo detectado.

De esta manera el sistema por estimar es el siguiente:

$$\begin{cases} \frac{dp}{dt} = v \\ \\ \frac{dv}{dt} = 0 \end{cases}$$

y para el caso discreto del modelo anterior se tiene:

$$\begin{cases} p_{k+1} = p_k + \Delta t \; v_k + w \\ \\ v_{k+1} = v_k + w \end{cases}$$

Este sistema permite estimar posición y velocidad de un vehículo detectado. A continuación, se representa este modelo aplicado a la componente x del sistema de referencia de la imagen 5.4 para fines prácticos. En el caso de la componente y es de la misma forma. Entonces, el sistema de posición y velocidad para la componente x se expresa como:

$$x = \begin{pmatrix} p_x \\ v_x \end{pmatrix} = \begin{pmatrix} p_x + \Delta t \; v_x + w \\ v_x + w \end{pmatrix} \tag{5.5}$$

Y el estado estimado siguiente es:

$$x' = \begin{pmatrix} p'_x \\ v'_x \end{pmatrix} = \begin{pmatrix} p'_x + \Delta t \; v'_x + w \\ v'_x + w \end{pmatrix} \tag{5.6}$$

donde x es el vector de estados, Δt es el paso de muestreo, p_x y v_x son velocidad y posición actual respectivamente y w es ruido de proceso con matriz de covarianza Q. El modelo de observación para el filtro es:

$$z_k = x_k + w \tag{5.7}$$

con $v = 0$, w es ruido de medición con matriz de covarianza R. Las mediciones para z_k son las posiciones del conjunto de centroides calculados y empatados. Ahora, la estimación de posición y velocidad consta de los siguientes pasos:

Predicción: Con base en el modelo de posición y velocidad se predice el estado siguiente considerando ruido gaussiano para el modelo de estado y el modelo de medición, como sigue:

$$\begin{aligned}\hat{x}_{k|k-1} &= f(\hat{x}_{k-1|k-1}, u_k)\\ \hat{P}_{k|k-1} &= F_k P_{k-1|k-1} F_k^T + Q_k\\ \hat{z}_{k|k-1} &= \hat{x}_{k|k-1}\end{aligned}$$

donde P es la matriz de covarianza para el error de estimación, F_k es el Jacobiano (5.8) de la función f (5.5) y $\hat{z}$ es modelo de observación.

$$F_k = \begin{pmatrix}1 & \Delta t\\ 0 & 1\end{pmatrix} \tag{5.8}$$

Para comenzar la estimación es necesario asignar un valor inicial a la matriz P de ruido de proceso. En este ejemplo P se inicializa con la matriz identidad.

Actualización: La corrección de la estimación se calcula de acuerdo a los errores de observación y estimación. Para el paso de actualización se utilizan las siguientes ecuaciones:

$$\begin{aligned}y_k &= z_k - h(\hat{x}_{k|k-1})\\ S_k &= H_k P_{k|k-1} H_k^T + R_k\\ K_k &= P_{k|k-1} H_k^T S_k^{-1}\\ \hat{X}_{k|k} &= \hat{x}_{k|k-1} + K_k y_k\\ P_{k|k} &= (I - K_k H_k) P_{k|k-1}\end{aligned}$$

donde y_k es un residual de medición, H es el Jacobiano (5.9) del modelo de observación respecto a los estados, la matriz K_k es llamada ganancia de Kalman. Las matrices Q_k y R_k para ruido de proceso y ruido de medición respectivamente, son:

$$Q_k = \begin{pmatrix}0.01 & 0\\ 0 & 0.01\end{pmatrix}$$

y $R_k = 0.1$ es un escalar. El paso de muestreo Δt es $0.05[s]$.

$$H_k = \begin{pmatrix}1 & 0\end{pmatrix} \tag{5.9}$$

Las ecuaciones correspondientes a los pasos de predicción y actualización en forma numérica para la primera iteración son de la forma:

Predicción

$$\hat{x} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ v_x \end{pmatrix} = \begin{pmatrix} p'_x \\ v'_x \end{pmatrix}$$

$$\hat{P} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \Delta t & 1 \end{pmatrix} + \begin{pmatrix} 0.01 & 0 \\ 0 & 0.01 \end{pmatrix}$$

Actualización

$$y = \begin{pmatrix} p_{ox} \\ v_{ox} \end{pmatrix} - \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} p'_x \\ v'_x \end{pmatrix}$$

$$S = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 0.1$$

$$K = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \frac{1}{S}$$

$$x = \begin{pmatrix} p'_x \\ v'_x \end{pmatrix} + Ky$$

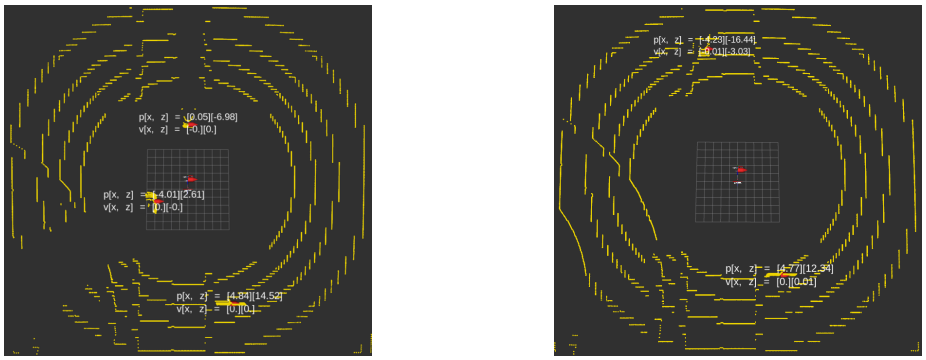
$$P = \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - k \begin{pmatrix} 1 & 0 \end{pmatrix} \right) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Para efectos de implementación en código del Filtro de Kalman Extendido, se optó por crear una clase que contuviera las etapas de predicción y actualización. Después de realizar el proceso de empatado, cada centroide tendrá su propio filtro de Kalman. Mientras c_a y c_p sean el mismo objeto su respectivo filtro de Kalman actualizará posición y velocidad. En el caso de que c_a y c_p sean objetos distintos se creará un nuevo filtro de Kalman para c_a.

```
1  # CREATE N OBJECTS
2  filters = [ [ c, Kalman_Filter() ] for c in centroids ]
3
4  for nc in centroids:
5      # COMPUTE DISTANCES
6      distances = [
7          math.sqrt(
8              (nc[0] - lc[0])** 2 +
9              (nc[1] - lc[1])** 2 +
10             (nc[2] - lc[2])** 2
11         ) for lc in last_centroids
12     ]
13     # GET CLOSER CENTROID (c)
14     min_distance = min(distances)
15     closer_centroid = last_centroids[
16         distances.index(min_distance)
17     ]
18     closer_idx = distances.index(min_distance)
19     if min_distance > threshold:
20         # NC IS A NEW CENTROID, CREATE A NEW OBJECT
21         filters[closer_idx] = [ nc, Kalman_Filter() ]
22     else:
23         # NC AND C ARE THE SAME, UPDATE MEASURE
24         filters[closer_idx][0] = nc
25
26 # APPLY EKF TO FILTERS MATCHED
27 for c,kalman_filter in filters:
28     kalman_filter.ekf(c)
```

La porción de código anterior representa las tareas de empatado de centroides y estimación de posición y velocidad con el EKF. Al inicio (línea 2) se crean n filtros de Kalman para los objetos detectados. Posteriormente, a partir de la línea 4 y hasta la línea 24 se implementa el algoritmo 2 para el empatado de centroides actuales y previos. Finalmente, las líneas 27 y 28 efectúan el proceso de filtrado de posición y velocidad con Kalman. Donde la línea 18 hace uso de la clase 'Kalman_Filter' que calcula cada una de las ecuaciones para predicción y actualización.

Anteriormente, en la sección 2.2 y en específico con la figura 5.2 se mostró el resultado de la detección de objetos mediante el algoritmo de clusterización *k-means*. La imagen 5.5 es una visualización con RViz que muestra la conclusión de empatar centroides y realizar una estimación de posición y velocidad con el filtro de Kalman extendido, a cada objeto se le asigna una etiqueta con sus correspondientes valores de posición y velocidad $[x, z]$ filtrados. En 5.5a se ilustra el seguimiento de objetos estáticos, se estima la posición $p[x, z]$ de un objeto mientras que la velocidad $v[x, z] = [0, 0]$. El caso de seguimiento de objetos en movimiento se muestra en 5.5b donde; la velocidad $v[x, z] \neq [0, 0]$. Es importante mencionar que las velocidades del vehículo solo aparecen en la componente z porque los vehículos en seguimiento se desplazan solo en dirección vertical.



(a) Seguimiento de objetos estáticos.

(b) Seguimiento de objetos dinámicos.

Figura 5.5: Seguimiento de objetos estáticos y dinámicos.

