UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Implementación de un vehículo inteligente a escala en redes IoT

TESIS

Que para obtener el título de

Ingeniero en Telecomunicaciones

P R E S E N T A

Alejandro Zambrano Espinosa

DIRECTOR DE TESIS

Dr. Luis Francisco García Jiménez

Ciudad Universitaria, Cd. Mx., 2020

Agradecimientos

A mi familia, gracias por su apoyo y soporte a lo largo de mi educación universitaria, por siempre impulsarme a seguir adelante.

Agradezco también al Dr. Luis Francisco García Jiménez, por su consejo y ayuda como asesor de esta tesis, así como su apoyo brindado a lo largo de la carrera.

Así mismo agradezco a mis sinodales por las aportaciones y recomendaciones que surgieron durante la revisión de este trabajo para mejorarlo. Por otro lado agradezco el apoyo brindado por parte del proyecto DGAPA-PAPIIT IA105520.

Índice general

Índice de figuras

Índice de tablas

Resumen

Uno de los objetivos principales de las redes IoT es automatizar una vasta cantidad de tareas de la vida cotidiana. Por ejemplo, el área de los vehículos inteligentes ha tomado una gran importancia ya que, en conjunto con las ciudades inteligentes, pretenden mejorar la calidad de vida, tanto de los conductores como, de los peatones que transitan por las ciudades.

Algunos de los grandes problemas que actualmente se estudian en el tema de tránsito vehicular son, por un lado, el gran número de accidentes derivados de una mala respuesta por parte de los conductores, por otro lado, los grandes tiempos de traslado que se reflejan en un alto índice de contaminación ambiental. Muchos de estos problemas se han atacado desde varias perspectivas, por ejemplo, el uso de inteligencia artificial. Sin embargo, sigue siendo un tema abierto, ya que en este tipo de problemas intervienen una gran cantidad de variables que hace muy compleja su solución. Debido a esto, en esta tesis se propone la implementación de un vehículo a escala que es capaz de seguir a otro vehículo que se encuentra enfrente de él sin la intervención humana, valiéndose únicamente de sus propios medios para su conducción. Esto, con el fin de reducir el número de accidentes. Para ello, se propone un modelo de 4 capas que toman información desde diferentes perspectivas, para después conjuntarlas y así obtener una respuesta que minimice la probabilidad de colisión entre vehículos y de está manera reducir los accidentes.

Específicamente, en la capa 1 se utiliza un sensor infrarrojo, el cual mide la distancia a partir del rebote de un rayo que es detectado por un receptor. En la capa 2, por medio de una cámara montada en el vehículo autónomo y utilizando la biblioteca OpenCV, se mide la distancia entre los dos vehículos tomando como referencia la placa del auto que se encuentra enfrente. En la capa 3, se mide la potencia de recepción emitida por una antena WiFi montada en el auto a seguir, y a partir de este dato se obtiene la distancia estimada entre los dos vehículos mediante un modelo de propagación. Finalmente, la cuarta capa obtiene la probabilidad de tener un buen enlace de comunicación entre los dos vehículos a partir de un umbral de recepción establecido.

Con la integración de toda esta información, el protocolo propuesto en esta tesis reduce la probabilidad de colisión entre los dos vehículos.

Capítulo 1

Introducción

Actualmente, los temas relacionados con el transporte inteligente cada vez son más importantes con el paso del tiempo. Dentro de las ramas del transporte inteligente se tienen diversos estudios enfocados a la seguridad, tanto de los conductores, como de los peatones que transitan por las calles diariamente. En los últimos años se han realizado diversos estudios para encontrar soluciones a diversos problemas que se presentan en este sector, como lo son el gran número de accidentes, los tiempos de traslado, la contaminación generada por los vehículos, entre muchos otros.

Para atacar estos problemas se han diseñado algunos sistemas como por ejemplo, el control de velocidad para los conductores de la tercera edad [4], con el cual se pretende reducir una gran cantidad de accidentes y dotar a las personas de la tercera edad de herramientas para poder seguir siendo independientes al momento de transitar en sus vehículos por las ciudades. En relación al tema de los tiempos de traslado, se han diseñado modelos de control vehicular a partir del uso de sensores y la manipulación de semáforos [5], o modelos de conducción cooperativa [6] en donde se busca reducir tiempos a partir de la comunicación *vehicle to vehicle* (V2V), de tal forma que los automóviles toman decisiones sobre su siguiente acción cuando se tienen situaciones como accidentes o eventos repentinos.

El avance en el ámbito de los vehículos autónomos se remonta a los inicios de la década de los años 20's con el diseño de los primeros vehículos a control remoto, después se diseñaron sistemas embebidos para el control de los vehículos. En la década de los años 80's con la introducción de diferentes sistemas de visión se pudieron realizar sistemas de control más precisos, así como la creación de nuevos sistemas para el control de frenado [7]. Hoy en día el uso de sistemas de visión artificial, como la biblioteca OpenCV, en conjunto con diversos sistemas de sensores, son el futuro en el diseño e implementación de mejores sistemas para los vehículos autónomos.

Tomando como base estas tendencias de tecnología, en esta tesis se propone la implementación de un vehículo a escala que sea capaz de seguir a otro que se encuentra enfrente de él sin la intervención humana, valiéndose únicamente de sus propios medios para su conducción. Esto con el fin de reducir el número de accidentes. Para ello se propone un modelo de 4 capas que toman información desde diferentes perspectivas. Con ello se podrá obtener una respuesta que reduzca la probabilidad de colisión entre vehículos y de está manera evitar accidentes.

Específicamente, en la capa 1 se utiliza un sensor infrarrojo, el cual mide la distancia a partir del rebote de un rayo que es detectado por un receptor. Este valor se determina a través de triangulación. Este método consiste en medir uno de los ángulos que forma el triángulo emisor-objeto-receptor, donde el receptor detecta el punto de incidencia el cual depende del ángulo y, a su vez, de la distancia del objeto.

En la capa 2, por medio de una cámara montada en el vehículo autónomo y utilizando la biblioteca OpenCV se mide la distancia entre los dos vehículos tomando como referencia la placa del vehículo que se encuentra enfrente. Este proceso de medición se lleva a cabo mediante la identificación del contorno de la placa y comparando el tamaño de este contorno con respecto a una referencia tomada previamente.

En la capa 3, se mide la potencia de recepción emitida por una antena WiFi montada en el auto a seguir y a partir de este dato se obtiene la distancia estimada entre los dos vehículos mediante un modelo de propagación. Finalmente, la cuarta capa obtiene la probabilidad de tener un buen enlace de comunicación entre los dos vehículos a partir de un umbral de recepción establecido, con esto se asegura que haya una buena comunicación entre los vehículos en caso de algún intercambio de información.

Los modelos de cada una de las diferentes capas, así como el protocolo de integración se implementaron en una Raspberry Pi 3 B+, este dispositivo es el encargado de conducir el vehículo a escala de forma autónoma.

1.1. Definición del problema

Hoy en día la gran sobrepoblación de las ciudades hace que el tránsito vehicular crezca a un ritmo más acelerado, esto genera un gran número de problemas como por ejemplo la gran cantidad de accidentes derivados de una mala respuesta al manejar por parte de los conductores. Para ello en esta tesis, se propone la realización de un vehículo a escala capaz de seguir a otro vehículo valiéndose de sus propios medios con el fin de reducir la probabilidad de colisión entre los vehículos y de esta forma reducir el número de accidentes.

1.2. Hipótesis

Un modelo de cuatro capas que estiman la distancia y el ángulo entre dos vehículos reduce la probabilidad de colisión.

1.3. Metas

1.3.1. Meta general

Implementar un vehículo a escala que sea capaz de seguir a otro que se encuentre enfrente de él sin ningún tipo de intervención humana, valiéndose de sus propios medios para su conducción y de esta forma poder prevenir posibles colisiones.

1.3.2. Metas particulares

- Obtener un modelo para la medición de distancias a partir del uso de un sensor infrarrojo.
- Obtener un modelo de propagación a través de la potencia de recepción de una antena WiFi montada en el vehículo a seguir con el fin de estimar la distancia de separación entre los vehículos.
- Obtener un mapa de probabilidad a partir del modelo de propagación que permita conocer el estado del enlace de WiFi entre los dos vehículos.
- Obtener un mapa de distancias a partir del uso de una cámara montada en el vehículo autónomo a través de la biblioteca OpenCV de Python.
- Implementar un protocolo que integre toda la información de los mapas y modelos mencionados para la obtención de un valor de distancia con un mínimo de error que le permita al vehículo autónomo tomar decisiones sobre su dirección y velocidad con el fin de minimizar la probabilidad de colisión entre los dos vehículos.

1.4. Metodología

El proceso para la implementación del vehículo autónomo se divide en dos etapas. En la primera etapa se obtienen los modelos con los cuales se miden distancias y ángulos, estos modelos corresponden a las 4 capas previamente mencionadas: sensor infrarrojo, cámara controlada con OpenCV, medición de potencia de recepción WiFi y probabilidad de enlace WiFi. Cada una de las capas trabaja de manera independiente.

En la segunda etapa se implementa el protocolo que integra la información obtenida de las 4 capas para la obtención de un solo valor de distancia que permita al vehículo tomar decisiones sobre su dirección y velocidad.

Para la implementación del protocolo y el control de los vehículos se optó por el uso de la plataforma Raspberry Pi debido a que este hardware cuenta con una antena WiFi y fácilmente se le puede adaptar una cámara que es compatible con los elementos ocupados para el funcionamiento del protocolo (OpenCV). Además de su versatilidad al momento de programar en diferentes lenguajes. En el caso de esta tesis, los programas que conforman el protocolo fueron implementados en el lenguaje orientado a objetos Python.

1.5. Contribución

- En este trabajo se analizan diferentes formas de obtener información con la cual se pueda dotar a un vehículo a escala de la capacidad de tomar decisiones de forma independiente al momento de seguir a un vehículo.
- Se propone un protocolo que pueda recabar información de las capas anteriormente mencionadas y, de toda esta información obtener un solo valor de distancia y dirección con el cual el vehículo pueda tomar decisiones inmediatas y reducir la probabilidad de colisión entre vehículos.

1.6. Descripción del contenido

- El capítulo 2 presenta algunos de los trabajos relacionados en el ámbito de transporte inteligente.
- El capítulo 3 presenta un análisis de las 4 capas a utilizar para la implementación del vehículo a escala. Se muestra el desarrollo para la obtención de los diferentes modelos correspondientes a las capas descritas.
- El capítulo 4 presenta la estructura del protocolo implementado a partir de las diferentes capas, se muestran algunos de los algoritmos empleados.
- El capítulo 5 presenta las pruebas realizadas con cada una de las capas y los resultados obtenidos de la implementación del protocolo.
- El capítulo 6 presenta las conclusiones generales y la verificación de la hipótesis.

Capítulo 2

Antecedentes

En este capítulo se presentan diversos trabajos relacionados al tema de vehículos autónomos en donde se abordan los temas de frenado inteligente, prevención de accidentes, tiempos de traslado, implementación de protocolos por medio del uso de OpenCV, y vehículos a escala.

2.1. Protocolos de prevención de accidentes

Dentro del gran número de problemas que se pueden presentar al momento de manejar, existen 3 problemas que actualmente tienen una mayor importancia y por consiguiente se les presta un mayor interés. Estos problemas como ya se han mencionado son: el gran número de accidentes que ocurren diariamente, los grandes tiempos de traslado derivados de una mala planeación, tanto de los sistemas de semáforos y señalización como de los conductores que transitan, y por último, el aumento de la contaminación ambiental; resultado de la conjunción de los dos problemas anteriores. Sin duda, de los problemas anteriores, el que tiene aun más prioridad es el que se refiere al gran número de accidentes que ocurren diariamente, esto se debe a que en muchas ocasiones este problema conlleva a la pérdida de vidas humanas.

Para contrarrestar este problema se han diseñado diferentes sistemas que logran tener respuestas más rápidas a las acciones de los seres humanos con el fin de prevenir los posibles accidentes. Un ejemplo de estos sistemas son los que se refieren al control de velocidad. Uno de los sectores de la población que son más vulnerables a sufrir accidentes vehiculares son los adultos de la tercera edad, tanto como conductores como peatones. Por ello se han desarrollado sistemas de control de velocidad con el fin de apoyar a este sector de la población a la hora de manejar de tal forma que puedan seguir siendo independientes al momento de conducir un auto [4]. Estos sistemas funcionan a partir del reconocimiento de señalizaciones en cuanto a los límites de velocidad, en caso de que el vehículo empiece a exceder los límites establecidos, el sistema de control a partir de la identificación del límite disminuye la velocidad del vehículo hasta que esté dentro de los valores de velocidad permitidos. Con esto se reducen las probabilidades de que ocurra un accidente debido a la respuesta tardía de los conductores.

Otro de los sistemas diseñados para la prevención de accidentes consiste en la conducción cooperativa [6], donde los vehículos usan comunicación V2V para el intercambio de información y de esta manera alertar a los demás vehículos en caso de que se haya presentado algún percance. Si bien, aunque uno de los objetivos de implementar vehículos autónomos es disminuir las tasas de accidentes, éstos se siguen presentando debido a que los vehículos pueden presentar fallas en sus sistemas o porque siguen interactuando con vehículos que aun son manejados por humanos.

Otro ejemplo se presenta en [8], donde el objetivo de esta propuesta es poder reconstruir los eventos en cuestión de minutos cuando un vehículo autónomo llega a sufrir algún accidente. Para ello, el auto recopila información sobre la ruta y lo que sucedió en su entorno, y a partir de simulaciones se puede obtener una secuencia de eventos de cómo ocurrieron los hechos.

2.2. Protocolos para la disminución de tiempo en trayectos vehiculares

Para atacar el problema de los grandes tiempos de traslado, también se han llevado a cabo desarrollo de sistemas que tratan de disminuir estos tiempos a partir de la implementación de protocolos o de recopilación de información con el fin de dotar a los vehículos inteligentes de la información necesaria para recorrer las diferentes rutas de una manera más rápida y eficiente.

Una de las alternativas que se han diseñado para eficientar los tiempos de traslado ha sido la implementación de protocolos de conducción cooperativa, además de ayudar a la prevención de accidentes como ya se ha mencionado anteriormente, también a través del intercambio de información los vehículos que se encuentran circulando tienen la capacidad de prever embotellamientos. En caso de que algún vehículo llegue a detenerse, éste avisará a los demás vehículos, los cuales evitaran al vehículo en cuestión y mantendrán un flujo constante de tránsito. De acuerdo al estudio realizado sobre conducción cooperativa [6], el uso de este sistema redujo en un $35\,\%$ los embotellamientos causados por los vehículos que se detenían en el camino.

La solución a este problema es importante para algunos de los sectores de la industria como por ejemplo las compañías que se dedican al envío de mensajería y paquetería. Para reducir los tiempos de entrega de este tipo de empresas se han diseñado protocolos para la recopilación de información sobre las rutas que siguen los vehículos que ofrecen estos servicios.

Otro ejemplo se presenta en [9], donde a partir de la implementación de una infraestructura de red, los vehículos son capaces de transmitir información sobre su localización, velocidad y sobre lo que acontece en el transcurso de sus rutas. Con esta información se realizan protocolos para que los vehículos puedan escoger sus traslados de manera más rápida, tomando decisiones sobre nuevas rutas.

Otro escenario donde se presenta el problema de tiempo es en los estacionamientos, ya que muchas veces el acomodo de los vehículos en los diferentes espacios llega a ser tardado. Además, los vehículos autónomos que actualmente circulan no cuentan con una gran capacidad para realizar este tipo de tareas. En [10] por ejemplo, a través del uso de mapas que indican cuales de estos lugares se encuentran disponibles, los vehículos pueden acceder a un lugar de forma más rápida.

2.3. OpenCV y otros sistemas de visión artificial para la implementación de protocolos en vehículos autónomos

En los últimos años una de las herramientas que ha cobrado gran importancia para la implementación de protocolos y sistemas dentro del ámbito del transporte inteligente como de otro tipo de sectores han sido los sistemas de visión artificial. En lo que respecta a los vehículos inteligentes, este tipo de sistemas han sido de gran ayuda para la recopilación de información sobre el entorno que rodea a los vehículos y que, a partir de lo que suceda en este entorno, el vehículo tiene la capacidad de tomar decisiones evitando así la posibilidad de un accidente.

El uso de sistemas de visión artificial en conjunto con inteligencia artificial (IA) y *deep learning* han impulsado la creación de sistemas que ayudan al reconocimiento de señalizaciones y semáforos [11]. Por medio del procesamiento de imágenes de los distintos tipos de señalización y semáforos que se pueden encontrar, el vehículo aprende a reconocer los tipos de señales de transito y de esta forma puede tomar decisiones de manera correcta al momento de conducir. Dentro de las pruebas realizadas con este tipo de sistema, el porcentaje referente a los errores cometidos por el vehículo al momento de identificar las señalizaciones se encuentra alrededor del 1 %.

Algunos otros sistemas implementados a partir de visión artificial tienen como objetivo mejorar la seguridad tanto de los pasajeros del vehículo como de los peatones. Con relación a este propósito se han diseñado sistemas para verificar que los conductores tengan la máxima atención posible al momento de manejar y evitar posibles distracciones. Tal es el caso mostrado en [12], en el cual por medio de la detección del rostro, ojos y boca se determina si el conductor presta atención al camino o está distraído, en caso de que ocurra lo segundo el sistema manda una alerta para que el conductor vuelva a prestar atención al conducir y de esta forma se reduzcan las probabilidades de sufrir un accidente.

Otros sistemas están orientados al reconocimiento de obstáculos y del entorno en general al momento de conducir, de igual forma para identificar los posibles obstáculos y prevenir colisiones que podrían terminar en accidentes. Por ejemplo la propuesta en [13], donde el sistema es capaz de identificar las lineas que limitan los carriles en las diferentes vías con las cuales los vehículos pueden mantener sus carriles, además de identificar otros vehículos que circulan, así como peatones, bicicletas, motocicletas e incluso árboles.

Algunos otros sistemas que se han diseñado con ayuda de visión artificial están orientados a la identificación de conductores y vehículos en estacionamientos u otros establecimientos. Uno de estos sistemas se basa en el reconocimiento de placas con OpenCV, al obtener la información de los datos de la placa, estos se corroboran con los obtenidos del RFID del conductor y se puede verificar en una base de datos [14]. De esta forma se le da el ingreso a los usuarios a los diferentes establecimientos.

