Doti, Santiago Andrés

Sistema de guiado de un robot móvil mediante visión artificial

2021

Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Ingeniería en Informática

TRABAJO FINAL DE LA PRÁCTICA PROFESIONAL SUPERVISADA

Sistema de guiado de un robot móvil mediante visión artificial

Estudiante:

Doti Santiago Andrés

Tutores:

Lic. Prof. Kelly Carolina

Prof. Ing. Salvatore Juan

Mg. Ing. Osio Jorge

Buenos Aires, noviembre de 2021

PRÁCTICA PROFESIONAL SUPERVISADA (PPS)

Sistema de guiado de un robot móvil mediante visión artificial.

Informe de Avance

DATOS DEL ESTUDIANTE

Apellido y Nombres: Doti Santiago Andrés

DNI: 40059539

Nº de Legajo: 23912

Correo electrónico: santiagoandresdoti@gmail.com

Cantidad de materias aprobadas al comienzo de la PPS: 45

PPS enmarcada en artículo (4 ó 7) de la Resolución (CS) 103/16.

DOCENTE SUPERVISOR

Apellido y Nombres: Prof. Ing. Salvatore Juan

Correo electrónico: jsalvatore@unaj.edu.ar

DOCENTE TUTOR DEL TALLER DE APOYO A LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ

Apellido y Nombres: Lic. Prof. Kelly Carolina

Correo electrónico: kellygcarolina@gmail.com

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS

Nombre o Razón Social: Universidad Nacional Arturo Jauretche

Dirección: Av. Calchaquí 6200, Florencio Varela, (1888) Buenos Aires, Argentina Teléfono: +54 11 4275-6100

Sector: Programa Tecnologías de la Información y la Comunicación (TIC) en aplicaciones de interés social, Instituto de Ingeniería y Agronomía

TUTOR DE LA ORGANIZACIONAL

Apellido y Nombres: Prof. Mg. Osio Jorge

Correo electrónico: josio@unaj.edu.ar

FIRMA DEL COORDINADOR DE LA CARRERA

DESARROLLO: Sistema de guiado de un robot móvil mediante visión artificial.

AGRADECIMIENTOS

Quisiera aprovechar este espacio para agradecer a todas aquellas personas que me acompañaron y confiaron en mí durante esta travesía universitaria.

Agradecimientos infinitos a toda mi familia, mis padres, mis hermanos, mi novia, mis primos; que siempre se mantuvieron muy cercanos a mí, al pie del cañón, en cada uno de los momentos difíciles y buenos a lo largo de mi desempeño universitario.

Gracias totales a todos mis compañeros de cursada que siempre me acompañaron en cada materia que compartíamos, siendo cada uno de ellos un gran apoyo en mi cursada y hoy en día son mis amigos.

Muchísimas gracias a todos aquellos profesores que clase a clase nos compartían sus conocimientos con pasión por enseñar y ante cada consulta se tomaron el tiempo para solventarlas. Mención especial para mis tutores en este proyecto que siempre se mostraron predispuestos a darme una mano con cada inconveniente o incertidumbre que tenía.

No tengo más que agradecimientos para todos ellos y cada uno de las personas que me cruce por la Universidad Nacional Arturo Jauretche que se mostraron proclives a dar una mano. Y, por último, pero no menos importante, gracias a la UNAJ; la institución que me formó como persona e ingeniero.

RESUMEN

Las técnicas y algoritmos de procesamiento de imagen llegaron para quedarse. Hoy en día son mucho más normales de que lo que podemos imaginar, ya que están incluidas en muchos ámbitos de la informática de las distintas tecnologías que consumimos actualmente. Si vamos al caso de la conducción de vehículos podemos encontrar marcas como Tesla que desde el año 2013 viene desarrollando y probando programas de conducción autónoma, ya sean tecnologías para estacionar en paralelo y perpendicular de forma independiente, para convocar automáticamente el vehículo hacia donde se encontrase el dueño y/o para cambiar automáticamente de carril mientras se conduce por la autopista.

El objetivo de este proyecto, además de utilizar tecnologías de procesamiento de imágenes, es el emplear la robótica con el objetivo principal de unir ambas, con el fin de crear un sistema autónomo capaz de operar en un entorno determinado. El sistema estará integrado por un robot compuesto por una cámara y un programa de software que, a partir de las imágenes obtenidas por ese dispositivo, guiará al robot por el medio de las líneas de un carril en un entorno determinado.

El robot estará compuesto por chasis de plástico, una Raspberry Pi, la cámara de la misma organización Raspberry Pi Camera, dos motores de corriente continua conectados a una rueda cada uno, una batería externa que alimentará todo el sistema y varios elementos más; todos incorporados y conectados de alguna forma a la placa, la cual debe soportar y ejecutar un programa de software desarrollado en el lenguaje de programación Python que será el encargado de llevar al robot hacia el objetivo.

ABSTRACT

Image processing techniques and algorithms are here to stay. Today they are much more normal than we can imagine, since they are included in many areas of computing of the different technologies that we currently consume. If we go to vehicle driving we can find brands such as Tesla that since 2013 has been developing and testing autonomous driving programs, such as technologies for parallel and perpendicular parking independently, to automatically make the vehicle go where the owner is and/or to automatically change lanes while driving on the highway.

The aim of this project, in addition to the use of image processing technologies, is to use robotics with the main objective of joining both, in order to create an autonomous system capable of operating in a given environment. The system will consist of a robot composed of a camera and a software program that, based on the images obtained by this device, will guide the robot along the lines of a lane in a given environment.

The robot will consist of a plastic chassis, a Raspberry Pi, the camera of the same organization Raspberry Pi Camera, two DC motors connected to wheels, an external battery that will power the entire system and several other elements; all incorporated and connected in some way to the board, which must support and run a software program developed in Python programming language that will be responsible for taking the robot to the target.

1. ÍNDICE

2. MARCO TEÓRICO

2.1 INTRODUCCIÓN

Cuando hablamos de un robot lo primero que muchos nos imaginamos es un objeto metálico con extremidades que imitan la forma física humana. Muchas veces emitiendo sonidos denominados *tech* y con un movimiento algo tosco que intenta imitar al ser humano. Indudablemente hoy en día esa visión de robot cada vez va quedando más atrás cuando vemos todo tipo de robots que son capaces de moverse a grandes velocidades, abrir puertas, realizar saltos y movimientos de parkour, aspirar todas las habitaciones de nuestra casa y hasta de investigar la superficie de distintos planetas de nuestro sistema solar, entre otras**.**

Este trabajo se enfocará principalmente en el desarrollo e implementación de un sistema de guiado de un robot móvil mediante visión artificial. En nuestro caso, el robot móvil consta de una Raspberry Pi 3, un ordenador de placa simple (*SBC* por sus siglas en inglés), conectada a una cámara web y una estructura vehicular, (cuatro ruedas con sus respectivos motores), que posibilitará su movimiento hacia cualquier dirección (izquierda, derecha, reversa y hacia delante). A través de la webcam es que entra en juego la visión por computadora y se convierte en los ojos del robot dándole toda la información correspondiente para que se pueda guiar en un entorno determinado, persiguiendo un objetivo móvil. Esto lo logra mediante el procesamiento de imágenes, el cual le permite detectar y seguir la posición de determinados objetos en las imágenes. A continuación, se envían los datos al SBC para que direccione las ruedas en pos de conseguir un objetivo previamente predeterminado. También es posible que se encuentre con un obstáculo en su camino que le impida tomar una decisión en base a su capacidad, ya sea directamente un objeto que le corte el trayecto o la falta de señalización de la carretera que lo guía. En los casos que no le sea posible planificar la próxima acción a realizar, se utilizará el sistema de monitoreo y control remoto del robot para indicarle las acciones a llevar a cabo con el fin de que pueda volver a funcionar de forma autónoma.

2.2 OBJETIVOS

El presente proyecto consiste en el desarrollo e implementación de un sistema de guiado de un robot móvil mediante visión artificial. El mismo tiene como principal objetivo unir la robótica y la visión por computadora con la finalidad de crear un sistema autónomo capaz de operar en un entorno determinado. Para poder lograr esto se determinaron una serie de objetivos específicos, que se detallan a continuación:

- Evaluar las distintas alternativas de procesamiento de imágenes para visión artificial (librería OpenCV, Deep Learning, aplicaciones por visión por computadora, entre otros.).
- Analizar las diferentes funciones de OpenCV que se van a utilizar para el procesamiento de imágenes. Probar funcionalidades similares con Deep Learning.
- Analizar las diferentes funciones y librerías disponibles para el control de dirección del robot.
- Desarrollar el *software* en lenguaje Python, que sea capaz de procesar las imágenes y obtener los correspondientes datos, a través de los cuales el robot recibirá instrucciones de dirección.
- Desarrollar una aplicación híbrida en un lenguaje de programación web que permita monitorear y controlar la dirección del robot de forma remota, mediante el acceso a la cámara y a los comandos del robot para controlar sus cuatro movimientos básicos (descritos anteriormente).
- Realizar distintas pruebas de funcionamiento al robot y a la aplicación de control y monitoreo.

3. CASO DE ESTUDIO

3.1 ESTADO DE LA TÉCNICA

En la siguiente sección se describen los aspectos técnicos del presente proyecto. Esta descripción abarca tanto las descripciones de cada función, herramienta o algoritmo de software implementado, como así también las especificaciones técnicas de cada dispositivo de hardware utilizado.

3.2 ROBÓTICA

La robótica como ciencia concierne a todo el estudio de aquellas máquinas que buscan reemplazar a los humanos en la ejecución de una tarea, tanto lo que respecta a su actividad física como a la toma de decisiones. Muchas veces se la define comúnmente como aquella que estudia la conexión inteligente entre percepción y acción. El término "robot" apareció por primera vez en una obra de ciencia ficción checa de 1920, *"Rossum's Universal Robots"* de Karel Čapek y su significado en la lengua checa era el de "trabajo de siervo" o, mejor dicho, trabajo duro o trabajo pesado. A lo largo de los años, la literatura y las historias de robots siguieron moldeando la opinión pública sobre estos nuevos aparatos: la mayoría finalizaban con los robots rebelándose, batallando contra la humanidad y destruyendo todo a su paso. Como la famosa serie de robots del escritor y profesor de bioquímica ruso Isaac Asimov que comenzó publicando en 1942 con el cuento corto *"*Círculo vicioso*"*, en los cuales también introdujo por primera vez las tres leyes básicas de la robótica:

1. Un robot no hará daño a un ser humano ni, por inacción, permitirá que un ser humano sufra daño.
2. Un robot debe cumplir todas las órdenes dadas por los seres humanos, a excepción de aquellas que entren en conflicto con la primera ley.
3. Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la primera o con la segunda ley.

Estas leyes establecieron reglas de comportamiento que luego fueron consideradas especificaciones al momento de diseñar un robot.

Consideramos un robot a toda máquina orientada que puede sentir, pensar y actuar. Éste detecta su entorno y usa esa información para planear la próxima acción a realizar en base a un objetivo previo que tenga definido, como puede ser el querer agarrar un objeto y trasladarlo. Para ello la máquina podría planear mover su brazo-robot y cogerlo, o bien si se encontrase lejos, movilizarse hacia el lugar donde se encuentre dicho objeto para posteriormente agarrarlo, así podrían ser miles las decisiones que planifique para cumplir su objetivo. Pero, de hecho, un *sistema robótico* es en realidad un sistema complejo, funcionalmente representado por múltiples sistemas (ver figura 3.1).

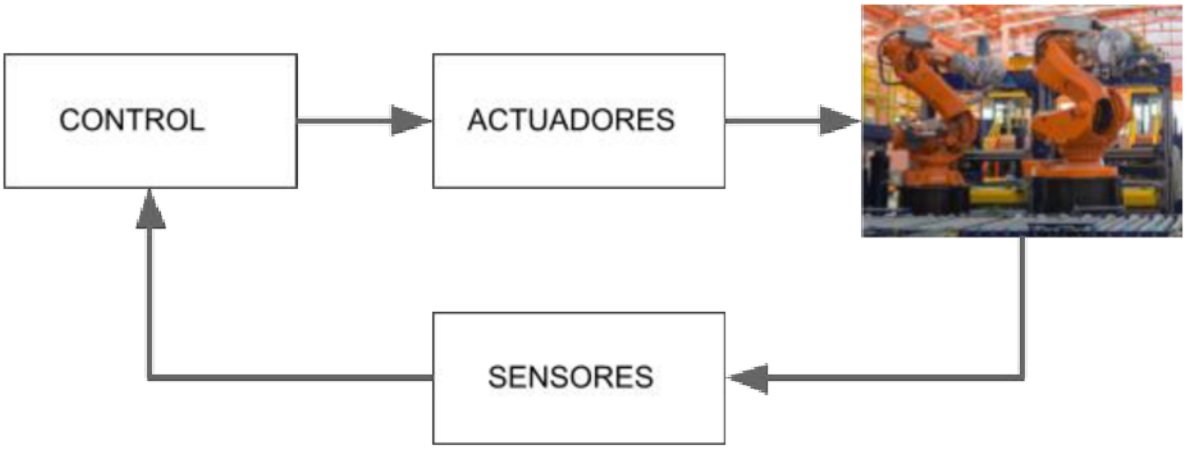

Figura 3.1 - Gráfico sobre los componentes de un sistema robótico. Adaptada del libro *Robotics – Modelling, Planning and Control* (2010, pág. 3).

El componente esencial de un robot es el "sistema mecánico", dotado generalmente de un aparato de locomoción (ruedas, orugas, piernas mecánicas) y de un aparato de manipulación (brazos mecánicos, efectores finales, manos artificiales).

La capacidad de ejercer una acción, ya sea de locomoción o manipulación, es provista por el "sistema de actuación", que anima a los componentes mecánicos del robot. El concepto principal de este sistema se refiere al contexto del control de movimiento, que se relaciona con servomotores, drivers y transmisores.

La capacidad de percepción se deposita en el "sistema sensorial", el cual puede adquirir datos sobre el estado interno del sistema mecánico (tanto sensores exteroceptivos como sensores de fuerza y cámaras). El concepto principal de este sistema se refiere al contexto de las propiedades de los materiales, el acondicionamiento de la señal, el procesamiento de datos y la obtención de la información.

La capacidad de conectar acción con la percepción de una forma inteligente es provista por un “sistema de control” que puede ordenar la ejecución de una acción, con respecto a los objetivos establecidos, por una técnica de planificación de tareas, a partir de las restricciones impuestas por el robot y el entorno.

3.3 VISIÓN POR COMPUTADORA

El campo de la visión por computadora es una combinación de diferentes campos, que incluyen (sin limitarse) a las ciencias de la computación, las matemáticas y la ingeniería electrónica. Dicho campo en particular estudia formas de capturar, procesar y analizar imágenes y videos del mundo real para transformarlos en información que puedan ayudar en la toma de decisiones de quien lo necesite o requiera. En otras palabras, es la transformación de datos procedentes de una cámara fija o de video en una decisión o en una nueva representación. Estos datos de entrada pueden incluir información contextual, como por ejemplo: “la cámara está montada en un coche” o “el visor láser indica que hay un objeto a un 1 metro de distancia”, y la decisión puede ser, por ejemplo: “hay una persona en esta escena” o “hay 14 células tumorales en esta dispositiva”. El objetivo final de la mayoría de estos sistemas de visión por ordenador es la de extraer información útil de las imágenes fijas y los videos (pudiendo ser pregrabados o transmitidos en directo) con el fin de tomar decisiones en base a esa información obtenida.

Si prestamos atención a una visión más general, podemos observar que este campo en concreto se relaciona muy fuertemente con los de la inteligencia artificial y la visión artificial. Muchas áreas de estudio de dichos campos se solapan y se comparten como, por ejemplo, el procesamiento de imágenes, el reconocimiento de patrones y el aprendizaje automático. Esto se puede ver representado gráficamente en la figura 3.2.

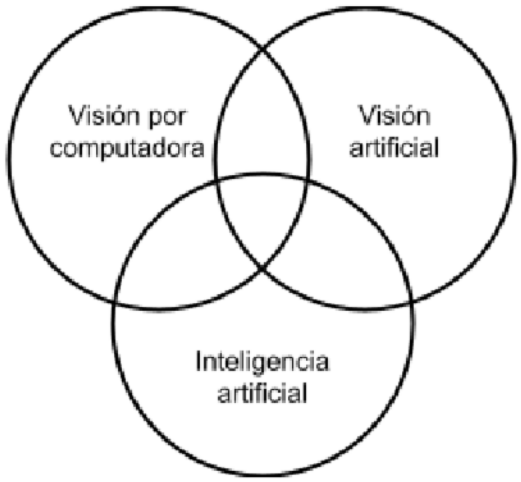

Figura 3.2 - Gráfico sobre las relaciones entre los diferentes dominios científicos, adaptado del libro *Raspberry Pi computer vision programming* (2020, pág. 2) de Aswin Pajankar.

Como somos seres muy visuales, es fácil pensar que en este campo las cosas son relativamente fáciles, pero las intuiciones y pensamientos iniciales pueden darnos una idea errónea. El cerebro humano suele dividir la señal de visión en muchos canales que transmiten diferentes tipos de información al cerebro y, en base a esto, dispone de un sistema de atención que identifica, en función de la tarea, las partes importantes de una imagen que hay que examinar y descarta otras. En esta instancia, hay mucha retroalimentación en el flujo visual que aún no se conoce en profundidad. También los sensores de control muscular y el resto de los sentidos aportan una gran cantidad de información asociativa que permite al cerebro establecer diferentes tipos de asociaciones cruzadas, tras años de experiencia en el mundo. Los bucles de retroalimentación del cerebro se remontan a todas las etapas de procesamiento, incluidos los propios sensores de *hardware* (los ojos) que controlan, mecánicamente, la iluminación a través del iris y afinan la recepción en la superficie de la retina.

En un sistema de visión artificial, sin embargo, las cosas son bastante diferentes: una computadora recibe una cuadrícula de números de la cámara o el disco y eso es todo. En su mayoría, no hay reconocimiento de patrones incorporado, ni control automático del enfoque y la apertura, ni asociaciones cruzadas con años de experiencia. En su mayoría, los sistemas de visión siguen siendo bastante ingenuos. En la figura 3.3 podemos ver un ejemplo de lo que nombramos anteriormente: para una computadora, el espejo lateral de un automóvil es sólo eso, una cuadrícula de números. Además de que por sí sola nos da bastante poca información, hay que

tener en cuenta que cada uno de esos números tiene un componente de ruido bastante grande. Esta corrupción de los datos se debe a las variaciones del mundo (clima, iluminación, reflejos, movimientos, etc.), a las imperfecciones del objetivo, la configuración mecánica, al desenfoque de movimiento, al ruido eléctrico que pueden generar los sensores u otros componentes electrónicos de la cámara y a los artefactos de comprensión tras la captura de la imagen, entre otras.

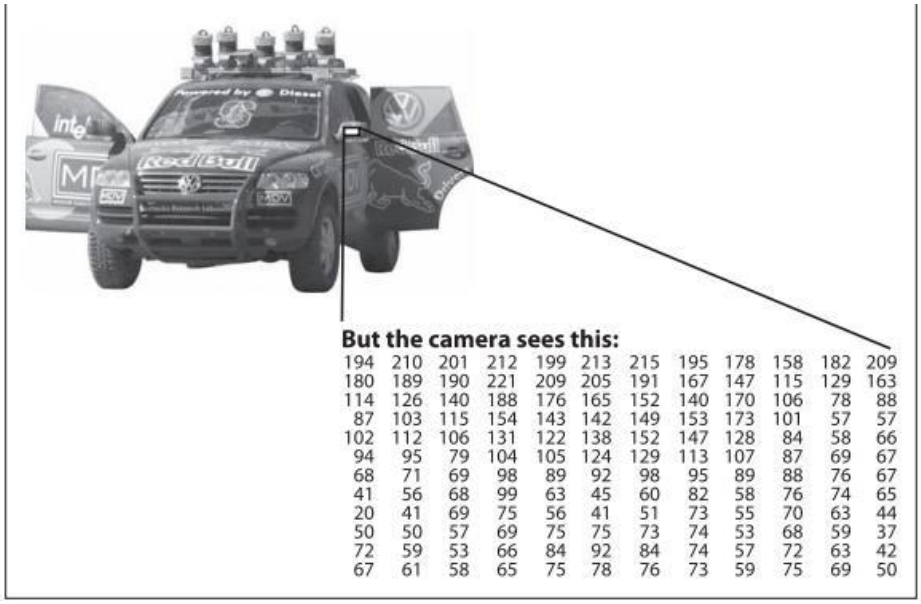

Figura 3.3 - Gráfico que representa lo que una cámara lateral es para una computadora. Extraído del libro *Learning OpenCV - Computer vision with the OpenCV library* (2008, pág. 3).

A raíz de las grandes dificultades planteadas en el último párrafo, surge la gran duda de cómo diseñar y desarrollar un sistema práctico de visión por computador en la actualidad. Esto se logra gracias a la utilización de conocimientos contextuales adicionales para poder sortear las limitaciones que nos imponen los sensores visuales. Si pensamos, por ejemplo, en un robot que debe encontrar y recoger bolígrafos en un edificio, éste puede utilizar el hecho de que un escritorio es un objeto que se encuentra en las oficinas y que las lapiceras se encuentran sobre todo en los escritorios. Esta información proporciona a la máquina una referencia de tamaño implícita: las biromes deben poder colocarse en dichos muebles. También ayuda a eliminar los falsos “reconocimientos” de bolígrafos en lugares imposibles, como por ejemplo en un techo o en una ventana. El robot, inclusive, puede ignorar con seguridad un cartel publicitario de lapiceras de 60 metros porque este carece del fondo de madera de un escritorio. En contraste, cuando se trata de tareas de

recuperación de imágenes, todas las fotografías de bolígrafos de una base de datos pueden ser bolígrafos reales, por lo que los tamaños grandes y otras configuraciones inusuales pueden haber sido excluidos explícitamente por las suposiciones de quienes tomaron esas fotografías. Muy posiblemente el fotógrafo sólo tomó imágenes de lapiceras reales de tamaño normal, además de que la gente tiende a centrar los objetos al sacar fotos y a colocarlos en orientaciones características. Por lo tanto, a menudo hay bastante información implícita no intencionada en las fotos tomadas por la gente.

La información contextual también puede modelarse explícitamente con técnicas de aprendizaje automático, y así variables ocultas, como el tamaño, la orientación a la gravedad, etc. pueden correlacionarse con sus valores en un conjunto de entrenamiento etiquetado. Como así también, se pueden intentar medir las variables ocultas de sesgo mediante la utilización de sensores adicionales como, por ejemplo, un telémetro láser que, para medir la profundidad, nos permite medir con precisión el tamaño de un objeto.

Lo próximo a enfrentar es el ruido, al que normalmente se lo trata con métodos estadísticos, los cuales tienen en cuenta, por ejemplo, estadísticas a lo largo del tiempo. Otros construyen modelos explícitos que aprenden directamente de los datos disponibles, como por ejemplo las bien conocidas distorsiones en las lentes; con sólo conocer los parámetros de un modelo polinómico sencillo es suficiente para describir y contrarrestar, casi por completo, esas distorsiones.