Al finalizar este capítulo el vehículo de pruebas cuenta con un sistema de navegación más completo, al incluir detección de objetos con ayuda de *k-means* y seguimiento de objetos con el filtro de Kalman extendido se puede añadir más complejidad al escenario de pruebas descrito en 3.3 y utilizado en el capítulo anterior. En primer lugar, con la aparición de nuevos vehículos en distintas posiciones y en algunos casos con velocidades controladas. Anteriormente, el vehículo autónomo solo podía navegar en un ambiente sin obstáculos utilizando las líneas del carril como guía. En este punto el vehículo aún no es capaz de navegar de manera segura en un ambiente que presente obstáculos, pero con la inclusión de detección y seguimiento de vehículos el vehículo autónomo cuenta con una mejor percepción del ambiente.

En el capítulo siguiente serán esenciales las implementaciones recién añadidas más las del capítulo 4, esto con el objetivo de diseñar e implementar diferentes comportamientos y estrategias que le permitan al vehículo autónomo navegar por el circuito que ahora cuenta con obstáculos. Además, se buscarán definir reglas para la toma decisiones en situaciones cercanas a la realidad.

Capítulo 6

Comportamientos

En este capítulo se definirán y diseñarán los comportamientos que necesita el vehículo no tripulado para realizar una conducción autónoma. Al comenzar, en la sección 6.1 se da una introducción para los comportamientos esperados del vehículo, enseguida la sección 6.2 expone los diferentes paradigmas de la robótica y específica que paradigma implementará el robot (vehículo). A continuación, en la sección 6.3 se definen tres comportamientos que el vehículo debe realizar, las subsecciones 6.3.1, 6.3.2 y 6.3.3 de esta sección responden a las preguntas de ¿Qué?, ¿Cómo? y ¿Cuándo? se realiza cada comportamiento. Finalmente, la subsección 6.3.4 explica el sistema de decisión que controla los comportamientos.

Para este capítulo las implementaciones en código de los comportamientos así como algún cambio en las leyes de control previamente establecidas son con el lenguaje de programación Python.

6.1. Introducción a comportamientos

Tal como se explicó en la sección 2.1.2 de capítulo 2, un conductor requiere de diferentes habilidades para realizar funciones tácticas y operativas durante el acto de conducir. También, se mencionó que un vehículo autónomo no cuenta de manera natural con estas habilidades, pero si tiene distintos sensores y algoritmos que le ayudan a percibir y conocer el ambiente donde se encuentra. La respuesta o conjunto de respuestas que puede presentar un vehículo no tripulado en situaciones de viales se conoce como comportamiento, la forma en que el robot (vehículo) utiliza las herramientas con las que cuenta son clave para definir uno o varios comportamientos. Al igual que un conductor humano, el vehículo autónomo debe responder correctamente ante diferentes situaciones vehiculares, por ejemplo; mantener una ruta, realizar un rebase, frenar repentinamente o incluso controlar cuidadosamente la velocidad y distancia en situaciones de tráfico.

En este trabajo se pretenden desarrollar tres comportamientos que ayuden al vehículo de pruebas a completar la DDT (*Dynamic Driving Task*) para na-

vegación autónoma con situaciones y condiciones cercanas a la realidad, estos comportamientos son:

1. **Crucero:** Para navegación autónoma sin obstáculos.
2. **Rebase:** Para navegación autónoma con obstáculos.
3. **Mantener Distancia:** Para navegación en situación de tráfico.

Retomando los conceptos de funciones tácticas y operativas descritas en el capítulo 2, el vehículo puede y debe cambiar de comportamiento según las condiciones actuales, para ello es necesario contar con un sistema de decisión que haga el cambio entre comportamientos. En secciones posteriores se describirán estos tres comportamientos además de un árbitro encargado para la toma de decisiones.

6.2. Paradigmas de la robótica

Una filosofía o conjunto de suposiciones y técnicas que caracterizan un enfoque para una clase de problemas en particular se conoce como paradigma. Un paradigma es además, una forma de ver el mundo como un conjunto implícito de herramientas en la resolución de problemas. Con este concepto, algunos problemas parecen más adecuados para diferentes enfoques. Por lo tanto, ningún paradigma es del todo correcto y mucho menos absoluto, simplemente tienen procesos diferentes [23].

Un ejemplo de esto son los paradigmas de la programación. Cualquier solución a un problema que pueda ser expresada en forma de algoritmo pude ser implementada mediante algún lenguaje de programación y posteriormente ser procesada por una computadora. En programación existen diferentes paradigmas de programación como: imperativo, declarativo, orientado a objetos, reactivo, entre otros. Cada uno de estos paradigmas posee características propias y están diseñados para implementar soluciones a cuestiones particulares. Problemas lógico-matemáticos sencillos pueden ser resueltos mediante cualquiera de estos paradigmas, pues todos permiten alcanzar una solución correcta y la diferencia radica en la cantidad de esfuerzo que requiere cada paradigma.

El ecosistema de la robótica recupera esta misma idea de paradigma. Aplicar el paradigma correcto permite facilitar la resolución de problemas. Por lo tanto, conocer los paradigmas de la robótica e inteligencia artificial es clave para programar con éxito un robot y lograr una meta particular. En la actualidad existen tres paradigmas en robótica: Jerárquico, Reactivo e Híbrido (Deliberativo - Reactivo). Los paradigmas de la robótica se describen de dos maneras diferentes.

1. Por la relación que existe entre las tres primitivas de la robótica: Sentir, Planear y Actuar. Por ejemplo, si una función del robot percibe información de los sensores que lo instrumentan y produce una salida útil para otras funciones, entonces esa función entra en la clasificación de Sentido.

Si la función está recibiendo información y produce una o más tareas por realizar (avanzar 2 metros, girar a la izquierda y detenerse) se trata de una función en la categoría de Plan. Finalmente, las funciones que producen comandos de salida hacia los actuadores del robot caen dentro de la categoría de Actuar (girar n grados en sentido horario, con velocidad angular ω).

2. Por la forma en que la información de los sensores es procesada y distribuida a través del sistema. En algunos paradigmas la información de los sensores está restringida para ser utilizada de forma especial en cada función del robot. Otros paradigmas esperan que toda la información del o los sensores primero sea procesada por un modelo global y después los subconjuntos del modelo se distribuyan a otras funciones conforme sea necesario.

6.2.1. Paradigma jerárquico

El paradigma jerárquico es el método históricamente más antiguo para organizar inteligencia robótica tradicional. Prevaleció fuertemente desde el año de 1967, con la aparición del primer robot con IA, Shakey[17], hasta principios de la década de 1990. Este paradigma funciona de manera descendente, es decir, de arriba hacia abajo y con mucha planificación. Se basa en una visión introspectiva de como piensan los humanos."Veo una silla, decido sentarme en ella y genero una trayectoria hacia ella evitando obstáculos". Un robot que implementa el paradigma jerárquico, percibe el mundo, planifica la siguiente acción y después actúa (Sensa, Planea y Actúa), ver 6.1. Dicho de otro modo, este paradigma es secuencial y ordenado. En cada paso el robot planifica explícitamente el movimiento siguiente. Otra característica fundamental del modelo jerárquico es que toda la información percibida por los sensores tiende a reunirse en un modelo global o una sola representación que el sistema planificador (Planear) utiliza para encaminarse a realizar acciones (Actuar) [23]. La detección o el efecto de sentir en el paradigma jerárquico es de manera monolítica: todas las observaciones de los sensores se fusionan en una estructura de datos global , a la que puede acceder el planificador. Esta estructura de datos se conoce como modelo de mundo, donde el término mundo hace referencia tanto al mundo exterior como cualquier otro significado que el robot le atribuya. Dentro de este paradigma el modelo de mundo contiene típicamente:

1. Una representación *a priori* del entorno en que el robot opera.

2. Información de detección del ambiente.

3. Cualquier conocimiento cognitivo adicional que resulte necesario para realizar una tarea.

El paradigma Jerárquico ofrece diferentes ventajas como: orden en la relación Sensar-Planear-Actuar, fomentar la programación monolítica y funcional. Sin embargo, al ser el más antiguo cuenta con varias desventajas entre las que

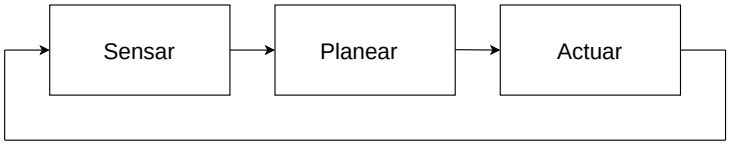


Figura 6.1: Secuencia del Paradigma Jerárquico.

destacan: lenta ejecución en algoritmos de detección y planificación, Requiere la mayor cantidad de información posible del entorno, además de no toma en cuenta la incertidumbre en sensores y actuadores.

6.2.2. Paradigma Reactivo

El Paradigma Reactivo surgió a finales de la década de 1980 como solución a problemas que el Paradigma Jerárquico no podía resolver. El modelo reactivo asume que la entrada de una tarea (Actuar) siempre será una salida directa de un sensor (Sensar). Su enfoque fundamental es que todas las acciones del robot son modeladas y realizadas a través de comportamientos.