Otro de los sistemas desarrollados con OpenCV tiene como objetivo llevar un control de los lugares y los vehículos que se encuentran en un estacionamiento [15]. Por medio del procesamiento de las imágenes de las cámaras se obtienen datos para estimar los lugares disponibles dentro del estacionamiento y además por medio del uso de una base de datos se lleva un control sobre los vehículos que ingresan o que salen del lugar.

2.4. Uso de vehículos a escala

Algunas de las forma para poner a prueba los protocolos y sistemas diseñados para la solución tanto de los problemas mencionados en los apartados anteriores, así como de algunos otros, es el uso de simulaciones o de vehículos a escala. Por medio de esta última alternativa es posible realizar pruebas por medio de una implementación física sin que exista el riesgo de que, en caso de producirse algún tipo de falla o accidente, lleguen a ponerse en riesgo vidas humanas.

Uno de los sistemas implementados por medio de estos vehículos prototipo se muestra en [6], donde se prueba un sistema de conducción cooperativa por medio de una flotilla de 16 vehículos a escala, los cuales interactúan entre sí con el fin de intercambiar información por medio de comunicación V2V, y de esta manera, mejorar el flujo de los vehículos en caso de que exista algún percance.

En [16], por me dio del uso de vehículos a escala se prueba un algoritmo para la navegación autónoma de los vehículos, el cual está basado en diferentes protocolos y modelos como los son, la navegación por estima (*Dead Reckoning*), persecución pura (*Pure Pursuit*), inteligencia artificial, en donde se usan herramientas como, algoritmos de búsqueda y redes neuronales. Todo esto con el fin de diseñar un algoritmo de navegación que permita al vehículo prototipo seguir ciertas rutas.

Uno de los sectores automotrices que tiene un constante avance tecnológico es el referente a los vehículos de carreras, ya que en esta rama del automovilismo, siempre se busca mejorar el desempeño de los autos con el fin de que sean competitivos dentro de los circuitos de carreras. Para probar los sistemas con los que se pretenden mejorar los vehículos, también llegan a usarse vehículos a escala. Como se muestra en [17] con un vehículo a escala 1:43. En este trabajo se prueba un sistema para estabilizar los autos cuando estos pasan pon las curvas de las pistas a grandes velocidades y, de esta forma, ayudar a los pilotos a tener un mejor control de los vehículos.

Otro ejemplo del uso de los vehículos a escala para el sector de carreras se muestra en [18], donde se uso un vehículo con una escala de 1:10 respecto a un auto de formula 1 (*F1*). Por medio de odometría, que consiste en el estudio de la estimación de la posición de las ruedas de un vehículo durante su desplazamiento, y del uso de un escáner láser, se busca mejorar la navegación de los autos de carreras sobre una pista específica apoyándose también en una base de datos con los mapas de las pistas y localización en tiempo real.

Capítulo 3

Análisis y modelado de las cuatro capas

En este capítulo se presentan las 4 capas que componen el protocolo para el funcionamiento del vehículo autónomo a escala. Se presenta un análisis de cada una de las capas y se muestra el modelo obtenido para cada una de ellas. Es importante mencionar, que el auto a seguir se controla mediante una aplicación vía el protocolo WiFi. En este vehículo se monta una placa Raspberry con antena WiFi que constantemente transmite *beacons*. Mientras que en el vehículo autónomo se monta placa Raspberry, una cámara, el sensor infrarrojo y una antena WiFi que constantemente está leyendo los beacons transmitidos por el auto a seguir.

3.1. Sensor infrarrojo (Capa 1)

En la primera capa se utiliza un sensor infrarrojo (montado en el vehículo autónomo) que estima la distancia que existe entre los dos vehículos. El sensor utilizado en esta tesis es un SHARP GP2Y0A710kF [19], este tiene un rango teórico de medición de 100 a $550\,\mathrm{cm}$, alimentado con un rango de voltaje V_{in} de $4.5\,\mathrm{V}$ a $5.5\,\mathrm{V}$.

El sensor obtiene los valores de distancia a partir del proceso de triangulación. Este método consiste en la medición de uno de los ángulos que forma el triángulo emisor-objeto-receptor (ver figura 3.1). El receptor es un *position sensitive detector* (PSD) que detecta el punto de incidencia del haz infrarrojo, el cual depende del ángulo y a su vez de la distancia del objeto.

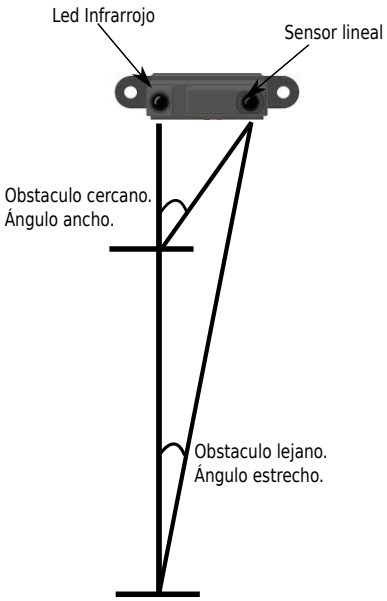


Figura 3.1: Medición de distancia.

Con ayuda de un convertidor analógico digital (ADC) MCP3008 el cual cuenta con una interfaz serial (SPI) [20] se obtienen los valores digitales de 0 a 1023, los cuales son procesados por la Raspberry Pi para que sean traducidos en valores de distancia.

La configuración entre el sensor SHARP GP2Y0A710kF, el ADC MCP3008 y la Raspberry Pi se muestra en la figura 3.2.

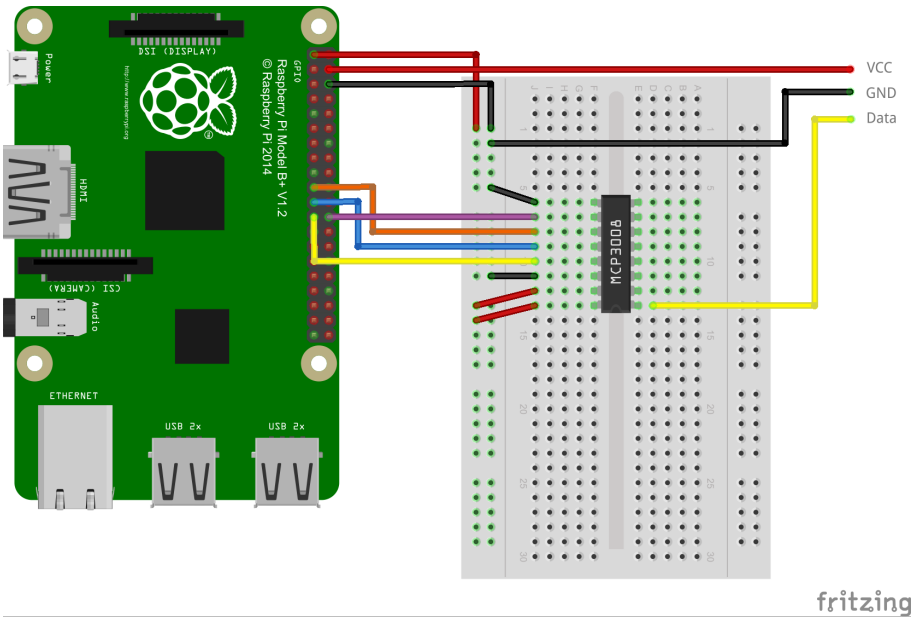


Figura 3.2: Conexión de sensor SHARP GP2Y0A710kF con Raspberry Pi [1].

Debido a las condiciones de luz en las cuales se usó el sensor, luz natural, este obtuvo un rango de medición efectivo entre $60\,\mathrm{cm}$ y $300\,\mathrm{cm}$. Estos dos valores mencionados se obtuvieron por medio del mapeo de diferentes distancias usando el sensor, por medio de este experimento se observo que los limites de distancia diferían respeto a los valores teóricos mencionados en la hoja de datos del sensor. Para la obtención del modelo se midió la distancia a partir del sensor y se graficaron los valores arrojados por el sensor a diferentes distancias con una variación

de $10\,\mathrm{cm}$ entre cada una de ellas. Posteriormente, mediante el uso de regresión polinomial se obtuvo el modelo que genera los valores de distancia (ver tabla 3.1).

Valor del sensor	Distancia [cm]
927	60
879	70
820	80
765	90
721	100
676	110
648	120
620	130
596	140
579	150
561	160
545	170
527	180
522	190
510	200
504	210
498	220
492	230
486	240
480	250
475	260
469	270
463	280
457	290
450	300

Tabla 3.1: Datos obtenidos con sensor infrarrojo.

El modelo obtenido a partir de la regresión polinomial y de los datos de la tabla 3.1 es el siguiente,

$$d = 1{,}6533 \times 10^{-8} * x^4 - (5{,}060346 \times 10^{-5} * x^3) + (0{,}05786 * x^2) - (29{,}515 * x) + 5802{,}812 \qquad (3.1)$$

En la figura 3.3 se muestra la gráfica de los puntos obtenidos de la tabla 3.1 y la curva resultante de la regresión polinomial.

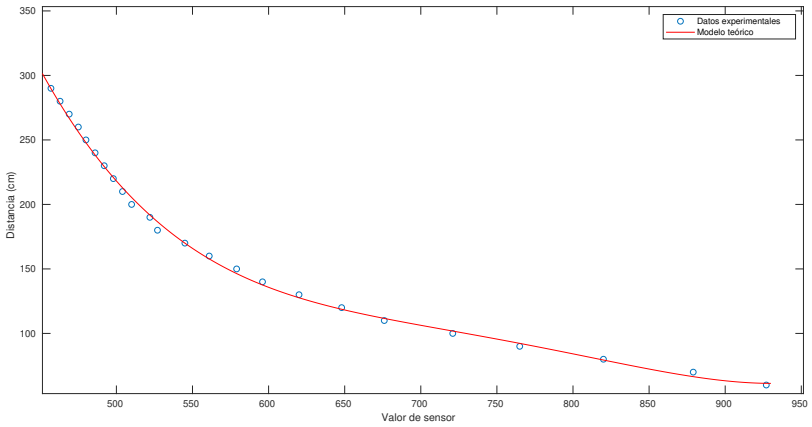


Figura 3.3: Modelo sensor infrarrojo.

A partir del modelo anterior se construyó el programa en Python para el cálculo de distancias por medio del sensor, este programa se muestra en el apéndice D.

3.2. Cámara (Capa 2)

OpenCV es una biblioteca de software libre para el uso de visión artificial y *machine learning*. El propósito de OpenCV es proporcionar una infraestructura para aplicaciones con visión artificial que ayudan a producir productos que posteriormente pueden comercializarse. Al ser de código abierto es sencillo para las empresas usar y modificar los códigos de la biblioteca con el fin de generar productos [21].

Actualmente, la biblioteca de OpenCV cuenta con más de 2500 algoritmos funcionales. El uso de estos algoritmos van desde el reconocimiento facial, identificación de objetos, clasificación de actividades, identificación de imágenes similares, seguimiento en el movimiento de ojos, reconocimiento de ambientes, entre muchos otros. Muchos de estos algoritmos están apoyados en el uso de *machine learning*.

Para esta tesis el hardware usado consiste en una Raspberry Pi 3 B+ conectada con una cámara V1.3 5MP. El sensor de la cámara tiene una resolución nativa de 5 megapixeles y puede tomar vídeos con una resolución de 1080p30 y 720p60.

Para esta capa se obtuvieron valores de distancia y ángulos por medio del uso de la biblioteca OpenCV en Python. La distancia entre los dos vehículos se obtiene a través de identificar el contorno de la placa del auto a seguir y las variaciones en el tamaño de este contorno conforme se separan los vehículos.

3.2.1. Cálculo de distancia

A partir de un *frame* se dibuja un contorno alrededor de la placa. De este contorno se obtiene el valor del ancho de la placa en pixeles. Este ancho se compara con el ancho de una imagen muestra tomada previamente a $30\,\mathrm{cm}$ de distancia del vehículo a seguir.

Figura 3.4: Contorno de la placa.

En la figura 3.4 se muestra el contorno dibujado al rededor de la placa, la cual sirve como referencia para las mediciones de distancia. El proceso para la obtención de los contornos y las áreas con las cuales se obtienen las distancias se muestra a continuación:

1. Con base a una imagen de la placa, primero se obtiene la imagen en escala de grises y se reduce el ruido por medio de un filtro, para después obtener todos los bordes que se encuentran en la imagen por medio del algoritmo de Canny (ver figura 3.5).

Figura 3.5: Imagen con algoritmo Canny.

2. A partir de esta imagen se obtienen todos los contornos que se pueden medir, estos se muestran en color rojo en la figura 3.6.

Figura 3.6: Contornos dentro de la imagen.

3. De todos los contornos obtenidos se seleccionan los que tengan un área poligonal de cuatro lados y que estén dentro de las proporciones del tamaño de la placa, estos están definidos por los puntos azules en la figura 3.7.

Figura 3.7: Puntos de los contornos más grandes.

4. Finalmente se dibuja el contorno con el área más grande (ver figura 3.8). Este contorno es usado para el cálculo de distancias.

Figura 3.8: Contorno alrededor de la placa.

Una vez obtenido el contorno tanto de la imagen muestra (a una distancias entre vehículos de $30\,\mathrm{cm}$) como de la imagen captada en tiempo real, se calcula la distancia de la siguiente forma.

$$focalLength_{model} = \frac{Width_{model} * KNOWN\_DISTANCE}{KNOWN\_WIDTH}, \tag{3.2}$$

$$DISTANCE = \frac{KNOWN\_WIDTH * focalLength_{model}}{Width\_frame}, \tag{3.3}$$

donde:

$focalLength_{model}$: es la distancia de separación en pixeles.

$Width_{model}$: es el ancho en pixeles de la placa muestra.

$KNOWN\_DISTANCE$: es la distancia de separación de la placa muestra en [cm].

$KNOWN\_WIDTH$: es el ancho de la placa muestra en [cm].

$Width_{frame}$: es el ancho en pixeles de la placa captada en tiempo real.

$DISTANCE$: es el valor de distancia calculada.

El cálculo del ángulo se realiza a través de la comparación de los centros de la imagen muestra con respecto a la posición del centro de la imagen captada en tiempo real. Aplicando el teorema de Pitágoras se obtiene el ángulo con el cual se esta moviendo el vehículo que se encuentra enfrente. En la figura 3.9 se muestran los puntos y distancias que se consideran para la obtención del ángulo donde cada punto representa lo siguiente,

C_0**:** Centro de la placa base.

C_1**:** Centro de la placa captada en tiempo real.

D_1**:** Distancia entre centros.

D_2 Distancia de separación entre los dos vehículos.

V_2**:** Vehículo de enfrente

Θ**:** Ángulo medido.

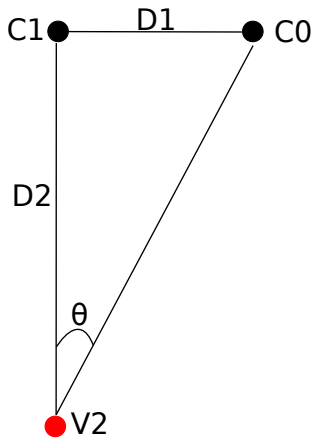


Figura 3.9: Obtención del ángulo.

El reconocimiento de la placa, el cálculo de distancia y de ángulo se da mientras la placa se encuentre de frente a la cámara, si el vehículo a seguir llega a desplazarse ligeramente a la izquierda o a la derecha, la cámara sigue siendo capaz de reconocer la placa y obtener la información. Sin embargo, si el vehículo a seguir da una vuelta completa en cualquier dirección, la cámara pierde de vista la placa y no podrá ser reconocida. Para poder identificar cuando es

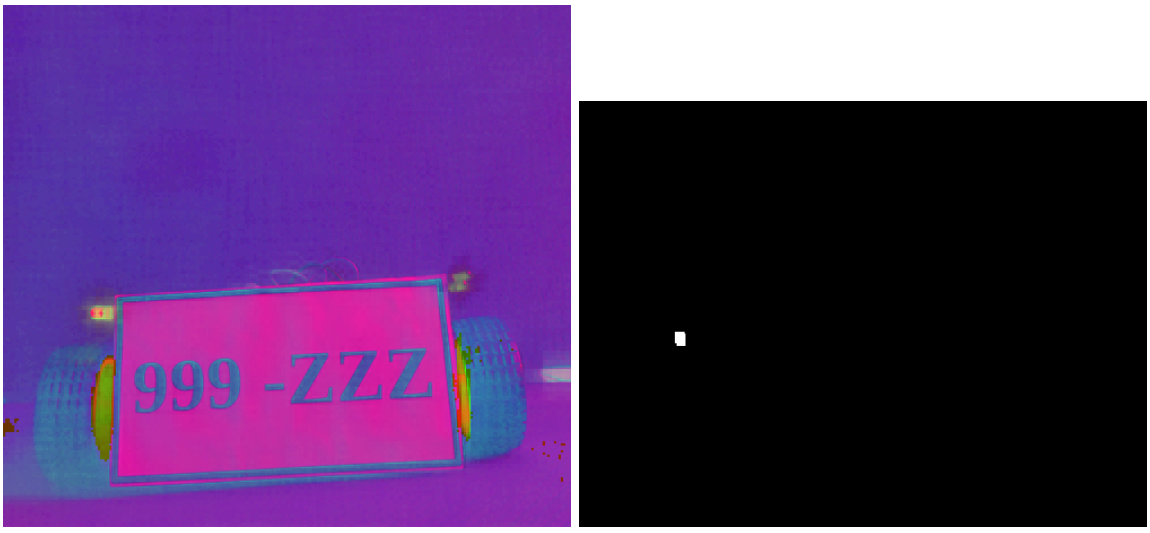
que el vehículo iba a realizar un giro a la izquierda o a la derecha se usó la identificación de direccionales.

Todas las funciones usadas para el calculo de distancias se pueden consultar en los apéndices B y C.

3.2.2. Identificación de vueltas por direccionales

Aunado al cálculo de ángulo, y con el fin de tener un parámetro más certero sobre si el vehículo a seguir toma la decisión de dar vuelta, se propuso la identificación de las luces direccionales. Para la identificación de las direccionales se hizo uso de filtros de color sobre la escala de color HSV. Esta escala se basa en 3 parámetros principales mismos que le dan el nombre.