En cuanto a las decisiones o acciones que la visión por computadora intenta tomar, basándose en los datos de la cámara, siempre se realizan dentro del contexto de una tarea o propósito especificado previamente. Por ejemplo, podemos querer eliminar el ruido que haya en las imágenes para que nuestro sistema de seguridad emita una alerta, si alguien intenta escalar una valla del perímetro, o porque queremos que nuestro sistema, como en el desarrollado en el presente proyecto, guíe a un robot móvil a través de un camino especificado. El software siempre actuará en consecuencia del objetivo planteado con anterioridad: esto significa que aquel que fue pensado para que los robots deambulen por los edificios de oficinas empleará, estrategias diferentes a las de aquel que fue planeado para las cámaras de seguridad fijas, ya que ambos sistemas tienen un contexto y objetivos

significativamente diferentes. Por regla general, cuanto más restringido sea el contexto de la visión por ordenador, más podemos confiar en esas restricciones para simplificar el problema y más fiable será la solución final del problema planteado.

4. SOLUCIÓN PROPUESTA

4.1 ARQUITECTURA DEL ROBOT.

En esta sección se detallan los elementos que se utilizaron para armar el robot móvil. La descripción de dichos elementos comprende los componentes de su estructura como el sistema de locomoción, sistema de alimentación, chasis y los componentes electrónicos, como los motores, la cámara y los módulos, entre otros. En primera instancia se reutilizaron elementos del robot educativo llamado *"*Robot N4*"* pero, debido a varias complicaciones, se tuvo que desarrollar una segunda versión con elementos del robot de la siguiente generación de la misma empresa RobotGroup, el llamado "Robot N6*"*.

Las dificultades encontradas y las decisiones tomadas, durante el armado de todo el dispositivo, se detallan a continuación en cada apartado de las siguientes secciones.

4.1.1 Estructura (chasis y ruedas).

Para sostener todos los componentes que conforman el robot es necesario contar con una estructura que los soporte y provea de ciertas habilidades al robot para que pueda cumplir con el objetivo que tiene previsto.

4.1.2 Primera versión.

Primeramente, se reutilizó chasis del robot N4 (ver figura 4.1) que ya se encuentra montado y listo para funcionar de fábrica, con un microcontrolador de Arduino para la aplicación de distintos ejercicios educativos de la organización productora.

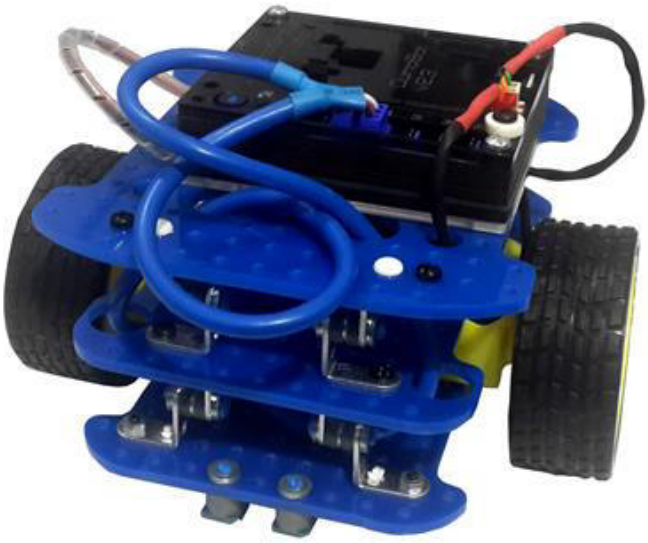

Figura 4.1 - Robot N4, vista trasera. Extraída del twitter oficial de RobotGroup.

Algunos de los demás componentes que incluía el robot original son:

- Dos ruedas motrices de 65mm que se ubican a ambos lados del dispositivo.
- Una rueda simple sin tracción que se ubica en la parte delantera del dispositivo y le permite realizar giros.
- Dos motores de corriente continua de 3v - 6v para controlar la velocidad de las ruedas.
- Portapilas con espacio para 3 pilas, que provee una alimentación de 6v.
- Tornillería y cables conectores.

En esta versión, la estructura de plástico se mantuvo igual y lo que se reemplazó fue el microcontrolador por una Raspberry Pi 2 y el portapilas por una batería de 10000 mAh. Dado que la parte superior posee una gran superficie fue posible colocar allí el driver L298N junto a la placa mencionada previamente. Además, se instaló un chasis rectangular en forma vertical para sostener la cámara y así, mediante la transmisión de video en vivo, poder estudiar los próximos pasos a seguir.

Una vez que se instaló todo y se realizaron las primeras pruebas con el robot, pudo observarse que la rueda derecha tenía una leve inclinación, la cual todo parecía indicar que era producida porque el eje estaba torcido. Este desperfecto provocaba que, cuando debía seguir una trayectoria recta, se desviaba un poco y no terminaba respetando las indicaciones iniciales. Es por esto que se decidió cambiar el modelo del robot que se usa como base y, en lugar de utilizar el robot N4, utilizar el robot N6.

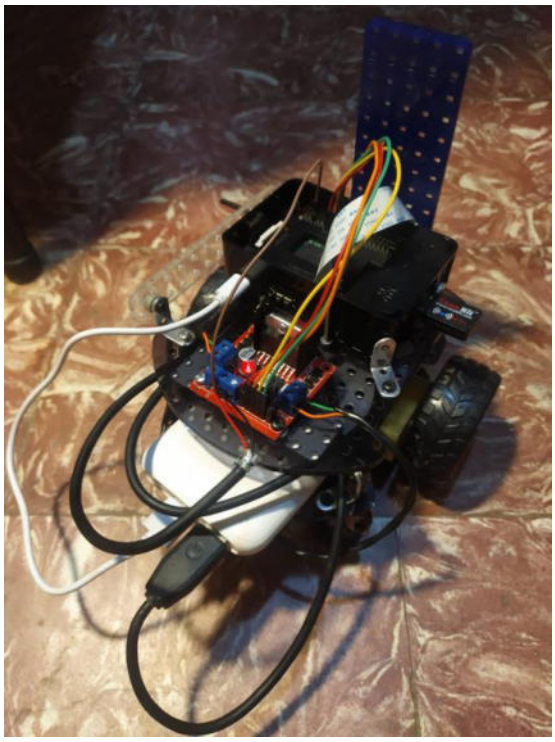

Figura 4.2 - Montaje final de la primera versión del robot móvil, vista trasera. Imagen de mi autoría.

4.1.3 Segunda versión.

En la segunda versión, se utilizó de base el robot N6 (ver figura 4.3), que es un modelo mucho más robusto y completo que el anterior. De fábrica incluye un microcontrolador Arduino para utilizarlo en ejercicios educativos, como así también otros componentes:

- Portapilas con espacio para 3 pilas, el cual provee una alimentación de 6v.
- Dos ruedas motrices de 65mm que se ubican a ambos lados del dispositivo.

- Una rueda simple sin tracción que se ubica en la parte delantera del dispositivo y le permite realizar giros.
- Dos motores de corriente continua de 2.5 a 13.5 V.
- Cuatro sensores reflexivos para detectar cambios de contrastes en superficies claras y oscuras.
- Un sensor ultrasónico que permite medir la distancia a la que se encuentra un objeto mediante ondas ultrasónicas.

Algo a destacar de esta versión del robot es que, en comparación, los motores son de una mejor calidad. Su material de construcción es muy superior, ya que están construidos en metal, mientras que los del N4 lo están en plástico.

Figura 4.3 - Robot N6, vista delantera. Imagen obtenida del sitio gubernamental de Chubut
https://ciencia.chubut.gov.ar/2016/09/06/talleres-de-robotica-y-programacion-de-videojuegos/

De igual forma que en el caso anterior, se reemplazó el microcontrolador por la Raspberry Pi. Por una cuestión de espacio y comodidad, en la parte superior del robot, y en forma horizontal, se colocó una de las plataformas del robot N4 para poder posicionar allí la placa junto al driver L298N. También se instaló otra en la parte inferior trasera para alojar el sistema de alimentación. De esta forma, se logró

posicionar, de manera más cómoda, todos los componentes necesarios para el funcionamiento del robot.

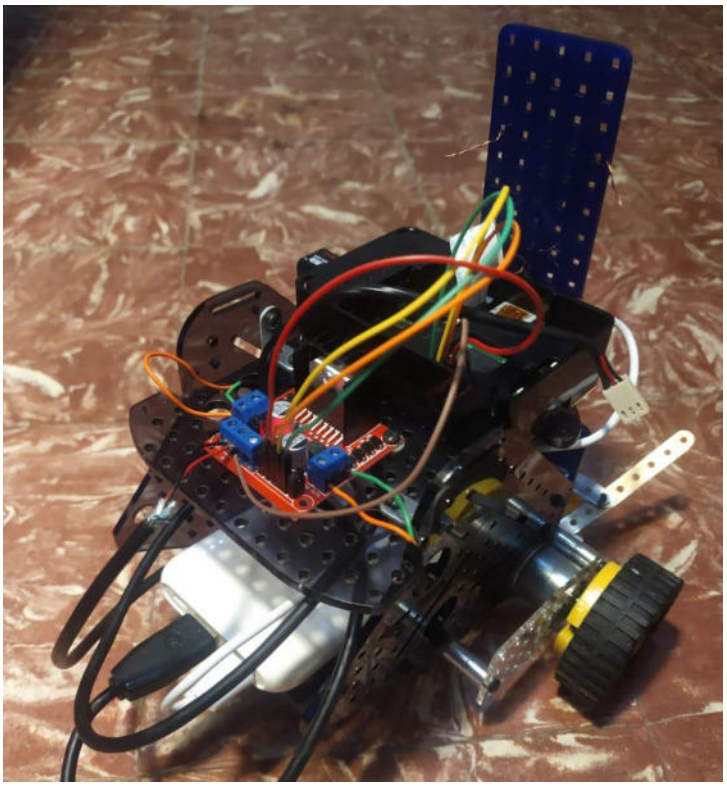

Figura 4.4 - Vista trasera del robot armado. Imagen de mi autoría.

4.1.4 Sistema de locomoción diferencial.

El sistema de locomoción de un robot es el encargado de la traslación del mismo en una superficie determinada: como en nuestro caso el robot circulará por superficies mayormente planas, no hará falta contar con un sistema demasiado complejo. Así se optó por utilizar el sistema de locomoción diferencial que ya empleaban los robots N4 y N6. La estructura de estos cuenta con dos ruedas laterales con tracción y una trasera sin tracción. Las dos primeras están situadas diametralmente opuestas en un eje perpendicular a la dirección del robot y cada una de ellas cuenta con un motor, de forma que los giros se realizan otorgándole diferentes velocidades a las ruedas o frenando una mientras gira la otra, permitiéndole al robot ir recto, girar sobre sí mismo y trazar curvas. Mientras que la restante, comúnmente llamada “rueda loca”, no posee ningún motor y, por lo tanto, gira libremente según la velocidad del robot. Además, permite mantener el equilibrio del robot, siendo el apoyo adicional que necesitan las ruedas laterales (diseño triangular).

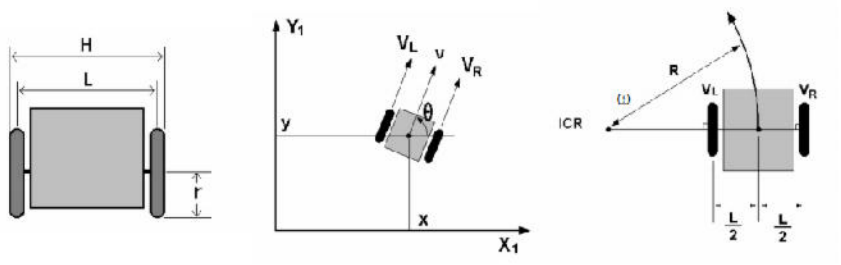

Figura 4.5 - Sistema de locomoción diferencial, no posee ruedas directrices. Extraído de la web (sin fuente reconocible)

4.1.5 Sistema de alimentación.

Originalmente se pensó alimentar todo el sistema del proyecto con dos fuentes de alimentación. Una principal, que se encargaría de alimentar a la Raspberry Pi, es una batería externa que se utiliza comúnmente para alimentar dispositivos móviles mediante conexión USB (ver figura representativa 4.6). En nuestro caso, contábamos con dos de ellas (una actuando como reserva), con distintas capacidades cada una, de 5000 mAh y de 10000 mAh, respectivamente. La otra fuente de alimentación se encargaría de proveer energía al driver L298N, encargado, a su vez, de controlar los motores de corriente continua que accionan las ruedas para que el robot se pueda mover. Ese era el portapilas original del robot, que posee lugar para tres pilas doble A (cada una de ellas de aproximadamente 2500 mAh en promedio), entregando alrededor de 6V a los motores.

Figura 4.6 - Batería externa de 10000 mAh. Imagen representativa, extraída de la web (sin fuente reconocible).

Por lo tanto, se pensó solamente añadir una nueva fuente de alimentación (la principal) para montar el nuevo robot, manteniendo el portapilas (segunda fuente de

alimentación) y reemplazando la placa Arduino por la Raspberry Pi conectada a la batería externa. Al momento de montarlo en la primera versión del robot (modelo N4) se realizaron las primeras pruebas y al principio se obtuvieron resultados satisfactorios, pero al repetirlas los motores ya no funcionaban. Por lo tanto, en primera instancia, se evaluó si había un problema en el conexionado y/o en el código de prueba. Al no encontrar problema alguno se procedió a inspeccionar el sistema de alimentación: fue aquí donde pudo detectarse que las pilas se descargaban muy rápido y, por lo tanto, se debían recargar constantemente a fin de lograr un correcto arranque de los motores. Es así que se decidió rediseñar el sistema de alimentación por completo, reemplazando las pilas por una batería de 10000 mAh. Inicialmente no se utilizó debido a su gran tamaño y peso, pero con el nuevo espacio ganado se la pudo incorporar al sistema, sin que cause demasiadas alteraciones en el equilibrio del mismo. Además, teniendo en cuenta que posee dos puertos USB (dos salidas de carga), es posible alimentar dos dispositivos a la vez y, por consiguiente, no es necesario utilizar la otra batería de 5000 mAh. Lo que significa que queda como única y principal fuente de alimentación de todo el sistema completo. Asimismo, cuenta con un botón de encendido y apagado que facilita la utilización del robot y provoca que no se malgaste su energía.

Cuando se armó la segunda versión del robot (modelo N6), directamente se utilizó la batería de 10000 mAh, previamente habiendo montado un chasis de forma horizontal en la parte inferior trasera para apoyarla y que el peso de la misma no desequilibre a todo el robot. Se puede, así, alimentar de forma óptima a todo el sistema.

4.1.6 Raspberry Pi 2 Model B.

Raspberry Pi es un ordenador de placa simple (SBC por sus siglas en inglés, *Single Board Computer*) o microcontrolador, desarrollado por la fundación del mismo nombre, con el objetivo de estimular la enseñanza de la informática en los colegios. Por ende, una de las características principales que la hace sumamente interesante es su bajo costo. Posee un tamaño similar al de una tarjeta de crédito (85mm x 53mm), en su versión base no posee periféricos y hoy en día ya van por la cuarta generación de placas (Raspberry Pi 4 model B fue anunciada en 2019), habiendo ya alrededor de 30 millones de unidades repartidas alrededor de todo el mundo.

Figura 4.7 - Raspberry Pi 2 Model B. Extraída de la página oficial de Raspberry Pi
https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

Todas las versiones cuentan con un procesador de la marca Broadcom, memoria RAM, GPU, puertos USB, HDMI, conector para la cámara, y un puerto Ethernet junto con 40 pines GPIO desde la segunda generación.

En cuanto al modelo utilizado en este proyecto, la Raspberry Pi 2 Model B (ver figura representativa anterior), contiene las siguientes especificaciones técnicas:

CPU	Broadcom BCM2836 de 900 Mhz ARM Cortex-A7 de cuatro núcleos
Memoria	SDRAM LPDDR de 1 Gb
Almacenamiento	1 ranura para MicroSD
Consumo energético	800 mA (4.0 W)

Tabla 4.1 - Especificaciones técnicas de la Raspberry Pi 2 Model B.

La elección de esta placa por sobre otras se basó, principalmente, en la gran inserción que tiene en el mercado, facilitando las herramientas de *software* necesarias para el proyecto, como así también la gran cantidad de material, recursos y ejemplos prácticos que se encuentran en internet. Además de que es un material con el que ya contaba la Universidad Nacional Arturo Jauretche (UNAJ) y, por lo tanto, en este sentido, es interesante mencionar que ahorramos recursos para el presente proyecto.

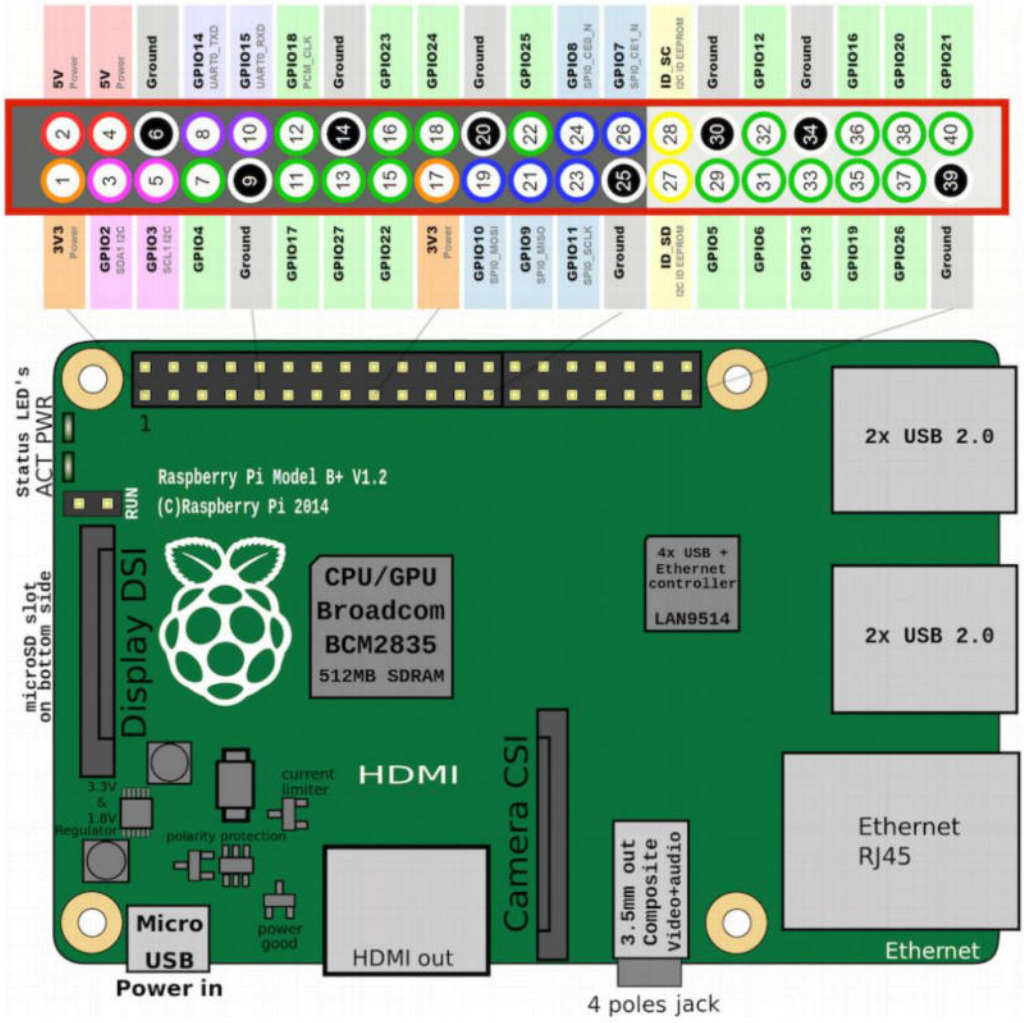

Figura 4.8 - Esquema de Raspberry Pi con diagrama de sus pines GPIO. Imagen extraída de https://www.hackster.io/the-swiftpi-team/swift-3-0-for-raspberry-pi-gpio-getting-started-393dd4

4.1.7 Raspberry Pi Camera Board v1.3.

Es el módulo oficial de la fundación para acompañar a la placa y proveerla de un dispositivo capaz de capturar imágenes y videos a 1080p. Cuenta con una cámara de 5mp (megapíxeles) que se conecta directamente a la Raspberry Pi mediante el puerto CSI (*Camera Serial Interface*).

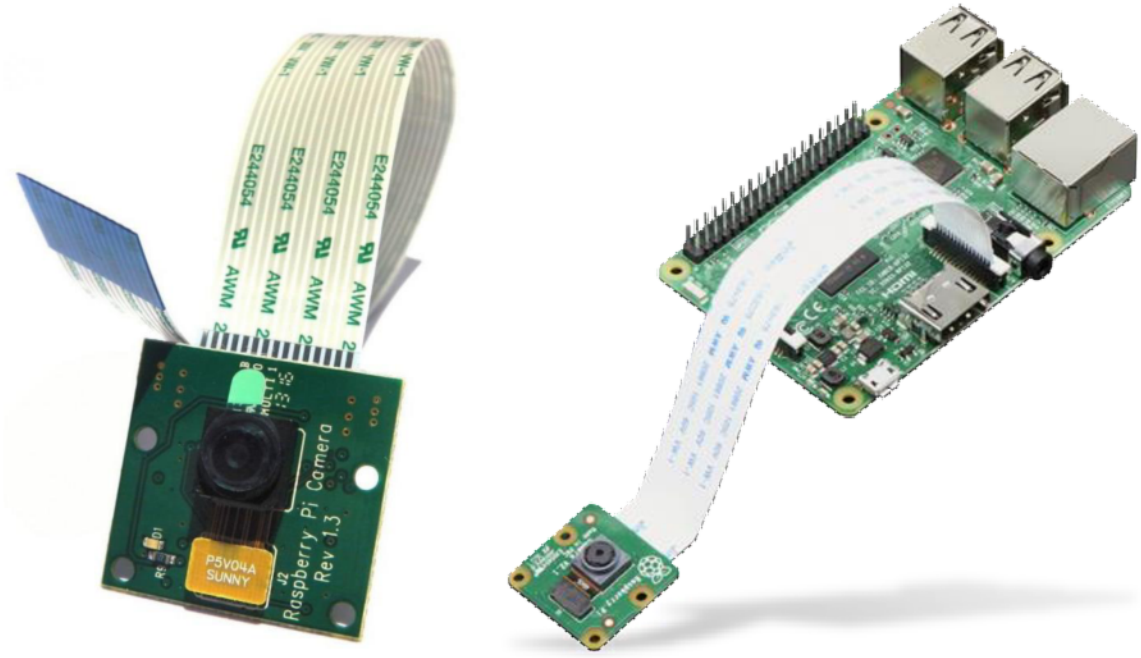

Figura 4.9 - Rasperry Pi Camera Board v1.3. Imagen extraída de
https://www.mouser.com/images/marketingid/2018/img/176394500.png?v=051720.0826

En nuestro caso contamos con la versión 1.3, la cual pesa tan sólo 3 gramos y tiene un tamaño alrededor de 20 x 25 x 9mm. Permite tomar fotografías nítidas gracias a su sensor Omnivision OV5647 de 5 megapíxeles de resolución, como así también grabar un video HD en 1080p a 30 FPS (*Frame per second*) o a 720p a 60 FPS. Su conector, el bus CSI, fue diseñado especialmente para las interfaces de cámaras y es capaz de alcanzar velocidades de datos extremadamente altas, trasladando exclusivamente los datos de los píxeles directamente al procesador. Algunas de sus características técnicas se resumen en el siguiente cuadro:

Tamaño	Aproximadamente 25 x 24 x 9mm
Peso	3 gramos
Modos de video	1080p @ 30, 720p @ 60 y 480p 60/90
Resolución de imagen fija	2592 x 1944
Sensibilidad	680 mV/lux-sec
Sensor	OmniVision OV5647
Rango dinámico	67 dB @ 8x de ganancia

Tabla 4.2 - Especificaciones técnicas de la Raspberry Pi Camera Board v1.3

El porqué de su elección se debe a su diminuto tamaño y escaso peso que la hacen perfecta para su aplicación en robots móviles, como así también su gran compatibilidad con la placa, dado que son del mismo fabricante y su instalación es de muy fácil aplicación. Para conseguir un buen rendimiento se montó la cámara sobre un chasis de plástico rectangular que se encuentra instalado en forma vertical al frente del robot para, de esta forma, lograr captar unas buenas imágenes, en vivo, del camino que tiene por delante.