Un comportamiento es el conjunto de respuestas que presenta un individuo con relación a su mundo o entorno. Los comportamientos son un mapeo directo de las entradas sensoriales a un patrón de acciones motoras que después se unen para alcanzar una meta. Desde el punto de vista matemático, los comportamientos representan una función de transferencia que transforma entradas sensoriales en comandos de activación [23].

Los sistemas dentro de un paradigma reactivo se componen de comportamientos que mantienen unidos estrechamente la percepción (Sensar) y acción (Actuar), todas las actividades robóticas surgen como resultado de estos comportamientos que operan simultáneamente o en secuencia. La organización de este modelo es Sensar-Actuar sin tomar en cuenta la planificación, la figura 6.2 muestra esto. La percepción en el Paradigma Reactivo es específica o local para comportamiento. Es decir, cada comportamiento tiene acceso directo a uno o más sensores que son independientes de otros comportamientos. En situaciones específicas, más de un comportamiento puede tomar el resultado de un sensor y procesarlo de manera diferente. En esencia, un comportamiento no conoce lo que hace o percibe otro comportamiento. El paradigma reactivo puede combinar comportamientos en una arquitectura mediante dos métodos dominantes: subsunción y suma de campos potenciales.

Las ventajas más reconocidas de este paradigma son los buenos principios de Ingeniería de Software que exhiben los sistemas que lo implementan, los comportamientos se pueden probar de manera independiente. Además, se pueden construir comportamientos más complejos a partir de comportamientos primitivos, existe bajo acoplamiento y alta cohesión. Los comportamientos son inherentemente modulares y fáciles de probar de manera aislada al sistema. Por el contrario, si los comportamientos se implementan de manera deficiente puede derivar en una ejecución lenta. Dentro del paradigma reactivo un robot se vuelve

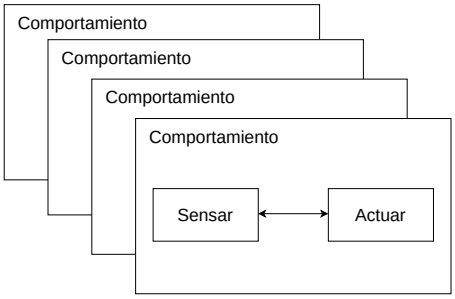


Figura 6.2: Secuencia del Paradigma Reactivo.

más inteligente mientras más y mejores comportamientos existan.

Es importante mencionar que el vehículo autónomo de este trabajo toma como base el paradigma reactivo, cada comportamiento que implemente el vehículo debe hacer uso de las herramientas previamente implementadas para efectuar tareas específicas y mientras más comportamientos se definan, el vehículo autónomo tendrá un nivel de independencia más alto.

6.2.3. Paradigma Híbrido

El paradigma Híbrido o Deliberativo/Reactivo emergió a principios de la década de 1990, es una combinación de muchos de los fundamentos del Paradigma Reactivo y Jerárquico. Bajo las tres primitivas de la robótica, el modelo Deliberativo/Reactivo se puede pensar como Planear y luego Sensar-Actuar (P, S-A), ver figura 6.3. Donde, la parte de Sensar-Actuar siempre es realizada mediante comportamientos reactivos, mientras que la planificación es resultado de una amplía gama de actividades inteligentes. El término híbrido se da porque la planificación se puede intercalar con la ejecución; luego, en mayor medida, el robot planea un conjunto de comportamientos que más tarde ejecutará por un largo tiempo. La detección también es híbrida; la parte reactiva utiliza representaciones locales en cada comportamiento, mientras que la parte deliberativa usa modelos del mundo global como en el paradigma jerárquico.

Generalmente una arquitectura híbrida tiene los siguientes módulos:

- Planificador de misiones
- Secuenciador
- Administrador de comportamiento
- Cartógrafo
- Monitor de Rendimiento

La regla general para clasificar funciones en deliberativas o reactivas es; funciones que operan con información simbólica forman parte de la capa deliberativa, mientras que las funciones que transforman información de los sensores en comandos de activación van a la capa reactiva. El componente reactivo del modelo está compuesto por comportamientos, por otro lado el deliberativo a menudo se subdivide en capas.

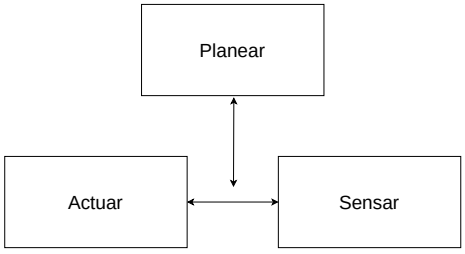


Figura 6.3: Secuencia del Paradigma Híbrido.

6.3. Comportamientos del vehículo

6.3.1. Crucero

De forma similar a como sucede en los sistemas de crucero de un vehículo real, el comportamiento de crucero tiene como objetivo navegar dentro de un circuito vehicular con situaciones controladas y con una velocidad previamente establecida. Estas situaciones incluyen tener un circuito sin obstáculos y mostrar los carriles de la carretera con sus debidas delimitaciones en todo momento, tanto en líneas rectas como en curvas. La figura 6.4 muestra un escenario con estas condiciones. En otras palabras, el comportamiento de crucero buscar realizar concretamente un seguimiento de carril. Para ello, este comportamiento utiliza el detector y seguidor de carril explicados en el capítulo 4, también hace uso de las ecuaciones (4.15)-(4.16) previamente diseñadas que modelan las leyes de control para las funciones operativas de movimiento lateral y longitudinal respectivamente. Este comportamiento se mantiene activo mientras existan condiciones sin obstáculos en el camino del vehículo.

6.3.2. Mantener distancia

Al incluir vehículos al circuito de pruebas la complejidad en la navegación autónoma aumenta. Este comportamiento como su nombre lo indica pretende mantenerse a distancia con los vehículos presentes en la carretera. El comportamiento de mantener distancia actúa en una situación de tráfico vehicular manteniendo al vehículo autónomo a una distancia segura respecto al vehículo de enfrente para evitar colisiones, ver 6.5. Este comportamiento se logra utilizando la detección y seguimiento de carril desarrollados en el capítulo 4 con

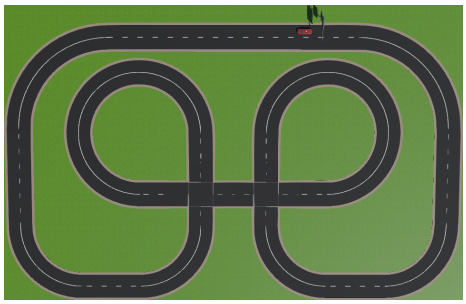


Figura 6.4: Circuito de pruebas sin obstáculos.

una mínima modificación. Primero, la ley de control para el movimiento lateral del vehículo es con la ecuación (4.16), que es la misma utilizada para el seguimiento de carril. La diferencia aparece en la ley de control para el movimiento longitudinal, anteriormente se había mencionado que la velocidad del vehículo autónomo sería constante durante todo el recorrido, esto funciona bien para el comportamiento de crucero. Sin embargo, en el caso de seguimiento de vehículos la velocidad debe variar respecto a la distancia del coche de enfrente para no chocar.

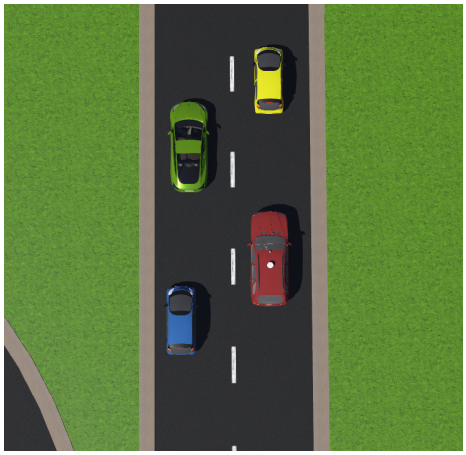


Figura 6.5: Ejemplo de posible situación de tráfico.

Considerar la figura 6.6, en ella se ilustra una posible situación de tráfico, el vehículo en color rojo es el vehículo autónomo y lo llamaremos a, el coche amarillo lo nombraremos b. Es evidente que en está situación no se puede manejar libremente solo siguiendo las líneas del carril con velocidad constante. Por ello en este comportamiento cuando el vehículo b se encuentre a una distancia muy cercana con a se debe de disminuir la velocidad de a, cuando el vehículo b esté más lejos, la velocidad de a aumenta. Es decir, el coche a aumenta o disminuye

su velocidad proporcionalmente a la distancia d. Con esta proporción entre velocidad y distancia se logra seguir al vehículo del frente a una distancia segura y evitar colisiones, por otro lado, las colisiones en los laterales se evitan con el seguimiento de carril, pues la ley de control del movimiento lateral mantiene al vehículo autónomo dentro de su carril.

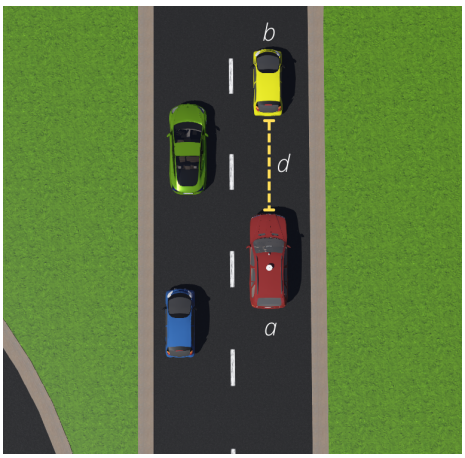


Figura 6.6: Distancia Segura, $d(a, b)$.