En las figuras 3.10 (a), (b), (c) y (d) se muestran las imágenes captadas bajo la escala HSV identificando las luces direccionales que indican si se da vuelta a la derecha o la izquierda. Los filtros utilizados para la identificación de las direccionales corresponden al color azul debido a que este es el color de las luces ocupadas en el vehículo a seguir. Los programas se pueden consultar en el apéndice H.



(a) Vista HSV izquierda.

(b) Filtro de color.

(c) Vista HSV derecha.

(d) Filtro de color.

Figura 3.10: Filtro para identificar vuelta a la derecha o izquierda.

3.2.3. Escala de color HSV

El espacio de color HSV o escala HSV (ver figura 3.11) es un modelo utilizado para representar el espacio de color similar al modelo RGB. Esta escala se compone de tres parámetros:

H: Matiz.

S: Saturación.

V: Valor.

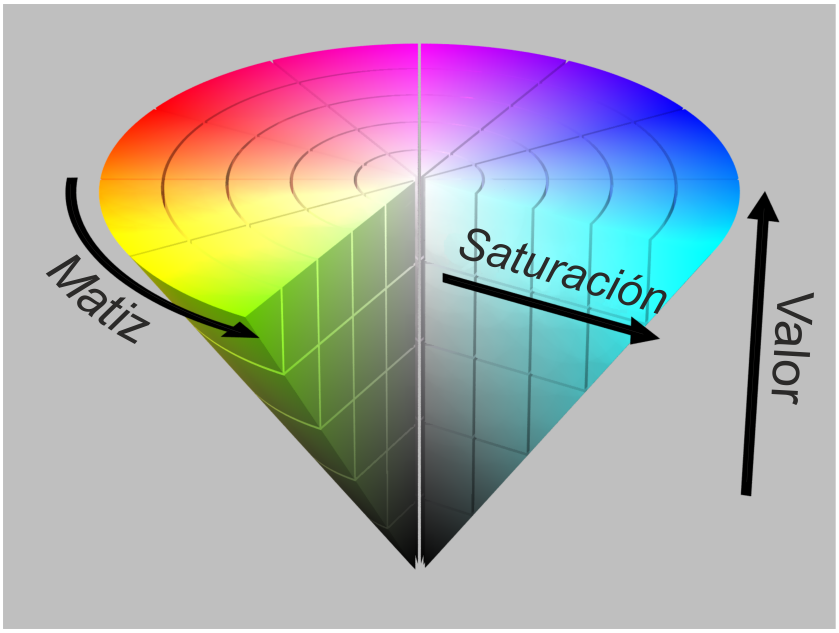


Figura 3.11: Escala de color HSV [2].

El parámetro **H** se refiere al tipo de color, el parámetro **S** se refiere a la saturación del color, que va desde no saturado (representa sombras de grises) a completamente saturado (sin componente blanco), y finalmente el parámetro **V** se refiere al brillo, que permite la intensidad del color. Por ejemplo la figura 3.12 muestra la saturación del color rojo.

Figura 3.12: Saturación de color rojo.

3.3. RSSI (Capa 3)

En esta capa, el valor de distancia entre los vehículos se obtiene a partir de medir la potencia de recepción con respecto a la antena WiFi montada en el vehículo a seguir. Estas mediciones de potencia se realizaron en separaciones de $50\,\mathrm{cm}$ y cada 45^o con respecto a la

antena montada en el vehículo a seguir. En la figura 3.13, se muestra los puntos en los que se tomaron los valores de RSSI. El punto verde que se encuentra en el centro representa la antena transmisora de una de las Raspberry Pi, mientras que los puntos rojos representan las posiciones en las que se coloco la Raspberry Pi receptora para la lectura de valores de RSSI .

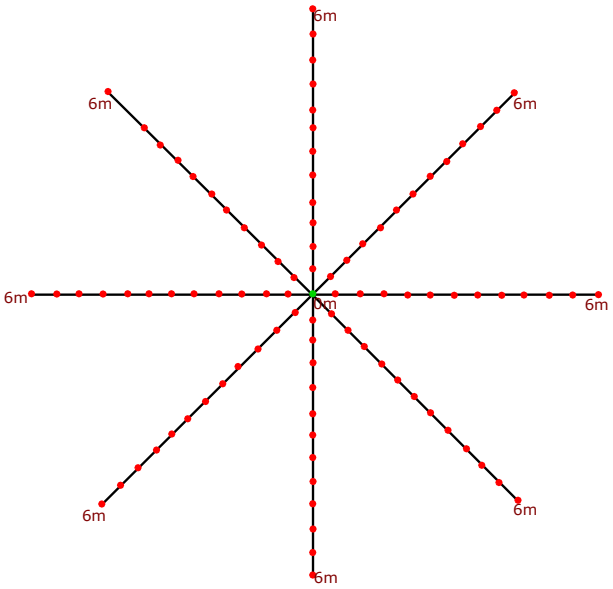


Figura 3.13: Mapeo de RSSI.

Cabe mencionar que en cada punto se tomaron 100 mediciones y se reporta el promedio. A partir del conjunto de mediciones de potencia obtenidas se propone un modelo de propagación con el cual se obtiene el factor de atenuación γ correspondiente al medio de propagación.

Distancia	N	NE	E	SE	S	SO	O	NO	PROMEDIO [dBm]
0 cm	-12	-14	-13	0	-9	-3	-3	-8	-7,75
.5 m	-37	-35	-33	-32	-42	-37	-26	-33	-34,375
1 m	-40	-47	-35	-43	-41	-37	-37	-30	-38,75
1.5 m	-42	-38	-46	-57	-50	-47	-44	-30	-44,25
2 m	-51	-56	-42	-40	-41	-40	-57	-34	-45,125
2.5 m	-44	-41	-48	-50	-51	-47	-43	-48	-46,5
3 m	-54	-42	-47	-44	-48	-45	-38	-52	-46,25
3.5 m	-39	-57	-52	-45	-48	-58	-40	-52	-48,875
4 m	-52	-54	-47	-57	-51	-56	-61	-44	-52,75
4.5	-54	-44	-45	-49	-48	-53	-45	-47	-48,125
5 m	-45	-50	-51	-60	-59	-44	-59	-48	-52
5.5 m	-47	-43	-46	-48	-43	-49	-54	-47	-47,125
6 m	-51	-54	-58	-45	-54	-45	-50	-48	-50,625

Tabla 3.2: Datos muestreados de potencia recibida.

En la figura 3.14 se muestra un mapa de calor donde se puede observar la distribución de los valores de potencia de recepción mostrados en la tabla 3.2. En este mapa los valores de distancia se muestran en el eje x, mientras que los ángulos corresponden al eje y, sobre el eje z se encuentran los valores de potencia de recepción en [dBm], los valores de mayor potencia están representados en color rojo, mientras que los valores de menor intensidad se muestran en color azul, la escala de colores se puede observar en la regla de la derecha mostrada en la figura.

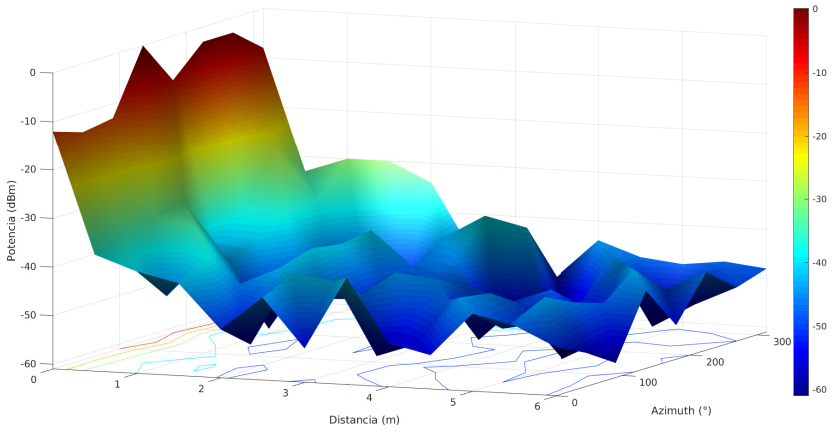


Figura 3.14: Potencia de recepción.

A partir de los datos mostrados en la tabla 3.2, se puede obtener una visualización del patrón de radiación de la antena Wi-Fi de la Raspberry Pi, y de esta forma tener una perspectiva más clara en cuanto a la distribución de los valores de potencia de recepción (ver figura 3.15), donde la antena está localizada en la coordenada (0,0) m. De esta forma, a partir de la figura se puede observar una distribución más clara de la intensidad de la potencia de la señal inalámbrica. Al igual que en la figura anterior, el eje z corresponde a los valores de potencia en dBm, los cuales están coloreados de acuerdo a la escala mostrada a la derecha de la figura. El color rojo corresponde a los puntos con mayor intensidad, mientras que los puntos de color azul representan los puntos con la señal más débil. En esta figura tanto el eje x como el eje y contienen valores de distancia en m, esto con el fin de observar el comportamiento de la potencia de la señal en todas direcciones en un radio de $6\,\mathrm{m}$.

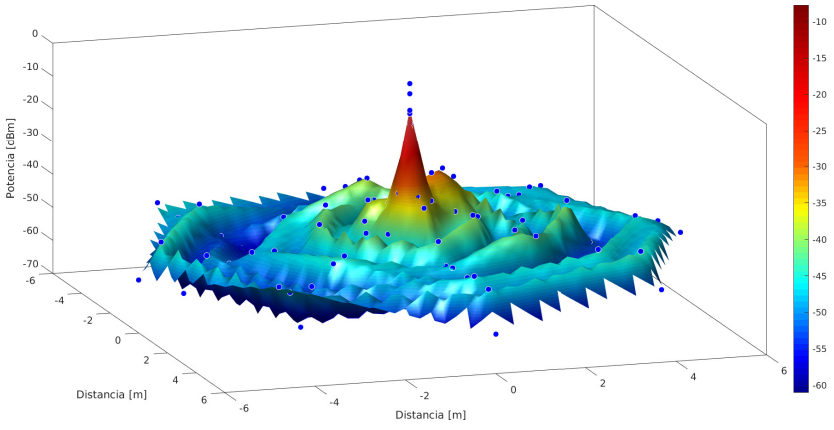


Figura 3.15: Patrón de radiación.

3.3.1. Modelo de propagación

Comúnmente los modelos de propagación toman en cuenta parámetros como la potencia de transmisión, potencia de recepción, la distancia de separación entre las antenas, la frecuencia, entre otros. Para esta tesis se tomó el modelo de atenuación propuesto en [22],

$$P_r = P_t K \left(\frac{d_o}{d}\right)^{\gamma}, \tag{3.4}$$

manejando las variables en [dB] se tiene,

$$P_r = P_t + K - 10\gamma \log_{10}\left(\frac{d}{d_0}\right), \tag{3.5}$$

donde:

P_r: Potencia de recepción en [dB].

P_t: Potencia de transmisión en [dB].

γ: Factor de atenuación.

d: Distancia de separación entre antenas [m].

d_o: Distancia de referencia [m].

K: Factor constante.

El factor K es una constante adimensional que depende de las características de la antena como es la longitud de onda (λ) y la atenuación del espacio libre, definida como:

$$K = 20 \log_{10}\left(\frac{\lambda}{4\pi d_o}\right), \tag{3.6}$$

donde d_0 para escenarios en exteriores puede tomarse de 1 a $10\,\mathrm{m}$ [22]. Para esta tesis se tomo $1\,\mathrm{m}$ y con una frecuencia de $2.4\,\mathrm{GHz}$. Para obtener el factor de atenuación (γ) se realiza una regresión cuadrática usando los datos medidos experimentalmente (ver tabla 3.3) y el modelo de propagación mostrado en 3.5. Con lo cual se obtiene de la siguiente forma,

$$F(\gamma) = \sum_{i=1}^{n} \left[P_{r(medida)_i} - P_{r(modelo)_i}\right]^2, \tag{3.7}$$

donde $F(\gamma)$ representa la variable a buscar (factor de atenuación), $P_{r(medida)}$ la potencia medida experimentalmente y $P_{r(modelo)}$ la potencia de recepción obtenida del modelo de propagación. En la Tabla 3.3 se muestran los valores con los cuales se realizó la regresión cuadrática.

P_r(medida) [dB]	P_r(modelo) [dB]
-34,375	$65{,}046 + 3{,}0103\gamma$
-38,75	$65{,}046 + 0\gamma$
-44,25	$65{,}046 - 1{,}7609\gamma$
-45,125	$65{,}046 - 3{,}0103\gamma$
-46,5	$65{,}046 - 3{,}9794\gamma$
-46,25	$65{,}046 - 4{,}7712\gamma$
-48,875	$65{,}046 - 5{,}4407\gamma$
-52,75	$65{,}046 - 6{,}0206\gamma$
-48,125	$65{,}046 - 6{,}5321\gamma$
-52	$65{,}046 - 6{,}9897\gamma$
-47,125	$65{,}046 - 7{,}4036\gamma$
-50,625	$65{,}046 - 7{,}7815\gamma$

Tabla 3.3: Datos usados para la regresión.

A partir de los datos mostrados en la Tabla 3.3 y utilizando la ecuación 3.7 se obtiene la siguiente expresión:

$$F(\gamma) = 332{,}56354\gamma^2 - 3008{,}4459\gamma + 8534{,}9121. \tag{3.8}$$

Derivando la ecuación anterior respecto a γ e igualando a cero se tiene:

$$\frac{\delta F(\gamma)}{\delta \gamma} = 665{,}12708\gamma - 3008{,}4459 = 0, \tag{3.9}$$

despejando γ se obtiene un valor de $4{,}52$, el cual corresponde al factor de atenuación encontrado para el espacio de propagación de la señal WiFi.

Una vez obtenido el valor de γ, se puede obtener la desviación estándar para el modelo empleado, la cual representa las fluctuaciones lentas de la señal. Para ello se sustituye el valor de γ en la ecuación 3.8 y se obtiene un valor de $E(n) = 1731{,}1$. Finalmente la desviación estándar σ se obtiene como:

$$\sigma = \sqrt{\frac{E(n)}{k}} = \sqrt{\frac{1731{,}1}{12}} = 12{,}01dB, \tag{3.10}$$

donde k es el número de mediciones y σ indica el error que se puede tener al momento de tomar los valores de potencia recibida [22].

3.4. Probabilidad de enlace (Capa 4)

Dentro de los sistemas de comunicación inalámbrica es importante tomar en cuenta los efectos que resultan de la propagación de la señal como por ejemplo las fluctuaciones lentas (σ) o la atenuación promedio (γ). A partir de estos efectos podemos obtener un valor de probabilidad que indique cuando ocurre una interrupción del enlace inalámbrico.

Esta probabilidad de corte o interrupción en inglés (*outage probability*) se comporta como una distribución normal dado un valor de potencia de recepción mínima P_{min} [22]. Por tanto, la probabilidad de corte $p_{out}(P_{min}, d)$ se refiere a la probabilidad de que dada una potencia de recepción a una distancia d, $P_r(d)$ este por debajo de la potencia de recepción mínima $P_{min} : p_{out}(P_{min}, d) = p(P_r(d) < P_{min})$. Combinando los efectos de σ y γ se obtiene el siguiente modelo [22],

$$p(P_r(d) \leq P_{min}) = 1 - Q\left(\frac{P_{min} - (P_t + 10\log_{10} K - 10\gamma \log_{10}(d/d_0))}{\sigma_{dB}}\right), \tag{3.11}$$

donde Q es la probabilidad de que una variable aleatoria gaussiana x con media 0 y varianza 1 sea más grande que z,

$$Q(z) \triangleq p(x > z) = \int_z^{\infty} \frac{1}{\sqrt{2\pi}} e^{-y^2/2} dy. \tag{3.12}$$

$$Q(z) = \frac{1}{2} erfc\left(\frac{z}{\sqrt{2}}\right). \tag{3.13}$$

A partir de este modelo y usando los valores de potencia promedio mostrados en la tabla 3.2, se obtuvieron los valores de probabilidad qué indican que tan factible es tener un enlace inalámbrico confiable a una cierta distancia. Estos valores se muestran en la tabla 3.4. Mientras que la figura 3.16 muestra el mapa de color de la probabilidad de corte con respecto a la distancia.

Distancia [cm]	Potencia promedio [dB]	Probabilidad
50	-34.375	.9999
100	-38.75	.9996
150	-44.25	.9967
200	-45.125	.9886
250	-46.5	.9734
300	-46.25	.9509
350	-48.875	.9218
400	-52.75	.8873
450	-48.125	.8489
500	-52	.8078
550	-47.125	.7654
600	-50.625	.7225

Tabla 3.4: Probabilidad de enlace de acuerdo a la distancia y P_{rx}.

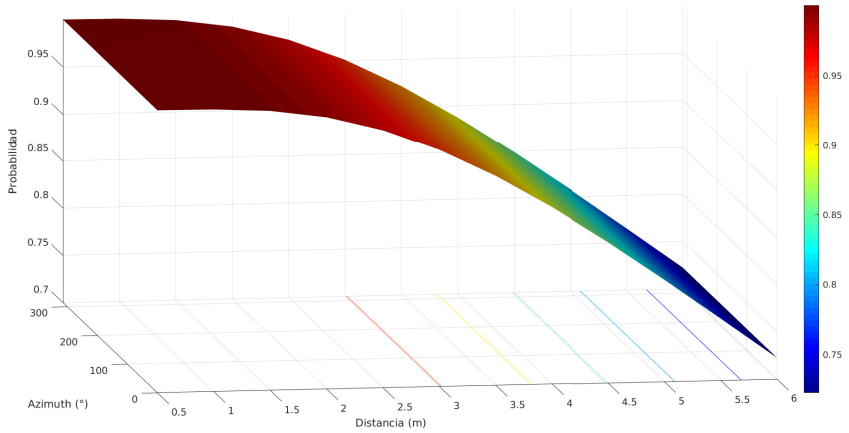


Figura 3.16: Mapa de probabilidades.

Por medio de la figura 3.16 se puede observar que a medida que nos alejamos de la antena transmisora la probabilidad de tener un enlace de comunicación estable disminuye, para el caso de la distancia medida más larga la probabilidad de tener una comunicación estable esta ligeramente por arriba del $70\,\%$.

La lectura de valores de *received signal strength indicator* (RSSI) para su uso tanto en la capa 3 y la capa 4 se obtienen a partir de un comando en la terminal bash de la Raspberry Pi. Sin embargo, se mandó a llamar este comando por medio de un script en Python para facilitar la manipulación de los valores obtenidos y poder usar estos dentro de la integración del protocolo diseñado en Python. Este script se encuentra en el apéndice F.