4.1.8 Módulo L298N.

Es el dispositivo encargado de controlar el funcionamiento de dos motores de corriente continua, permitiendo manejar su velocidad y sentido de giro. Esto es gracias a que posee dos canales de Puente H y está formado por un driver L298N que controla el giro de los motores haciendo uso de la modulación por ancho de pulso (*PWN* por sus siglas en inglés), diodos de protección, un regulador de voltaje de 5V (78M05), un conector de 6 pines para recibir señales TTL que le envía la Raspberry Pi para poder controlar los motores, un conector de 3 pines para la alimentación del módulo (6 -12v, GND y 5v) y dos borneras de 2 pines para la salida hacia los motores (ver referencias de la figura 4.12). También cuenta con la posibilidad de trabajar con motores de un voltaje que oscile entre los 12v y los 35v. Para esto debemos desactivar el regulador, quitando el jumper de 5v, y utilizar dos fuentes de alimentación diferentes. Para el proyecto sólo se utilizará una fuente conectada a la entrada de 6 – 12v por lo que no hará falta quitarlo para habilitarlo.
Originalmente no se planificó utilizar este *driver*, pero las prestaciones que ofrece este dispositivo hicieron que se lo incluya en la implementación del proyecto. Su controlador de motor sigue la configuración de puente H., lo cual es útil para controlar la dirección de motor de corriente continua (CC). Un esquema de puente en H se ve de la siguiente manera:

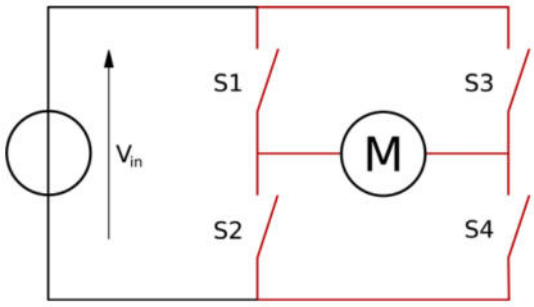

Figura 4.10 - Esquema de puente H, de elaboración propia.

El beneficio principal de utilizarlo es que puede proporcionar una fuente de alimentación separada a los motores. Esto toma principal relevancia cuando se utiliza una placa Raspberry Pi donde la fuente de alimentación de 5V simplemente no es suficiente para alimentar dos motores de CC.

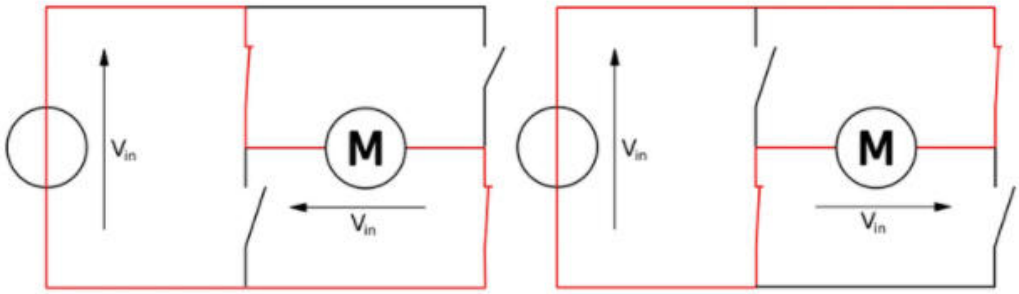

Figura 4.11 - Esquema de la dirección del motor en determinados casos, de elaboración propia.

El motor gira en la dirección que dictan los interruptores. Cuando S1 y S4 están encendidos, el terminal del motor izquierdo es más positivo que el terminal derecho y el motor gira en una dirección determinada. Por otro lado, cuando S2 y S3 están encendidos el terminal del motor derecho es más positivo que el terminal del motor izquierdo, haciendo que el motor gire en la otra dirección.

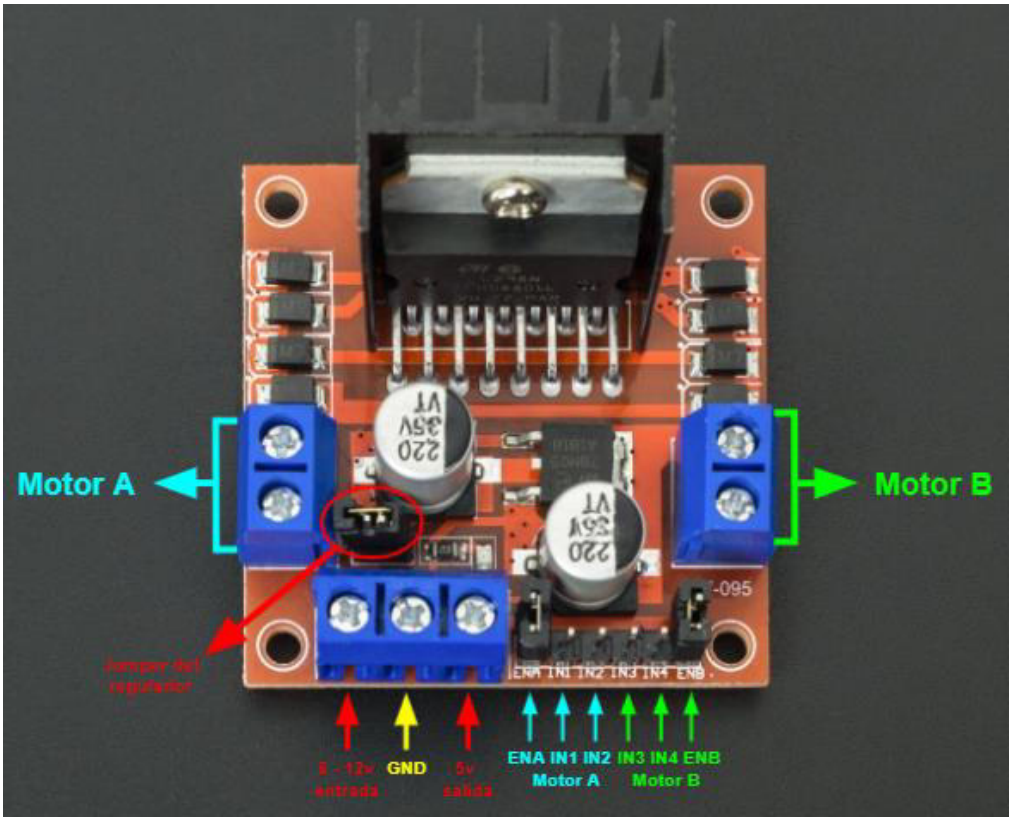

Figura 4.12 - Módulo L298N. Esquema de mi autoría.

El conexionado del control del módulo se hace mediante el conector de 6 pines mencionado previamente, de los cuales ENA, IN1 y IN2 corresponden a las entradas para controlar el motor A (OUT1 y OUT2) y IN3, IN4 y ENB permiten controlar el motor B (OUT3 y OUT4); todos ellos están conectados con sus correspondientes pines GPIO de nuestra placa SBC. A continuación, se detalla el conexionado de los pines y el esquema simple del conexionado general:

L298N	GPIO (Raspberry Pi)
IN1	27
IN2	22
IN3	23
IN4	24

Tabla 4.3 – Conexiones entre la Raspberry Pi (GPIO) y el módulo L298N.

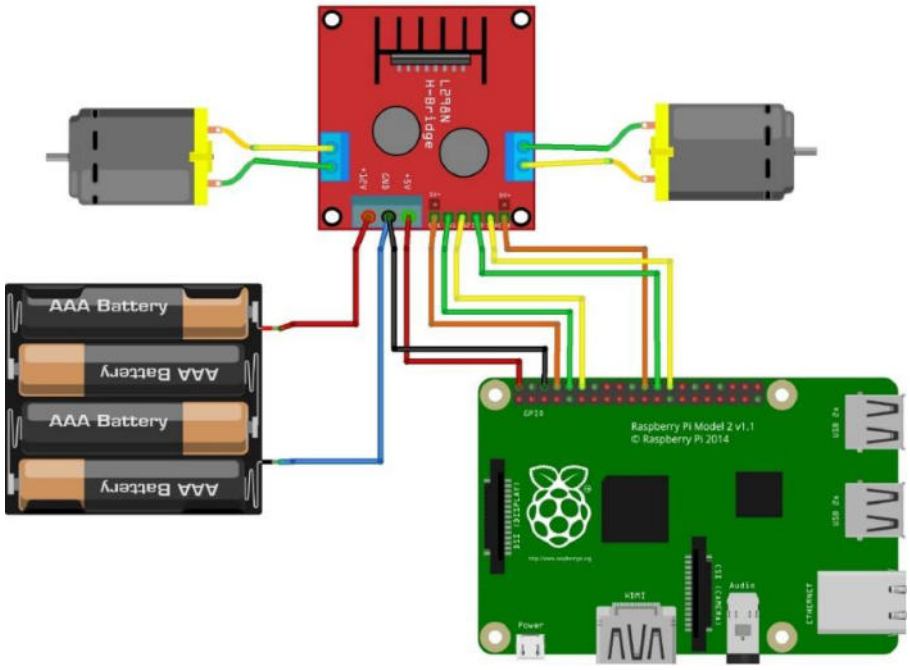

Figura 4.13 - Esquema simple del conexionado. Extraída de los foros de Raspberry Pi (sin fuente reconocible).

4.1.9 Sensor ultrasónico HC-SR04.

Es un sensor de distancia que utiliza ultrasonido para determinar la distancia de un objeto que se encuentre en rango de 2 a 450 cm. Se destaca principalmente por su bajo consumo energético, buena precisión y un bajo costo.

Figura 4.14 - Sensor ultrasónico HC-SR04. Imagen obtenida de https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html

Posee dos transductores: un emisor y un receptor piezoeléctricos. El primero emite ocho pulsos de ultrasonido (40 KHz), luego de recibir la orden a través del pin TRIG. Dichas ondas viajan en el aire y rebotan al encontrar un objeto y, al hacerlo, producen un sonido de rebote que es detectado por el receptor del dispositivo y lo

traducen en un pulso por el pin ECHO. Con esto podemos saber la cantidad de tiempo que demoró la onda desde que fue emitida hasta que fue detectada, pudiendo así calcular la distancia al objeto mediante la siguiente fórmula (siendo la velocidad del sonido $343.2\ \frac{m}{s}$):

$$Distancia\ (m) = \frac{(Tiempo\ del\ pulso\ ECO) * (Velocidad\ del\ sonido)}{2}$$

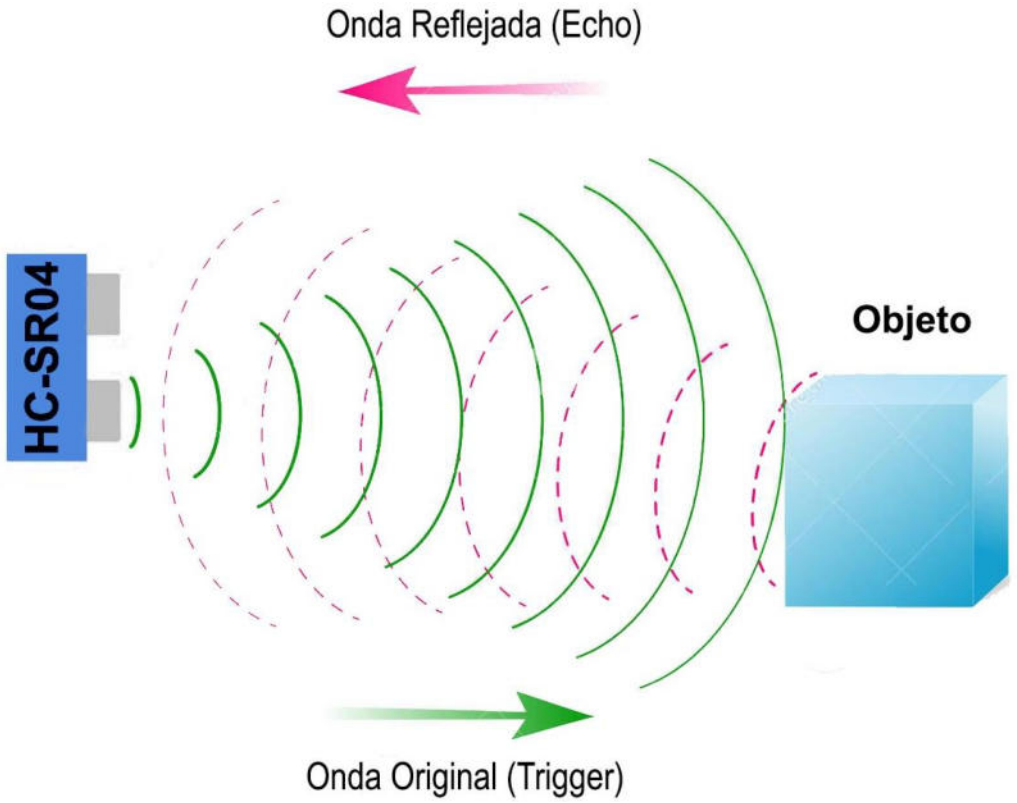

Figura 4.15 - Esquema de funcionamiento del sensor ultrasónico HC-SR04. Imagen sacada de https://www.zonamaker.com/arduino/modulos-sensores-y-shields/ultrasonido-hc-sr04

Además de los pines de Trigger (TRIG) y Echo (ECHO), el microcontrolador cuenta con uno de alimentación (VCC) y uno de tierra (GND).

Algunas de sus características técnicas son:

Alimentación	+5V DC
Frecuencia de trabajo	40 KHz
Rango de distancia	2 cm a 400 cm (a partir de 250cm la resolución no es buena)
Consumo	2mA (suspendido) y 15mA (trabajando)
Resolución	0.3cm

Tabla 4.4 - Características técnicas del sensor ultrasónico HC-SR04

En nuestro robot este sensor fue colocado en la parte frontal del mismo, justo por debajo de donde se encuentra la Raspberry Pi Camera. Allí se encuentra tomando constantemente mediciones para asegurarse de que ningún objeto se acerque lo suficiente como para provocar una colisión y poner en peligro al robot. Si el sensor ultrasónico detectase que un objeto se encuentra a una distancia menor a 5 centímetros detendrá cualquier movimiento del robot y solicitará la actuación externa (nosotros) para salir de esa situación, ya sea mediante la conducción manual o el reposicionamiento del robot en sí. El robot recién volverá a funcionar normalmente una vez que el sensor ultrasónico le informe que no hay objetos peligrosos frente a él.

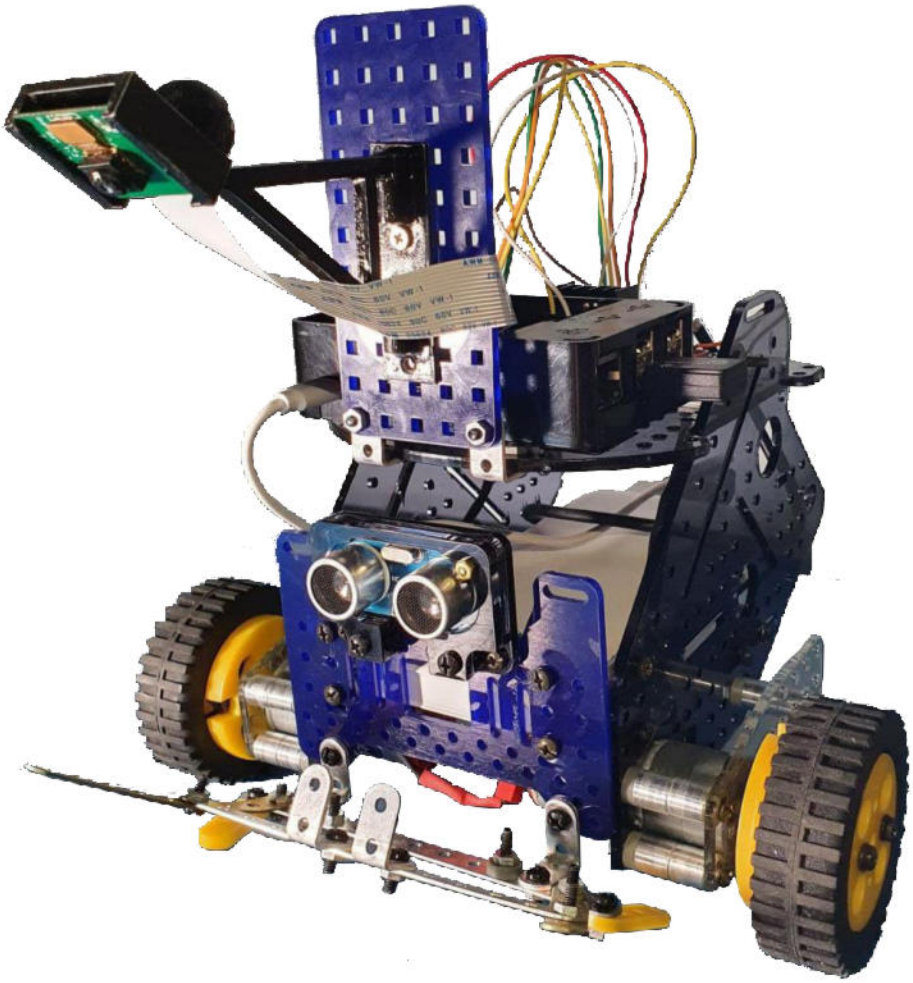

Figura 4.16 - Imagen lateral del robot. Debajo de la Raspberry Pi Camera se puede observar el sensor HC-SR04. Imagen de mi autoría.

4.2 OpenCV.

La librería OpenCV (*Open Source Computer Vision Library*) es una librería de código abierto multiplataforma que funciona en diferentes lenguajes de programación. Su objetivo principal es proporcionar las herramientas básicas necesarias para resolver problemas de visión por ordenador y el aprendizaje automático. Posee una gran colección de funciones relacionadas al procesamiento de imágenes, la visión por ordenador y el aprendizaje automático, como así también varias funciones de interfaz gráfica de usuario (GUI) y manejo de eventos. Su distribución y utilización es gratuita para el uso comercial y académico, ya que se encuentra bajo la licencia Berkley Software (BSD).

El código del proyecto se desarrolla, compila y prueba, primero, en una computadora para luego cargarlo en el robot. Esta metodología es mucho más rápida y práctica, dada las limitaciones de hardware de este último, que hacerlo directamente sobre el Raspbian (sistema operativo de la Raspberry Pi). Para probar cada algoritmo y/o técnica utilizada de la librería OpenCV se utilizaron varios videos de vehículos circulando en carreteras. Cada modificación hecha al código, primero se probó sobre uno de estos debido a su practicidad, lo que, además, permitió asegurarse de que funcionara correctamente según lo esperado. El principal de ellos, el más utilizado, se obtuvo de la web (sin fuente reconocible) y fue grabado con una cámara posicionada en la parte delantera del coche (desde del interior). Dura tan sólo 2 segundos, por lo que se repite constantemente para observar bien los resultados de cada ejecución del código. A continuación, en la figura 4.17, podemos observar una imagen sacada de ese video que utilizaremos, junto a otras, como referencia en las siguientes secciones para ver las diferentes aplicaciones de las técnicas y/o algoritmos de OpenCV.

Figura 4.17 – Imagen de mi autoría, tomada del video de carretera que se utiliza como test.

4.2.1 Espacios de colores.

Los espacios de color son una forma de representar los canales de color que se presentan en una imagen. El más conocido es el modelo RGB (Rojo, Verde, y Azul por sus siglas en inglés) pero, en este caso, el utilizado por la librería OpenCV es

algo distinto. Utiliza el modelo BGR, es decir, intercambia el canal Azul (B) por el Rojo (R) y la razón de ello, según los propios desarrolladores, es porque en aquel entonces (la librería tuvo su origen en 1999) dicho modelo era el más popular entre los fabricantes de cámaras y los proveedores de *software*. A continuación, en la figura 4.18, se puede observar cómo varían los colores dependiendo de las distintas variables.

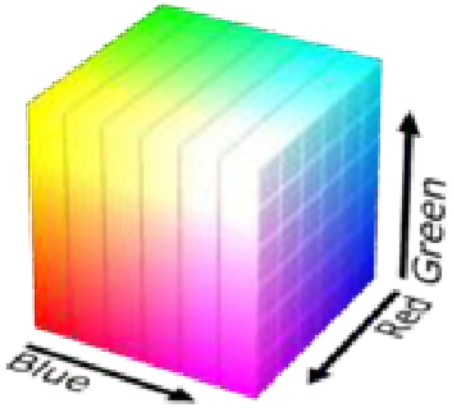

Figura 4.18 - Representación del espacio de color BGR. Extraída de la documentación oficial de OpenCV - https://docs.opencv.org/4.5.2/da/d97/tutorial_threshold_inRange.html

Para poder realizar una mejor tarea de procesamiento de imágenes, se realiza una conversión de la imagen del anterior espacio de colores a otro modelo que nos facilite la tarea. En este proyecto hemos utilizado dos y ambos, partiendo del modelo BGR que utiliza la librería.

El primero es el modelo HSV (este término hace referencia a sus siglas en inglés, *Hue, Saturation y Value*, es decir, Matiz, Saturación y Valor, respectivamente). El primer canal indica el color; el segundo, la variación de saturación u oscuridad, y el tercero, el brillo o la intensidad del color, variando cada uno de ellos de la siguiente manera: [H:0-179, S:0-255, V:0-255].

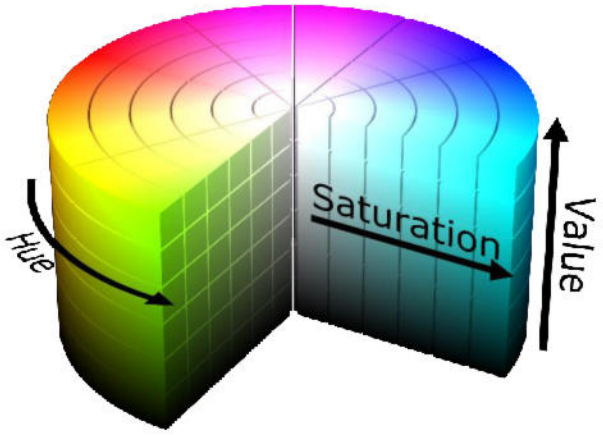

Figura 4.19 - Representación del espacio de color HSV. Extraída de la documentación oficial de OpenCV - https://docs.opencv.org/4.5.2/da/d97/tutorial_threshold_inRange.html

La razón de esta transformación radica en el primer canal (Hue o Matiz), que permite modelar el color fácilmente, simplificando el procesamiento de imágenes de aquellos que necesitan segmentar objetos en función de su color. En la figura 4.19, se puede ver la representación cilíndrica del espacio de color HSV a diferencia de la cúbica de BGR, mostrada anteriormente. Los colores en el espacio de color RGB se codifican utilizando los tres canales y, por lo tanto, es más difícil segmentar un objeto en la imagen basándose en su color.