La distancia d que existe entre el vehículo autónomo y el vehículo de enfrente la llamaremos "distancia segura", ver 6.6. Con esta distancia se calcula la velocidad del vehículo autónomo. Entonces, para este comportamiento la ecuación (4.15) que representa la ley de control de movimiento longitudinal se expresa de la siguiente forma:

$$v = d * k \tag{6.1}$$

donde, d es la distancia entre a y b, mientras que k es una constante de proporcionalidad para aumentar o disminuir velocidad. El comportamiento mantener distancia entra en acción cuando se presentan situaciones donde no es posible mantenerse en crucero o cuando no se pueda realizar un rebase.

6.3.3. Rebase

El rebase es un comportamiento especial porque cambia considerablemente las leyes de control previamente establecidas para el movimiento lateral y longitudinal. Un rebase requiere de diferentes acciones que involucran el cambio de dirección en las llantas y mantener la velocidad en un rango aceptable, estas acciones deben de ser bien planeadas para evitar choques con el vehículo por rebasar.

Para implementar un comportamiento de rebase en este trabajo específico se optó por realizarlo en condiciones de lazo abierto, es decir, no existe ningún tipo de retroalimentación como en los comportamientos de crucero y seguimiento de vehículos donde la retroalimentación al sistema provenía del error en los

carriles detectados y la distancia segura respectivamente. Se entiende que bajo las normas de conducción vehicular solo se pueden efectuar rebases por el carril izquierdo. Por lo tanto, una vez que se detecta una posible situación de rebase se ejecuta una máquina de estados específica que cumple con esta acción, ver figura 6.7. A continuación se enuncia la secuencia de pasos a seguir para realizar un rebase:

1. Girar a la izquierda t segundos.
2. Girar a la derecha $t/2$ segundos para alinear.
3. Avanzar en línea recta t segundos.
4. Girar a la derecha t segundos.
5. Girar a la izquierda $t/2$ segundos para alinear.
6. Avanzar en línea recta t segundos.

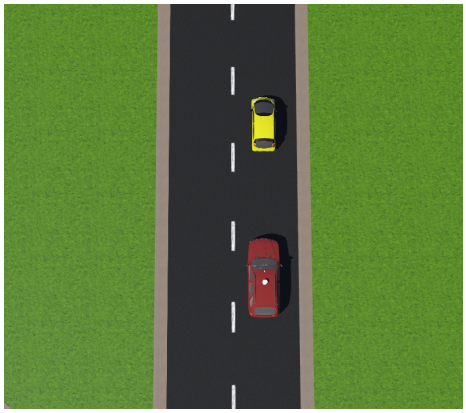


Figura 6.7: Situación de rebase.

Una vez conocidos estos pasos es fácil diseñar una máquina de estados para rebase, la figura 6.8 representa la carta ASM para esta máquina de estados.

Durante el rebase se considera una velocidad constante pero menor a la velocidad utilizada en el seguimiento de carril.

$$v_r < v \tag{6.2}$$

Para controlar el *steering* del vehículo en giros a la izquierda el ángulo de dirección es $\delta < 0$ y en giros a la derecha $\delta > 0$. Además, se toma en cuenta el ángulo de dirección actual del vehículo, de esto modo la ecuación (4.16) para movimiento lateral se transforma en:

$$\delta = \delta_{actual} + \theta \tag{6.3}$$

para giros a la derecha, y

$$\delta = \delta_{actual} - \theta \tag{6.4}$$

para giros a la izquierda.

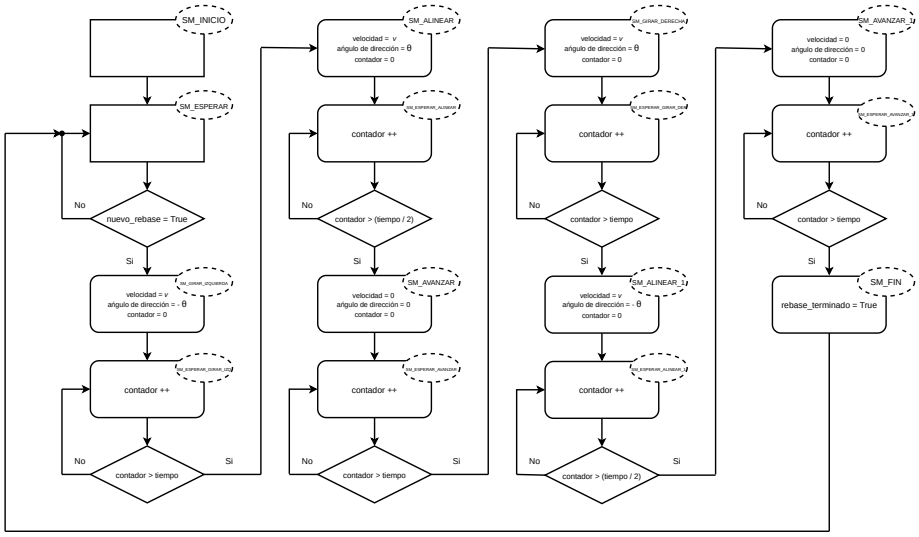


Figura 6.8: Máquina de estados para acción de rebase.

En las ecuaciones (6.3)-(6.4), θ representa un ángulo constante para provocar un giro a la izquierda o a la derecha según corresponda. Esta constante θ como el tiempo t fueron obtenidos mediante experimentación del comportamiento de rebase. El resultado esperado al implementar la carta ASM de la figura 6.8 es el que se muestra en 6.9, donde se observa la secuencia de pasos que debe de seguir el vehículo autónomo para lograr un rebase seguro. El comportamiento de rebase se activa siempre y cuando existan condiciones seguras para evitar choques, es decir, que no existan vehículos cercanos al frente, por detrás y al costado izquierdo del vehículo. También, se considera que lo más seguro después de completar un rebase es volver al comportamiento de crucero.

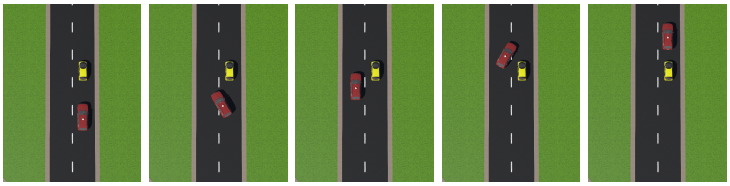


Figura 6.9: Secuencia para comportamiento de rebase.

6.3.4. Árbitro

El árbitro es un sistema que permite tomar decisiones y cambiar al comportamiento más adecuado en cada situación que se presente durante la navegación autónoma en el circuito. Para elegir entre los comportamientos crucero, mantener distancia y rebase, el árbitro percibe el ambiente del vehículo en todo momento. Mientras el vehículo avanza se ejecuta el algoritmo de agrupación *k-means* de la sección 5.3.1, posteriormente el EKF diseñado en 5.5.2 estima la posición y velocidad de los obstáculos (vehículos) detectados. Con base en estas mediciones el árbitro es capaz de definir diferentes escenarios y elegir el comportamiento que mejor se acople.

Dentro del circuito es posible que existan múltiples vehículos en diferentes posiciones, sin embargo, para este proyecto se toman en cuenta cuatro posibles obstáculos que influyan en el cambio de comportamiento del vehículo. Considerar la figura 6.10, en ella se muestra el vehículo de pruebas en color rojo y los objetos en color amarillo son las posibles posiciones de los vehículos obstáculo. Las etiquetas de los vehículos amarillos corresponden a la zona donde se encuentran, para estas zonas se considera al vehículo autónomo (rojo) como el punto de origen, las zonas son:

1. Norte (N)
2. Este (E)
3. Noroeste (NE)
4. Suroeste (SE)

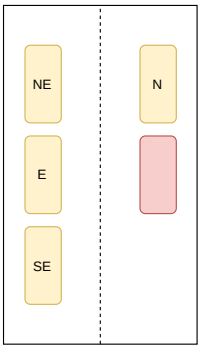


Figura 6.10: Zonas de obstáculos.

Teniendo en cuenta esta configuración de zonas es claro que un vehículo puede o no ocupar una zona, al considerar cuatro posibles vehículos como obstáculos se pueden presentar $4^2 = 16$ estados posibles. Cada uno de estos estados se aprecian abiertamente en la figura 6.11, donde 0 representa una zona ocupada y 1 para zona libre.

Figura 6.11: Estados considerados.

La información de los posibles estados de la figura 6.11 se puede ver en forma tabular. A continuación, el cuadro 6.1 indica cual es el comportamiento más adecuado para cada estado.