Capítulo 4

Implementación del protocolo de 4 capas

Este capítulo presenta el protocolo que conjunta la información de cada una de las capas previamente descritas con el fin de que esta información sea la entrada a los motores del vehículo autónomo. Para ello se hace uso del algoritmo kdTree [23].

4.1. Algoritmo kdTree

Un árbol kd es una estructura de datos que almacena y organiza información de un espacio Euclidiano de dimensión *d*. Los árboles kd emplean planos perpendiculares a los ejes del sistema cartesiano para organizar la información. Esto quiere decir que los árboles kd dividen el espacio Euclidiano de tal suerte que la búsqueda de información almacenada en el árbol se realice de manera muy eficiente. Para comprender de mejor manera la construcción de un árbol kd, la figura 4.1 muestra un arreglo de 10 pares de coordenadas:

$$[(14,5),(17,8),(20,10),(12,15),(8,1),(7,2),(2,3),(5,4),(4,7),(9,6)], \tag{4.1}$$

donde cada coordenada está representada por un punto en el plano $\mathcal{R}^2$.

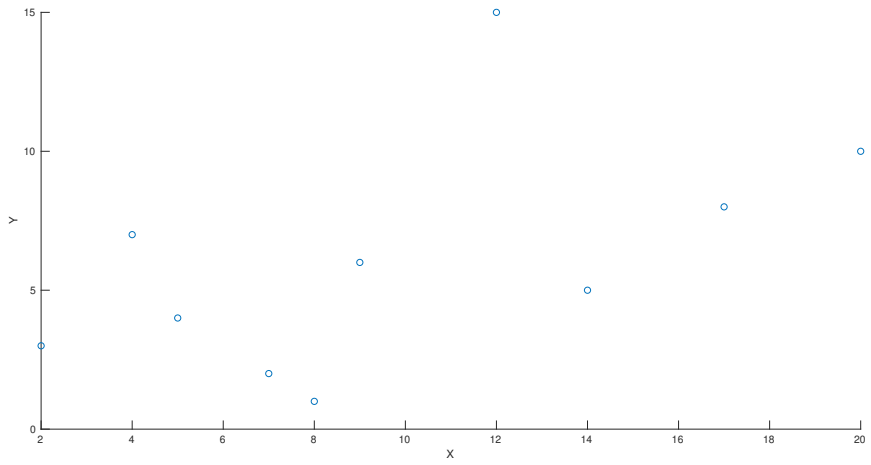


Figura 4.1: Puntos en $\mathcal{R}^2$.

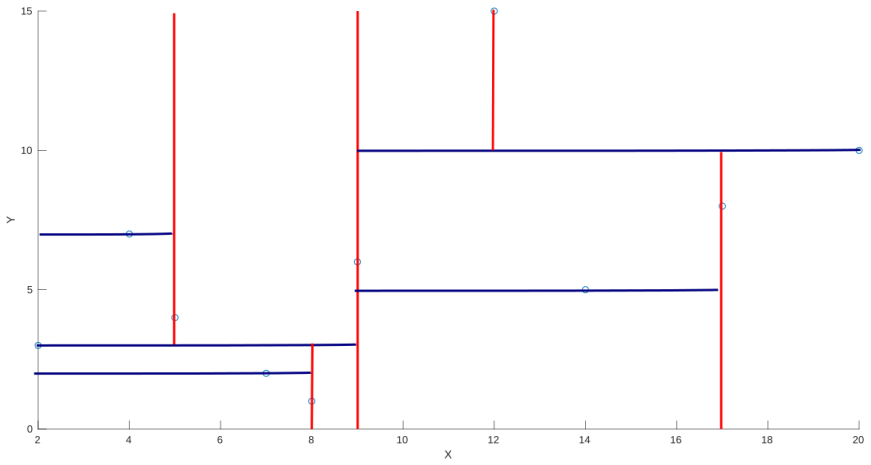


Figura 4.2: Divisiones perpendiculares a los ejes.

La partición del espacio Euclidiano se construye de la siguiente manera. Se ordena el arreglo de puntos con respecto a la coordenada x, y posteriormente se elige el punto que divide al arreglo en dos partes iguales de puntos. Por ejemplo, la figura 4.2 muestra la división del espacio en la coordenada $x = 9$ (ver línea vertical roja) que corresponde al punto medio del arreglo de puntos ordenados con respecto a su coordenada x. Este proceso divide al espacio Euclidiano en dos subespacios (izquierdo y derecho con respecto a la línea vertical roja). Cada subespacio contiene $\frac{n}{2}$ puntos. Posteriormente, para cada subespacio se ordenan los $\frac{n}{2}$ puntos, pero ahora con respecto a la coordenada y, y nuevamente se elige el punto medio. Este proceso nuevamente divide cada subespacio en dos nuevos subespacios, pero ahora con una línea horizontal. Para este ejemplo, la figura 4.2 muestra la línea azul a la izquierda con $y = 3$ y la línea azul a la derecha con $y = 10$. Este proceso se repite recursivamente hasta que ya no haya puntos en el arreglo. La representación de la división del espacio Euclidiano en un árbol se muestra en la figura 4.3, donde se muestran los puntos correspondientes a cada división del plano. El pseudocódigo de kdTree se presenta en el algoritmo 1.

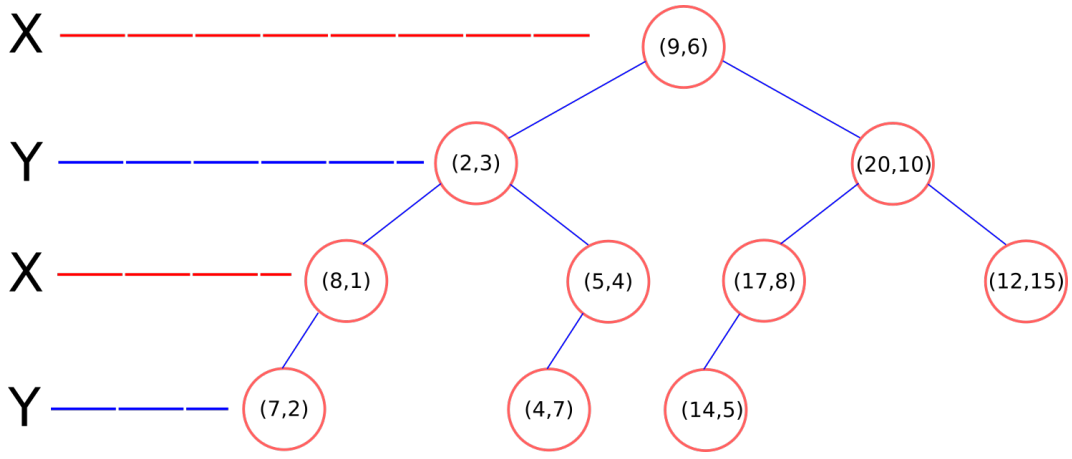


Figura 4.3: Árbol kd resultante.

Algorithm 1 KdTree

```
1: procedure KDTREE(pointList, depth) {
2:     if pointList is empty then
3:         return
4:     else
5:         // Select axis based on depth so that axis cycles through all valid values
6:         var int axis:=depth mod k
7:
8:
9:         // Sort point list and choose median as pivot element
10:        sort pointList using predicate: point1[axis] < point2[axis];
11:        choose median from pointList;
12:
13:
14:        //Create node and construct subtrees
15:        var tree_node node;
16:        node.location := median;
17:        node.leftChild := kdtree(points in
18:        pointList before median, depth+1);
19:        node.rightChild := kdtree(points in pointList after median, depth+1);
20:        return node;
21:    end if
22: }
23: end procedure
```

Por otro lado, dado un punto en el espacio Euclidiano, se puede encontrar un grupo de puntos cercanos o vecinos mediante el algoritmo *nearNeighbors* [23]. Esto se logra por medio de búsquedas recursivas en el árbol kd. Estos vecinos se encuentran a partir de un radio establecido al rededor del punto dado. El pseudocódigo de nearNeighbors se muestra en el algoritmo 2.

Algorithm 2 Serching

```
1: procedure NEARNEIGHBORS(q, node, radius, coord) {
2:     if node = None then
3:         return
4:
5:     end if
6:     if (isInside(q, node.location, radius)==1) then
7:         coord.append(node.location)
8:
9:     end if
10:    if (node.location[node.axis] < q[node.axis]+radius) then
11:        nearNeighbors(q, node.rightChild, radius, coord)
12:
13:    end if
14:    if (node.location[node.axis] > q[node.axis]-radius) then
15:        nearNeighbors(q, node.leftChild, radius,coord)
16:    end if
17: }
18: end procedure
```

4.2. Implementación del algoritmo kdTree para la obtención de datos de Capa 1

A partir de los datos obtenidos en capa 1, se creó una lista que contiene pares de coordenadas (x, y) en los cuales se almacenó la información correspondiente a los valores de distancia y ángulos. Con lo cual cada punto mostrado en la figura 4.4 contiene:

- Distancia de separación entre vehículos.
- Ángulo.

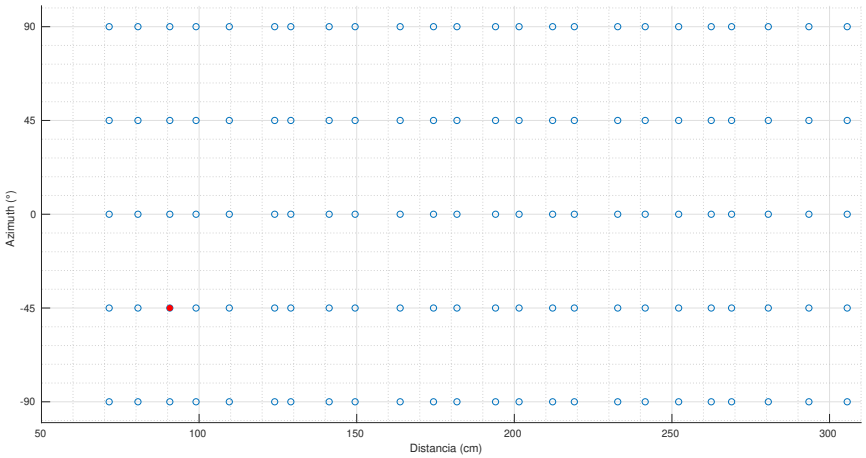


Figura 4.4: Distribución de datos del sensor.

Para el caso del punto rojo marcado en la figura 4.4, le corresponden los valores de distancia de $90.71\,\mathrm{cm}$ y ángulo -45^o.

En esta capa, el objetivo de usar el algoritmo kdTree es mejorar el valor de distancia estimado en cada momento mediante el uso de la función nearNeighbors, ya que dado un radio se puede obtener una lista de valores que son cercanos al valor estimado y, a través de esta lista se puede obtener un promedio con el cual mejorar el valor estimado. Por ejemplo, en la figura 4.5 se muestran diferentes circunferencias que ilustran los nodos vecinos con respecto a los puntos coloreados en rojo. La circunferencia A encierra a 2 nodos vecinos, donde el punto rojo es el valor estimado. Este punto tiene como coordenadas $(90.71, -45^o)$. El radio para este ejemplo tiene un valor de $10\,\mathrm{cm}$. Este valor corresponde a la distancia de separación que hay entre los valores medidos experimentalmente. Por otro lado, la circunferencia B tienen un radio de 45. Este valor corresponde a la separación entre ángulos tomados experimentalmente. Claramente se observa que con este segundo radio se puede considerar un mayor número de vecinos con el fin de mejorar el valor estimado. Por último, en esta figura se muestra la circunferencia C y D que encierran a los vecinos correspondientes al punto resaltado en color rojo; el cual no pertenece a la lista de puntos que conforman el árbol kd. Sin embargo, a partir del árbol, se puede encontrar el punto más cercano al valor estimado. Cabe resaltar que entre más puntos vecinos se tengan la exactitud con la que se puede determinar el valor real de distancia es mayor.

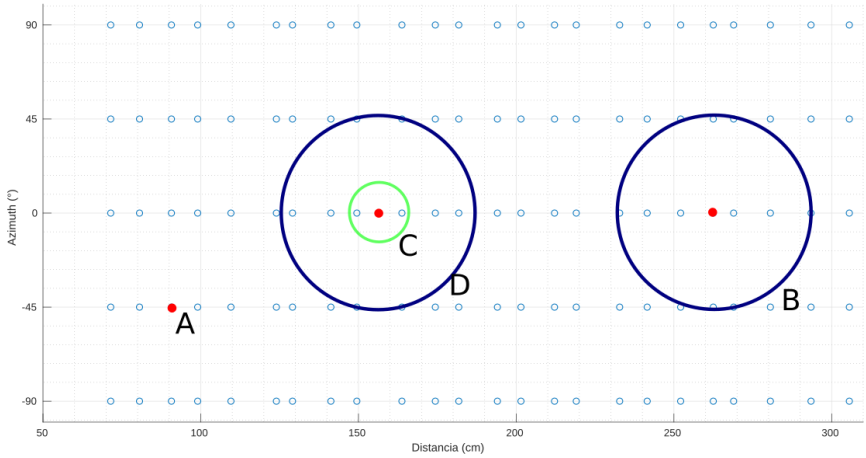


Figura 4.5: Circunferencias para la búsqueda de vecinos cercanos.

4.3. Implementación del algoritmo kdTree para obtención de datos de las Capas 3 y 4

A partir de los datos de potencia de recepción (capa 3) y los valores de probabilidad de enlace (capa 4) se implemento el algoritmo kdTree, donde cada nodo contiene:

- Valor de potencia de recepción.
- Ángulo de recepción.
- Distancia de separación.
- Valor de probabilidad de enlace.

Todos los datos que contiene cada uno de los nodos son información obtenida de las tablas 3.2 y 3.4. La lista de puntos para la creación del árbol kdTree consta de pares de coordenadas (x, y) que representan un valor de potencia y uno de ángulo, respectivamente. Por ejemplo, la figura 4.6 muestra los valores de distancia y ángulo. El punto rojo contiene los siguientes valores:

Distancia: $150\,\mathrm{cm}$.

Ángulo: -90^{o}.

Probabilidad: $0{,}9967$.

P_{rx}: $-44\,\mathrm{dBm}$.

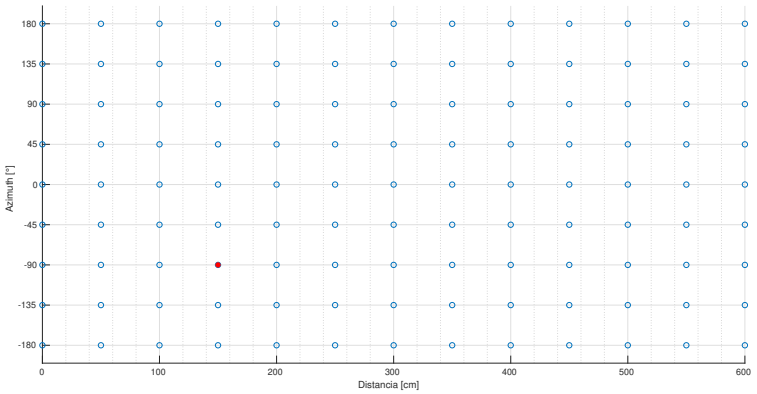


Figura 4.6: Distribución de datos en distancia y ángulo.

Sin embargo, si se observa la distribución de puntos desde los parámetros de potencia y ángulo, se tiene la distribución de puntos mostrada en la figura 4.7.

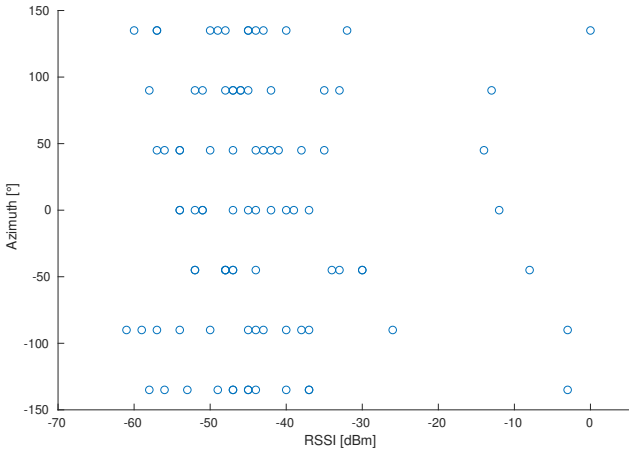


Figura 4.7: Distribución de datos en potencia y ángulo.

Como se mencionó, la capa 3 y 4 dependen de los valores de RSSI, los cuales fueron recopilados y mostrados en el capítulo 3, por lo que a partir del árbol, se busca obtener un valor más cercano de potencia con el cual se pueda obtener un valor de distancia aproximado al real. Para la función *nearNeighbors* se usó un radio con un valor de 12, ya que este valor representa la desviación estándar σ obtenida en el capítulo 3 que hace referencia al posible error que presenta el medir valores de RSSI.

4.4. Integración de las capas

A partir de la recopilación de toda la información de las 4 capas se implementó un protocolo para obtener un valor de distancia y un ángulo que permitan al vehículo tomar decisiones sobre su trayectoria en tiempo real.

Por medio del uso de hilos cada capa obtiene de manera constante valores tanto de distancia como de grados correspondientes al ángulo. Con el valor obtenido por cada capa, se usa el algoritmo kdTree para encontrar los datos más similares o que se encuentran cercanos al dato recibido. Una vez que este algoritmo arroja una lista de datos obtenidos, que está en función del posible error que tiene cada una de las capas, se realiza un promedio con el cual se obtienen los valores finales correspondientes a dicha capa.

Este procedimiento se realiza para cada una de las cuatro capas de tal modo que después de cada ciclo de medición se obtiene un valor de distancia y un ángulo correspondiente a cada una de las capas. A partir de estos 4 valores finales de distancia y ángulo es como se decide la velocidad y dirección del vehículo a escala.

4.5. Configuración del Protocolo

Para el funcionamiento del protocolo se tienen 5 hilos con los cuales se trabaja de manera paralela con los datos obtenidos por cada una de las capas para la obtención de los valores de distancia y ángulo, con estos valores se controla la velocidad en los motores del vehículo para que este puede avanzar o frenar en caso de ser necesario.

En la figura 4.8 se muestran los cinco hilos utilizados, los cuales conforman el protocolo propuesto. En cada uno de los hilos se muestran las acciones que realizan cada uno de ellos dependiendo de la capa a la cual correspondan. Por ejemplo el primer hilo corresponde a la capa 1 referente al sensor de distancia. En la figura se muestra que este hilo realiza el cálculo de distancia y ángulo resultado de los datos obtenidos por el sensor, y de aplicar el algoritmo kd Tree, los cuales son utilizados en el hilo correspondiente al control de los motores (hilo 4). En este último hilo se realiza el cálculo de la velocidad de los motores a partir de los valores obtenidos de los otros hilos referentes a cada una de las capas.