Si tomamos la imagen de referencia que tenemos y le aplicamos este nuevo espacio de colores, obtendremos una nueva imagen transformada, como se puede ver en la figura 4.20.

Figura 4.20 - Imagen de referencia transformada al espacio de colores HSV. Imagen de mi autoría.

El otro modelo que utilizamos en este proyecto es el modelo HSL (*Hue, Saturation y Lightness,* es decir, Matiz, Saturación y Luminosidad, respectivamente). El primer canal representa los tres colores primarios (rojo, verde y azul); el segundo hace referencia a la cantidad de color (de tal manera que, si se pierde saturación, tenderá hacia el gris) y, por último, el tercero apunta a la cantidad de luz (por lo tanto, si la cantidad disminuye tenderá al negro mientras que, por el contrario, si aumenta tenderá al blanco).

El modelo HSL, comparado con el modelo HSV, permite detectar problemas de imagen debido a la iluminación, ya sean sombras, resplandor del sol, luces, etc. Es necesario eliminar todo ese ruido para facilitar la detección de las líneas del carril y este modelo es ideal ya que divide todos los colores en valores de tono, saturación y luminosidad. De esta forma se optó por trabajar con el modelo HSL para el desarrollo del presente proyecto.

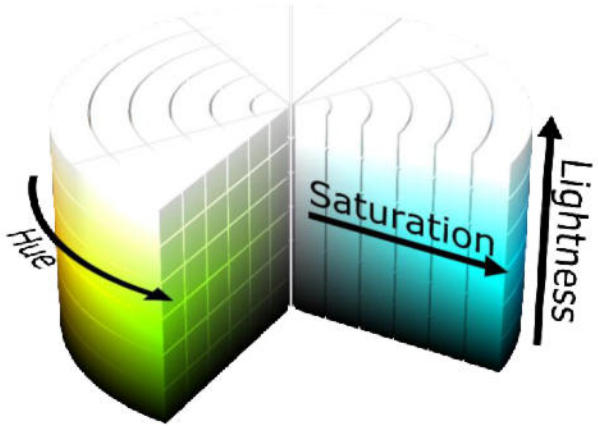

Figura 4.21 – Representación del espacio de color HSL. Extraída de https://en.wikipedia.org/wiki/HSL_and_HSV

A modo de visualización también se aplicó este espacio de colores sobre la imagen de referencia y su resultado se puede observar en la figura 4.22.

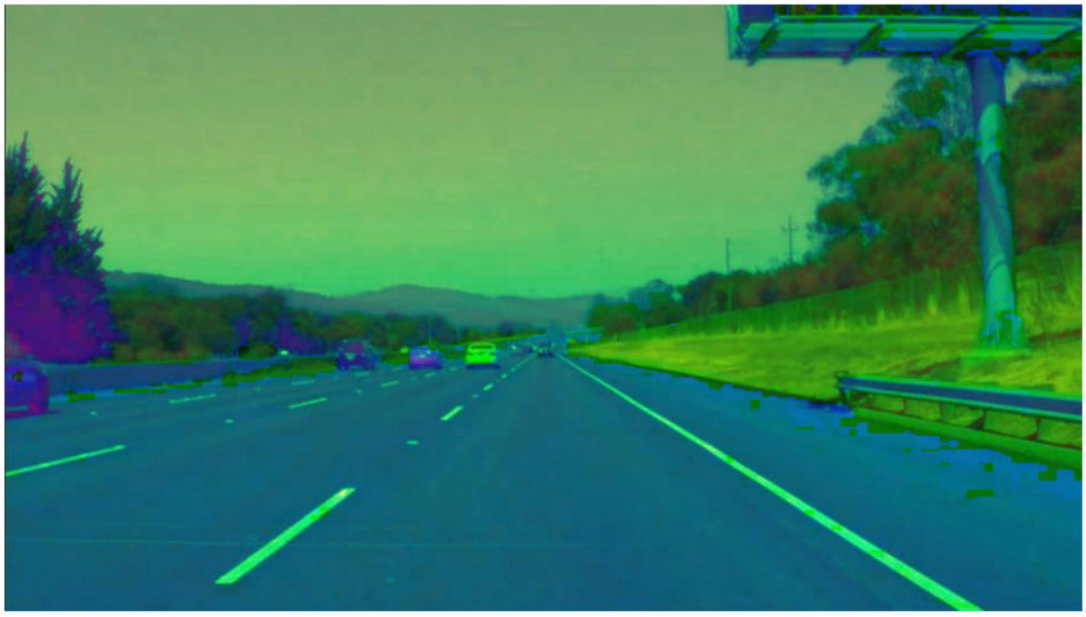

Figura 4.22 - Imagen de referencia transformada al espacio de colores HSL. Imagen de mi autoría

4.2.2 Filtro Blur Gaussiano

Como se mencionó previamente, muchas veces las imágenes que queremos procesar traen consigo ruido inherente de los dispositivos con los que fueron capturadas y/o efectos contraproducentes provocados por la iluminación. Para

contrarrestar estos efectos adversos hay diferentes técnicas para eliminar ese ruido: una de ellas es el método Gaussiano.

Este filtro se aplica con el objetivo de suavizar la imagen y hacerla más clara, eliminando los detalles. Funciona mediante la ponderación, siguiendo lógicamente la campana de Gauss y logrando así darle más importancia a los píxeles que se encuentran más cerca del centro que a los que están más alejados de éste. El resultado que se obtiene se podría ver como el desenfoque de una cámara fotográfica común. Se utiliza la campana de Gauss porque es una aproximación de cómo ve el ojo humano.

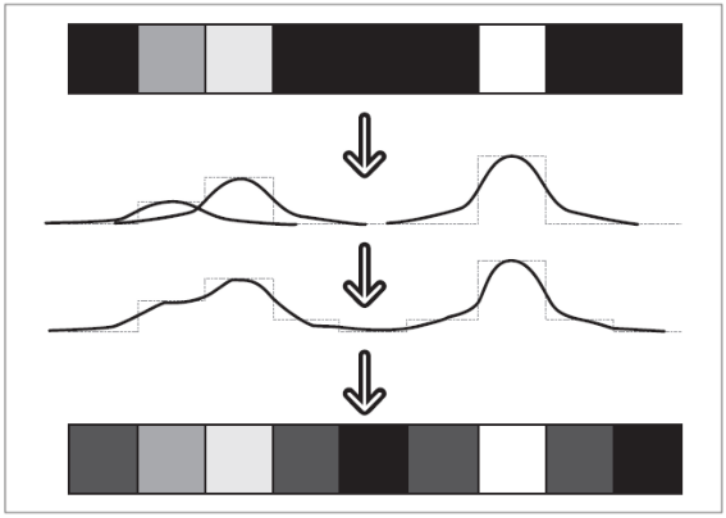

Figura 4.23 - Suavizado gaussiano sobre un array de píxeles en una dimensión. Extraído del libro *Learning OpenCV - Computer vision with the OpenCV library* (2008, pág. 113).

Para poder aplicar este suavizado debemos hacerlo en dos dimensiones; por eso es que se utiliza la convolución, una operación matemática que consiste en ir recorriendo píxel a píxel con una máscara o kernel de *NxN.* Dicho tamaño determina el número de píxeles con el que el algoritmo va a trabajar, y es muy importante que sea impar para siempre tener un píxel central que será el píxel en cuestión que estamos tratando. En nuestro caso, definimos que sea de 5x5.

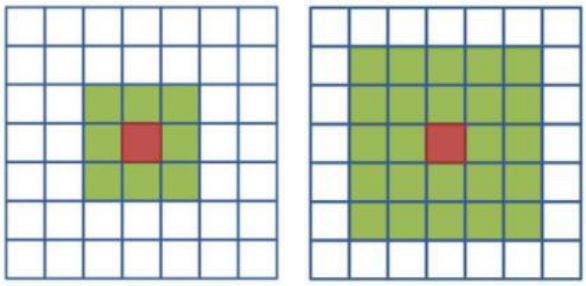

Figura 4.24 - Kernels impares, de 3x3 y 5x5 respectivamente. Al ser impares ambos cuentan con un pixel central. Imagen de mi autoría

Figura 4.25 - Suavizado gaussiano aplicado sobre la imagen de referencia, la diferencia es casi imperceptible. Imagen de mi autoría.

4.2.3 Operador Sobel

Utilizado para la detección de bordes, al igual que el filtro Gaussiano, emplea kernels y la operación matemática de convolución para dicha finalidad. Consiste en estimar la primera derivada de una imagen haciendo una convolución entre una imagen (entrada) y dos núcleos especiales:

1. Uno para aproximar el cambio de intensidad en un píxel en la dirección X (horizontal).
2. Uno para aproximar el cambio de intensidad en un píxel en la dirección Y (vertical).

Ambos se encuentran realizando convoluciones con cada píxel de la imagen original para identificar las regiones donde el cambio de gradiente de la intensidad se maximiza en magnitud de las direcciones X e Y. Así, en cada uno de esos puntos, este operador da la magnitud del mayor cambio posible, evidenciando cómo abrupta o suavemente cambia una imagen en cada píxel y, por consiguiente, cuán probable es que este represente un borde en la imagen.

Matemáticamente, el operador Sobel utiliza dos kernels de 3x3 elementos para aplicar convolución a la imagen original y poder calcular aproximaciones a las derivadas. Como se mencionó anteriormente, se utiliza uno para los cambios horizontales y otro, para los verticales. Si definimos A como la imagen original, el resultado, dos imágenes G_x y G_y que representan para cada punto las aproximaciones horizontal y vertical de las derivadas de intensidades, es calculado mediante:

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{y} \quad \mathbf{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$

En cada punto de la imagen, los resultados de las aproximaciones de los gradientes horizontal y vertical pueden ser combinados para obtener la magnitud del gradiente, a través de:

$$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$

Se puede también así calcular la dirección del gradiente:

$$\mathbf{\Theta} = \operatorname{atan2}(\mathbf{G_y}, \mathbf{G_x})$$

Donde, por ejemplo, Θ es cero para bordes verticales con puntos más oscuros en el lado izquierdo.

Al utilizarlo sobre la imagen de referencia con los métodos provistos por la librería OpenCV, podemos obtener todos los bordes de la misma. El resultado se puede apreciar en la figura 4.26.

Figura 4.26 - Operador Sobel aplicado sobre la imagen de referencia. Imagen de mi autoría

4.2.4 Algoritmo de Canny.

Originalmente creado en 1986 por John F. Canny, esta técnica tiene como objetivo principal detectar todos los bordes existentes en una fotografía. Para ello, basa su funcionamiento en la utilización de máscaras de convolución y en la primera derivada, buscando puntos de contorno en la imagen, los cuales son zonas de píxeles en las que existe un cambio brusco de nivel de gris.

La propuesta original de Canny en 1986 tiene tres principales criterios como base:

- Un criterio de detección que busca evitar la eliminación de bordes importantes y no suministrar falsos bordes.
- Un criterio de localización que busca minimizar la distancia entre la posición real y la localizada del borde.
- Un criterio de una respuesta que busca integrar las respuestas múltiples correspondientes a un único borde.

Anteriormente se mencionó que este algoritmo utiliza la primera derivada para buscar bordes. Esto se consigue mediante la búsqueda de aquellas regiones en las que se obtiene el valor de cero, es decir, en aquellas en donde no varía la intensidad y tiene un valor constante en toda la transición de intensidad. Un borde o, mejor

dicho, un cambio de intensidad se manifiesta como un cambio brusco en la primera derivada, y es aquí donde se lo detecta.
La manera en la que el algoritmo actúa está representada por tres grandes pasos:

- Obtención del gradiente: se calcula la magnitud y orientación del vector gradiente en cada píxel.
- Supresión no máxima: aquí se busca adelgazar el ancho de los bordes obtenidos en el paso anterior hasta lograr bordes de un píxel de ancho.
- Histéresis de umbral: se aplica una función de histéresis basada en dos umbrales, buscando reducir la posibilidad de que aparezcan contornos falsos.

La librería OpenCV nos provee métodos para aplicarlo sobre alguna imagen: en este caso se utilizó la de referencia para visualizar todos los bordes que se encuentren en ella (figura 4.27).

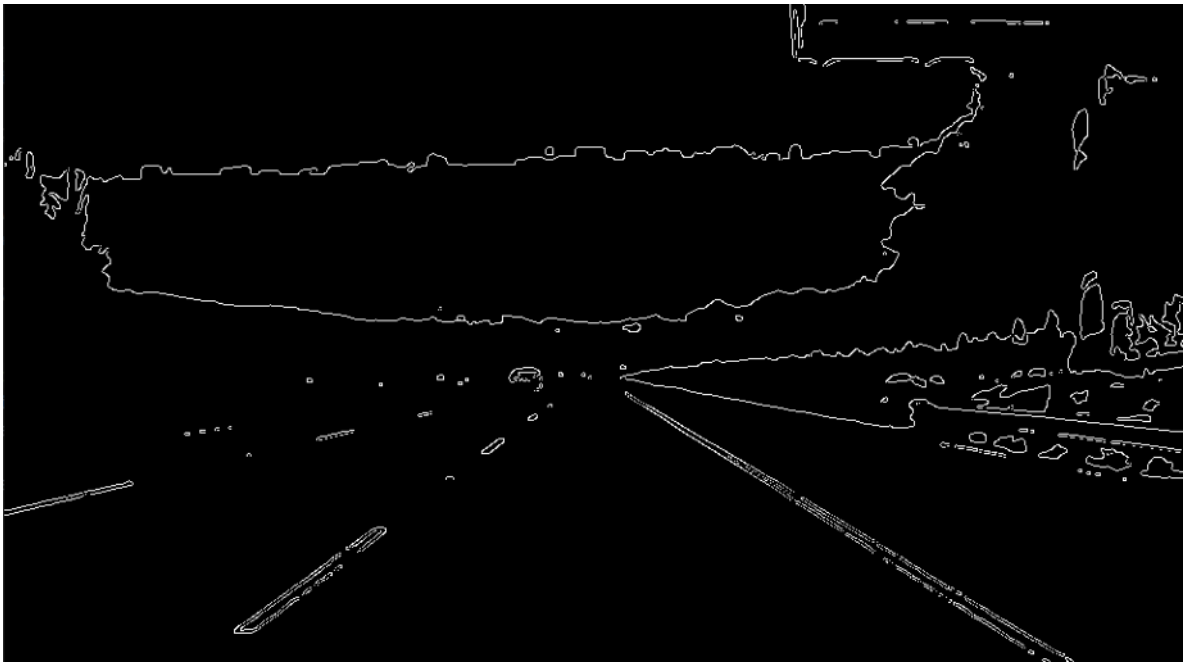

Figura 4.27 - Algoritmo de Canny sobre la imagen de referencia. Imagen de mi autoría

4.2.5 Transformada de Hough

Es una técnica que permite descubrir líneas, círculos u otras simples formas en una imagen. Se basa en transformar puntos de la imagen en un espacio de parámetros; cualquier punto de una fotografía binaria puede formar parte de algún conjunto de

líneas posibles. Si se parametriza cada línea con, por ejemplo, una pendiente *a* y un interceptor b, entonces un punto de la imagen original se transforma en un lugar de puntos en el plano *(a,b)*, correspondiente a todas las líneas que pasan por ese punto. Un ejemplo de esto puede verse en la figura 5.28, que muestra cómo se emplea dicha técnica y en la que se encuentran muchas líneas en cada imagen, aunque no todas sean las deseadas.

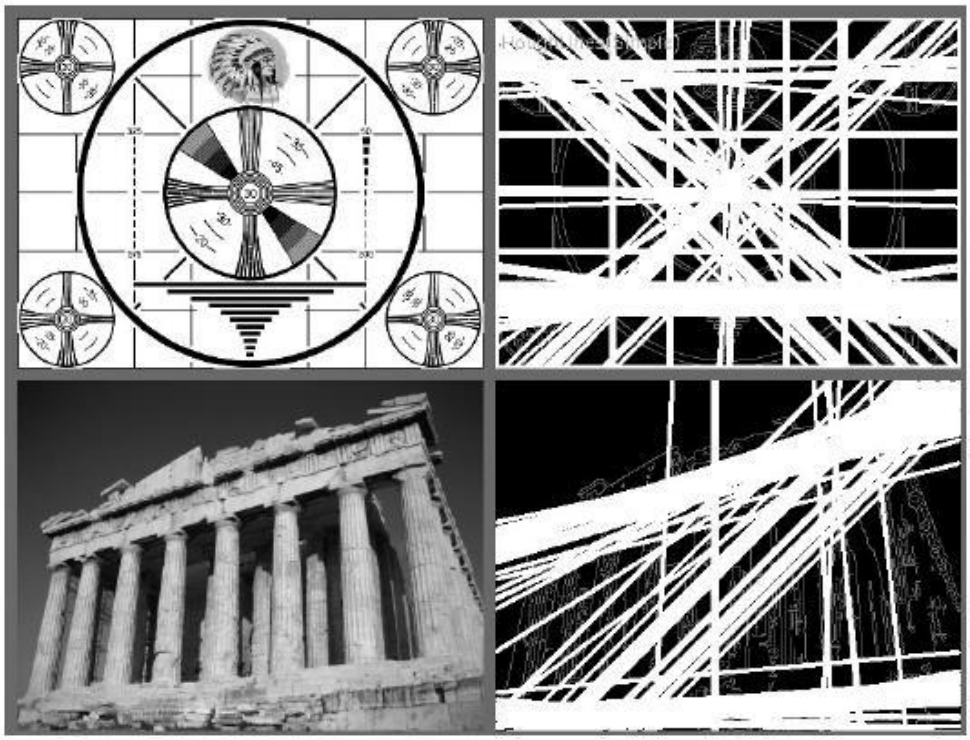

Figura 4.28 - Gráfico sobre la aplicación de la transformada de Hough. Extraído del libro *Learning OpenCV - Computer vision with the OpenCV library* (2008, pág. 155).

Si sumamos todos los píxeles no nulos de la imagen de entrada, convertidos en un conjunto de puntos en la imagen de salida, las líneas que aparecen en la imagen de entrada (plano *(x,y)*) aparecerán como máximos locales en la imagen de salida (plano *(a,b)*): este último es llamado, comúnmente, “plano acumulador”.

Dicha operación sería la elegida, si no se tuviera en cuenta la densidad considerablemente diferente de líneas en función de la pendiente y el hecho de que el intervalo de posibles pendientes va de -∞ a +∞. Por lo tanto, la parametrización real de la imagen de transformación que termina siendo utilizada es aquella que considera cada línea como un punto de coordenadas polares *(ρ, θ)*, siendo la línea

implícita la que pasa por el punto indicado perpendicular al radial, desde el origen hasta ese punto (ver la figura 4.29). La ecuación final de esa recta es:

$$\rho = x\ cos\ cos\ (\theta)\ + y\ sin\ sin\ (\theta)$$

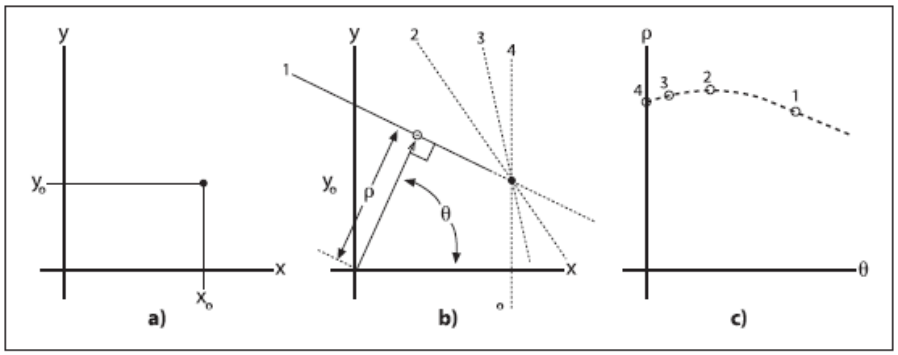

Figura 4.29 – Gráfico del proceso de la transformada de Hough. Imagen extraída de la web (sin fuente reconocible)

4.2.6 Región de interés.

Muchas veces cuando se trabaja con visión por computadora es necesario definir una región de interés (ROI por sus siglas en inglés), ya que hay muchas zonas de la imagen o video que realmente no nos interesa para nuestra tarea u objetivo y, si las tuviéramos en cuenta a la hora de realizar el procesamiento de los datos, no harían más que desperdiciar nuestro poder de cómputo con información innecesaria para nosotros. Además de ahorrar recursos, logramos descartar mucho ruido y un gran número de falsos positivos, reduciendo así una gran cantidad de errores de nuestro sistema.

Para poder aplicarlo en nuestro proyecto, primero, es necesario obtener las dimensiones (expresadas en píxeles) que tendrán nuestras imágenes. En nuestro caso, son imágenes en vivo tomadas por la Raspberry Pi Camera que tiene montada el robot, la cual está configurada para obtener fotografías con una dimensión de 720x480 píxeles (ancho x alto), pero para ejemplificar y explicar esta sección utilizaremos la misma foto de referencia que venimos utilizando, la cual posee una dimensión de 960x540 píxeles (figura 4.17).

Una vez que tenemos el tamaño de la imagen podemos pasar a calcular el área de nuestra región de interés; para ello, es útil utilizar una librería de Python llamada Matplotlib, más precisamente su módulo Pyplot, que nos permite dibujar ejes

cartesianos sobre una imagen (ver figura 4.30) y, de esta manera, poder determinar de una manera mucho más precisa el tamaño que debe tener nuestro ROI.