E	SE	NE	N	Comportamiento
0	0	0	0	Mantener Distancia
0	0	0	1	Crucero
0	0	1	0	Mantener Distancia
0	0	1	1	Crucero
0	1	0	0	Mantener Distancia
0	1	0	1	Crucero
0	1	1	0	Mantener Distancia
0	1	1	1	Crucero
1	0	0	0	Mantener Distancia
1	0	0	1	Crucero
1	0	1	0	Rebase
1	0	1	1	Crucero
1	1	0	0	Mantener Distancia
1	1	0	1	Crucero
1	1	1	0	Rebase
1	1	1	1	Crucero

Cuadro 6.1: Tabla de estados.

Con la tabla anterior se puede notar que en 6 de los 16 estados (37.5 %) el comportamiento más adecuado es mantener distancia, en 8 ocasiones (50.0 %) lo más apropiado es la conducción mediante el comportamiento de crucero y solo en 2 de los 16 estados (12.5 %) existen condiciones seguras para realizar un rebase de vehículo.

De forma similar a la implementación del comportamiento de rebase, el árbitro también hace uso de una máquina de estados donde se incluyen todas las condiciones vistas en la tabla 6.1 para escoger el comportamiento apropiado. La carta ASM que representa a la máquina de estados del árbitro se puede ver en 6.12.

Mediante esta máquina de estados el árbitro puede habilitar y deshabilitar comportamientos según sea necesario, además puede enviar y recibir las variables de “distancia segura” y “rebase terminado” para el control de los comportamientos de mantener distancia y rebase respectivamente.

Al concluir este capítulo se definieron los comportamientos con los que contará el vehículo autónomo, cada uno de ellos es utilizado en situaciones y tareas específicas. También, se mencionó que un árbitro estará a cargo de la elección. Es posible que el árbitro sea la parte más importante del sistema de navegación del vehículo autónomo porque es el encargado de decidir el comportamiento adecuado a cada situación, de modo que, si llega a fallar en la elección puede tener como consecuencia salir del camino o en un caso más extremo chocar con

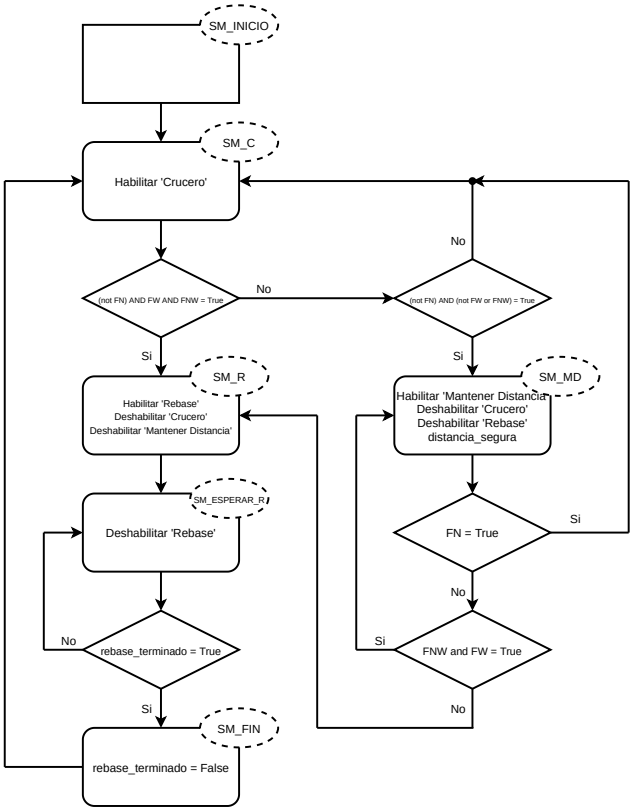


Figura 6.12: Máquina de estados para decidir comportamiento.

algún vehículo. Con estas nuevas características, el sistema de navegación del vehículo autónomo es mucho más completo y ahora debe ser capaz de conducir de forma autónoma en espacios supervisados.

En el capítulo siguiente se expondrán diferentes retos de navegación para el vehículo autónomo donde se pondrán a prueba todos los módulos diseñados e implementados como el detector y seguidor de carril, el sistema para detección y seguimiento de objetos, así como el árbitro que deberá seleccionar el comportamiento apropiado en cada situación como ya se ha mencionado reiteradamente.

Capítulo 7

Pruebas y resultados

El capítulo de pruebas y resultados como su nombre lo indica pretende observar el desempeño del sistema de navegación en diferentes situaciones para interpretar los resultados obtenidos. Al comienzo de este capítulo, en la sección 7.1 se describen de forma breve todos y cada uno de los nodos desarrollados para el sistema de navegación. En las posteriores secciones 7.2, 7.3 y 7.4 se exponen las pruebas para navegación sin obstáculos, navegación con obstáculos y navegación con obstáculos en movimiento respectivamente. En cada una de estas secciones se menciona qué se busca comprobar además de mencionar las condiciones que son requeridas, finalmente se exponen los resultados obtenidos durante cada prueba.

7.1. Integración mediante la plataforma ROS

El sistema navegación del vehículo autónomo es logrado a través de una combinación de diferentes tecnologías. Como se mencionó en capítulos anteriores Webots es el simulador utilizado para modelar los escenarios, vehículos de obstáculo y el propio vehículo autónomo. Los controladores y supervisores que utilizan los elementos modelados en el simulador fueron desarrollados con el lenguaje de programación Python. Por otro lado, los nodos que conforman al sistema de navegación fueron desarrollados en su mayoría con Python y otros más con C++, finalmente, la parte de comunicación por mensajes entre nodos es con la plataforma ROS.

A continuación se describen brevemente cada uno de los nodos que forman parte del sistema de navegación del vehículo.

ros_car_controller

- **Descripción:** Nodo responsable de obtener y comunicar la información de los sensores(Cámara y LIDAR), así como el ángulo de dirección actual del vehículo autónomo. Además, sitúa los valores de velocidad y ángulo de dirección en los actuadores correspondientes del vehículo autónomo.

- **Tópicos publicados:**
  - /current_steering
  - /camera/rgb/raw
  - /point_cloud
- **Tópicos suscritos:**
  - /goal_speed
  - /goal_steering

lane_detect

- **Descripción:** Este nodo tiene la finalidad de procesar una imagen RGB que proviene de la cámara del vehículo para hacer la detección de carril, hace uso de las herramientas de detector de bordes de Canny y detección de líneas rectas con transformada Hough explicadas en el capítulo 4. El resultado de la ejecución de este nodo son dos líneas rectas en forma polar que corresponden a los bordes del carril identificado.
- **Tópicos publicados:**
  - /left_lane
  - /right_lane
- **Tópicos suscritos:**
  - /camera/rgb/raw

object_detect

- **Descripción:** Nodo responsable de obtener la nube de puntos del sensor LIDAR, filtrarla y agruparla con el algoritmo *k-means*. La salida de este nodo es un conjunto de centroides que representan la posición media de los obstáculos detectados.
- **Tópicos publicados:**
  - /object_pose
- **Tópicos suscritos:**
  - /point_cloud

object_tracking

- **Descripción:** Este nodo se encarga de empatar y estimar posición y velocidad de los obstáculos detectados con el algoritmo *k-means*. Para el empatado utiliza el algoritmo 2 y para la estimación el filtro de Kalman diseñado en la sección 5.5.2 del capítulo 5. Como resultado de este proceso se obtiene un conjunto de pares [obstáculo, EKF], donde "obstáculo" es un identificador para cada obstáculo detectado y "EKF" es su correspondiente filtro de Kalman con posición y velocidad estimadas.
- **Tópicos publicados:**
  - /filter_pose
- **Tópicos suscritos:**
  - /object_pose

object_avoidance

- **Descripción:** El sistema de árbitro explicado en el capítulo 6 es representado por este nodo. Define el conjunto de zonas ocupadas utilizando las posiciones estimadas con el EKF, implementa la máquina de estados de la figura 6.12 y comunica las señales de habilitación para los comportamientos de crucero, mantener distancia y rebase. También, para el comportamiento de mantener distancia, calcula y comunica la distancia que existe entre el vehículo autónomo y el coche de enfrente, en el caso del comportamiento rebase, recibe una señal que indica cuando se ha completado esta acción.
- **Tópicos publicados:**
  - /enable_LT
  - /enable_KD
  - /enable_PS
  - /safe_distance
- **Tópicos suscritos:**
  - /filter_pose
  - /pass_finished

lane_tracking

- **Descripción:** Este nodo representa al comportamiento de crucero y su objetivo es implementar las leyes de control vistas en el capítulo 4 para las funciones tácticas de movimiento lateral y longitudinal del vehículo (ángulo de dirección y velocidad). Hace uso de los bordes de carril detectados con el nodo /lane_detect y cuenta con una señal de habilitación proveniente del árbitro para activarse o desactivarse.
- **Tópicos publicados:**
  - /goal_speed
  - /goal_steering
- **Tópicos suscritos:**
  - /left_lane
  - /right_lane
  - /enable_LT

keep_distance

- **Descripción:** Este nodo reproduce al comportamiento de mantener distancia. Calcula la ley de control para velocidad del vehículo con la ecuación (6.1) explicada en la sección 6.3.2 mientras que el ángulo de dirección es de la misma forma que en el comportamiento de crucero, se utiliza el detector y seguidor de carril del capítulo 4. Recibe por parte del nodo /object_avoidance un habilitador de comportamiento y una distancia que es utilizada en el cálculo de velocidad.
- **Tópicos publicados:**
  - /goal_speed
  - /goal_steering
- **Tópicos suscritos:**
  - /safe_distance
  - /left_lane
  - /right_lane
  - /enable_KD

overtake

- **Descripción:** Este nodo se encarga de representar al comportamiento de rebase, implementa la máquina de estados vista en la imagen 6.8 para lograrlo. Calcula las leyes de control mostradas en la sección 6.3.3 para el movimiento del vehículo en este comportamiento. Al igual que los comportamientos de crucero y mantener distancia, recibe un habilitador para activarse o desactivarse según corresponda, finalmente, envía una señal al árbitro para indicar cuando se ha terminado una acción de rebase.
- **Tópicos publicados:**
  - /goal_speed
  - /goal_steering
  - /pass_finished
- **Tópicos suscritos:**
  - /enable_PS
  - /current_steering

En la figura 7.1 se pueden observar cada uno de los nodos del sistema de navegación previamente descritos. Ahí, se pueden apreciar los tópicos a los que se suscribe cada nodo así como los tópicos publicados por cada uno.