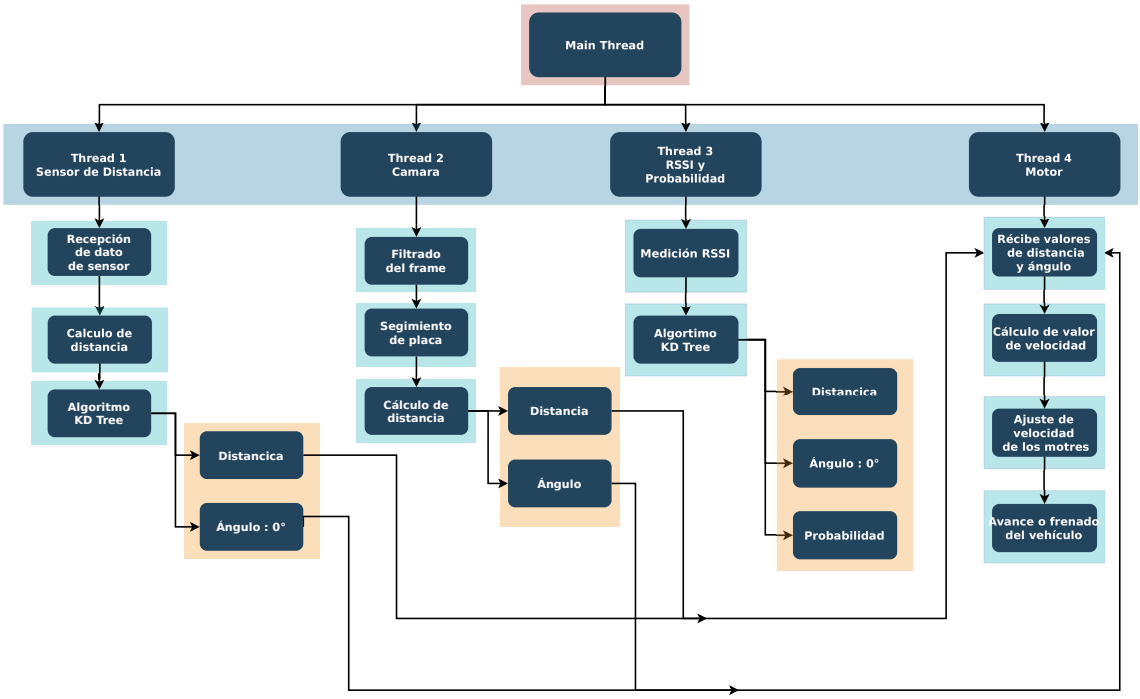


Figura 4.8: Hilos del protocolo.

4.6. Blynk

Por otro lado, la plataforma Blynk se utilizó para controlar el vehículo no autónomo desde un celular. Blynk es una plataforma diseñada para el uso en IoT, por la cual es posible

controlar hardware vía remota, desplegar información generada por sensores, almacenar y administrar esta información por medio de la nube [24]. Esta plataforma esta compuesta principalmente por de tres partes:

- **Blynk App.** Esta permite crear las interfaces para controlar los proyectos por medio de las diferentes herramientas que ofrece la aplicación.
- **Blynk Server.** Es el responsable de llevar a cabo todas las comunicaciones entre la aplicación y el hardware. Se puede acceder al servidor en la nube o también es posible tener un servidor local.
- **Blynk Libraries.** Permiten la comunicación entre el servidor y los procesos que se llevan a cabo entre el hardware y la aplicación.

Una de las grandes ventajas de esta plataforma es la facilidad de usar sus bibliotecas en distintos lenguajes de programación como lo son C, C++, Python, JavaScript, entre otros. Esto abre la posibilidad de controlar una gran variedad de hardware con el cual se pueden implementar proyectos de IoT. La aplicación móvil de Blynk está disponible en los dos sistemas operativos más comunes que son Android y IOS. En esté caso la plataforma de Blynk se uso para el control del vehículo no autónomo en un celular Android. Blynk puede controlar la velocidad de los motores, la dirección del vehículo y el uso de las luces direccionales. Se optó, por usar las bibliotecas de Blynk en el lenguaje Python debido a la facilidad de la implementación del código en este lenguaje, además de que las funciones creadas para el control de los motores se encuentran escritas en Python. Tanto el código usado con las bibliotecas de Blynk como las funciones para el control de los motores se pueden consultar en los apéndices H e I.

4.6.1. Pines virtuales

Para la comunicación entre la aplicación Blynk y el hardware a controlar (motores, luces, Raspberry Pi), se utilizó el concepto de *pines virtuales*. Estos pines virtuales son un concepto creado por Blynk Inc. para facilitar el manejo y la interacción con los diferentes sensores, actuadores o cualquier otro dispositivo que se pueda controlar a partir del hardware principal. La principal ventaja de estos pines virtuales (ver figura 4.9) es que extienden el rango de pines que se pueden usar para el control del hardware [3].

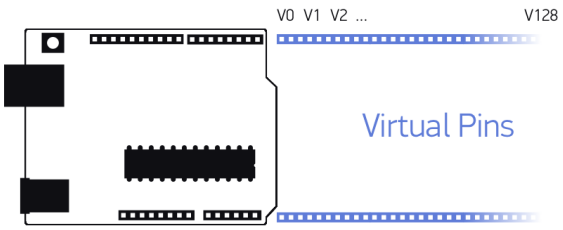


Figura 4.9: Pines virtuales [3].

Ya que la mayoría de los pines de la Raspberry Pi fueron usados para conectar los motores, las luces direccionales y otros componentes, se opto por usar pines virtuales para el intercambio de información desde la aplicación móvil de Blynk hacía la Raspberry Pi. La figura 4.10 muestra una captura de la aplicación diseñada para el control del vehículo a seguir.

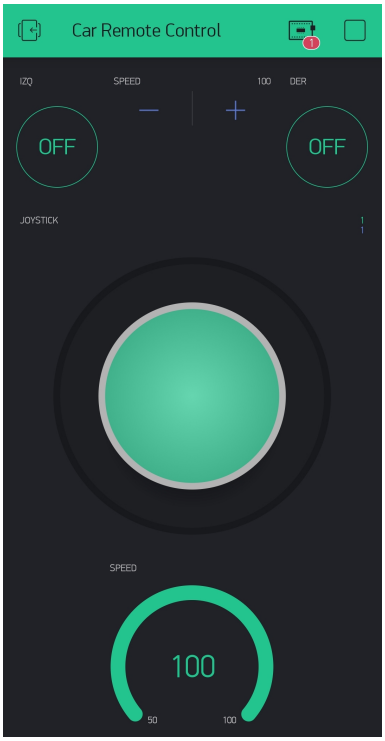


Figura 4.10: Interfaz de aplicación en Blynk.

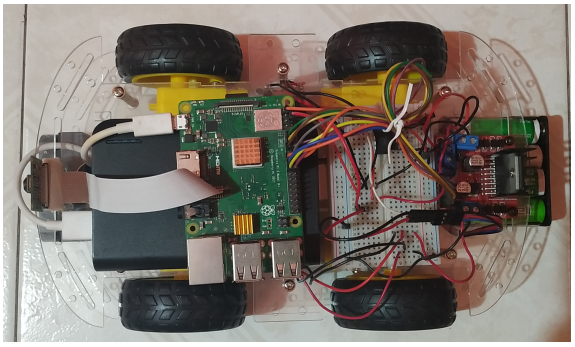
Capítulo 5

Análisis de resultados

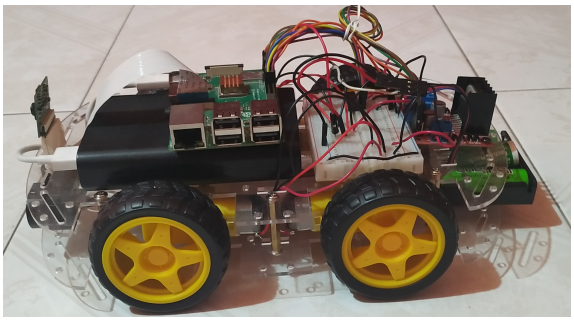
En este capítulo se presentan las pruebas realizadas en cada una de las capas con el fin de observar el desempeño del vehículo autónomo.

5.1. Construcción de los vehículos

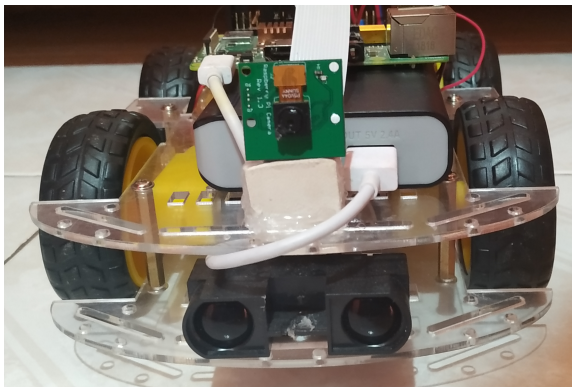
La figuras 5.1 (a), (b) y (c) muestran el vehículo autónomo en diferentes perspectivas. En estas figuras se muestra que el vehículo autónomo monta el sensor SHARP GP2Y0A710kF y la cámara V1.3, los cuales pertenecen a los dispositivos usados para la obtención de datos en las capas 1 y 2, respectivamente. Las datos de capa 3 y 4 son obtenidos a partir de la antena WiFi de la placa Raspberry Pi 3 B+.



(a) Vista aérea.



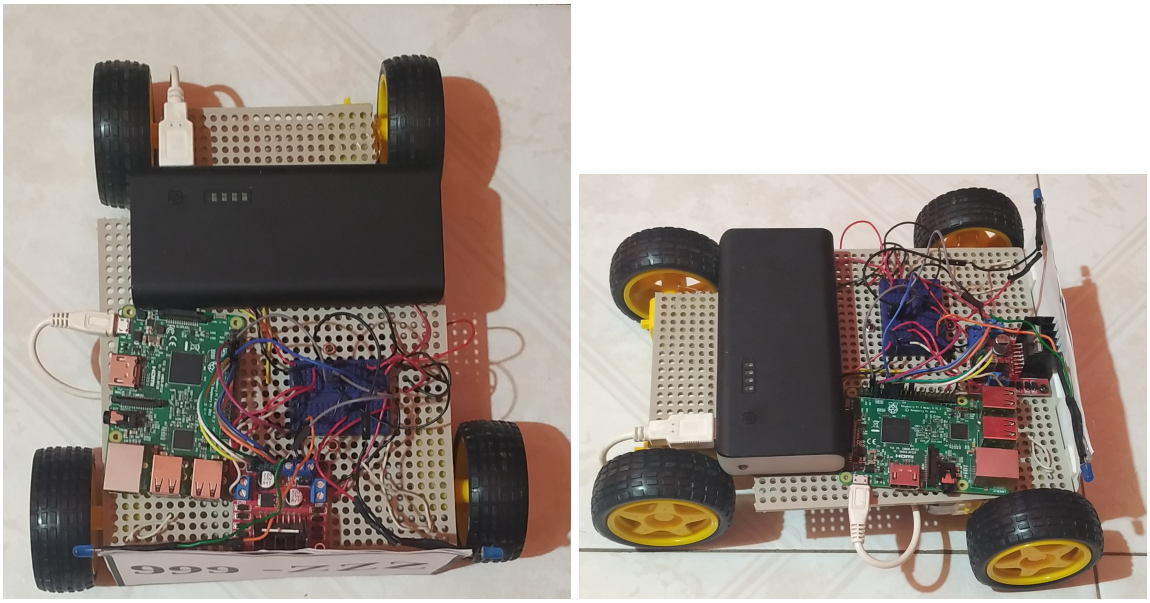
(b) Vista lateral.



(c) Sensor y cámara.

Figura 5.1: Vehículo autónomo.

Las figuras 5.2 (a) y (b) muestran el vehículo a seguir controlado mediante Blynk.



(a) Vista aérea. (b) Vista lateral.

Figura 5.2: Vehículo a seguir.

En la tabla 5.1 se muestran los componentes correspondientes a cada uno de los vehículos construidos.

Vehículo autónomo	Vehículo a seguir
Rasberry Pi 3B+	Rasberry Pi 3B+
Mini protoboard	Mini protoboard
Driver Puente H L298	Driver Puente H L298
Chasis de plástico	Chasis de plástico
Motores DC de 9 V	Motores DC de 9 V
Llantas de plástico	Llantas de plástico
Batería 9 V	Baterías AA
Power Bank de 10000 mAh	Power Bank de 10000 mAh
Cámara V1.3 5MP.	-
Sensor SHARP GP2Y0A710kF	-
–	Prototipo de placa
–	Diodos LED

Tabla 5.1: Tabla de componentes de los vehículos construidos

5.2. Pruebas con el sensor de distancia SHARP GP2Y0A710kF (Capa 1)

Como se mencionó en el capítulo 3 el rango alcanzado para el sensor infrarrojo fue de 60 cm a 300 cm. Sin embargo, ya en la implementación del vehículo autónomo se tuvo una disminución en el rango de medición de 10 cm con lo cual la distancia mínima medida fue de 70 cm. Para las pruebas con el sensor se realizaron 100 pruebas para diferentes distancias con el fin de evaluar el comportamiento del sensor y una vez más rectificar el rango de efectividad del sensor SHARP GP2Y0A710kF. La tabla 5.2 muestra las distancias en las cuales se probó el sensor y los valores obtenidos después de realizar 100 pruebas en cada una de las distancias.

Distancia real [cm]	Distancia medida [cm]	Error estimado
70	76.035	8,62 %
80	85.65	7,06 %
100	104.30	4,3 %
150	145.37	3,08 %
200	197.77	1,11 %
250	257.3	2,92 %
300	291.34	2,88 %

Tabla 5.2: Distancias obtenidas mediante el sensor SHARP GP2Y0A710kF.

Como se puede observar por medio de la tabla anterior, el error obtenido para las mediciones realizadas se encuentra por debajo del 10 %. Para estas pruebas solo se hace uso del hilo correspondiente a la capa 1, en la cual se hace uso del sensor y del algoritmo kdTree.

5.3. Pruebas con Cámara y Open CV (Capa 2)

Como se mencionó en el capítulo 3, la capa 2 corresponde a la obtención de distancias mediante la cámara montada en el vehículo autónomo. El rango efectivo que se experimentó se encuentra alrededor de 1 m. Tomando en cuenta este rango de distancia se realizaron pruebas para verificar la efectividad en el cálculo de distancias usando la cámara. De este modo se realizaron varias pruebas para obtener los valores de 4 distancias diferentes, estos valores y sus distancias estimadas se muestran en la tabla 5.3.

Distancia real [cm]	Distancia calculada [cm]	Error estimado
30	30	0 %
50	53.32	6,64 %
80	84.13	5,16 %
100	106.8	6,8 %

Tabla 5.3: Distancias calculadas mediante cámara.

De igual forma que los resultados obtenidos en la capa 1 el error estimado para los valores medidos en la capa 2 también se encuentran en un rango de 0 % a 7 %. Al igual que en las pruebas anteriores, para obtener los valores de distancia asociados con la capa 2 se habilitó el hilo correspondiente a esta capa dentro del protocolo. Para los resultados de la capa 2 se requiere usar las funciones que permiten la visualización de los frames y las funciones que realizan el cálculo de distancias a partir de los frames obtenidos.

Las figuras 5.3 (a), (b), (c) y (d) muestran diferentes imágenes de la placa captadas a la distancia 30, 50, 80 y 100 cm, respectivamente.

(a) 30 cm. (b) 50 cm.

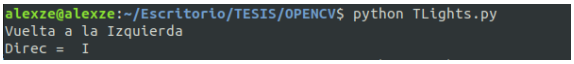
(c) 80 cm. (d) 100 cm.

Figura 5.3: Distancias medidas en capa dos.

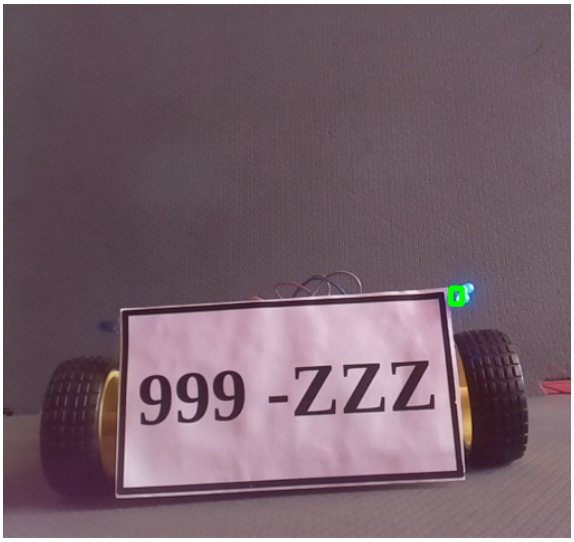
5.3.1. Pruebas para identificación de direccionales

Las pruebas realizadas para identificar si el vehículo a seguir realizará una vuelta se llevaron a cabo a una distancia de 40 cm, es decir, que la identificación se realizó cuando ambos vehículos se encontraban detenidos. Las figuras 5.4 (a) y (c) muestran la identificación de la luz direccional cuando el vehículo autónomo gira a la izquierda y a la derecha, respectivamente. Mientras que las figuras 5.4 (b) y (d) muestran la acción tomada por el vehículo.

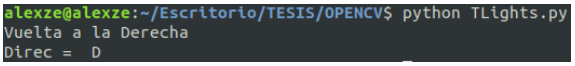
(a) Detección de direccional izquierda.



(b) 50 cm.



(c) Detección de direccional derecha.



(d) 100 cm.

Figura 5.4: Detección de luz direccional.

5.4. Pruebas con la integración de Capa 1 y Capa 2

Con las pruebas de capa 1 y 2, se diseñó una función para el control de velocidad de los motores del vehículo autónomo con base en los rangos de alcance de cada una de las capas como se muestra en la figura 5.5.

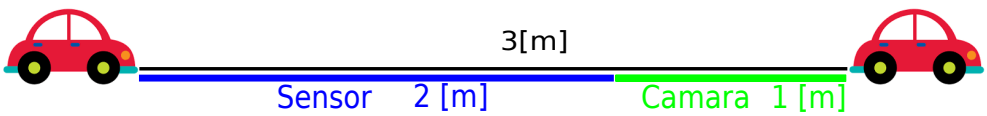


Figura 5.5: Alcance de las capas 1 y 2.