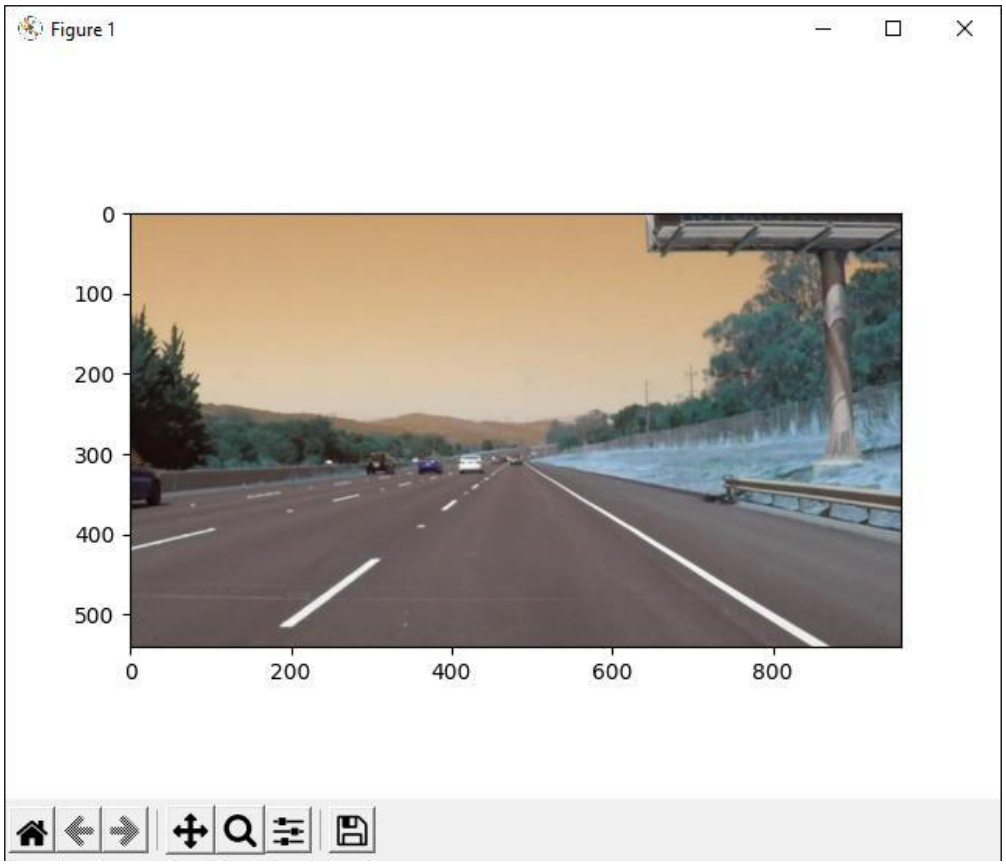

Figura 4.30 - Ejes cartesianos sobre la imagen de referencia, realizada con la librería Matplotlib (módulo Pyplot). Imagen de mi autoría

Teniendo en cuenta el área que realmente queremos analizar junto con los ejes cartesianos, que tienen su origen en la esquina superior izquierda, podemos determinar el área que tendrá nuestra región de interés. En el caso de la imagen de referencia sólo necesitamos cubrir todo el carril actual por donde circula el coche. Por lo tanto, con formar un trapecio que tenga los orígenes en los extremos inferiores y los restantes exactamente a la altura del centro de la imagen estamos cubriendo toda el área que necesitamos.

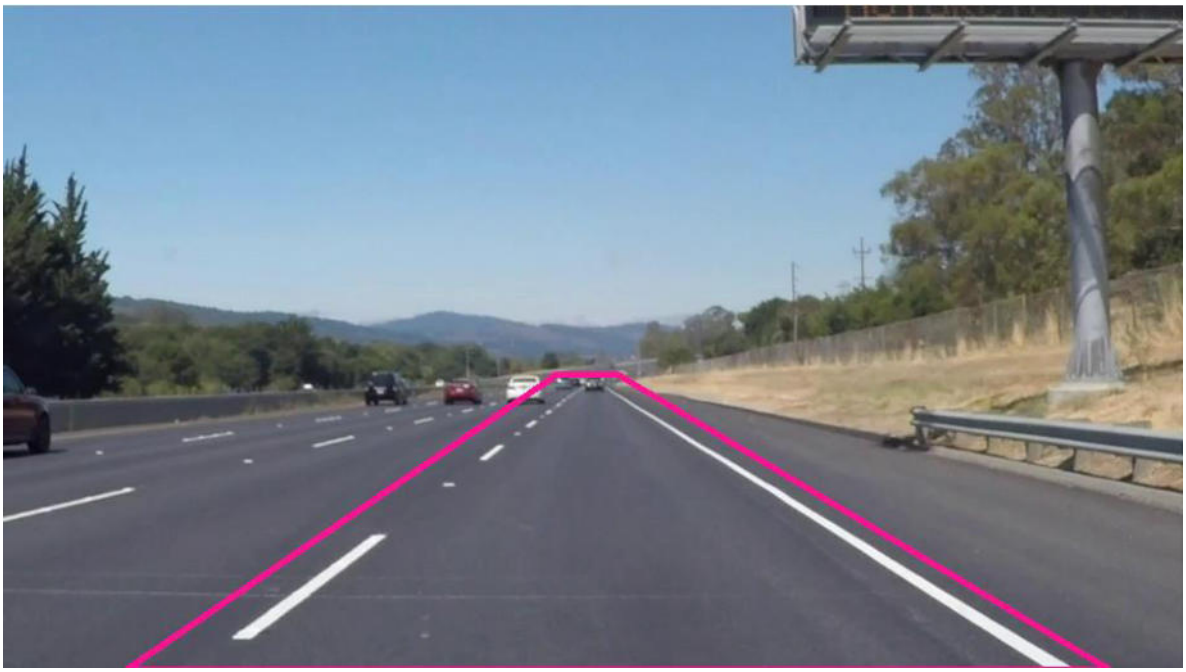

Figura 4.31 – Imagen de referencia con el área de ROI definida. Imagen de mi autoría

En la figura 4.31 se puede observar el contorno que contiene a toda la región de interés. En base a este se construirá una máscara que tendrá exactamente el mismo tamaño que la imagen original, con la particularidad de que la intensidad de cada pixel de la misma se lo transformará a cero, salvo el área del ROI que se mantiene sin modificación. Esto se hace con la finalidad de luego unirla a la imagen original y así sólo analizar dicha región ya que, cuando posteriormente OpenCV procese la imagen, se va a encontrar con todos píxeles que tienen intensidad cero y, por lo tanto, sin información que le permita encontrar bordes en esas zonas de la imagen. Esto se puede apreciar con más claridad en la figura 4.32.

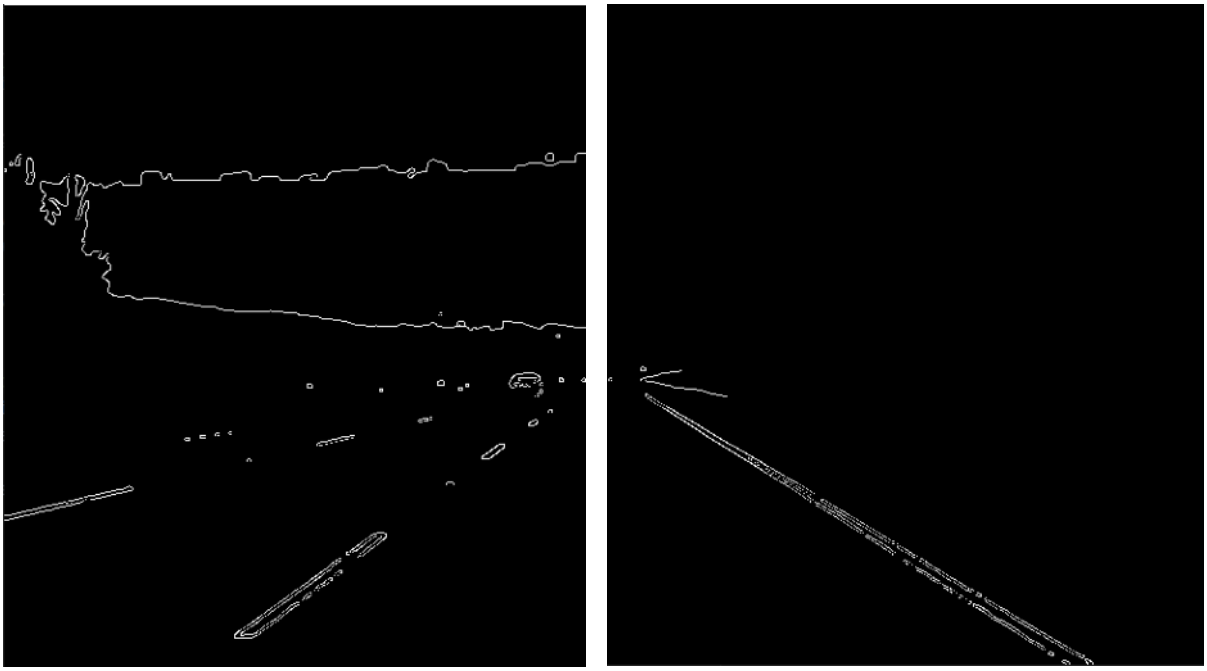

Figura 4.32 - Imagen procesada y detectados todos los bordes blancos. A la izquierda completa y a la derecha con ROI. Imagen de mi autoría.

4.2.7 Transformación de perspectiva.

Las imágenes obtenidas desde una cámara montada al frente del automóvil, como la de las imágenes de referencia, poseen una perspectiva que dificulta algunos cálculos necesarios para el proyecto. Uno de ellos es el cálculo del radio de curvatura del carril, el cual nos permite saber en qué dirección gira la vía. Desde este punto de vista, las líneas de los carriles tienen una forma similar a un trapecio y el ancho del carril parece disminuir a medida que se aleja del automóvil, haciendo que dicho cálculo no se pueda aplicar correctamente, ya que lo que estamos observando no es una representación precisa de lo que sucede realmente. Es por ello que se necesita transformar la perspectiva a una que nos permita obtener datos más exactos de la ruta, más precisamente a la llamada "vista de pájaro", o como se la conoce en inglés "*bird's eye view".* Este nuevo punto de vista nos permite observar mucho mejor las líneas de los carriles ya que, como su nombre lo indica, se examina la ruta desde la posición de un ave. Es decir, desde un punto más alto que permite ver a ambos carriles como líneas paralelas.

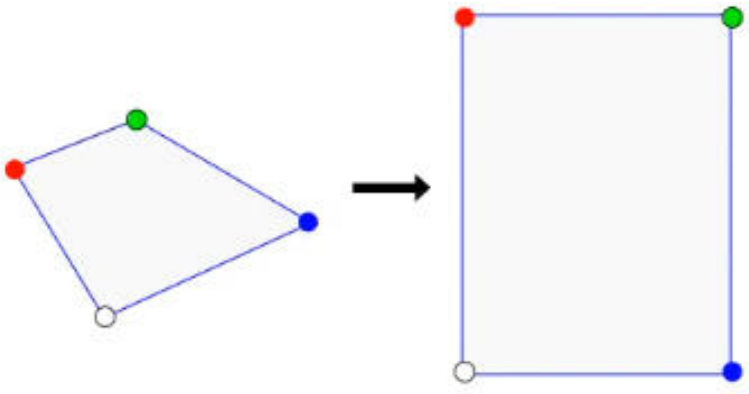

Figura 4.33 - Transformación de perspectiva, de una frontal (trapezoidal) a una vista de pájaro (rectangular). Imagen de mi autoría

La librería OpenCV facilita esta tarea ya que provee métodos que realizan esta transformación, de la perspectiva trapezoidal a una rectangular (ver figura 4.33). Al igual que con el ROI, se necesita definir los cuatro puntos externos para definir el área de la nueva imagen con perspectiva rectangular. Para visibilizar esta transición se escoge otro video de prueba del proyecto y se determinan sus correspondientes vértices como puede observarse en la figura 4.34.

Figura 4.34 - Imagen de referencia de otro video de prueba con los puntos de la nueva perspectiva definidos. Imagen de mi autoría

Después se procede a realizar el cambio de perspectiva y lo que se obtiene puede apreciarse en la figura 4.35: en ella se puede observar el carril amarillo a la izquierda

y el carril blanco a la derecha. Ambos con claridad, dado que, además de la transformación de perspectiva, la imagen ya se encuentra procesada para delimitar los bordes.

Figura 4.35 - Imagen procesada y deformada, perspectiva rectangular o vista de pájaro. Imagen de mi autoría

5. DESARROLLO.

En esta sección, se encuentra descrito detalladamente todo el proceso que se necesitó para implementar el sistema móvil guiado mediante visión artificial, desde los planteamientos originales, las dificultades encontradas a lo largo del mismo y las correspondientes decisiones que se tomaron para sortear dichos inconvenientes. Todo el proyecto fue documentado a medida que se iba desarrollando para posteriormente poder describirlo con mayor precisión.

Para una mejor comprensión, la descripción del proyecto se hará en dos partes. La primera abarcará el procesamiento de imágenes y visión artificial, cómo se procesan los distintos fotogramas de un video para obtener información relevante que nos ayude a guiar a un robot dentro de un entorno determinado. En tanto en la segunda comprenderá la utilización de esos datos, obtenidos en la etapa anterior, para entregarle instrucciones concretas de movimiento a un robot móvil para que pueda moverse de forma autónoma en un entorno determinado. De igual forma se detallará todo el hardware que compone al robot en sí y cómo se logró armarlo. Esta división no respeta el orden en que sucedieron ya que ambas se desarrollaron de forma casi paralela.

5.1 VISIÓN ARTIFICIAL.

Para desarrollar el software del presente proyecto se contó con las siguientes tecnologías y herramientas:

- Windows 10: sistema operativo en el cual se estableció el espacio de trabajo principal.
- Python 3.9[1]: lenguaje de programación de alto nivel utilizado para la construcción del programa del presente proyecto.
- PIP: manejador de paquetes de Python utilizado para instalar distintas librerías y dependencias.

[1] Página oficial de Python: https://www.python.org/

- PyCharm Community[2]: entorno de desarrollo integrado (Integrated Development Environment, IDE por sus siglas en inglés) utilizado para desarrollar todo el software de la aplicación ya que el mismo está especialmente diseñado para trabajar con el lenguaje de programación Python.
- GitHub[3]: plataforma de desarrollo colaborativo para utilizar el control de versiones de Git. En nuestro caso utilizado para mantener actualizado el código entre el robot (Raspberry Pi) y la computadora con Windows 10.
- OpenCV[4]: librería principal encargada del procesamiento de imágenes y utilizada para emplear visión artificial.
- NumPy: librería que ofrece la posibilidad de crear vectores y matrices multidimensionales, como así también una gran colección de funciones matemáticas de alto nivel.
- Matplotlib: librería especializada en la creación de gráficos de dos dimensiones. Permite crear y personalizar los gráficos más comunes como diagrama de barras, histogramas, diagramas de sectores, mapas de color, etc.
- Time: librería para Python que contiene una serie de funciones relacionadas con la medición del tiempo.
- Socket: librería que proporciona acceso a una interfaz de Socket BSD (implementación en Berkeley Software Distribution), la cual nos permite establecer una conexión TCP/IP entre la Raspberry Pi y la computadora con Windows 10.

5.1.1 Procesamiento de imágenes

Lo primero a realizar para poder procesar las imágenes del video es cambiar el espacio de colores en las que se encuentran, por defecto en OpenCV vienen en BGR por lo que necesitamos pasarlo a otro para resaltar o disminuir ciertos aspectos de la imagen que resulten más valiosos para su posterior análisis y procesamiento. En un principio se hizo uso del espacio de colores HSV dado que simplifica dicha

[2] Página oficial de PyCharm: https://www.jetbrains.com/es-es/pycharm/
[3] Página oficial de GitHub: https://github.com/
[4] Página oficial de OpenCV: https://opencv.org/

tarea segmentando objetos en función de su color, algo valioso para las posteriores aplicaciones, el algoritmo de Canny y la transformada de Hough. El primero se encarga de detectar los bordes en la imagen y el segundo de buscar todos aquellos puntos que puedan representar una línea. Al momento de realizar algunas pruebas preliminares con la cámara móvil se notó que el robot tenía muchas dificultades para detectar correctamente las líneas del carril, inclusive con una luz blanca alumbrando a donde apuntaba la cámara. Por lo tanto, se decidió buscar otra alternativa para poder obtener mejores resultados a la hora de detectar las líneas de los carriles, la cual se detalla a continuación.

Al igual que con el anterior filtro, el primer paso es representar la imagen (frame del video) en otro espacio de colores diferente al BGR. En este caso lo transformamos al espacio de colores HLS (utilizando funciones de OpenCV) ya que funciona mucho mejor para la detección de problemas en la imagen debido a la iluminación, ya sean sombras, resplandores del sol o de las luces, etc. Luego de esto, generamos varios umbrales binarios:

El primero, lo generamos a partir de la imagen HLS para quedarnos con los colores más intensos de la imagen. Será un umbral que contenga puras intensidades de blancos (ceros) y negros (255) ya que aquellos píxeles de la imagen relativamente claros se convertirán en blancos y aquellos oscuros en negros, de esta forma podemos aplicar el operador Sobel sobre este nuevo umbral mucho más definido.

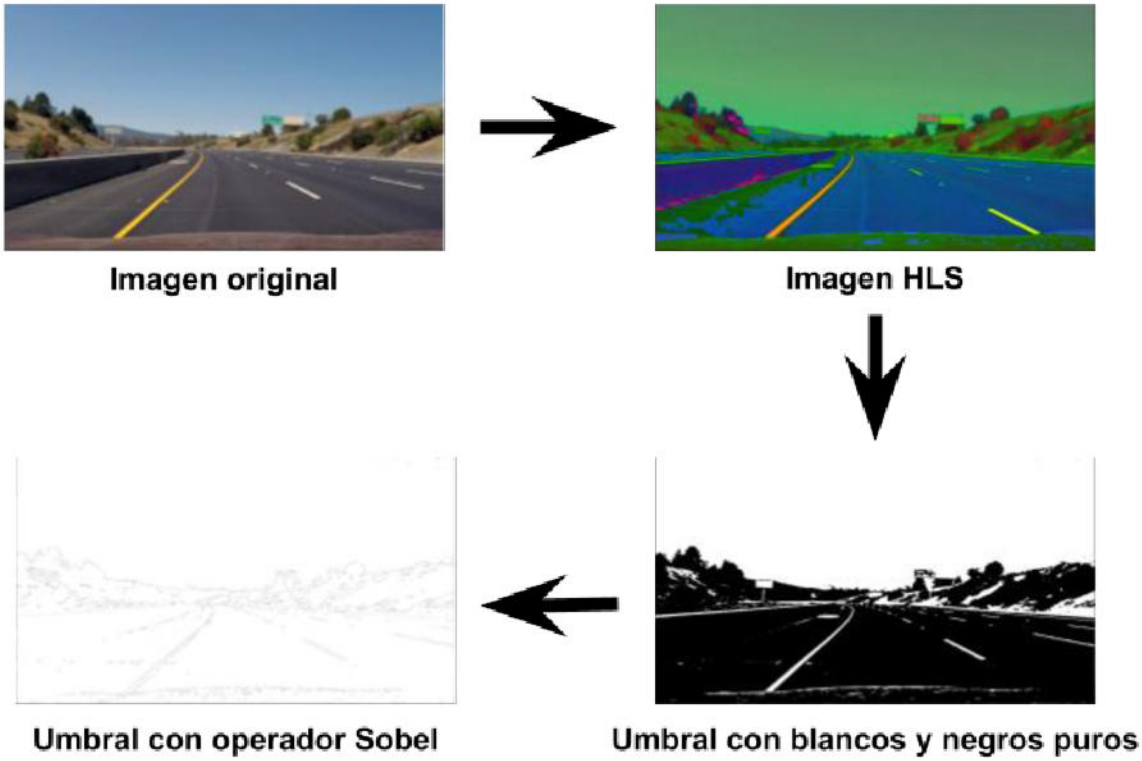

Figura 5.1 - Diagrama de creación del primer umbral. Imagen de mi autoría

El segundo, lo generamos a partir del canal S (saturación) de la imagen HLS con la intención de eliminar todos los colores apagados de la carretera, esperando que los colores de los carriles (amarillos y/o blancos), por el contrario, sean sólidos y puros. Un valor de saturación alto significa que el color del tono es puro. En este nuevo umbral vamos a obtener una imagen llena de valores de intensidades de la imagen, 0 para los negros y 255 para los blancos.

Finalmente, le aplicamos a este nuevo umbral una difusión de filtro gaussiano con el fin de suavizarlo y hacerlo aún más claro eliminando los detalles.

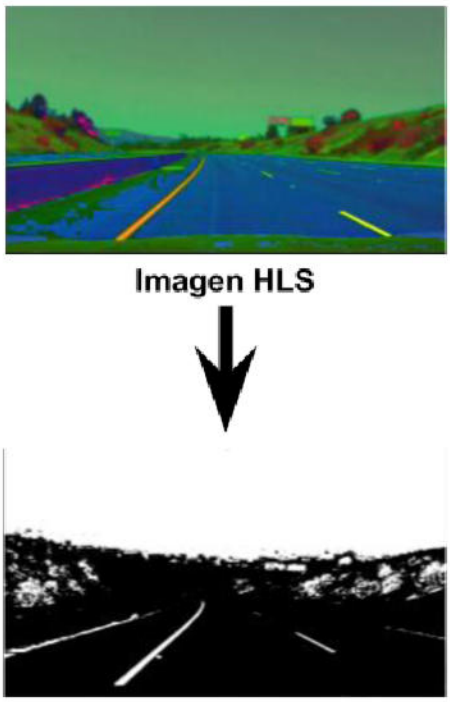

Figura 5.2 - Esquema de la generación del umbral sobre el canal S de la imagen HLS. Imagen de mi autoría.

El tercero, lo generamos sobre el canal rojo del canal BGR (R, tercer canal) del frame original con la intención de quedarnos con los carriles de la carretera en la imagen ya que sus colores poseen valores de rojos puros: el blanco [255, 255, 255] y el amarillo [0, 255, 255]. En este nuevo umbral aquellos que cuenten con una intensidad mayor a 120 se los establecen en blanco mientras que todos los demás en negro.

Figura 5.3 - Esquema de generación del umbral sobre canal R de la imagen original. Imagen de mi autoría.

Una vez creados los tres umbrales binarios procederemos a aplicar operaciones para combinar los umbrales y obtener como resultado el umbral definitivo que posee toda la información necesaria para detectar la posición del vehículo (robot) con respecto a la posición de los carriles de la carretera.