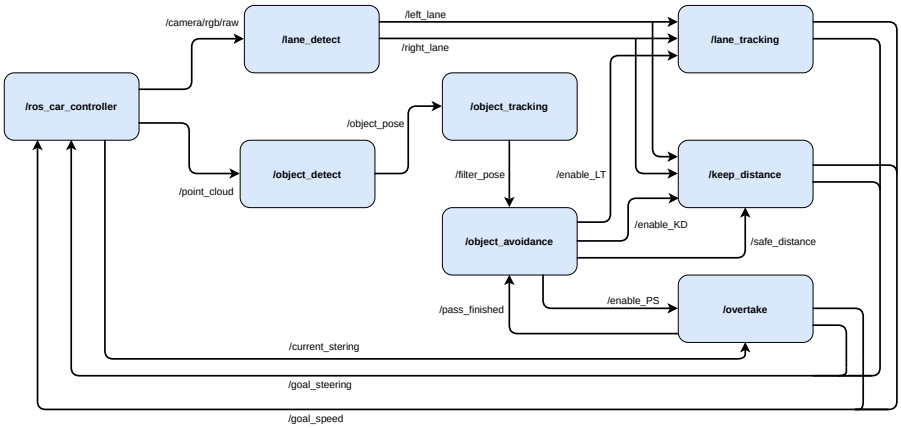


Figura 7.1: Nodos del sistema de navegación. Los rectángulos color azul representan a los nodos y las flechas es la comunicación entre nodos mediante tópicos.

Los nodos descritos anteriormente forman la arquitectura de autonomía de sistema de navegación del vehículo no tripulado de este trabajo. Esta arquitectura cuenta con dos principales sistemas: percepción y decisión. En la figura 4, se muestra a que sistema pertenece cada nodo desarrollado.

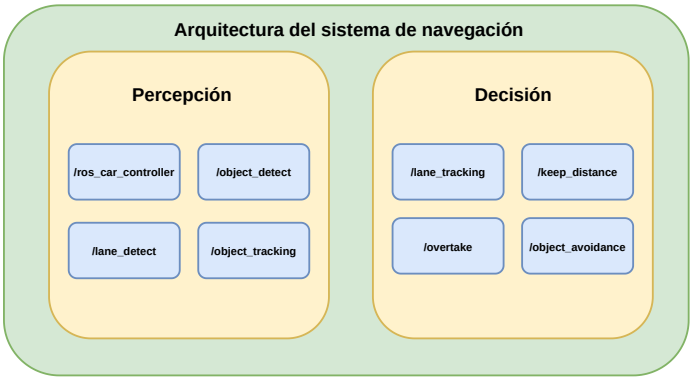


Figura 7.2: Arquitectura del sistema de navegación.

Antes de mencionar los resultados obtenidos en las diferentes pruebas es importante mencionar que el circuito de pruebas es el que se mencionó en el capítulo 3 en la figura 3.6a. En cada una de las pruebas se modificó el mundo creado para añadir elementos o modificar características de los obstáculos.

7.2. Pruebas de navegación sin obstáculos

La prueba de navegación sin obstáculos tiene la finalidad de verificar el comportamiento de "Crucero". Para esta prueba el vehículo autónomo debe completar en su totalidad el circuito sin salir de su carril y sin chocar con los bordes del mismo, en este escenario no existen obstáculos en la carretera, la figura 7.3 muestra el circuito para esta prueba.

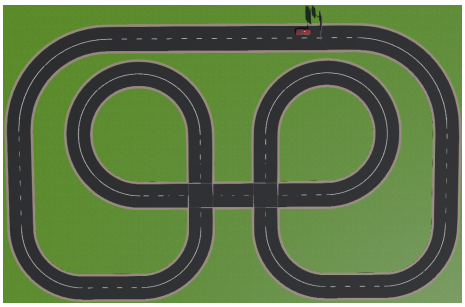


Figura 7.3: Circuito de pruebas para navegación sin obstáculos.

Como ya se ha mencionado, la velocidad v del vehículo en el comportamiento

de "Crucero" es constante en todo momento, la documentación de Webots indica que la velocidad del vehículo es en unidades de km/h. Teniendo esto en cuenta, se realizaron diferentes experimentos para este comportamiento con diferentes velocidades con el fin de observar y comprobar el desempeño de la conducción del vehículo. La siguiente tabla muestra algunas de las velocidades que se probaron. En la tabla 7.1 se puede observar que conforme aumenta la velocidad del vehículo

Velocidad [km/h]	Observación
30.0	Completó el circuito correctamente.
35.0	Completó el circuito correctamente.
40.0	Completó el circuito correctamente.
45.0	Salió del carril en curvas.
50.0	El vehículo se volteó en curvas.
55.0	El vehículo no giró en curvas y salió del circuito.
60.0	El vehículo no giró en curvas y salió del circuito.

Cuadro 7.1: Pruebas de velocidad para navegación sin obstáculos.

el control del ángulo de dirección es inestable. Esto es provocado por el error que existe entre los bordes de carril observados y originales, como consecuencia se obtienen cambios pequeños o muy grandes en el ángulo de dirección. En casos extremos cuando la velocidad v fue > 50 km/h el vehículo logró mantenerse en su carril en caminos rectos pero no logró girar en curvas. Con velocidades en el rango de $[40.0 - 50.0]$ km/h el vehículo se mantuvo en su carril en rectas pero en curvas a la izquierda o derecha se volteó por completo. Sin embargo, cuando la velocidad varió entre 30 y 40 km/h, el vehículo se comportó de muy buena forma, realizó giros suaves en curvas y se mantuvo bien alineado dentro de su carril en todo momento. En este rango de velocidades el vehículo logró completar el circuito en su totalidad sin salir de su carril y por su puesto sin chocar con los bordes del carril.

Con estos resultados se optó por mantener la velocidad en el rango de $[30.0 - 40.0]$ km/h para esta y las siguientes pruebas.

7.3. Pruebas de navegación con obstáculos

Para las pruebas de navegación con obstáculos sin movimiento se añadieron vehículos en diferentes posiciones del circuito pero en el mismo carril que el vehículo autónomo, la modificación al mundo es mostrado en la figura 7.4. En este escenario se busca probar el comportamiento de "Crucero" mientras no existan vehículos que impidan el camino del vehículo autónomo, cuando esto suceda se pondrá aprueba el comportamiento de "Rebase", el cual debe de realizarse sin chocar y sin salir del circuito.

Debido a los resultados obtenidos en las pruebas de navegación sin obstáculos se concluyó que la velocidad que ofrece mayor seguridad es de 30.0 km/h en

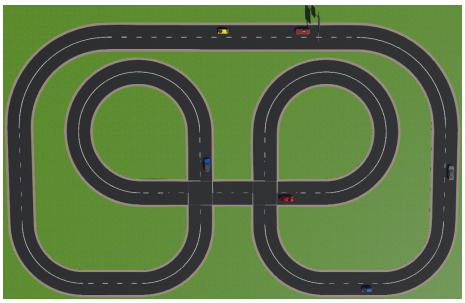


Figura 7.4: Circuito de pruebas para navegación con obstáculos estáticos.

el comportamiento de "Crucero". Para mantener la seguridad del vehículo autónomo durante una acción de rebase se optó por tener una velocidad constante de 20.0 km/h durante este comportamiento.

En la imagen 7.4 se observa que hay en total 5 vehículos como obstáculos. Para la prueba de navegación con obstáculos estáticos se realizaron 10 simulaciones de manera consecutiva, en la siguiente tabla se muestran los resultados con la cantidad de rebases bien hechos por el vehículo autónomo en cada simulación:

Simulación	Cantidad de rebases bien realizados
1	2
2	3
3	5
4	0
5	3
6	4
7	5
8	5
9	0
10	4

Cuadro 7.2: Pruebas de rebase para navegación con obstáculos estáticos.