Los algoritmos 3 y 4 muestran las dos funciones con las cuales se obtienen las velocidades

de los motores para que de esta manera el vehículo pueda conducirse de manera autónoma disminuyendo la probabilidad de colisión.

Algorithm 3 Speed

Require: $distCAM, distSEN$
Ensure: $speed$,x

```
1:                                  ▷ Indicación del motor 'r' para avanzar, 's' para frenar
2: if (distCAM == None) then
3:
4:     speed=(15/220)*distSEN - (120 / 22) + 85)
5:
6:     x='r'
7:     return x,speed
8:
9: end if
10: if (distCAM != None) then
11:
12:     x,s = SpeedCAM()
13:     return x,speed
14: end if
```

Algorithm 4 SpeedCAM

Require: $distCAM$
Ensure: $speed$,x

```
1:                                  ▷ Indicación del motor 'r' para avanzar, 's' para frenar
2: if (distCAM <= 40) then
3:
4:     x='s'
5:     speed=0
6:     return x,speed
7: end if
8:
9: if (another case) then
10:
11:     speed=(0.2)*distCAM + 69)
12:
13:     x='r'
14:     return x,speed
15:
16: end if
```

5.4.1. Pruebas a corta distancia

Con la integración de la capa 1 y 2, se colocaron los vehículos con una separación de $40\,\mathrm{cm}$, ya que esta distancia se definió como la mínima distancia a la que deben de estar separados los vehículos. Bajo este escenario se realizaron 20 pruebas, las cuales fueron exitosas ya que el vehículo identificó correctamente la distancia a la cual se encontraba. La tabla 5.4 muestra los resultados obtenidos en una de las pruebas realizadas.

DIST CAM [cm]	DIST SEN [cm]	DIST RSSI [cm]	ANG CAM	ANG SEN	ANG RSSI	DRIVE	SPEED	PROB ENLACE
39	71.47	200	1	0.0	0.0	STOP	0	0.9886
39	71.47	200	1	0.0	0.0	STOP	0	0.9886
39	71.47	200	1	0.0	0.0	STOP	0	0.9886
39	71.47	200	1	0.0	0.0	STOP	0	0.9886
39	71.47	200	1	0.0	0.0	STOP	0	0.9886
39	71.47	200	1	0.0	0.0	STOP	0	0.9886
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982
39	71.47	350	1	0.0	0.0	STOP	0	0.8982

Tabla 5.4: Pruebas a 40 cm.

Como se puede observar en la figura, la distancia obtenida mediante la capa 2 fue de 39 cm, mientras que la capa 1 registró un valor de 71.67 cm, el cual difiere con la distancia real a la cual se encontraba el vehículo (40 cm). Como se mencionó, el sensor de distancia tiene un rango de medición estable de 80 cm a 3 m aproximadamente. Es por esta razón que en distancias menores a 80 cm no se considera la capa 1 para la toma de decisiones en cuanto a si el vehículo debe avanzar o detenerse. También se puede observar en esta figura que la instrucción del auto es detenerse (*STOP*) ya que la capa 2 esta registrando un distancia menor a 40 cm.

5.4.2. Pruebas realizadas a 1 m

Para este experimento se llevaron a cabo 50 pruebas en las cuales se observó el comportamiento del vehículo autónomo. Se colocaron los vehículos con una separación de 1 m, una vez separados se inicio el protocolo de frenado en el vehículo autónomo. Para este caso el vehículo debe desplazarse hasta encontrarse a una distancia de 40 cm, al llegar a este punto el vehículo debe detenerse como se observó en la prueba anterior. El objetivo de estas pruebas es cuantificar cuantas veces se da el caso en que ocurre una colisión, así como medir la separación entre los vehículos en los casos en que el protocolo de frenado funciona de manera correcta.

Las tablas 5.5 y 5.6 muestran algunos de los resultados obtenidos en estas pruebas.

DIST CAM [cm]	DIST SEN [cm]	DIST RSSI [cm]	ANG CAM	ANG SEN	ANG RSSI	DRIVE	SPEED	PROB ENLACE
None	99.77	350	None	0.0	0.0	None	None	0.8982
None	94.87	350	None	0.0	0.0	RUN	87	0.8982
None	94.87	350	None	0.0	0.0	RUN	87	0.8982
None	80.93	350	None	0.0	0.0	RUN	86	0.8982
106	80.93	350	5	0.0	0.0	RUN	91	0.8982
74	80.93	350	6	0.0	0.0	RUN	84	0.8982
51	126.54	350	8	0.0	0.0	RUN	80	0.8982
36	187.92	350	9	0.0	0.0	STOP	0	0.8982
36	202.57	350	9	0.0	0.0	STOP	0	0.8982
36	202.57	350	9	0.0	0.0	STOP	0	0.8982
36	178.08	350	9	0.0	0.0	STOP	0	0.8982
36	197.77	350	9	0.0	0.0	STOP	0	0.8982
36	202.57	350	9	0.0	0.0	STOP	0	0.8982
36	197.77	350	9	0.0	0.0	STOP	0	0.8982
36	197.77	350	9	0.0	0.0	STOP	0	0.8982

Tabla 5.5: Pruebas a 1 m.

DIST CAM [cm]	DIST SEN [cm]	DIST RSSI [cm]	ANG CAM	ANG SEN	ANG RSSI	DRIVE	SPEED	PROB ENLACE
None	116.78	350	None	0.0	0.0	None	None	0.8982
None	109.58	350	None	0.0	0.0	RUN	88	0.8982
None	109.58	350	None	0.0	0.0	RUN	87	0.8982
None	94.87	350	None	0.0	0.0	RUN	86	0.8982
None	71.47	350	None	0.0	0.0	RUN	91	0.8982
107	109.58	350	3	0.0	0.0	RUN	84	0.8982
68	178.08	350	2	0.0	0.0	RUN	80	0.8982
51	187.92	350	6	0.0	0.0	RUN	0	0.8982
40	206.83	350	13	0.0	0.0	STOP	0	0.8982
40	206.83	350	13	0.0	0.0	STOP	0	0.8982
40	202.57	350	13	0.0	0.0	STOP	0	0.8982
40	206.83	350	13	0.0	0.0	STOP	0	0.8982
40	197.77	350	13	0.0	0.0	STOP	0	0.8982
40	197.77	350	13	0.0	0.0	STOP	0	0.8982
40	206.83	350	13	0.0	0.0	STOP	0	0.8982
40	206.83	350	13	0.0	0.0	STOP	0	0.8982
40	197.77	350	13	0.0	0.0	STOP	0	0.8982
40	206.83	350	13	0.0	0.0	STOP	0	0.8982
40	197.77	350	13	0.0	0.0	STOP	0	0.8982
40	197.77	350	13	0.0	0.0	STOP	0	0.8982
40	215.59	350	13	0.0	0.0	STOP	0	0.8982
40	206.83	350	13	0.0	0.0	STOP	0	0.8982
40	202.57	350	13	0.0	0.0	STOP	0	0.8982

Tabla 5.6: Pruebas a $1\,\mathrm{m}$.

En los casos mostrados en las tablas 5.5 y 5.6 la separación al inicio del experimento entre los vehículos fue de $1\,\mathrm{m}$. Se puede observar en estas figuras que los valores de distancia medidos durante el recorrido del vehículo, así como el seguimiento de la velocidad del mismo cambia a través del recorrido. También se puede observar que cuando el vehículo autónomo se encuentra a $40\,\mathrm{cm}$ del vehículo a seguir se detiene mediante la instrucción STOP. Es importante mencionar que las primeras 5 muestras la capa 1 toma el control del vehículo, ya que la distancia de la cámara aparece como *None*, lo cual significa que la cámara no detecta el contorno de la placa a esa distancia. Posteriormente la capa 2 toma control del vehículo hasta que lo detiene.

Del número total de pruebas realizadas, el $86\,\%$ de estas fueron exitosas. Lo que quiere decir que hubo pruebas en las que el protocolo falló y los dos vehículos colisionaron. El número de pruebas en las cuales existieron colisiones fue de 7, lo cual corresponde a un $14\,\%$ respecto al número total de pruebas realizadas.

En la siguiente tabla se muestran algunas de las distancias de separación entre los dos vehículos, después de frenar. Estas fueron recabadas de las 50 pruebas realizadas y en las cuales el protocolo funcionó de manera exitosa.

Prueba	Distancia de separación [cm]
1	18
2	40
3	13
4	25
5	25
6	26
7	25
8	20
9	40
10	26
11	26
12	33
13	28
14	20
15	16
16	24
17	16
18	18
19	20
20	10

Tabla 5.7: Pruebas de separación en frenado.

Como se puede observar en la tabla 5.7, aunque la distancia establecida en el protocolo para realizar la acción de frenado fue de $40\,\mathrm{cm}$, mediante las pruebas se obtuvieron distancias de separación que oscilan entre los 10 y $40\,\mathrm{cm}$. En promedio la distancia a la cual el vehículo autónomo se detiene por completo de acuerdo con los datos de la tabla anterior es de $23.5\,\mathrm{cm}$. La razón por la cual no en todas las pruebas realizadas el vehículo se detiene al llegar a los $40\,\mathrm{cm}$ es el tiempo de procesamiento requerido por la capa 2 para la obtención de los valores de distancia, lo cual hace más lento la toma de decisiones del vehículo en cuanto a las acciones de frenado, aunado a esto la inercia que tiene el vehículo en movimiento antes de detenerse por completo, hace que la distancia de separación final entre los dos vehículos sea menor a los $40\,\mathrm{cm}$ establecidos.

5.4.3. Pruebas con el vehículo a seguir en movimiento

El siguiente escenario analizado fue con el vehículo de referencia en movimiento constante con el fin de observar la regulación de velocidad del vehículo autónomo. Para este experimento el vehículo autónomo se colocó detrás de vehículo a seguir con una distancia de separación de $40\,\mathrm{cm}$, de esta forma como primera acción el vehículo autónomo permanece detenido. Por medio del uso de la aplicación de Blynk se controla el curso del vehículo de seguimiento haciéndolo avanzar en linea recta y deteniéndolo por momentos con el fin de que el vehículo autónomo siga la misma ruta y en el momento en el que el vehículo de enfrente se detenga, este se detenga a una distancia máxima de $40\,\mathrm{cm}$ y de esta forma prevenir una posible colisión. De igual forma para este experimento se llevaron a cabo 25 pruebas para observar el comportamiento del vehículo autónomo.

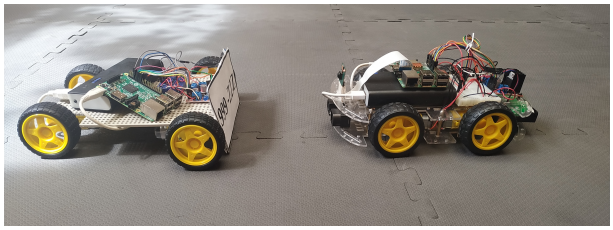
La tabla 5.8 presenta los resultados obtenidos en una de las pruebas respecto a los valores de distancia y velocidad obtenidos por el vehículo autónomo. En la primera fila de la tabla se puede observar los valores obtenidos por el vehículo autónomo al iniciar el protocolo. En la columna **DRIVE**, se puede verificar que en un primer momento el vehículo se encuentra detenido. Una vez que la distancia de separación entre los dos vehículos sobrepasa los $40\,\mathrm{cm}$, por medio de la siguiente fila resaltada en color amarillo se puede observar que el vehículo comienza a moverse. De esta forma, a través de la tabla, se puede observar el comportamiento del vehículo autónomo, donde las filas resaltadas en color amarillo muestran los momentos en que el vehículo cambia de estado, ya sea en movimiento o detenido.

DIST CAM [cm]	DIST SEN [cm]	DIST RSSI [cm]	ANG CAM	ANG SEN	ANG RSSI	DRIVE	SPEED	PROB ENLACE
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
29	126.54	350	17	0.0	0.0	STOP	0	0.8982
39	76.03	350	6	0.0	0.0	STOP	0	0.8982
39	76.03	350	6	0.0	0.0	STOP	0	0.8982
47	76.03	350	1	0.0	0.0	RUN	79	0.8982
47	76.03	350	1	0.0	0.0	RUN	79	0.8982
47	71.47	350	1	0.0	0.0	RUN	79	0.8982
47	71.47	350	1	0.0	0.0	RUN	79	0.8982
47	76.03	350	1	0.0	0.0	RUN	79	0.8982
47	76.03	350	1	0.0	0.0	RUN	79	0.8982
44	71.47	350	2	0.0	0.0	RUN	78	0.8982
44	71.47	350	3	0.0	0.0	RUN	78	0.8982
57	71.47	350	1	0.0	0.0	RUN	81	0.8982
38	85.66	350	10	0.0	0.0	STOP	0	0.8982
37	85.66	350	13	0.0	0.0	STOP	0	0.8982
37	71.47	350	13	0.0	0.0	STOP	0	0.8982
42	71.47	350	10	0.0	0.0	RUN	78	0.8982
42	71.47	350	10	0.0	0.0	RUN	78	0.8982
42	71.47	350	10	0.0	0.0	RUN	78	0.8982
42	71.47	350	10	0.0	0.0	RUN	78	0.8982
42	71.47	350	10	0.0	0.0	RUN	78	0.8982
42	71.47	350	10	0.0	0.0	RUN	78	0.8982
42	109.58	350	10	0.0	0.0	RUN	78	0.8982
36	145.38	350	15	0.0	0.0	STOP	0	0.8982
36	116.78	350	15	0.0	0.0	STOP	0	0.8982
38	116.78	350	15	0.0	0.0	STOP	0	0.8982
34	109.58	350	22	0.0	0.0	STOP	0	0.8982
34	116.78	350	22	0.0	0.0	STOP	0	0.8982
34	104.31	350	22	0.0	0.0	STOP	0	0.8982
34	94.87	350	22	0.0	0.0	STOP	0	0.8982
34	94.87	350	21	0.0	0.0	STOP	0	0.8982
35	85.66	350	16	0.0	0.0	STOP	0	0.8982
35	85.66	350	11	0.0	0.0	STOP	0	0.8982
36	85.66	350	10	0.0	0.0	STOP	0	0.8982
36	71.47	350	10	0.0	0.0	STOP	0	0.8982
47	71.47	350	3	0.0	0.0	RUN	79	0.8982
53	71.47	350	1	0.0	0.0	RUN	80	0.8982
52	71.47	350	1	0.0	0.0	RUN	80	0.8982
53	76.03	350	0	0.0	0.0	RUN	80	0.8982
69	94.87	350	1	0.0	0.0	RUN	83	0.8982
69	116.78	350	1	0.0	0.0	RUN	83	0.8982
90	141.26	350	1	0.0	0.0	RUN	87	0.8982
78	145.38	350	0	0.0	0.0	RUN	85	0.8982
78	145.38	350	2	0.0	0.0	RUN	85	0.8982
58	145.38	350	8	0.0	0.0	RUN	81	0.8982
53	145.38	350	10	0.0	0.0	RUN	80	0.8982
53	135.18	350	10	0.0	0.0	RUN	80	0.8982
53	145.38	350	10	0.0	0.0	RUN	80	0.8982
53	145.38	350	10	0.0	0.0	RUN	80	0.8982
17	156.63	350	32	0.0	0.0	STOP	0	0.8982
16	149.49	350	40	0.0	0.0	STOP	0	0.8982
16	156.63	350	40	0.0	0.0	STOP	0	0.8982
16	145.38	350	40	0.0	0.0	STOP	0	0.8982

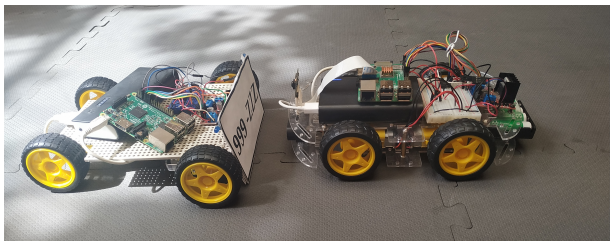
Tabla 5.8: Prueba continua.

Como se puede observar en la tabla anterior, el vehículo se detiene y avanza correctamente a lo largo del tiempo. Sin embargo, debido al procesamiento que se realiza en la capa 2, existe

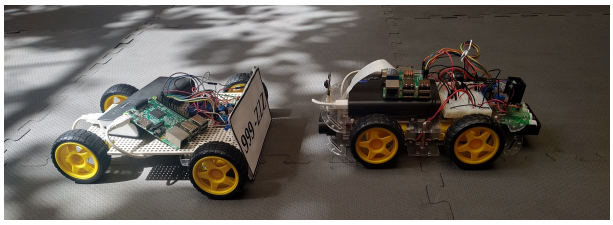
un retraso de 1 a 2 segundos en la toma de decisiones, esto se ve reflejado en la distancia a la cual se detiene el vehículo autónomo. Las figuras 5.6 (a), (b) y (c) muestran la separación entre los vehículos una vez que se aplicó la acción de frenado en distintos momentos.



(a) Distancia de separación 1.



(b) Distancia de separación 2.



(c) Distancia de separación 3.

Figura 5.6: Prueba continua.

Como se puede apreciar en las figuras anteriores en algunos de las casos la distancia de separación es menor que lo esperado (40 cm) debido al retraso mencionado.

Del total de pruebas realizadas en este experimento, el 80 % de ellas fueron exitosas, mientras que en el 20 % restante hubo por lo menos una colisión entre los 2 vehículos.

5.5. Pruebas con RSSI (Capa 3 y 4)

La capa 3 y 4 como se mencionó en el capítulo 3 hacen uso de RSSI para estimar la distancia y probabilidad de enlace. En ambas capas se realizaron 100 pruebas cada 50 cm y 45^o de la antena receptora con el fin de observar el comportamiento tanto de los valores de probabilidad como de los valores de distancia asociados a los valores de RSSI (ver valores en el capítulo 3). Sin embargo, se observó que la capa 3 no es viable para la determinación de las distancias de separación entre los dos vehículos debido a la gran variación que existe entre los valores recibidos de RSSI derivados de los problemas que se presentan en el medio de propagación, los efectos de multi trayectoria y fluctuaciones lentas, mencionados en el capítulo 3. En la figura 5.7 se muestra el patrón de radiación de la antena WiFi obtenido a partir de los datos de la tabla 3.2.