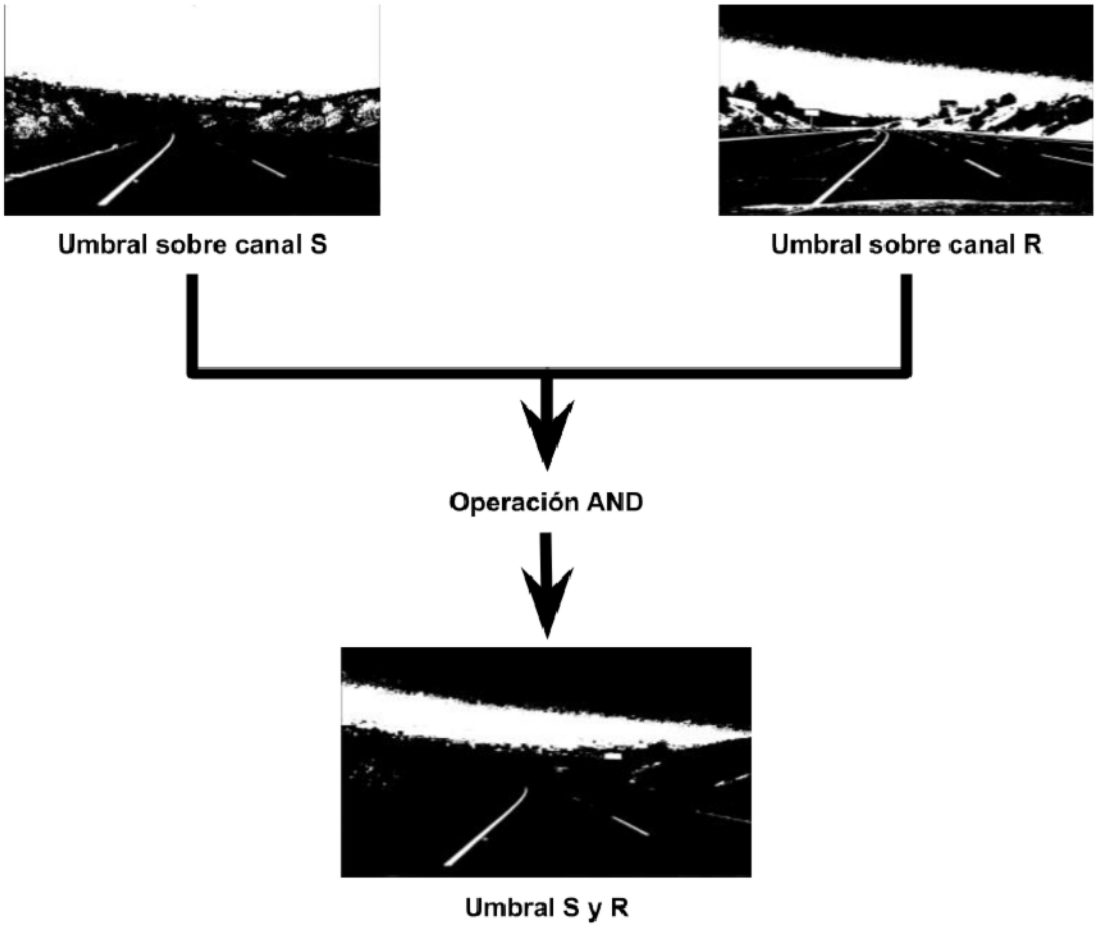

Figura 5.4 - Operación AND bit a bit entre el segundo umbral (sobre canal S) y el tercer umbral (sobre canal R). Imagen de mi autoría.

La primera que ejecutamos es una operación AND bit a bit entre el segundo y el tercer umbral, aquellos que se aplican sobre sobre el canal S de la imagen HLS y sobre el canal R de la imagen original respectivamente. Dicha operación binaria toma dos representaciones binarias de igual longitud y realiza la operación AND lógica en cada par de los bits correspondientes, lo que equivale a multiplicarlos. Por lo tanto, si ambos bits en la posición comparada son 1 la representación binaria resultante es 1 (1 x 1 = 1), por el contrario, el resultado es 0 (1 x 0 = 0 y 0 x 0 = 0).

ENTRADA A	ENTRADA B	RESULTADO
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 5 - Tabla de verdad de operación lógica AND

De esta operación obtenemos como resultado un nuevo umbral que contiene toda la información presente en ambos umbrales, es decir, una imagen con menos ruidos y sin píxeles que no parecen ser colores sólidos, puros y agradables (como las líneas de carriles blancas o amarillas). Finalmente, al umbral resultante le aplicamos otra operación AND con el primer umbral al que se le aplicó el operador Sobel para obtener el umbral resultante final.

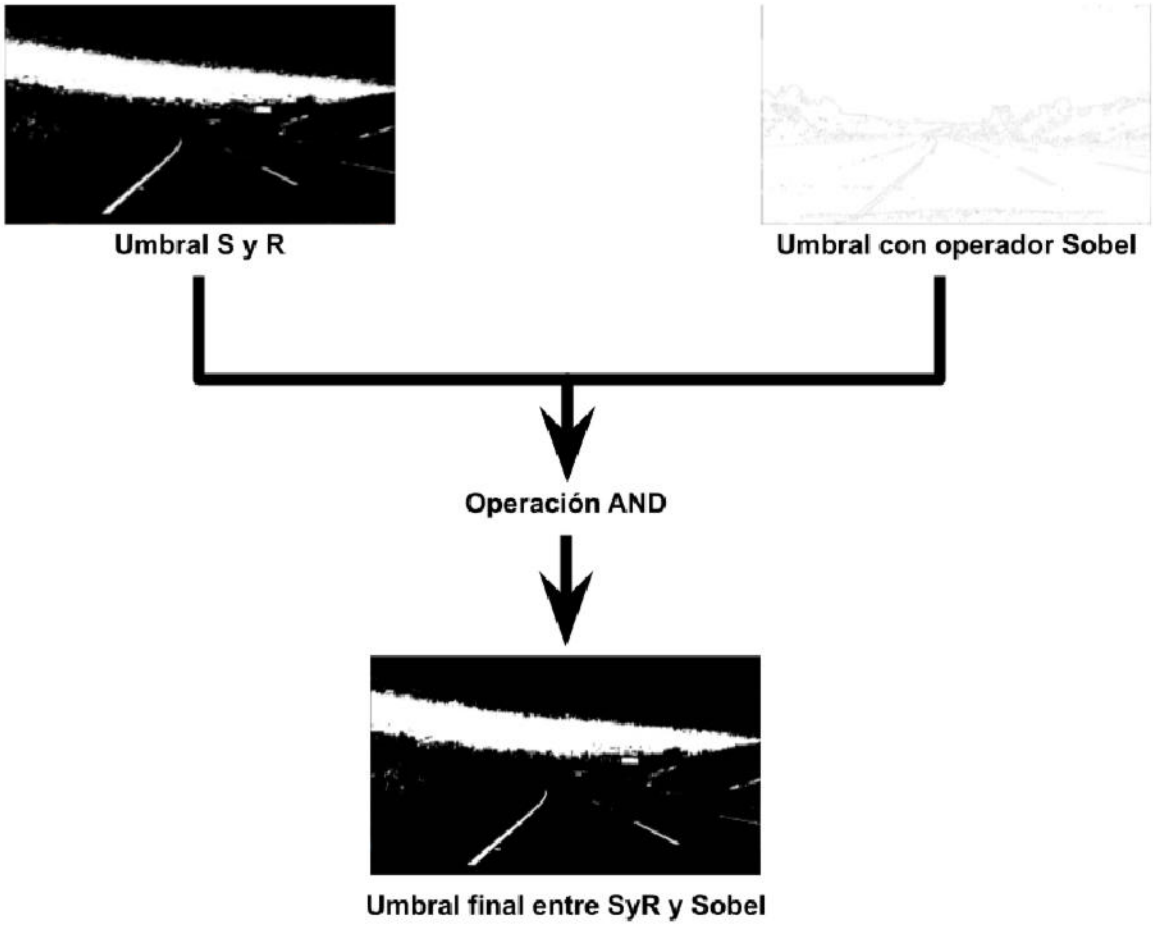

Figura 5.5 – Operación AND bit a bit entre el umbral resultante de la operación anterior y el umbral con operador Sobel. Imagen de mi autoría.

5.1.2 Transformación de perspectiva.

Una vez que tenemos el umbral con toda la información contenida necesaria para analizarla y procesarla procedemos a deformar la perspectiva de la cámara de esas imágenes a una vista de pájaro, o *Bird's-eye view* por su nombre en inglés. Lo primero que debemos hacer es identificar la región de interés (*Region of Interest*, ROI) de la imagen, esto nos ayuda a enfocarnos solamente en la información que queremos estudiar y procesar. En este caso nos centramos en cubrir sólo el área de la carretera porque nos interesa la posición de los carriles ya que son los que nos indican la dirección que debe tomar el robot, lo demás es irrelevante para tomar esas decisiones de direccionamiento. En otras palabras, para nuestro algoritmo lo que se encuentra fuera de las dimensiones de las carreteras es sólo ruido y, si lo tuviéramos en cuenta para el procesamiento, estaríamos desperdiciando poder de cómputo.

Cada video (pregrabado o en vivo) posee diferentes resoluciones de tamaño y, por lo tanto, cada uno de ellos posee diferentes regiones de interés. Inclusive un mismo video puede tener diferentes ROIs si se llegase a mover la dirección hacia donde apunta la cámara, ya sea inclinándola más hacia abajo o hacia arriba. Para el caso del video pregrabado definimos el área teniendo en cuenta la resolución del mismo y dónde se encuentra la zona de la carretera (proceso detallado en la sección 1.7.6).

El ROI resultante se muestra a continuación en la figura 5.6:

Figura 5.6 - Región de interés definida para el video pregrabado. Imagen de mi autoría.

Una vez que lo tenemos definido pasamos a aplicarlo generando una matriz de transformación que se crea con los puntos del ROI actuales y con unos nuevos deseados que se aplican sobre la imagen con la perspectiva cambiada como se detalla en la siguiente figura (ver figura 5.7). Con dicha matriz realizamos una transformación de la perspectiva de la imagen mediante métodos que nos provee la librería OpenCV y así obtenemos la imagen resultante que se puede ver en la figura 4.35 del presente informe.

Figura 5.7 - Área del ROI de la imagen deformada con la perspectiva de vista de pájaro. Imagen de mi autoría.

5.1.3 Identificación de los píxeles de las líneas de los carriles.

Una vez que contamos con la imagen deformada podemos proceder a analizarla en profundidad para conocer la posición de las líneas de los carriles. Si observamos con atención la imagen resultante (ver figura 4.35) podemos ver que los píxeles blancos representan la línea de los carriles o partes de ellos por lo que, podemos generar un histograma para ubicar esas áreas de la imagen que posean altas concentraciones de píxeles blancos y así conocer la ubicación de los carriles. Idealmente cuando generemos el histograma éste contará con dos picos o máximos, uno izquierdo y otro derecho, correspondientes a la línea del carril izquierdo y al carril derecho, respectivamente. Gracias a la librería matplotlib de Python podemos crearlo de una forma sencilla y observar con mejor claridad cuáles áreas está detectando, el resultado gráfico se puede observar en la siguiente figura 5.8.

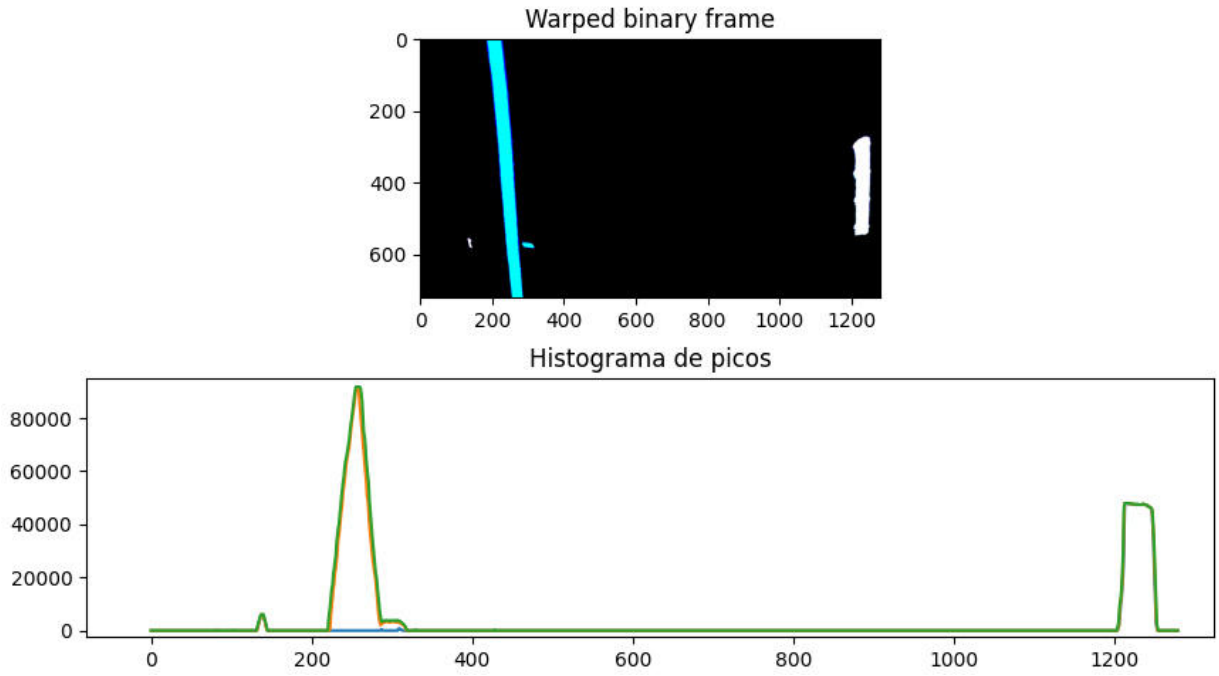

Figura 5.8 - Histograma de picos o máximos para la identificación de las líneas de los carriles. Imagen de mi autoría.

5.1.4 Ventanas deslizantes.

El próximo paso es utilizar la técnica de ventanas deslizantes (o S*liding windows* en inglés), la cual consiste en comenzar a escanear la imagen desde la parte inferior de la imagen hasta la parte superior de la misma. A partir de la posición inicial, la primera ventana mide cuántos píxeles no nulos (blancos) se encuentran dentro de la ventana, si esa cantidad supera cierto umbral desplaza la siguiente ventana a la posición lateral media de los píxeles detectados, en otras palabras, se recentra la siguiente ventana en la nueva posición media. En caso contrario, si no se detectan los suficientes píxeles comienza la siguiente ventana en la misma posición lateral. Esto puede observarse con mayor claridad en la primera imagen de la figura 5.9: las ventanas deslizantes siempre se centran en la posición donde existe la mayor cantidad de píxeles detectados (carril izquierdo) y, en caso de no hallar la suficiente cantidad de píxeles (carril derecho), se colocan en la posición media.

Una vez identificados los píxeles que corresponden a las líneas del carril izquierdo y derecho, dibujamos una línea polinomial de mejor ajuste a través de los píxeles, la cual representa la mejor estimación de las líneas de carril.

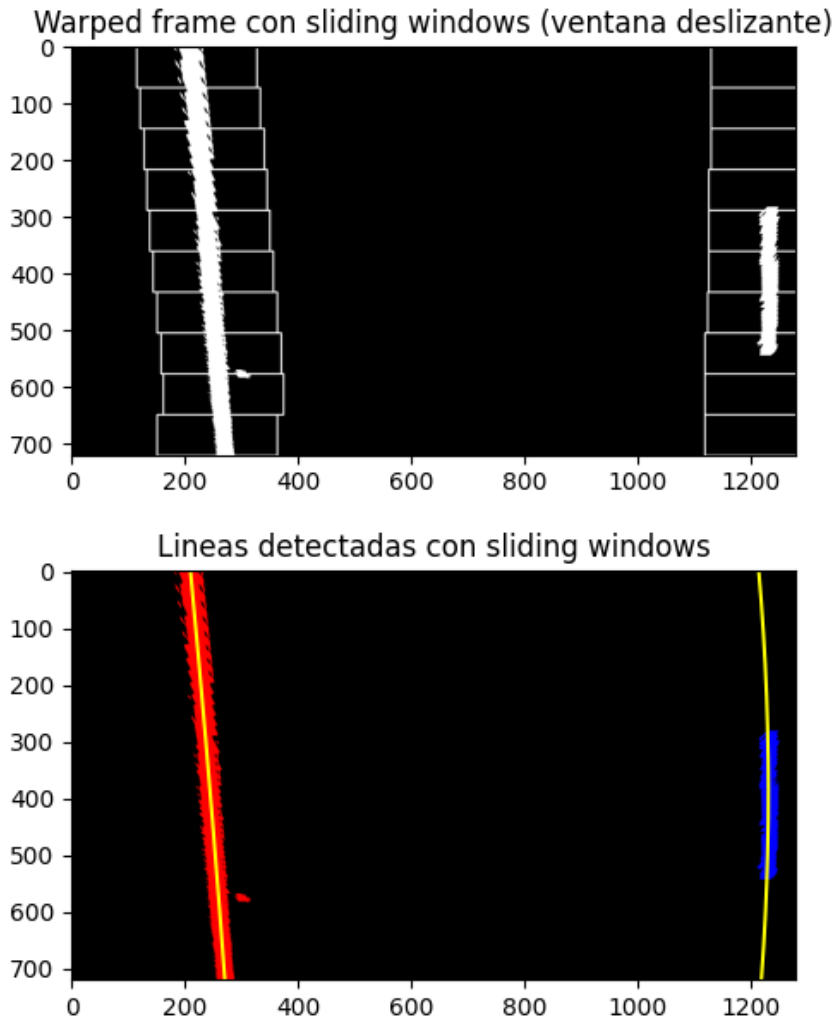

Figura 5.9 - Procesamiento de las líneas del carril con la técnica de ventanas deslizantes. Imagen de mi autoría.

5.1.5 Superposición de líneas de carril en la imagen original.

Utilizando la línea de carril de la ventana deslizante anterior obtenemos los parámetros de la línea polinómica para rellenar la línea de carril y así obtener unos límites mucho más definidos que utilizaremos para realizar la superposición de líneas (ver figura 5.10).

Con las líneas de los carriles identificadas podemos superponer esa información en la imagen original, para ello generamos una matriz nueva completamente llena de ceros del mismo tamaño que la imagen original para dibujar las líneas del carril sobre ella. A esa matriz dibujada le deformamos el espacio en blanco de vuelta al espacio de la imagen original utilizando la matriz inversa y, combinándola con la imagen original, obtenemos el resultado de la figura 5.11: un frame igual al original pero con la superposición de carriles, es decir, con el carril por donde debe ir el vehículo en la carretera marcado con color.

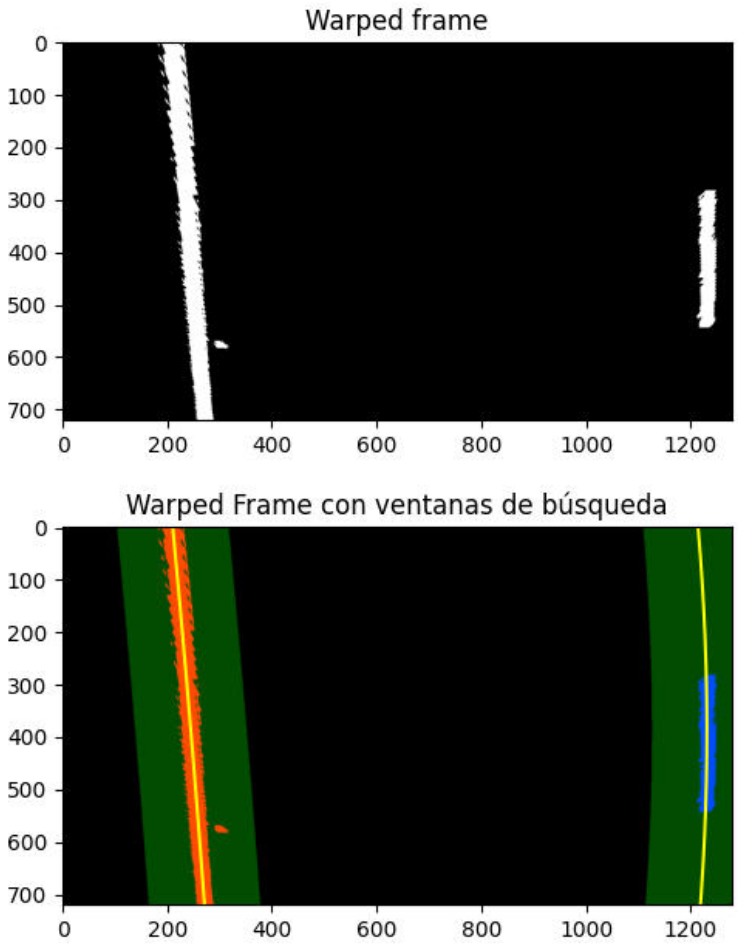

Figura 5.10 – Procesamiento de las líneas de carril con ventanas de búsqueda. Imagen de mi autoría.

Figura 5.11 - Frame original con superposición de carriles. Imagen de mi autoría.

5.1.6 Cálculo de varios elementos en base a la información procesada.

Con el área del carril demarcada ahora podemos calcular algunos elementos que nos servirán para guiar al robot móvil. Como, por ejemplo, el radio de curvatura de la línea de los carriles, la cual podemos calcular con la siguiente fórmula:

$$\frac{\left[1+\left(\frac{dy}{dx}\right)^2\right]^{3/2}}{\left|\frac{d^2y}{dx^2}\right|}$$

Esto es posible ya que la misma se da en coordenadas cartesianas y contamos con todas las variables necesarias para calcularla.

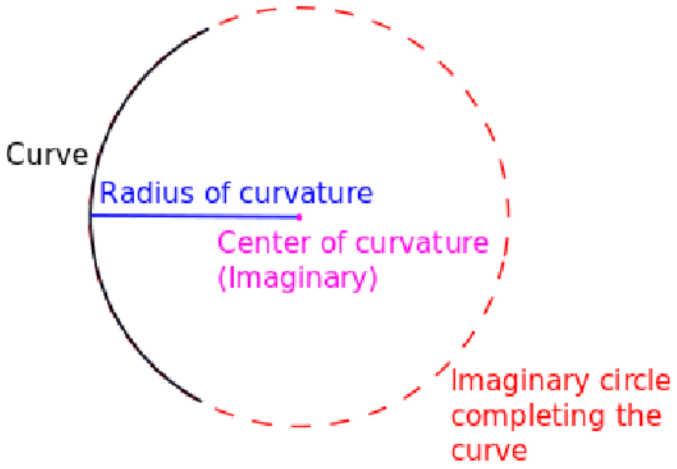

Figura 5.12 - Radio de curvatura y centro de curvatura. Imagen obtenida de https://en.wikipedia.org/wiki/Radius_of_curvature

Otro elemento muy importante para el guiado del robot que necesitamos calcular es la posición del vehículo o robot con respecto al centro de la carretera (punto medio entre las dos líneas de los carriles). Para ello, asumimos que la cámara está centrada en la imagen y, junto con la posición de la línea de los carriles, podemos determinar a cuanta distancia se encuentra desplazado el vehículo o el robot del centro de la carretera (ver figura 5.13).

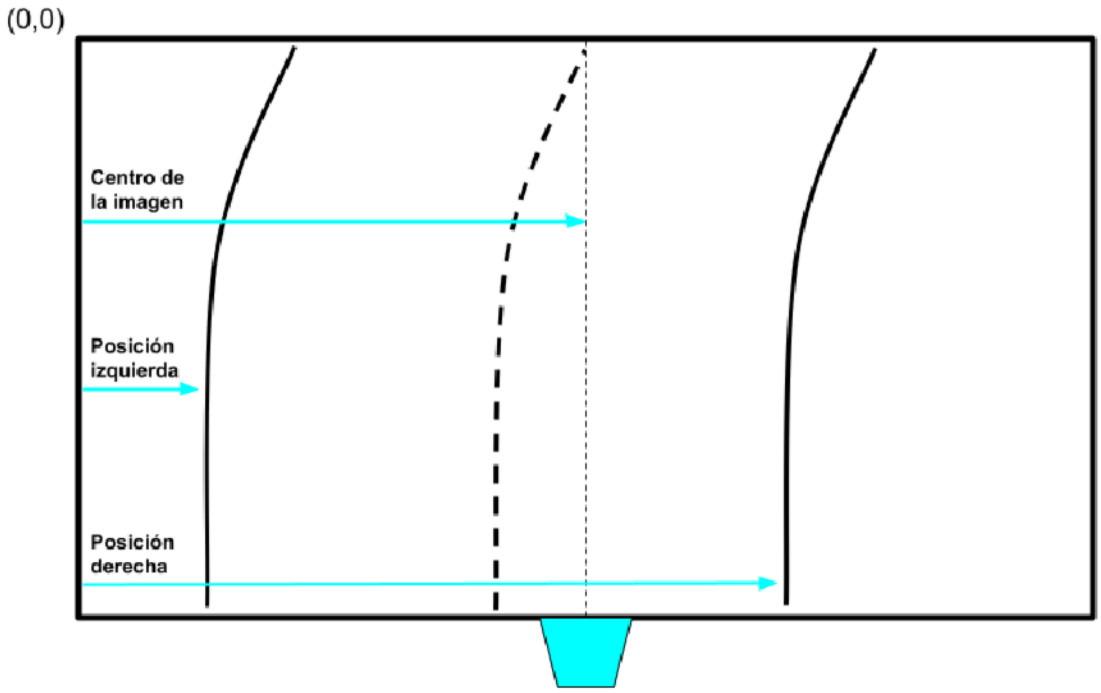

Figura 5.13 - Diagrama del cálculo de la distancia al centro de la carretera. Imagen de mi autoría.