Con los resultados de la tabla 7.2 se entiende que en un 30 % de las simulaciones se completó el circuito realizando de manera correcta los rebases, en 20 % de las ocasiones se lograron 4 rebases que equivale a recorrer 3/4 del circuito, otro 20 % de las ocasiones indica que solo se realizaron 3 rebases, es decir, recorrió poco más de la mitad del circuito. Los casos más extremos se presentaron cuando solo se realizaron dos rebases (10 %) y ningún rebase (20 %).

Como se mencionó en la sección 6.3.3, el comportamiento de "Rebase" se realiza en lazo abierto. Dicho de otro modo, todas las situaciones de rebase que detecte el vehículo se realizarán de la misma forma, sin ningún tipo de retroali-

mentación. Esto tiene como consecuencia que el vehículo no logré completar un rebase en algunas ocasiones. Sin embargo, durante estas simulaciones se verifica el buen funcionamiento del sistema de árbitro porque se nota el cambio entre comportamientos, el comportamiento de "Crucero" funciona correctamente antes y después de una acción de rebase. En el caso del comportamiento "Rebase" ya se han mencionado los resultados obtenidos.

7.4. Pruebas de navegación con obstáculos en movimiento

Las pruebas de navegación con obstáculos en movimiento tiene dos propósitos principales. Primero, observar el desempeño del comportamiento de "Rebase" cuando el vehículo por rebasar se encuentre en movimiento. El segundo objetivo es verificar el comportamiento de "Mantener distancia" cuando se presente una posible situación de tráfico. Para estos dos propósitos se realizaron algunas modificaciones a los escenarios previos.

Para el comportamiento de "Rebase" se mantuvo la misma cantidad de obstáculos en el circuito, a diferencia del circuito de la figura 7.4 donde el primer vehículo (en color amarillo) se mantiene estático ahora se puede desplazar de forma longitudinal. Para ello, se le asigna una velocidad constante para avanzar hacia adelante, la velocidad de este coche es más pequeña que la velocidad de rebase del vehículo autónomo. Tanto la velocidad como el tiempo que dura el movimiento del vehículo amarillo son manipulados a través de un controlador supervisor. El resto de los vehículos en el circuito se mantienen estáticos pero en diferentes posiciones, en 7.5 se muestra el escenario para esta prueba de navegación.

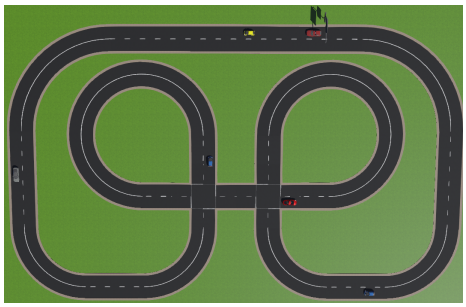


Figura 7.5: Circuito de pruebas para navegación con obstáculos en movimiento.

Al igual que en las pruebas de navegación con obstáculos estáticos se realizaron 10 simulaciones consecutivas para navegación con obstáculos dinámicos. En la tabla 7.3 se muestran los resultados de la acción realizada por el vehículo autónomo solo para el primer rebase, porque este vehículo es el que tiene movimiento.

Simulación	Observación del vehículo autónomo
1	Realizó el rebase correctamente.
2	Realizó el rebase pero choca al costado.
3	Realizó el rebase correctamente.
4	Realizó el rebase correctamente.
5	No realizó rebase.
6	Realizó el rebase pero choca al costado.
7	Realizó el rebase correctamente.
8	Realizó el rebase correctamente.
9	No realizó rebase.
10	Realizó el rebase correctamente.

Cuadro 7.3: Pruebas de rebase para navegación con obstáculos dinámicos.

Como lo indica la tabla anterior en el 60 % de las simulaciones el vehículo autónomo logró realizar el rebase correctamente, solo en dos ocasiones (20 %) se realizó el rebase pero con un ligero choque al costado, lo cual no es correcto y en el otro 20 % no se logró realizar el rebase.

La segunda adecuación tiene que ver con el comportamiento de "Mantener distancia". Retomando el circuito anterior, a este circuito se añadió un vehículo más que se encuentra en el carril izquierdo, ver 7.6. La posición de estos vehículos simulan una posible situación de tráfico como las propuestas en la imagen 6.10. A estos dos vehículos también se les asignaron velocidades constantes para que puedan desplazarse hacia adelante, el movimiento inicia al comienzo de la simulación.

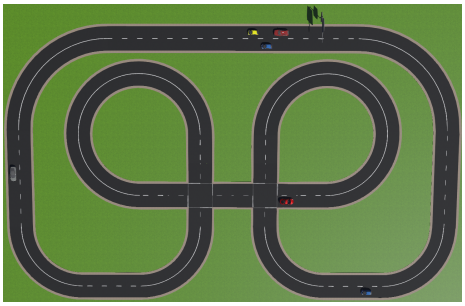


Figura 7.6: Circuito de pruebas para navegación con obstáculos en movimiento (Mantener distancia).

Para la prueba de este comportamiento también se ejecutaron 10 simulaciones de manera continúa, en todas estas simulaciones el vehículo autónomo detectó el comportamiento de "Mantener distancia" y por lo tanto reguló la velocidad original logrando mantenerse a una distancia segura de los vehículos (amarillo y azul) y por ello evitar percances.

Con el conjunto de pruebas realizadas para la navegación del vehículo autónomo en un circuito cerrado sin obstáculos y con obstáculos tanto estáticos como en movimiento se logró observar el desempeño de los comportamientos de "Crucero", "Rebase" y "Mantener distancia" además del árbitro para la elección de estos. Durante estas pruebas los resultados indicaron que se puede completar el circuito propuesto manteniendo situaciones controladas, en el caso de navegación sin obstáculos solo se requirió que fueran visibles las líneas que delimitan el carril y obviamente no existan obstáculos en el camino. El rebase se comprobó con las pruebas de navegación con obstáculos con y sin movimiento, el desempeño de este comportamiento fue positivo en más de la mitad de las ocasiones que fue requerido. Sin embargo, en algunas ocasiones el balance no fue satisfactorio por razones anteriormente expresadas. Finalmente y en específico con la prueba de navegación con obstáculos en movimiento se logró observar el funcionamiento del comportamiento "Mantener distancia", el cual fue exitoso en las simulaciones realizadas.

Estas pruebas también permitieron verificar el funcionamiento de todos los nodos creados, trabajando y comunicándose en conjunto mediante la plataforma ROS además de los controladores desarrollados para algunos elementos presentes en las simulaciones. Como resultado de esto se puede afirmar que el sistema de navegación del vehículo autónomo funciona adecuadamente siempre y cuando se encuentran en escenarios y con condiciones controladas como los vistas a lo largo de este trabajo. En el siguiente capítulo se abordarán conclusiones más específicas en los resultados obtenidos durante todo el desarrollo. También, se mencionarán posibles mejoras por implementar en el sistema de navegación en la búsqueda de obtener mejores resultados.

Cabe mencionar que este es un proyecto de código abierto y por lo tanto, se encuentra libre, en específico, dentro de un repositorio de la plataforma GitHub [1]. Dentro del repositorio se incluyen los códigos fuente de cada nodo, mundos creados en el simulador Webots para las pruebas de navegación así como los controladores y supervisores para el vehículo autónomo y obstáculos. También se incorporan las instrucciones y requisitos necesarios para ejecutar el proyecto completo.

[1] https://github.com/Davidtrejo590/Webots-ROS

Capítulo 8

Discusión

8.1. Conclusiones

Al termino de este trabajo se desarrolló un sistema de visión artificial capaz de reconocer carriles a partir de imágenes RGB. Con esto se comprobó que las herramientas de detector de bordes de Canny y transformada Hough son bastante efectivas en el desarrollo de sistemas de visión artificial. Este sistema sirvió como imitación del sentido de la vista humana para el vehículo y en base a ello se diseñaron leyes de control que le permitieron al vehículo desplazarse lateral y longitudinalmente sin abandonar su carril. Para que el vehículo tuviera un mejor conocimiento del ambiente se implementó un algoritmo de agrupación (*k-means*) sobre una nube de puntos y así obtener información acerca de los objetos presentes en los escenarios de pruebas. También, se diseñó un filtro de Kalman extendido para estimar posición y velocidad de obstáculos potencialmente peligrosos en el camino. En conjunto, los subsistemas de detección de carril, detección y seguimiento de objetos formaron el sistema de percepción para el vehículo autónomo.

El sistema de decisión del vehículo no tripulado se logró en primer lugar por el sistema de percepción, como consecuencia del reconocimiento del ambiente se pudo desarrollar un sistema de árbitro que lograra elegir el comportamiento que mejor convenga en diferentes situaciones viales. La segunda parte de este sistema fue gracias a la elección e implementación de comportamientos similares (seguir carril, rebasar, mantener distancia) a los que realiza un conductor humano.