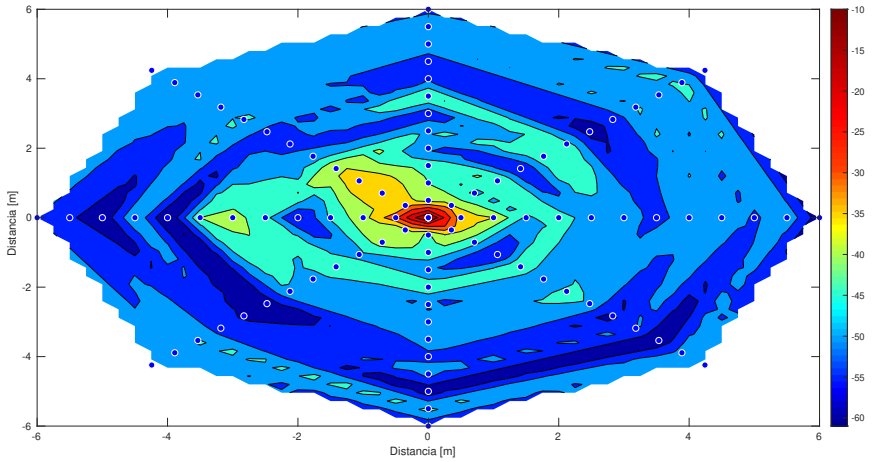


Figura 5.7: Patrón de radiación de la antena Wi-Fi.

Como se puede observar a través del patrón de radiación existen áreas en donde los valores de potencia son muy parecidos o incluso iguales por lo cual es muy difícil determinar la distancia exacta a la cual se encuentra la antena receptora.

En la tabla 5.9 se muestran los valores de distancia promedio aparentes obtenidas mediante las 100 pruebas para cada uno de los puntos establecidos.

Distancia real (cm)	P_r(obtenida) (dBm)	Probabilidad	Distancia aparente (cm)	Error estimado
50	-50.19	0.964192	240.5	381 %
100	-50.97	0.987696	201.5	101,5 %
150	-48	0.898200	350	133 %
200	-49.53	0.944304	273.5	36,75 %

Tabla 5.9: Valores obtenidos por medio del uso del RSSI.

Por lo anterior, se decidió usar el parámetro de RSSI solo para establecer la probabilidad de que el enlace de comunicación se mantenga estable entre los dos vehículos, lo cual corresponde solo a la capa 4.

Capítulo 6

Conclusiones

6.1. Conclusiones generales

Actualmente con el auge de las nuevas tecnologías en IoT y la búsqueda de soluciones a diferentes problemas de la vida cotidiana como es el transito vehicular y peatonal, se han desarrollado diferentes sistemas apoyados en estas tecnologías para mejorar las condiciones de vida de los usuarios que estan involucrados en el sector vehicular. Dentro de las tecnologías ocupadas actualmente para este propósito se encuentran el uso de sensores y de sistemas de visión artificial. En esta tesis se hace uso del sistema de visión artificial OpenCV en conjunto con un sensor de distancia para la implementación de un protocolo que permita reducir la probabilidad de colisión entre dos vehículos a escala a partir de los datos obtenidos a cada momento. Esta tesis propone un modelo de cuatro capas que permite estimar la distancia y ángulo entre dos vehículos con el fin de que uno de ellos siga los movimientos del otro. Sin embargo, como se mencionó al principio de esta tesis una de las capas que proporcionaría información al protocolo para la toma de decisiones sería el parámetro Received Signal Strength Indicator (RSSI), el cual sería utilizado para predecir la distancia de separación entre los dos vehículos. Sin embargo, a lo largo de las pruebas realizadas se observó que este no era un indicador fiable para la toma de decisiones debido a la inconsistencia de los valores recibidos, esta inconsistencia se puede observar en el mapa de la figura 5.7. Por lo que se decidió eliminar esa capa. Una opción para aprovechar mejor este parámetro y tratar de evitar los problemas mencionados en el capítulo 5 sería el uso de antenas direccionales con el fin de redireccionar toda la potencia de transmisión de la antena y de igual forma reducir el radio de recepción de la antena y con esto obtener lecturas más fiables sin el riesgo de sufrir tantas discrepancias. Por lo anterior el parámetro de RSSI solo se uso para saber qué tan confiable era el enlace de comunicación establecido entre los dos vehículos por medio de valores de probabilidad mostrados en el capítulo 3.

Por otro lado, las pruebas mostradas en el capítulo 5 referentes a la capa 1 y 2 muestran resultados positivos en cuanto al calculo de distancias y la toma de acciones por parte del vehículo autónomo. Por ejemplo, en las pruebas realizadas a corta distancia se tuvo un $100\,\%$ de efectividad ya que en las 20 pruebas realizadas el vehículo siempre detecto que se encontraba a una distancia de $40\,\mathrm{cm}$ o menos, por lo cual siempre permanecía detenido. En las pruebas realizadas a $1\,\mathrm{m}$ se tuvo una respuesta exitosa del $86\,\%$, mientras que el $14\,\%$ restante hubo colisiones entre los dos vehículos. Por otro lado, en las pruebas realizadas de manera continua, en las cuales el vehículo a seguir se encontraba en movimiento constante y durante el trayecto este se detenía alrededor de 3 veces con el fin de observar el comportamiento del vehículo autónomo, se tuvo una eficiencia del $80\,\%$, ya que en el $20\,\%$ de las pruebas existió alguna colisión.

Como se mencionó anteriormente el cálculo de distancias en la capa 2 corresponde al uso de la cámara y la biblioteca OpenCV. El problema con esta biblioteca es que conllevan cierto tiempo de procesamiento lo cual llega a ser un factor importante a la hora de tomar decisiones en situaciones repentinas. Este factor de tiempo puede disminuirse a partir de el uso de dispositivos que puedan ofrecer un mejor rendimiento al momento de realizar tareas

como lo es el procesamiento de imágenes, o utilizando lenguajes de más bajo nivel como lo es C o C++.

6.2. Verificación de la hipótesis

Retomando la hipótesis que se presentó en la sección 1.2:

Un modelo de cuatro capas que estiman la distancia y el ángulo entre dos vehículos reduce la probabilidad de colisión.

Para verificar la hipótesis planteada, en esta tesis se llevaron a cabo dos procesos. El primer proceso consistió en el desarrollo de los cuatro modelos que conformarían las 4 capas usadas para la implementación del protocolo de frenado. Estas cuatro capas consisten en, el sensor de distancia infrarrojo, la cámara en conjunto con la biblioteca de OpenCV, la lectura de valores RSSI para la medición de distancias en la capa 3 y en la capa 4 para el cálculo de probabilidades sobre la estabilidad del enlace de comunicación.

Una vez implementado el protocolo, el segundo proceso consistió en las pruebas experimentales, las cuales demostraron la eficacia del protocolo como se mostró en el capítulo 5, en donde se muestra el comportamiento del vehículo en diferentes situaciones y cómo es que éste toma la decisión de frenar cuando es requerido. Tomando en cuenta los porcentajes obtenidos en los diferentes grupos de pruebas referentes a la probabilidad de evitar colisiones o, a la eficiencia del protocolo, realizadas con la interacción de las capas 1 y 2, se puede señalar que existe un $88{,}6\,\%$ de probabilidad de que el protocolo logre evitar una colisión entre los dos vehículos.

A pesar de que la capa 3 se contemplo en un principio para que contribuyera a la determinación de valores de distancia, esta se tuvo que eliminar debido al error tan grande que genera la medición de RSSI. Por lo que se puede concluir que a pesar de la eliminación de una capa del modelo originalmente planteado, este es capaz de evitar colisiones y seguir correctamente al vehículo frente a él.

Apéndice A

Clase Principal

En esta clase se concentran todas las funciones y programas mostrados en los siguientes apéndices para el funcionamiento del protocolo.

```
# PROTOCOLO

# CARGANDO MODELOS

import threading
from math import ceil
from kdTreeMod import *
from kdTreeRSSI import *
from rssiScan2 import *
from camaraPi import *
from dist import *
from motorfun import *
import time
import RPi.GPIO as GPIO
from time import sleep


class Sensado():

    def __init__(self):

        ##INICIACIÓN DE VARIABLES

        # RSSI
        self.angRSSI = None
        self.distRSSI = None
        self.probaRSSI = None
        self.levelRSSI = None

        # CAMERA
        self.distCAM = None
        self.angCAM = None
        self.dataCAM = None

        # SENSOR
        self.distSEN = None
        self.angSEN = None
        self.drive = None
        self.speed = None

        self.placa = None

    def config(self):

        ##CREACION E INICIALIZACION DE HILOS

        t1 = threading.Thread(name="Picam", target=self.CamaraT)
        t2 = threading.Thread(name="Measure", target=self.MeasureCam)
        t3 = threading.Thread(name="RSSI", target=self.RSSI)
```

```
        t4 = threading.Thread(name="Sensor", target=self.Sensor)
        t5 = threading.Thread(name="Resultados", target=self.Results)
        t6 = threading.Thread(name="Motor", target=self.Motor)

        t1.start()
        time.sleep(8)
        t2.start()
        t3.start()
        t4.start()
        time.sleep(7)
        t6.start()
        t5.start()


    def CamaraT(self): ##INICIALIZACIÓN DE LA CAMARA PARA LA CAPA2

        print ("INICIAR CAMARA")
        Camara()


    def MeasureCam(self):  ##MEDICION DE DISTANCIA POR MEDIO DE CAPA 2
        print ("INICIANDO CAMARA")
        coord = []
        while True:

            for x in range (3):
                self.dataCAM=Data()
                if self.dataCAM !=None:
                    coord.append(self.dataCAM)
                    time.sleep(.4)

            distance = []
            angle = []
            for x in range(len(coord)):
                distance.append(coord[x][0])
                angle.append(coord[x][1])

            if(len(distance)!=0):

                self.distCAM = int(self.Average(distance))
                self.angCAM = int(self.Average(angle))
                if self.distCAM <=15:
                    self.distCAM = None

            del coord[:]


    def RSSI(self): ##MEDICIÓN DE RSSI Y CALCULO DE DISTANCIA PARA CAPA 3 Y CAPA 4

        print ("INICIANDO RSSI")
        listaRSSI = ReadRSSI()
        nodeRSSI = buildKdtreeRSSI(listaRSSI)
        while True:

            val = RssiLevel()
            coorde = []
            searchNeighborsRSSI((val, 0), nodeRSSI, 15, coorde)

            rssi = []
            distance = []
            angle = []
            proba = []
            for x in range(len(coorde)):
                rssi.append(coorde[x][0][0])
                angle.append(coorde[x][0][1])
                distance.append(coorde[x][1])
                proba.append(coorde[x][2])

            if(len(distance)!=0):
              self.distRSSI = self.Average(distance)
              self.angRSSI = self.Average(angle)
```

```
121             self.probaRSSI = self.Average(proba)
                self.levelRSSI = self.Average(rssi)
123

125     def Sensor(self): ##MEDICION DE DISTANCIAS CON SENSOR PARA CAPA 1
            print ("INICIANDO SENSOR")
127         listaSensor = Read()
            nodeSensor = buildKdtree(listaSensor)
129         while True:
                DIST = None
131             val = DistSensor(DIST)
                coorde = []
133             searchNeighbors((val, 0), nodeSensor, 10, coorde)
                distance = []
135             angle = []
                for x in range(len(coorde)):
137                 distance.append(coorde[x][0])
                    angle.append(coorde[x][1])
139
                if(len(distance)!=0):
141                 self.distSEN = self.Average(distance)
                    self.angSEN = self.Average(angle)
143


145
    def Average(self, list): ##CALCUL0 DE PROMEDIOS
147     a = sum(list)
        b = float(len(list))
149     avg = a / b
        return avg
151

153 def Results(self): ##IMPRESION Y VISUALIZACIÓN DE RESULTADOS

155     print ("INICIANDO RESULTADOS")

157     print ("################################################")
        print("{9:^5}{0:^10}{9:^5}{1:^10}{9:^5}{2:^10}{9:^5}{3:^10}{9:^5}{4:^10}
159         {9:^5}{5:^10}{9:^5}{6:^10}{9:^5}{7:^10}{9:^5}{8:^10}{9:^5}
            ".format("DIST CAM","DIST SEN","DIST RSSI","ANG CAM","ANG SEN","ANG RSSI","DRIVE",
"SPEED","PROB ENLACE","||"))
161     while True:

163         if(self.distSEN==None):

165

                print ("{9:^5}{0:^10}{9:^5}{1:^10}{9:^5}{2:^10}{9:^5}{3:^10}{9:^5}{4:^10}
167             {9:^5}{5:^10}{9:^5}{6:^10}{9:^5}{7:^10}{9:^5}{8:^10}{9:^5}
                ".format(str(self.distCAM),str(self.distSEN),str(self.distRSSI),str(self.
angCAM),str(self.angSEN),str(self.angRSSI),str(self.drive),str(self.speed),str(self.
probaRSSI),"||"))
169
            else:
171             print ("{9:^5}{0:^10}{9:^5}{1:^10.2f}{9:^5}{2:^10}{9:^5}{3:^10}{9:^5}{4:^10}
                {9:^5}{5:^10}{9:^5}{6:^10}{9:^5}{7:^10}{9:^5}{8:^10}{9:^5}
173             ".format(str(self.distCAM),self.distSEN,str(self.distRSSI),str(self.angCAM),
str(self.angSEN),str(self.angRSSI),str(self.drive),str(self.speed),str(self.probaRSSI),"||
"))

175         time.sleep(1)

177


179 def Speed(self): ##CALCULO DE VELOCIDAD CON CAPA 1
        if(self.distCAM == None):
181
                s = ceil((15 / 220.) * self.distSEN - (120 / 22.) + 85)
183             x = 'r'
                if(s > 100):
185                 s = 100
                return x, s
```

```
187         elif(self.distCAM != None):
189
                x, s = self.SpeedCam()
191             return x, s
193
        def SpeedCam(self):  # CALCULO DE VELOCIDAD CON CAPA 2
195
            if (self.distCAM <= 40):
197
                x = 's'
199             s = 0
                return x, s
201         else:
203             s = ceil(0.2 * self.distCAM + 69)
                x = 'r'
205             return x, s
207
        def Motor(self):  ## CONTROL DE VELOCIDAD DE LOS MOTORES
209         print ("INICIANDO MOTOR")
            if(self.distCAM ==None or self.distCAM > 50):
211
                RunForward(100)
213             print ("ACELERANDO")
                time.sleep(1)
215         while True:
217             if(self.distSEN!=None):
                    x, speed = self.Speed()
219                 if (x == 'r'):
                        self.speed = speed
221                     self.drive = "RUN"
                        RunForward(speed)
223                 if (x == 's'):
                        self.speed = 0
225                     self.drive = "STOP"
                        Stop()
227                     if(self.placa==None):
                            self.placa =Placa()
229
                        self.distCAM==None
231                     time.sleep(3)
sensado = Sensado()
233 sensado.config()
```

Apéndice B

Lectura de frames para el cálculo de distancias en capa 2

```
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2
from funcam import *

data = None

def Camara():
  global data
  # initialize the camera and grab a reference to the raw camera capture
  camera = PiCamera()
  camera.resolution = (640, 480)
  camera.framerate =5
  rawCapture = PiRGBArray(camera, size=(640, 480))
  # allow the camera to warmup
  time.sleep(.01)

  focalLength,C_cal,dBase_cal = Calibration()
  # capture frames from the camera
  for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
    # grab the raw NumPy array representing the image, then initialize the timestamp
    # and occupied/unoccupied text
    fr = frame.array
    data = Measure(fr,focalLength,C_cal,dBase_cal)

    # show the frame
    #cv2.imshow("Frame", fr)
    key = cv2.waitKey(1) & 0xFF
    # clear the stream in preparation for the next frame
    rawCapture.truncate(0)
    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
      break

def Data():
  return data
```