Lo primero que debemos calcular es la posición del centro de la carretera, esto se puede obtener sencillamente sumando las posiciones del carril izquierdo y derecho y dividiéndolo por dos, es decir:

$$centroCarretera = \frac{(posIzq + posDer)}{2}$$

Lo segundo y último que tenemos que hacer es restar el resultado anterior con la posición del centro de la imagen que ya la tenemos (dividiendo la resolución de la imagen por dos), es decir:

$$distanciaAlCentro = centroImagen - centroCarretera$$

De esta forma obtenemos el desplazamiento del coche con respecto al centro de la carretera, un dato que nos será vital para poder dirigir al robot móvil.

5.2 SISTEMA DE GUIADO DEL ROBOT.

Para esta etapa se utilizaron las siguientes tecnologías y herramientas de desarrollo:

- Raspberry Pi OS: anteriormente conocido como Raspbian es el sistema operativo oficial de la Raspberry Pi. Es una distribución del SO GNU/Linux.
- Thonny: entorno de desarrollo integrado (IDE) para desarrollar en Python que viene preinstalado en el SO de Raspberry Pi.
- RPi.GPIO: librería que provee de una interfaz para controlar los pines GPIO de la Raspberry Pi.
- VNC: programa de software libre que permite el acceso remoto a distintos dispositivos. Utilizado para acceder remotamente a la Raspberry Pi desde una computadora conectada a la misma red.

5.2.1 Configuración inicial del robot.

Concluido el armado del robot (comentado en la sección 4.1) se procedió a configurarlo para así poder operar con él. Lo primero es instalar el sistema operativo para la Raspberry Pi, esto se realiza descargando el S.O de la página oficial de Raspberry[5] e instalándolo en una tarjeta SD que luego se le inserta a la placa.

Una vez instalado el sistema operativo se necesita habilitar el VNC que ya viene preinstalado para poder conectarnos remotamente desde otra PC que se encuentre en la misma red. De esta forma podemos configurar y manejarnos dentro del S.O sin la necesidad de tenerla conectada a una pantalla y un mouse, permitiendo así también la total movilidad del robot cuando se accionen sus motores.

Para comprobar que todo el hardware del robot funciona correctamente se utilizaron algunos ejemplos de códigos de prueba para accionar diferentes piezas del mismo, como por ejemplo, aquellos que permiten accionar los motores del robot para que vaya hacia delante, de una vuelta completa o gire hacia derecha e izquierda.

[5] Página oficial de Raspberry Pi para descargar S.O: https://www.raspberrypi.com/software/operating-systems/

5.2.2 Implementación y uso del sensor ultrasónico HC-SR04.

En un principio para poder instalar el sensor ultrasónico en el robot se realizó su conexión de la siguiente manera:

HC-SR04	GPIO (Raspberry Pi)
VCC	4
TRIG	7
ECHO	12
GND	9

Tabla 5.6 - Referencia de conexionado entre los pines del sensor y la Raspberry Pi

Pero luego de un análisis más detallado se concluyó que de esta forma no iba a funcionar correctamente porque la señal de salida ECHO del HC-SR04 es de 5 voltios mientras que los pines GPIO de entrada de la Raspberry Pi son de 3,3 voltios. Por lo tanto, si se envía una señal de 5V a un puerto desprotegido de entrada de 3,3V éste último puede resultar dañado. Entonces, para poder realizar una conexión apropiada es necesario disminuir el voltaje de salida del sensor para permitir que la placa reciba la señal correspondiente sin problemas. Esto se puede realizar de dos formas:

- Con un divisor resistivo que consiste de dos resistencias en serie conectadas a un voltaje de entrada, el cual debe ser reducido a un voltaje de salida.
- Con un convertidor adaptador de niveles lógicos de 3,3V a 5V que consiste en un pequeño circuito integrado que convierte dichos niveles, ya sea de 3,3V a 5V o viceversa. Permitiendo adaptar señales SPI, I2C o cualquier señal digital.

En nuestro caso se optó por la segunda opción ya que en la universidad se contaba con un adaptador de niveles lógicos ensamblado por otros estudiantes y, mediante una conexión como se aprecia en la figura 5.14, se puede realizar una instalación acorde en el robot.

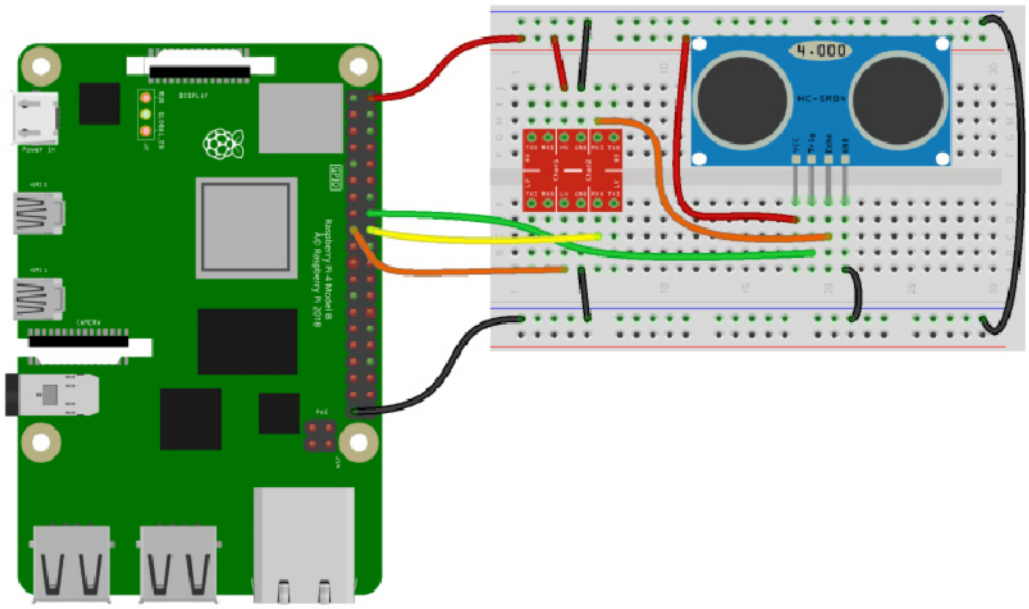

Figura 5.14 - Diagrama del conexionado entre el sensor HC-SR04 y la Raspberry Pi. Imagen obtenida de https://blog.330ohms.com/2020/06/17/como-conectar-un-sensor-ultrasonico-a-raspberry-pi/

Una vez hecho el conexionado se procede a probar el funcionamiento del sensor con distintos códigos de ejemplo sencillos obtenidos de internet[6] para medir la distancia. Aquí se comprobó que no funciona correctamente, el dispositivo simplemente no arroja ninguna medición y es como si no estuviese conectado. Se sugiere que el adaptador ensamblado no funciona correctamente y de ahí viene la falla. Por lo tanto, se descarta utilizar el sensor ultrasónico en el proyecto ya que dado las limitaciones de tiempo no es posible intentar otra conexión. Habría que adquirir otro adaptador de niveles lógicos o armar un divisor resistivo para intentar que funcione.

5.2.3 Compilación y ejecución del código.

Todo el código del desarrollo primero se compila y ejecuta en un ordenador personal que cuenta con mejores características técnicas que la Raspberry. Se optó por esta forma de trabajo dado que la placa posee muchas limitaciones para poder compilar y depurar el código. Una vez que se prueba una nueva modificación en el código, la misma se sube al repositorio del proyecto alojado en GitHub para luego descargarse en la Raspberry Pi y poder compilarla y ejecutarla allí con el programa Thonny. Y,

[6] https://blog.330ohms.com/2020/06/17/como-conectar-un-sensor-ultrasonico-a-raspberry-pi/

en caso de que haga falta dependiendo de los resultados de las pruebas, se realizan las correcciones pertinentes.

Como se explicó previamente, toda interacción con la Raspberry Pi se realiza a través del VNC Viewer instalado en una computadora personal y, a través de este programa, pudo apreciarse la gran dificultad de la placa para renderizar el video de la cámara a medida que lo procesaba. Es por ello que tomó mucha más relevancia la idea de quitarle procesamiento a la placa, idea pensada desde el inicio del proyecto debido a que se especulaba que la misma no iba a poder soportar correctamente todo el procesamiento del proyecto por sí sola y así pudo comprobarse.

5.2.4 Conexión mediante socket con ordenador que procese la información.

Una vez comprobado que el robot procesando todo el proyecto por sí sólo no puede funcionar correctamente se decidió realizar una conexión TCP/IP mediante socket entre la placa y mi computadora personal con el fin de que aumente su rendimiento. Esto se puede lograr de la siguiente manera:

1. La Raspberry Pi captura el video a través de la Raspberry Pi Camera y lo envía mediante un socket (puerto 8000) hacia la computadora.
2. El ordenador recibe el video en crudo y procesa cada frame para obtener la información pertinente que muestra en pantalla.
3. En base a esa información construye las instrucciones para poder guiar al robot. Una vez fabricadas se las envía mediante otro socket diferente (puerto 5500).
4. Apenas envía el video en crudo (primer paso) la placa ya está esperando por el puerto 5500 que la computadora le envíe las instrucciones. Una vez que las recibe acciona los motores en base a las órdenes que recibió.

De esta forma podemos quitarle toda la carga de procesamiento a la placa para aumentar el rendimiento del robot. Dirigiendo aquellos procesos que requieren mayor poder de cómputo a la computadora, haciendo una gran diferencia debido a sus especificaciones de hardware.

El funcionamiento general de esta comunicación entra mi computadora personal y el robot se puede ver en la siguiente figura 5.15, del lado izquierdo se encuentran los pasos realizados por el robot (Raspberry Pi) y del lado derecho aquellos hechos por el ordenador.

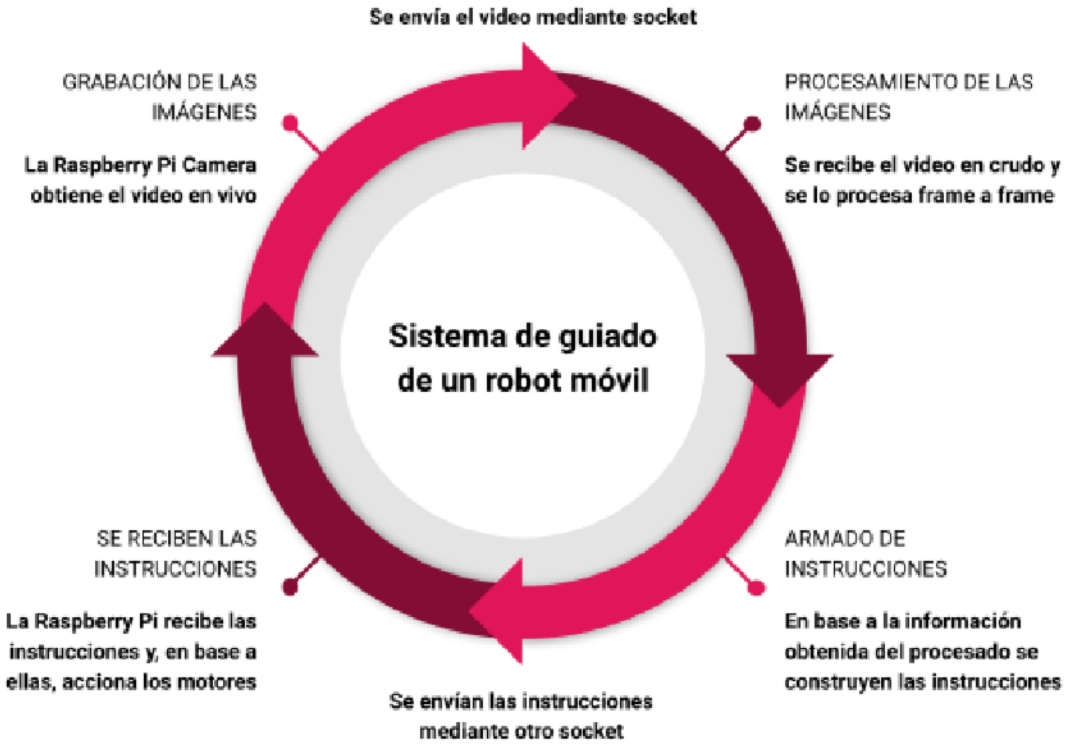

Figura 5.15 - Diagrama del funcionamiento general para guiar al robot móvil. Imagen de mi autoría.

6. RESULTADOS

Habiendo desarrollado el proyecto con más de 1500 líneas de código escritas en Python y ensamblado el robot en reiteradas veces hasta conseguir la versión final podemos pasar a la ejecución de las pruebas finales. Para ello se armó una pista en la que el robot pueda desenvolverse, se utilizaron cintas adhesivas de color amarillo sobre cartones para demarcar el carril, que cuenta con aproximadamente 8 centímetros de ancho. Al realizarlo sobre cartones se pueden superponer de distintas formas y así ir cambiando el circuito.

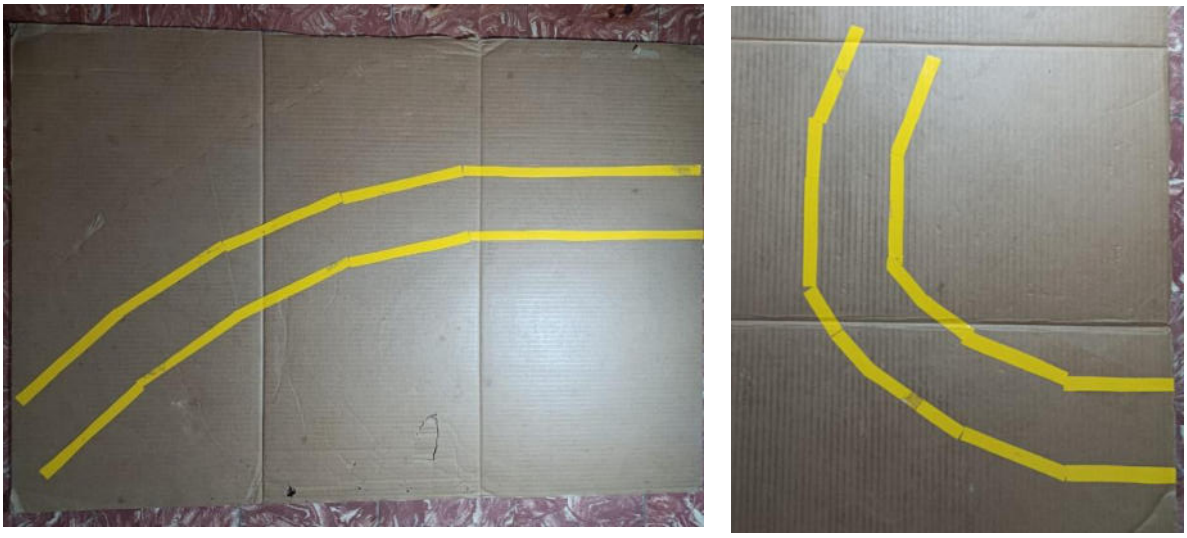

Figura 6.1 - Pista con líneas amarillas para demarcar los carriles, armable por partes. Imagen de mi autoría.

6.1 EJECUCIÓN DEL PROGRAMA.

Posicionamos el robot al inicio de la pista y allí ejecutamos en la Raspberry Pi, a través del VNC, los archivos *videoStreamRaspberry.py* y *movementControllerRaspberry.py* para enviar el video en vivo sin procesar por el puerto 8000 y para controlar el robot en base a las instrucciones que recibe por el puerto 5500, respectivamente. A su vez en la PC ejecutamos el archivo *pcClientProcessor.py* que se encarga de recibir el video en crudo, procesarlo y enviarle las instrucciones al robot.

Figura 6.2 - Video en vivo procesado, con la información pertinente en pantalla. Imagen de mi autoría.

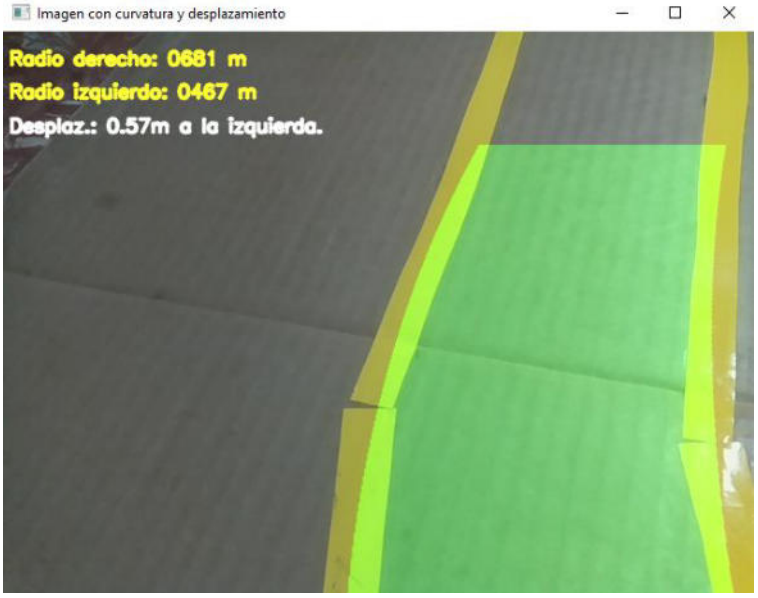

Figura 6.3 - Video en vivo procesado, con un desplazamiento mayor hacia la izquierda. Imagen de mi autoría.

Como se pueden apreciar en las figuras 6.2, 6.3, 6.4 y 6.5 el robot es capaz de seguir las líneas de carril amarillas para guiarse en el circuito. En las dos primeras figuras se puede ver el área verde pintada que indica la superficie detectada del carril, como así también en la esquina superior izquierda de la pantalla los datos que se obtuvieron del procesamiento de las imágenes, los radios calculados en base a las líneas del carril y el desplazamiento del centro que tiene actualmente el robot. En la primera imagen se encuentra posicionado 1 centímetro hacia la izquierda,

mientras que en la segunda unos 57 centímetros, es decir, se desvió aún más. El robot interpreta esa desviación y para corregirla o contrarrestarla debe maniobrar hacia el lado opuesto, este es el funcionamiento básico para guiar al robot en el circuito. En las dos últimas figuras de esta serie (figuras 6.4 y 6.5) podemos apreciar al robot desde otra perspectiva siguiendo el camino en base a lo que se recibe y se procesa, mostrado en las figuras anteriores.

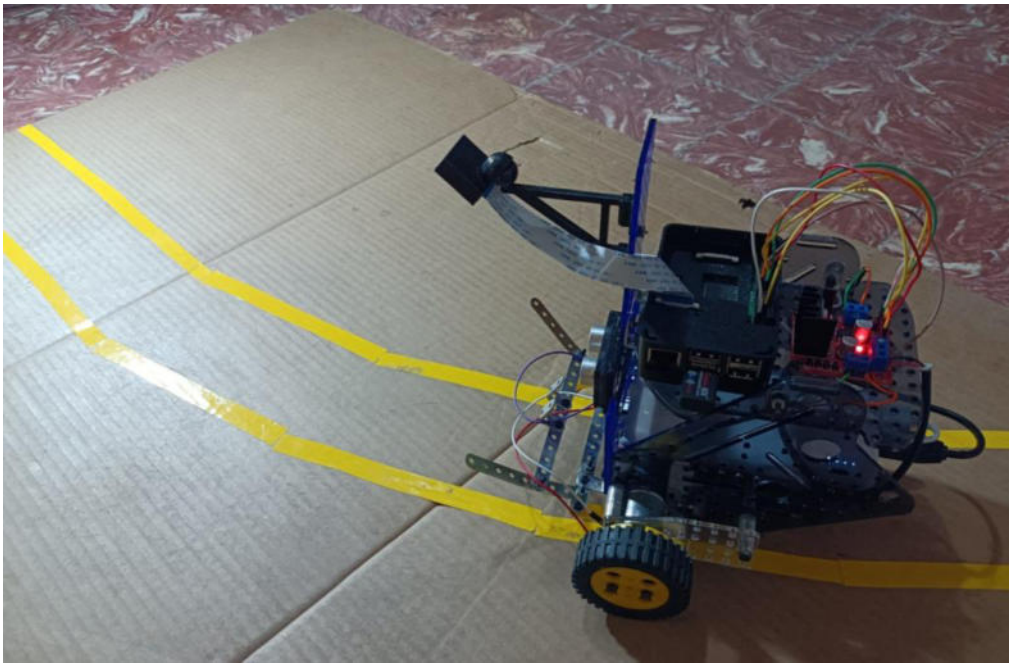

Figura 6.4 - Robot utilizando las líneas del carril para guiarse, vista lateral. Imagen de mi autoría.

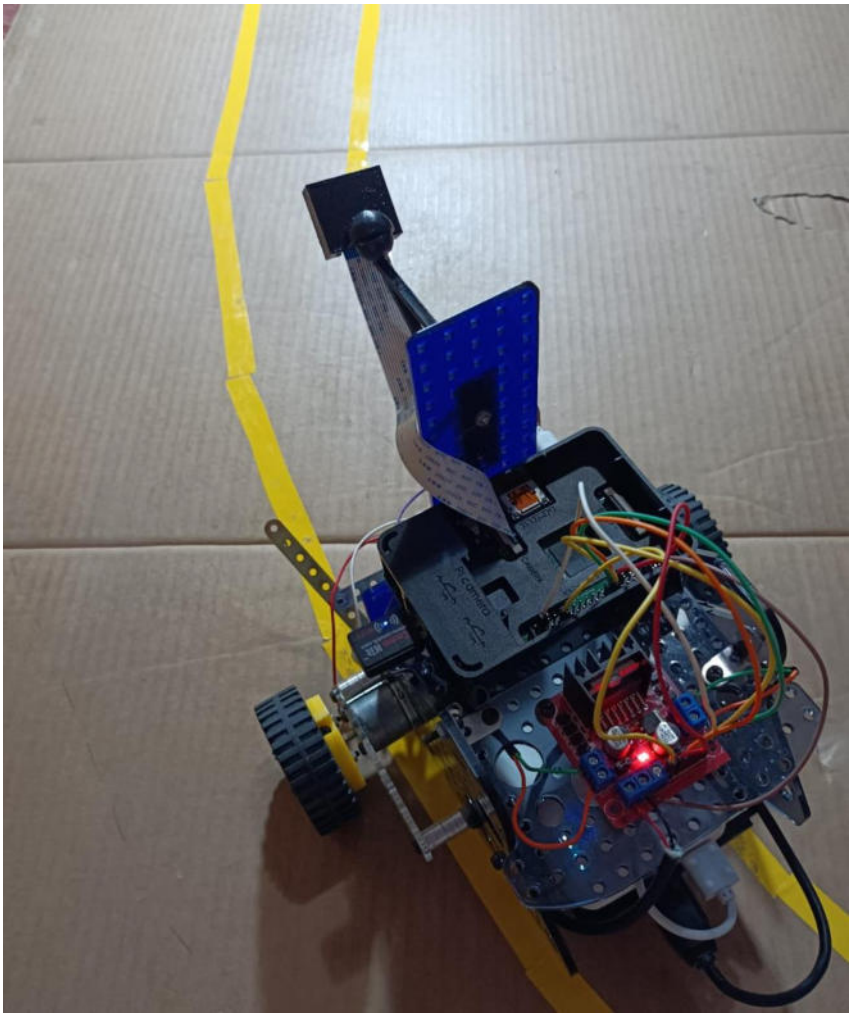

Figura 6.5 - Robot usando las líneas del carril para guiarse, vista trasera. Imagen de mi autoría.