Se crearon diferentes escenarios de prueba utilizando el simulador Webots tratando de cubrir una pequeña parte de los múltiples escenarios que se pueden presentar en un ambiente vial cotidiano. Con esto se consiguieron comprobar diferentes hipótesis. Primero, al utilizar el simulador Webots durante todo el proyecto se obtiene un claro ejemplo de que los simuladores son alternativas muy rentables en el desarrollo de tecnologías relacionadas con la conducción autónoma, pues los sensores, vehículos y otros elementos del mobiliario urbano fueron esenciales en el desarrollo de los subsistemas que forman parte de la arquitectura

de navegación del vehículo inteligente. Por otro lado, para las diferentes pruebas de navegación el simulador generalmente entregó buenos resultados respecto al control del vehículo, lo cual fue muy importante en la detección de errores y posibles escenarios catastróficos. También, se resolvió el problema de no contar con los elementos físicos necesarios para el desarrollo de estas tecnologías, con el uso del simulador Webots u algún otro, el problema se reduce a explorar, diseñar e implementar conceptos teóricos del mundo de la robótica para generar posibles soluciones que sumen a la conducción autónoma.

Los comportamientos "Seguir Carril", "Rebase" y "Mantener distancia" definidos para el vehículo además de el sistema de árbitro fueron implementados mediante máquinas de estados finitos, en la mayoría de las simulaciones estos comportamientos ofrecieron resultados positivos y se comprobó que las máquinas de estados son funcionales para implementar comportamientos reactivos. Sin embargo, tal como se explicó en el capítulo de pruebas y resultados el comportamiento de "Rebase" fue el más irregular durante las pruebas de navegación como consecuencia de realizarlo en lazo abierto. A pesar de esto, los resultados fueron satisfactorios tanto pruebas de conducción autónoma como para descubrir posibles mejoras a implementar en el futuro.

Es importante mencionar que el trabajo realizado con el simulador Webots en el desarrollo de ambientes urbanos y uso de los sensores para instrumentar al vehículo autónomo fueron útiles durante el Torneo Mexicano de Robótica 2022 en su modalidad virtual. Así mismo, el sistema de navegación autónoma desarrollado fue puesto a prueba en la misma competencia, donde se obtuvieron resultados positivos en las pruebas de navegación autónoma sin obstáculos, navegación autónoma con obstáculos con y sin movimiento además de una prueba adicional para estacionamiento autónomo, la cual no se incluye en este trabajo. El desempeño realizado durante esta competición ayudó a confirmar el funcionamiento estable del sistema de navegación en diferentes situaciones.

8.2. Trabajo futuro

En cuanto al trabajo futuro se pretenden mejorar o en su defecto diseñar nuevas leyes de control para movimiento lateral y longitudinal del vehículo autónomo, sin dejar de lado el modelo cinemático del mismo. En específico se requiere de un control más estable para el control de dirección (*steering*) con velocidades más altas. Como se mencionó en el capítulo de pruebas y resultados, el control del vehículo se vio comprometido en curvas debido a la alta velocidad del vehículo, teniendo como resultado no realizar giros en curvas o en casos extremos provocar una volcadura.

Con el fin de obtener mejores resultados en acciones de rebase es necesario mejorar el comportamiento de "Rebase". Se esperan obtener características de los vehículos por rebasar como: medidas horizontales y verticales, posición real en lugar de una posición media. Con base en estas características calcular un control de velocidad y dirección más adecuado para el vehículo autónomo y obtener acciones de rebase más estables sin depender de aspectos controla-

dos como se probó en algunas simulaciones. Además, se buscarán mejorar los sistemas de detección y seguimiento de objetos para tomar en cuenta no solo la posición sino también la velocidad de vehículos obstáculo y decidir de mejor manera cuando es posible realizar un rebase manteniendo la seguridad tanto del vehículo no tripulado como la de los demás vehículos.

Añadir capas para detección de señales viales y reconocimiento de personas son algunas propuestas para mejorar el sistema de visión artificial. Con esto, se busca añadir más detalle a los escenarios de prueba dentro del simulador y como consecuencia obtener resultados cada vez más cercanos a una conducción autónoma en el mundo vial real.

El trabajo futuro también va enfocado en la búsqueda de complementar las funciones operativas y tácticas del vehículo, algunas funciones estratégicas como selección de destinos y planificación de rutas son tópicos interesantes en el tema de conducción autónoma. Por último, es importante realizar múltiples pruebas y adecuaciones al sistema de navegación con otro tipo de sensores utilizados por vehículos autónomos, por ejemplo; sensores RADAR, GPS e incluso utilizar más cámaras de diferentes tecnologías en puntos estratégicos del vehículo.

Bibliografía

[1] T. Bräunl. *Embedded robotics Mobile Robot Design and Applications with Embedded Systems*. Springer, 2006.

[2] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard. *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[3] S. Daily. Self-driving deep learning with lex fridman holiday repeat. https://softwareengineeringdaily.com/2018/12/27/self-driving-deep-learning-with-lex-fridman-holiday-repeat/, 2018. Consultado en 2022.

[4] F. M. de Robótica. Automodelcar. https://www.femexrobotica.org/tmr2022/automodelcar/, 2022. Consultado en 2022.

[5] D. E. S. S. Dr. Marco Negrete, Dr. José Martínez Carranza. Categoría automodelcar torneo mexicano de robótica, libro de reglas, modalidad virtual. https://github.com/mnegretev/TMR-2022-AutoModelCar/blob/main/Reglas_Virtual/ReglasVirtual.pdf, 2022. Consultado en mayo 2022.

[6] A. Eskandarian. *Handbook of intelligent vehicles*, volume 2. Springer, 2012.

[7] D. Forsyth and J. Ponce. *Computer vision: A modern approach*. Prentice hall, 2011.

[8] GeeksforGeeks. Matlab | rgb image representation. https://www.geeksforgeeks.org/matlab-rgb-image-representation/, 2018. Consultado en 2022.

[9] Geotab. What is gps? https://www.geotab.com/blog/what-is-gps/, 2020. Consultado en 2022.

[10] D. González, J. Pérez, V. Milanés, and F. Nashashibi. A review of motion planning techniques for automated vehicles. *IEEE Transactions on intelligent transportation systems*, 17(4):1135–1145, 2015.

[11] R. C. Gonzalez. *Digital image processing*. Pearson education india, 2009.

[12] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[13] INEGI. La georreferenciación de accidentes de tránsito en zonas urbanas. https://www.inegi.org.mx/contenidos/saladeprensa/boletines/2021/accidentes/ACCIDENTES_2021.pdf, 2021. Consultado en 2022.

[14] INTEL. Beginner's guide to depth. https://www.intelrealsense.com/beginners-guide-to-depth/, 2019. Consultado en 2022.

[15] S. J3016. Surface vehicle recommended practice-taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. *SAE International*, pages 6–41, 2021.

[16] U. Jaume. Fundamentos de visión por computador. https://www.vision.uji.es/courses/Doctorado/FVC/FVC-T2-FormacionImagen-2p.pdf, 2010. Consultado en 2022.

[17] B. Kuipers, E. A. Feigenbaum, P. E. Hart, and N. J. Nilsson. Shakey: from conception to history. *Ai Magazine*, 38(1):88–103, 2017.

[18] B. Lenz. *Autonomous driving: Technical, legal and social aspects*. Springer Open, 2016.

[19] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE intelligent vehicles symposium (IV)*, pages 163–168. IEEE, 2011.

[20] T. Luettel, M. Himmelsbach, and H.-J. Wuensche. Autonomous ground vehicles—concepts and a path to the future. *Proceedings of the IEEE*, 100(Special Centennial Issue):1831–1839, 2012.

[21] V. M. Martínez, G. Gil-Gómez, and A. G. Cerezo. Modelado cinematico y dinamico de un robot móvil omni-direccional. *Malaga*, page 9, 2003.

[22] J. A. Michon. A critical view of driver behavior models: what do we know, what should we do? In *Human behavior and traffic safety*, pages 485–524. Springer, 1985.

[23] R. R. Murphy. *Introduction to AI robotics*. MIT press, 2019.

[24] OGRE. Ogre documentation. https://www.ogre3d.org/about/features, 2022. Consultado en 2022.

[25] OpenGL. Opengl documentation. https://www.khronos.org/opengl/, 2022. Consultado en 2022.

[26] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):4–6, 2016.

[27] S. D. Rathod. An autonomous driverless car: an idea to overcome the urban road challenges. *Journal of Information Engineering and Applications*, 3(13):34–38, 2013.

[28] R. Smith. Open dynamics engine. https://www.ode.org/, 2022. Consultado en 2022.

[29] Stackoverflow. How to produce rgb cube matrix in python? https://stackoverflow.com/questions/22991809/how-to-produce-rgb-cube-matrix-in-python, 2014. Consultado en 2022.

[30] J. Stechschulte. Converting between ros images and opencv images (python). http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython, 2020. Consultado en 2022.

[31] Tesla. Tesla-autopilot. https://www.tesla.com/es_MX/autopilot, 2022. Consultado en 2022.

[32] Waymo. Faq-waymo. https://waymo.com/intl/es/faq/, 2022. Consultado en 2022.

[33] Webots. Webots documentation. http://www.cyberbotics.com, 2022. Consultado en marzo 2022.

[34] P. Wilson and H. A. Mantooth. *Model-based engineering for complex electronic systems*. Newnes, 2013.