Apéndice C

Funciones creadas en Python con la biblioteca OpenCV para la medición de distancias

```
import imutils
import cv2
import numpy as np
import pytesseract
import time
import math
from PIL import Image
from scipy.spatial import distance as dist

KNOWN_DISTANCE = 30.0 ##DISTANCIA DE SEPARACIÓN DE LA IMAGEN MUESTRA EN [cm]
KNOWN_WIDTH = 15.0 ###ANCHO DE LA PLACA EN [cm]
KNOWN_HIGH = 8.0 ##ALTO DE LA PLACA EN [cm]
placa = None

def find_marker(frame):
  global placa
  gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #convert to grey scale
  gray = cv2.bilateralFilter(gray, 11, 17, 17) #Blur to reduce noise
  edged = cv2.Canny(gray, 100, 200) #Perform Edge detection
  edged = cv2.dilate(edged,None, iterations=1)


  # find contours in the edged image, keep only the largest
  # ones, and initialize our screen contour

  cnts,_ = cv2.findContours(edged.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
  #cv2.drawContours(frame, cnts, -1, (0,255,0), 2)

  for c in cnts:
    area = cv2.contourArea(c)
    epsilon = 0.09*cv2.arcLength(c,True)
    approx = cv2.approxPolyDP(c,epsilon,True)

    if len(approx)==4 and area > 3000:

      x,y,w,h = cv2.boundingRect(c)
      aspect_r = aspect_ratio(w,h)

      if aspect_r > 1.5:



        placa = gray[y:y+h,x:x+w ]

        return cv2.minAreaRect(c)
```

```
47 def aspect_ratio(w,h):
     return(float(w)/h)
49
   def distance_to_camera(knownWidth, focalLength, perWidth): ##CALCULO DE DISTANCIA
51
     return (knownWidth * focalLength) / perWidth
53
   def midpoint(ptA, ptB): ##CALCULO DEL PUNTO MEDIO
55   return ((ptA[0] + ptB[0]) * 0.5, (ptA[1] + ptB[1]) * 0.5)

57
   def Placa():##LECTURA DE PLACA DEL VEHÍCULO
59     global placa
       text = pytesseract.image_to_string(placa,config ='--psm 6')
61     ("text = ",text)
       return text
63

65 def Calibration(): ##CALIBRACIÓN DE LA IMAGEN MUESTRA

67   img = cv2.imread('p301.png',cv2.IMREAD_COLOR)
     img = cv2.resize(img, (620, 480))
69   marker_Cal = find_marker(img)

71   focalLength = (marker_Cal[1][0] * KNOWN_DISTANCE) / KNOWN_WIDTH
     box = cv2.cv.BoxPoints(marker_Cal) if imutils.is_cv2() else cv2.boxPoints(marker_Cal)
73   box = np.int0(box)

75
     c,dBase=Center(img,box)
77

79   return focalLength,c,dBase

81
   def Center(frame,box):##OBTENCION DEL CENTRO DE LA PLACA
83   pixelsPerMetric = None
     # unpack the ordered bounding box, then compute the midpoint
85   # between the top-left and top-right coordinates, followed by
     # the midpoint between bottom-left and bottom-right coordinates
87   (tl, tr, br, bl) = box
     (tltrX, tltrY) = midpoint(tl, tr)
89   (blbrX, blbrY) = midpoint(bl, br)

91   # compute the midpoint between the top-left and top-right points,
     # followed by the midpoint between the top-righ and bottom-right
93   (tlblX, tlblY) = midpoint(tl, bl)
     (trbrX, trbrY) = midpoint(tr, br)
95
     c=(tltrX,trbrY)
97
     dBase = dist.euclidean((tlblX, tlblY), (trbrX, trbrY))
99   return (tltrX,trbrY), dBase

101
   def Measure(frame,focalLength,C_cal,dBase_cal):##OBTENCION DEL FRAME DELA PLACA Y CALCULO DE
       DISTANCIA
103
     frame = cv2.resize(frame, (620,480) )
105  marker=find_marker(frame)

107  if marker != None:

109    distance= distance_to_camera(KNOWN_WIDTH, focalLength, marker[1][0])

111
       box = cv2.cv.BoxPoints(marker) if imutils.is_cv2() else cv2.boxPoints(marker)
113    box = np.int0(box)

115    C,dBase=Center(frame,box)
```

```
117
119     pixelsPerMetric = None
        if pixelsPerMetric is None:
121         pixelsPerMetric = dBase_cal / 15

123
        # compute the Euclidean distance between the midpoints
125     dC = dist.euclidean(C_cal, C)
        dista_x = dC / pixelsPerMetric
127
        data =[]
129     data.append(distance)

131
        ang=math.atan(dista_x/distance)
133     ang=math.degrees(ang)
        data.append(ang)
135
        return data
137

139 def Turn_Lights(frame):##DETECCION DE LUCES DIRECCIONALES
      global direc
141
      kernel = np.ones((5, 5), np.uint8)
143   while True:

145
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
147
        ##FILTROS PARA COLOR AZUL
149     umbral_bajo = np.array([100, 200, 100], dtype=np.uint8)
        umbral_alto = np.array([130, 250, 255], dtype=np.uint8)
151     # CREACION DE LA MASCARA Y FILTRADO DE FRAMES
        mask = cv2.inRange(hsv, umbral_bajo, umbral_alto)
153     mask = cv2.erode(mask, None, iterations=2)
        mask = cv2.dilate(mask, None, iterations=2)
155     #cv2.imshow('MASK', mask)
        res = cv2.bitwise_and(frame, frame, mask=mask)
157
        opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
159     closing = cv2.morphologyEx(opening, cv2.MORPH_OPEN, kernel)

161     x, y, w, h = cv2.boundingRect(closing)
        #print x
163     if(x != 0):
          if(x < 320):
165         direc = "I"

167         print("Vuelta a la Izquierda")

169         print "Direc = ", direc

171         return
          else:
173         print("Vuelta a la Derecha")
            direc = "D"
175         print "Direc = ", direc
            return
177
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 3)
179     cv2.circle(frame, (x+w/2, y+h/2), 5, (0, 0, 255), -1)
```

Apéndice D

Código en Python para el cálculo de distancias con el sensor SHARP GP2Y0A710kF

```
import time
import math
# Import SPI library (for hardware SPI) and MCP3008 library.
import Adafruit_GPIO.SPI as SPI
import Adafruit_MCP3008


def DistSensor(DIST):

  const = 10
  SPI_PORT   = 0
  SPI_DEVICE = 0
  mcp = Adafruit_MCP3008.MCP3008(spi=SPI.SpiDev(SPI_PORT, SPI_DEVICE))
  val = []
  val2 = []
  while True:
    v = mcp.read_adc(0)

    time.sleep(.1)
    ###Función para el calculo de distancia con luz
    dist = 1.6533e-8*v**4-(5.06034e-5*v**3)+(.0579*v**2)-(29.515*v)+5802.8-30
    ###Función para el calculo de distancia con poca luz
    dist2 = 6.42936e-9*v**4-2.18185e-5*v**3+0.027*v**2-14.7212*v+3105.55

    val2.append(dist2)
    if dist <132:
      dist=dist-2

      D1 = dist
    elif dist < 215:
      dist=dist
      D1 = dist

    elif dist > 215 or dist<244.5:
      dist=dist
      D1 = dist


    val.append(D1)


    if len(val)==10:
      avg2 = sum(val2)/float(len(val2))
```

```
46      avg = sum(val)/float(len(val))

48      D3 = (avg+avg2)/2
        val =[]
50      val2 =[]
        return D3
52
```

Apéndice E

Código en Python con la implementación del algoritmo kdTree para los valores del sensor SHARP GP2Y0A710kF

```
#! /usr/bin/env python

################################################################################
#
#       This class implements kd-tree. Given a point and range, this class looks for
#        a set of nearest neighbor.
#
#
################################################################################
from Sensordat import *
#from read_csv import *
class Node(object):

  def __init__(self):
    self.leftChild = None
    self.rightChild = None
    self.location = None
    self.axis = None

def buildKdtree(pointList, depth=0):
  if not pointList:
    return

  #Select the axis based on depht.
  k = len(pointList[0])
  axis = depth %k

  #Sort points depended on axis and choose the midpoint
  pointList.sort(key=lambda point:point[axis])
  median = len(pointList) // 2
  node = Node()
  node.location = pointList[median]
  node.axis = axis
  node.leftChild = buildKdtree(pointList[0:median], depth+1)
  node.rightChild = buildKdtree(pointList[median+1:], depth+1)
  return node

def searchNeighbors(q, node, radius,coord):
  if (node == None):
    return

  if (isInside(q, node.location, radius) == 1):
```

```
    coord.append(node.location)
44
  if (node.location[node.axis] < q[node.axis]+radius):
46    searchNeighbors(q, node.rightChild, radius, coord)

48  if (node.location[node.axis] > q[node.axis]-radius):
    searchNeighbors(q, node.leftChild, radius,coord)
50

52 def isInside(pointA, pointB, dist):

54  if (((pointB[0] - pointA[0])**2 + (pointB[1] - pointA[1])**2) <= dist**2):
    return 1
56  else:
    return 0
```

Apéndice F

Código en Python para la lectura de valores de RSSI

```
#!/usr/bin/env python
import subprocess
import time
import argparse


def RssiLevel():

    arraylevel = []
    while True:
        ##COMANDO EN TERMINAL PARA LA LECTURA DE VALORES RSSI
        cmd = subprocess.Popen(
            '/sbin/iwlist wlan0 scanning | grep "level\|ESSID:INFINITUM439A\|Address:b8:27:eb
    :80:41:55"', shell=True,
                               stdout=subprocess.PIPE)

        for line in cmd.stdout:


            field = str(line),

            field = field[0].strip().split(" ")


            level = field[23]
            level = level.strip().split("=")

            rssi = level[1]

            arraylevel.append(int(rssi))
            if len(arraylevel) == 10:
                avgrssi = sum(arraylevel) / len(arraylevel)

                return avgrssi

        time.sleep(.1)
```

Apéndice G

Código en Python con la implementación del algoritmo kdTree para los valores de RSSI

```
#! /usr/bin/env python

################################################################################
#
#       This class implements kd-tree. Given a point and range, this class looks for
#        a set of nearest neighbor.
#
#
################################################################################
from RSSIdat import *
class Node(object):

    def __init__(self):
        self.leftChild = None
        self.rightChild = None
        self.location = None
        self.axis = None
        self.coord=None


def buildKdtreeRSSI(pointList, depth=0):
    if not pointList:
        return

    #Select the axis based on depht.
    k = len(pointList[0][0])
    axis = depth %k

    #Sort points depended on axis and choose the midpoint
    pointList.sort(key=lambda point:point[axis])
    median = len(pointList) // 2

    node = Node()
    xcoor =int(pointList[median][0][0])
    ycoor = pointList[median][0][1]
    node.location = [xcoor,ycoor]

    node.axis = axis
    node.distRSSI=pointList[median][1]
    node.prob = pointList[median][2]
    node.leftChild = buildKdtreeRSSI(pointList[0:median], depth+1)
    node.rightChild = buildKdtreeRSSI(pointList[median+1:], depth+1)
    return node

def searchNeighborsRSSI(q, node, radius,coord):
```

```
46
    if (node == None):
48      return

50  if (isInside(q, node.location, radius) == 1):

52      data=[]
        data.append(node.location)
54      data.append(node.distRSSI)
        data.append(node.prob)
56      coord.append(data)

58
    if (node.location[node.axis] < q[node.axis]+radius):
60      searchNeighborsRSSI(q, node.rightChild, radius,coord)

62  if (node.location[node.axis] > q[node.axis]-radius):
        searchNeighborsRSSI(q, node.leftChild, radius,coord)
64
def isInside(pointA, pointB, dist):
66
    if (((pointB[0] - pointA[0])**2 + (pointB[1] - pointA[1])**2) <= dist**2):
68      return 1
    else:
70      return 0
```

Apéndice H

Código en Python para el control de motores con Raspberry Pi

Este código es usado tanto para el control de los motores del vehículo autónomo como los del vehículo de seguimiento.

```
import RPi.GPIO as GPIO
from time import sleep


#MOTOR1
in1 = 24
in2 = 23
en = 25

temp1=1
#MOTOR 2
in3 =17
in4 =27
en2 =22


GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

##INICIALIZACION DE PINES
#MOTOR1
GPIO.setup(in1,GPIO.OUT)
GPIO.setup(in2,GPIO.OUT)
GPIO.setup(en,GPIO.OUT)
GPIO.output(in1,GPIO.LOW)
GPIO.output(in2,GPIO.LOW)
p1=GPIO.PWM(en,2000)

#MOTOR2

GPIO.setup(in3,GPIO.OUT)
GPIO.setup(in4,GPIO.OUT)
GPIO.setup(en2,GPIO.OUT)
GPIO.output(in3,GPIO.LOW)
GPIO.output(in4,GPIO.LOW)
p2=GPIO.PWM(en2,2000)

def RunForward(speed):

    #print("run")
    if(temp1==1):

        p1.start(speed)
        p2.start(speed)

        #MOTOR1
```

```
        GPIO.output(in1,GPIO.HIGH)
        GPIO.output(in2,GPIO.LOW)
        #MOTOR2
        GPIO.output(in3,GPIO.HIGH)
        GPIO.output(in4,GPIO.LOW)
        #print("forward")
        x='z'
    else:


            #MOTOR1
        GPIO.output(in1,GPIO.LOW)
        GPIO.output(in2,GPIO.HIGH)
        #MOTOR2
        GPIO.output(in3,GPIO.LOW)
        GPIO.output(in4,GPIO.HIGH)
        #print("backward")

def RunLeft(speed):

    #MOTOR1

    GPIO.output(in3,GPIO.LOW)
    GPIO.output(in4,GPIO.LOW)
    GPIO.output(in1,GPIO.HIGH)
    GPIO.output(in2,GPIO.LOW)

def RunRigth(speed):

    GPIO.output(in3,GPIO.HIGH)
    GPIO.output(in4,GPIO.LOW)
    GPIO.output(in1,GPIO.LOW)
    GPIO.output(in2,GPIO.LOW)

def RunBackward(speed):
    print("backward")
    #MOTOR1
    GPIO.output(in1,GPIO.LOW)
    GPIO.output(in2,GPIO.HIGH)
    #MOTOR2
    GPIO.output(in3,GPIO.LOW)
    GPIO.output(in4,GPIO.HIGH)
    temp1=0

def Stop():

    #MOTOR1
    GPIO.output(in1,GPIO.LOW)
    GPIO.output(in2,GPIO.LOW)
    #MOTOR2
    GPIO.output(in3,GPIO.LOW)
    GPIO.output(in4,GPIO.LOW)
```

Apéndice I

Código de Python para el control del vehículo no autónomo por medio de la aplicación Blynk

```
import BlynkLib
import RPi.GPIO as GPIO
from time import sleep
from motorfun import *
from ledfun import *
# Initialize Blynk
blynk = BlynkLib.Blynk('Xnz34XiGhHhCEAXSGp4JpzaZwX6Qgm5g')

#MOTOR1
in1 = 24
in2 = 23
en = 25
temp1=1
#MOTOR 2
in3 =17
in4 =27
en2 =22
speed= 100
SPEED=None
@blynk.VIRTUAL_WRITE(0) # Get Joystick X  and Y axis on V0

def my_write_handler(value):
    global SPEED
    x = str(int(value[0]))
    y = str(int(value[1]))
    val = x+y
    blynk.virtual_write(2,x)
    blynk.virtual_write(3,y)

    if (val =='12'):
        if SPEED==None:
            RunForward(speed)
        else:
            RunForward(SPEED)

    if (val =='10'):

        RunBackward()

    if (val =='11'):

        Stop()

    if (val =='01' or val=="02"):
```

```
        RunLeft()

    if (val =='21' or val =='22'):

        RunRight()

    if (val =='00'):

        RunBackLeft()

    if (val =='20'):

        RunBackRight()


@blynk.VIRTUAL_WRITE(2)  # Get TurnLight  Right on V2
def my_write_handler(value):

    led = int(value[0])
    print("led =", led)

    if(led == 1):
        ledIZQ()

    if(led == 0):
        ledOFFI()
#


@blynk.VIRTUAL_WRITE(3)  # Get TurnLight  Left on V2
def my_write_handler(value):

    led = int(value[0])
    print("led =", led)

    if(led == 1):
        ledDER()

    if(led == 0):
        ledOFFD()
#
#
# main loop that starts program and handles registered events
while True:
    blynk.run()
```

Bibliografía

[1] R. P. tutorials, "Infrared distance measurement with the raspberry pi (sharp gp2y0a02yk0f)." https://tutorials-raspberrypi.com/infrared-distance-measurement-with-the-raspberry-pi-sharp-gp2y0a02yk0f/, 2018. [Accesado 31-Mar-2020].

[2] Wikipedia, "Cono de la coloración hsv." https://commons.wikimedia.org/wiki/File:Cono_de_la_coloraci%C3%B3n_HSV.png#/media/Archivo:Cono_de_la_coloraci\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{o\global\mathchardef\accent@spacefactor\spacefactor}\accent19o\egroup\spacefactor\accent@spacefactor\futurelet\@let@token\penalty\@M\hskip\z@skipn_HSV.png, 2015. [Accesado 31-Mar-2020].

[3] Blink, "What is virtual pins." http://help.blynk.cc/en/articles/512061-what-is-virtual-pins, 2019. [Accesado 31-Mar-2020].

[4] W. Guo, D. Brennan, K. Pavkova, and P. T. Blythe, "Intelligent speed control technology for older drivers," in *IET and ITS Conference on Road Transport Information and Control (RTIC 2012)*, pp. 1–6, Sep. 2012.

[5] VanDung Nguyen, Oanh Tran Thi Kim, Tri Nguyen Dang, Seung Il Moon, and C. S. Hong, "An efficient and reliable green light optimal speed advisory system for autonomous cars," in *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1–4, 2016.

[6] N. Hyldmar, Y. He, and A. Prorok, "A fleet of miniature cars for experiments in cooperative driving," *CoRR*, vol. abs/1902.06133, 2019.

[7] K. Bimbraw, "Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology," in *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, vol. 01, pp. 191–198, July 2015.

[8] A. Gambi, T. Huynh, and G. Fraser, "Automatically reconstructing car crashes from police reports for testing self-driving cars," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 290–291, May 2019.

[9] A. Koike and Y. Sueda, "Contents delivery for autonomous driving cars in conjunction with car navigation system," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1–4, Sep. 2019.

[10] S. Tariq, Hyunsoo Choi, C. M. Wasiq, and Heemin Park, "Controlled parking for self-driving cars," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 001861–001865, Oct 2016.

[11] R. Kulkarni, S. Dhavalikar, and S. Bangar, "Traffic light detection and recognition for self driving cars using deep learning," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pp. 1–4, Aug 2018.

[12] V. Rathod and R. Agrawal, "Camera based driver distraction system using image processing," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pp. 1–6, Aug 2018.

[13] D. Dong, X. Li, and X. Sun, "A vision-based method for improving the safety of self-driving," in *2018 12th International Conference on Reliability, Maintainability, and Safety (ICRMS)*, pp. 167–171, Oct 2018.

[14] E. J. Sen, K. Deepa Merlin Dixon, A. Anto, M. V. Anumary, D. Mieheal, F. Jose, and K. J. Jinesh, "Advanced license plate recognition system for car parking," in *2014 International Conference on Embedded Systems (ICES)*, pp. 162–165, July 2014.

[15] A.áliková@ and E.íková@, "Parking system with image processing," in *2019 IEEE 17th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pp. 281–286, Jan 2019.

[16] M. Jara, A. Guerrero, and M. Arzamendia, "Study and implementation of an autonomous navigation algorithm for a scale electric car," in *2019 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, pp. 1–7, 2019.

[17] M. Kim, T. Lee, and Y. Kang, "Experimental verification of the power slide driving technique for control strategy of autonomous race cars," in *International Journal of Precision Engineering and Manufacturing (2020) 21:377–386*, pp. 1–10, 2019.

[18] A. Gotlib, K.ukojć@ Łukojć, and M. Szczygielski, "Localization-based software architecture for 1:10 scale autonomous car," in *2019 International Interdisciplinary PhD Workshop (IIPhDW)*, pp. 7–11, 2019.

[19] Global.Sharp, "Gp2y0a710k sensor." `https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a710k_e.pdf`, 2011. [Accesado 31-Mar-2020].

[20] Microchip, "Mcp3004/3008." `https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf`, 2008. [Accesado 31-Mar-2020].

[21] O. team, "Open cv." `https://opencv.org/about/`, 2019. [Accesado 9-Dic-2019].

[22] A. Goldsmith, *Wireless Communications*. Cambridge Universitiy, 2005.

[23] Wikipedia, "Kd tree." `https://en.wikipedia.org/wiki/K-d_tree`, 2019. [Accesado 31-Mar-2020].

[24] Blynk, "Blynk." `https://blynk.io/about`, 2020. [Accesado 31-Mar-2020].