Luego de ser ejecutado en reiteradas veces para observar su comportamiento se apreció que aproximadamente en un 60% de las pruebas realizadas cuando debía atravesar las curvas más pronunciadas, el robot no lograba direccionarse a tiempo y se desviaba de la pista. Lo que provocaba entrar en un ciclo de direcciones erróneas ya que sigue procesándose la información porque el mismo no detecta que se desvió.

6.2 ANÁLISIS DE LOS RESULTADOS.

En base a lo observado se puede analizar la situación para intentar descubrir las razones de los inconvenientes encontrados. Para ello se ejecutó varias veces más el programa solamente sobre la porción de la pista que posee una curva (primera

imagen de la figura 6.1) con el fin de estudiar a fondo la problemática. La principal causa que se detectó es que mientras el robot está ejecutando una instrucción que interpretó no llega a reaccionar a tiempo para la próxima y, en ese lapso de tiempo, se desvía de la pista. Para cuando terminó de ejecutarla ya se encuentra un poco desviado de la carretera y tiene en cola varias más que fueron tomadas en la posición anterior del trayecto logrando así desviarse aún más del carril hasta salirse por completo y entrar en el ciclo de direcciones erróneas nombrado anteriormente.

Esta situación se puede detallar mejor con las siguientes figuras, en el instante cero (figura 6.6) el robot interpreta que debe seguir recto para mantenerse dentro del carril. El camino ideal que debería seguir para ello es el marcado por la flecha roja de la figura y la dirección que interpreta o tomará para poder cumplirlo es el marcado por la flecha restante de color cian.

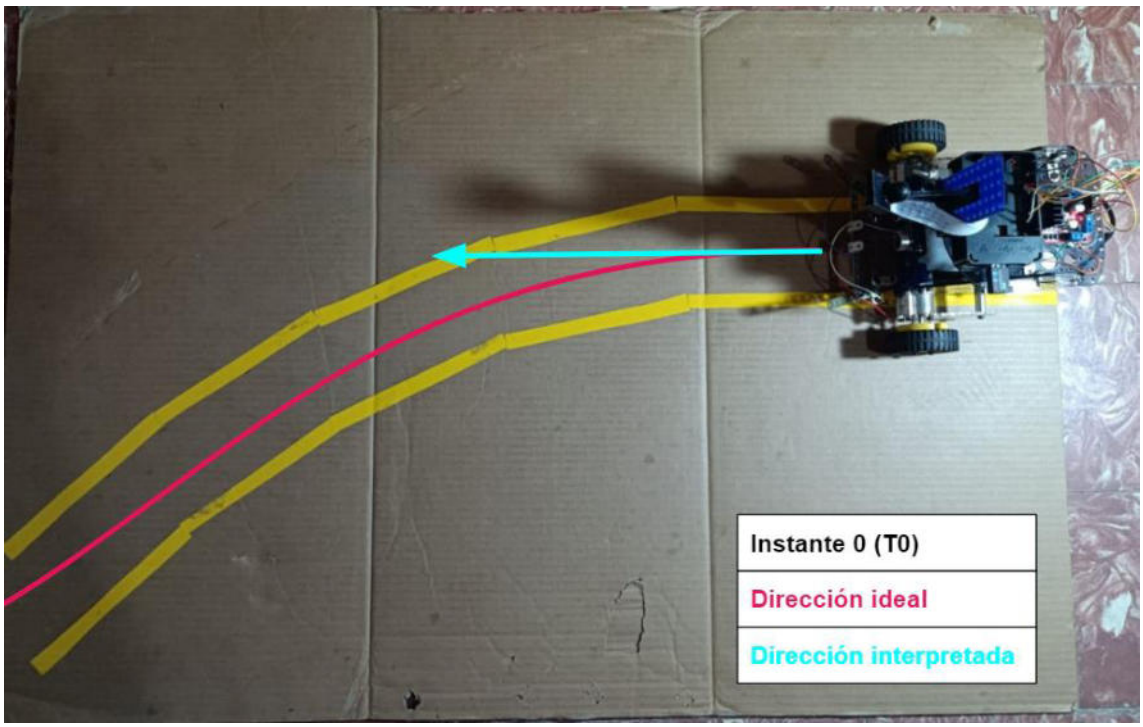

Figura 6.6 - Instante cero del trayecto analizado. Imagen de mi autoría.

Mientras continúa por esa dirección que entiende, el robot sigue procesando toda la información recibida y construyendo nuevas órdenes de dirección que se encolan para ejecutarse luego de finalizar la actual del instante cero.

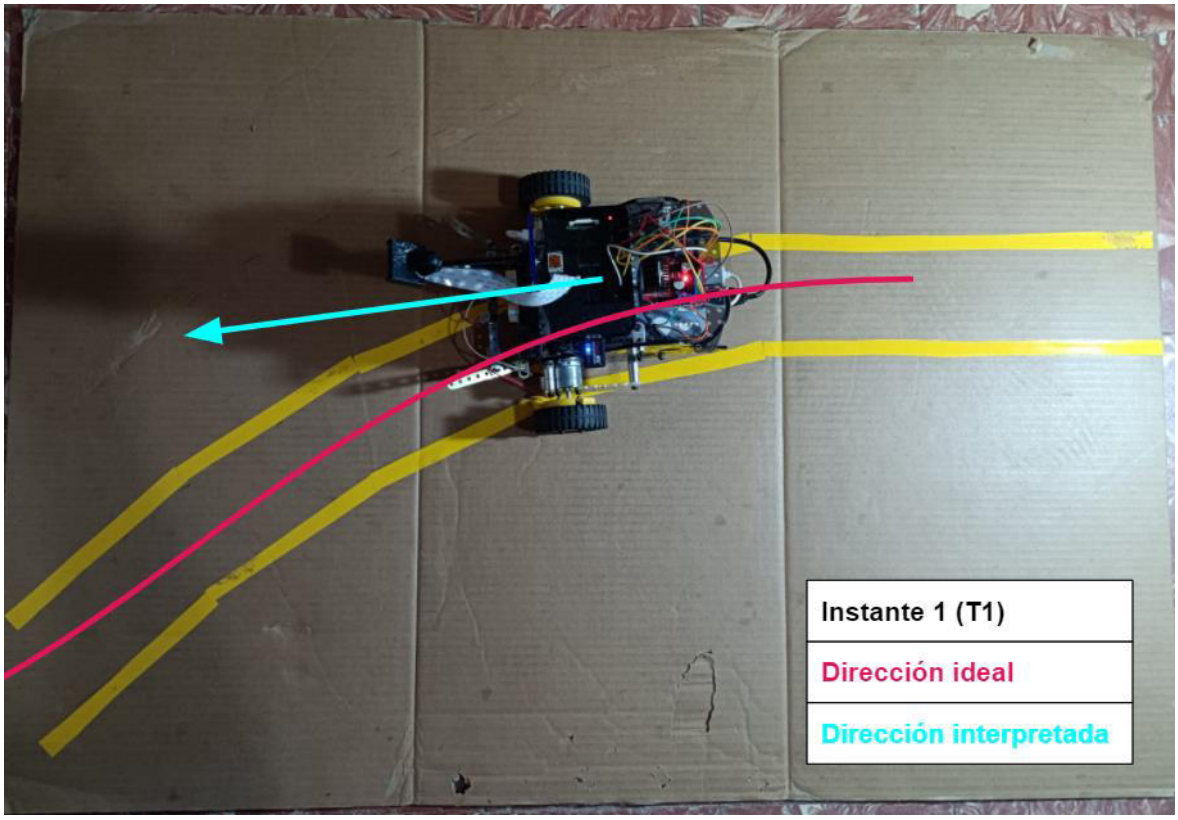

Figura 6.7 - Instante uno del trayecto analizado. Imagen de mi autoría.

Luego, en el instante uno (figura 6.7) cuando finaliza de ejecutar la primer orden, ya se encuentra un poco desplazado del camino y lo próximo a realizar es aquello que decidió en el anterior instante (marcado por la flecha de color cian) y, por lo tanto, incrementará aún más su desviación del camino ideal (marcado por la flecha de color rojo) ya que el robot se encuentra en otra posición diferente a la original.

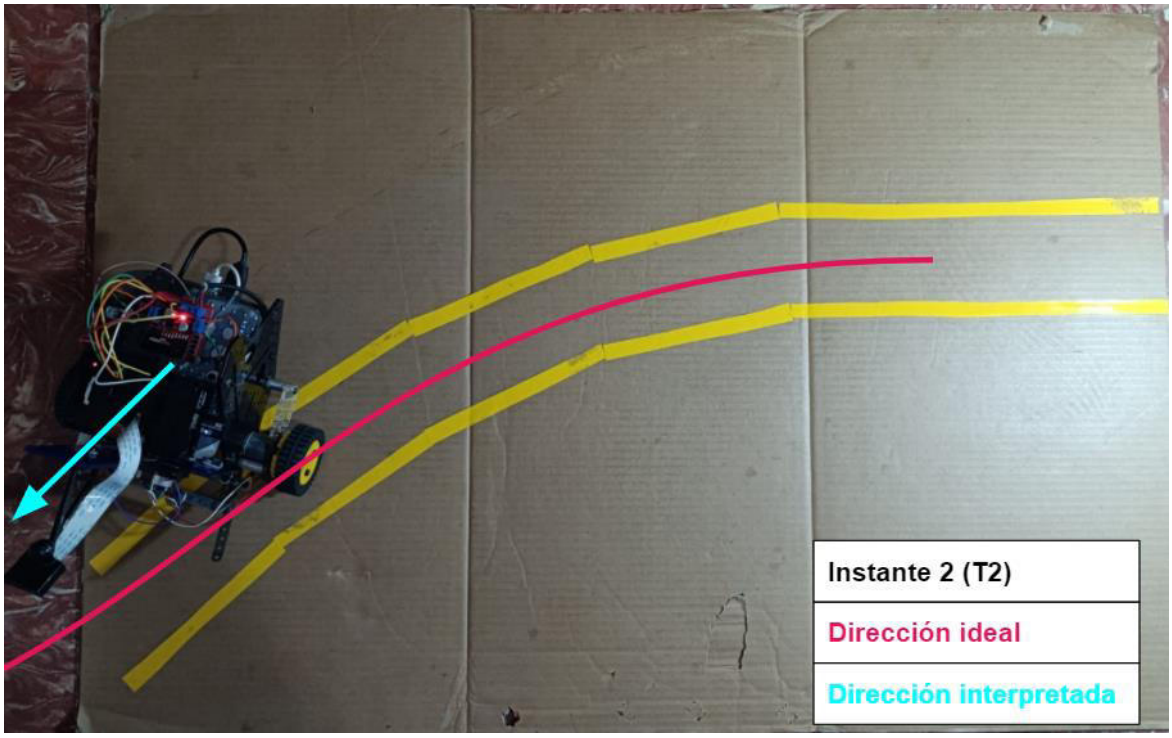

Figura 6.8 – Instante dos del trayecto analizado. Imagen de mi autoría.

Por último, en el instante dos (figura 6.8) cuando termina de llevar a cabo la segunda orden, ya se encuentra bastante apartado de la dirección ideal y debe interpretar la siguiente instrucción que también fue tomada en el anterior instante (instante uno). Por lo que, termina provocando que el robot se extravíe totalmente del camino ideal que debía seguir. Además de que al haberse alejado tanto de la pista ya comienza a procesar toda información errónea que no le servirá para volver al camino ideal y no hará más que seguir desviándolo.

6.3 ANÁLISIS DE SOLUCIONES Y MEJORAS.

Analizado el inconveniente en profundidad con el manejo del robot se analizaron distintas posibles soluciones. En un principio lo que se realizó fue un ajuste en las variables de guiado del mismo, es decir, se manipularon variables como:

- El ángulo de giro: si el robot no gira con tanta fuerza o amplitud se reduce la desviación promedio del mismo. Ya que de esta forma el dispositivo no realiza movimientos tan bruscos para corregir su dirección que lo suelen dejar desviado del camino, sino que los hace de una forma mucho más controlada.

- Máximo de desplazamiento: se determinó que como máximo el robot puede estar desplazado 10 centímetros hacia la derecha o la izquierda ya que el ancho del carril es pequeño en comparación al del dispositivo y, dentro de ese rango de 20 cm, se lo considera centrado dado que no puede caber entero en la carretera.
- Velocidad: se redujo la velocidad de las ruedas del motor prácticamente al mínimo para que el dispositivo pueda guiarse de una forma mucho más controlada y pausada, dándole el mayor tiempo posible para terminar de ejecutar la instrucción actual y darle lugar a la siguiente.

Al momento de volver a ejecutar el programa se pudieron notar algunas mejoras al atravesar las curvas de la pista, pero el robot seguía teniendo inconvenientes para mantenerse dentro de la carretera, seguía desviándose del trayecto ideal de una forma no determinista y al azar. Es decir, no había un patrón predeterminado en el cual ciertas acciones siempre daban con el mismo resultado o trayecto de desviación.

Ante esta nueva situación se comenzó a investigar mejores formas de controlar y dirigir el robot para planificar las mejoras futuras del proyecto actual. Se pudo hallar una muy interesante que será detallada en la próxima sección.

6.3.1 Control PID.

Se trata de un mecanismo de retroalimentación de bucle de control que calcula la diferencia entre un punto de ajuste deseado y la salida real de un proceso, y usa ese resultado para aplicar una corrección al proceso. Permitiendo regular no sólo la velocidad de un robot, sino también variables como la temperatura, presión y flujo de un proceso en general. Sus siglas PID vienen del idioma inglés y significan *Proportional, Integral* y *Derivative* (Proporcional, Integral y Derivado respectivamente) y funcionan de la siguiente manera:

- P – Proporcional: obtiene una parte del valor de error actual. Dicha proporción se especifica mediante una constante llamada valor de ganancia y su respuesta proporcional se representa con las letras *Kp*. Por ejemplo, puede establecerse en 0,3, lo que significa que utilizará un 30% del valor

del error para calcular la respuesta correctiva al proceso. Si no hay error, no hay una parte proporcional de la respuesta correctiva. Su fórmula está dada por $P_{sal} = K_p\, e(t)$ donde K_p es la ganancia proporcional (valor de sintonización) y $e(t) = SP - PV(t)$ es el error (SP es el punto de establecimiento, y PV(t) es la variable de proceso).

- I – Integral: toma todos los valores de error pasados y los integra a lo largo del tiempo, es decir, los acumula con el fin de que el término integral crezca hasta que el error llegue a cero. Cuando el error se elimina, el mismo dejará de crecer pero si aún existe después de aplicado el control proporcional, el término integral intenta eliminarlo agregando su valor de error acumulado. Dando como resultado que el efecto proporcional disminuya a medida que también lo hace el error, esto se logra compensar con el efecto integral. Su fórmula está dada por $I_{sal} = K_i \int_0^t e(t)\ dt$ donde K_i es la ganancia integral (parámetro de ajuste) y T es la variable de integración.
- D – Derivado: se utiliza para estimar la tendencia futura del error en función de su tasa de cambio actual. Su objetivo es el de contar con un efecto amortiguador en el sistema, cuanto más rápido sea el cambio, mayor será el efecto o control de amortiguación. Su fórmula está dada por $D_{sal} = Kd\ \frac{de}{dt}$ donde K_d es la ganancia derivativa (parámetro de ajuste) y T es el tiempo o tiempo instantáneo (el presente).

No todos los sistemas de control requieren el uso de los tres términos PID, en muchos casos emplean sólo las combinaciones PI o PD.

La fórmula general del PID (utilizando las tres variables) es la siguiente:

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(t')\,dt' + K_\mathrm{d} \frac{de(t)}{dt}$$

Figura 6.9 - Fórmula general del controlador PID utilizando sus tres variables. Imagen obtenida de https://medium.com/@jaems33/understanding-robot-motion-pid-control-8931899c31df

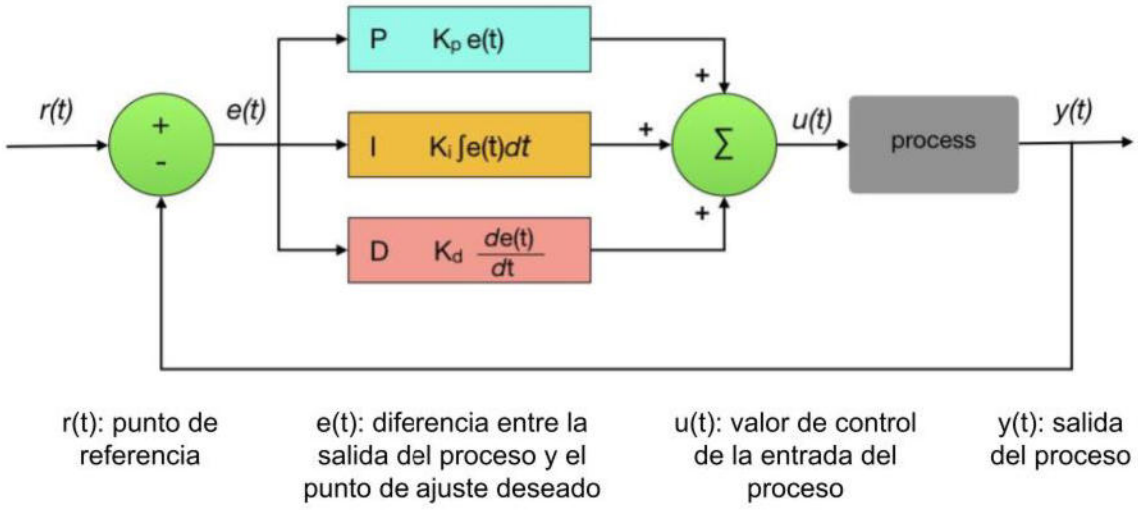

Figura 6.10 - Diagrama del funcionamiento de un controlador PID. Imagen adaptada de https://mjwhite8119.github.io/Robots/pid-control

Implementando y manipulando las variables del control PID creemos que puede ayudar a mejorar ampliamente el manejo del robot ya que, como bien se dijo, es un mecanismo de control que constantemente se está retroalimentando con las variables de fallo, actuando sobre ello y corrigiendo su trayecto para mantenerse dentro del ideal.

7. CONCLUSIONES.

Personalmente la robótica es un área que siempre me resultó interesante pero nunca había tenido la posibilidad de indagar mucho en ella, cuando se presentó la posibilidad de desarrollar este proyecto no tardé mucho en aceptarlo. Además de que se encontraba vinculado con el área de la visión artificial, un ámbito totalmente desconocido para mí pero que me intrigaba y despertaba mi interés para saber de qué se trataba.

A medida que fui avanzando en el desarrollo de la tesis pude ir comprendiendo una a una las diferentes técnicas y contenidos que hacen posible a cada una de estas tecnologías, pudiendo entenderlas, desarrollarlas y aplicarlas el presente proyecto, en pos de conseguir el primer objetivo planteado. Algo que resulto apasionante para mí, pero también me llevo muchísimo esfuerzo.

Esta investigación me llevó a descubrir numerosas tecnologías, técnicas y ámbitos relacionados con el tema, los cuales han despertado mi interés y motivación para continuar con esta línea de investigación y desarrollo en un futuro.

A nivel personal considero que esta instancia formativa es una excelente oportunidad para cerrar un ciclo académico, para hacer uso de todos los contenidos y aprendizajes obtenidos a lo largo del gran recorrido que hoy me lleva a esta instancia. Creo que es una buena forma de sintetizarlos y emplearlos en un proyecto.

8. ÍNDICE DE FIGURAS.

9. BIBLIOGRAFÍA.

- Siciliano, B., Sciavicco, L., Villani, L., Oriolo, G. (2010). *Robotics, Modelling, Planning and Control*, Editorial Stinger.
- Corke, P. (2011). *Robotics, Vision and Control*, Editorial Stinger.
- Laganière R. (2011). *OpenCV 2 Computer Vision Application Programming Cookbook,* Editorial Packt Publishing.
- Bradski, G. & Kaebler A. (2008). *Learning OpenCV. Computer Vision with the OpenCV Library*, Editorial O'Reilly.
- Pajankar A. (2020). *Raspberry Pi Computer Vision Programming*, Editorial Packt Publishing.
- Página oficial de la empresa *RobotGroup* - http://robotgroup.com.ar/
- Documentación oficial OpenCV. *"Thresholding Operations using inRange"* https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html
- Documentación oficial OpenCV. *"Canny Edge Detector"*. *https://docs.opencv.org/3.1.0/da/d5c/tutorial_canny_detector.html*
- Valverde Rebaza, Jorge. *"Detección de bordes mediante el algoritmo de Canny"*. Universidad Nacional de Trujillo.
- Documentación oficial OpenCV. *"Hough Line Transform". https://docs.opencv.org/3.1.0/d9/db0/tutorial_hough_lines.html*
- Curso *"The Complete Self-Driving Car Course - Applied Deep Learning"* - https://www.udemy.com/course/applied-deep-learningtm-the-complete-self-driving-car-course
- Curso *"Visión por computador con OpenCV y Python: Control de robots"* - https://www.udemy.com/course/control-de-robots-con-opencv-python-y-arduino
- *Detección de objetos por colores en imágenes con Python y OpenCV* - https://medium.com/@gastonace1/detecci%C3%B3n-de-objetos-por-colores-en-im%C3%A1genes-con-python-y-opencv-c8d9b6768ff
- *Omes, visión por computador* - https://omes-va.com/

- *Detector de bordes Canny cómo contar objetos con OpenCV y Python* - https://programarfacil.com/blog/vision-artificial/detector-de-bordes-canny-opencv/
- *The ultimate guide to real-time lane detection using OpenCV* - https://automaticaddison.com/the-ultimate-guide-to-real-time-lane-detection-using-opencv/
- *How the Sobel Operator Works* - https://automaticaddison.com/how-the-sobel-operator-works/
- *Operador Sobel* - https://es.wikipedia.org/wiki/Operador_Sobel
- *El círculo vicioso – Isaac Asimov* - *https://inteligenciaeducativa.net/descargas/runaround.pdf*
- *Midiendo distancias: el sensor ultrasónico* - http://edupython.blogspot.com/2016/07/midiendo-distancias-el-sensor.html
- *PID Control for robotics* - https://mjwhite8119.github.io/Robots/pid-control