



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

T R A B A J O
T E R M I N A L I I

No. 2020-A045

SISTEMA PARA LA NAVEGACIÓN DE UN
ROBOT TERRESTRE UTILIZANDO
PROCESAMIENTO DE IMÁGENES

Para obtener el grado de:

INGENIERO EN SISTEMAS COMPUTACIONALES

PRESENTAN:

BASTIDA PRADO JAIME ARMANDO
DÍAZ MEDINA JESÚS KAIMORTS
HERRERA RAMÍREZ ABSALOM



DIRECTORES:

M. EN C. NAYELI VEGA GARCÍA
DR. JULIO CÉSAR SOSA SAVEDRA

Ciudad de México a 15 de junio de 2021

Resumen

En este trabajo se describe el diseño, implementación y pruebas de un prototipo de sistema embebido para la navegación autónoma, en un ambiente controlado, de un móvil terrestre de 4 ruedas utilizando procesamiento digital de imágenes y un algoritmo de navegación basado en identificar el punto central del camino a recorrer, además de una página web de monitoreo a través de WiFi. El propósito del trabajo es proveer de una arquitectura modular y de código abierto para el futuro desarrollo de algoritmos de navegación de móviles terrestres basados en visión. Se trabajó utilizando la metodología en V realizando pruebas unitarias en cada componente del sistema con un acercamiento empírico para la optimización de los parámetros para la navegación. El resultado del trabajo está compuesto de un módulo de captura de imágenes, módulo de procesamiento digital de imágenes, módulo de navegación autónoma, módulo de manipulación de los motores del móvil y un módulo de transmisión de imágenes por medio de WiFi.

Palabras clave: Sistema Embebido, Procesamiento Digital de Imágenes, Algoritmo de Navegación Autónoma, Robótica.

Índice general

Resumen	I
1 Introducción	1
1.1 Antecedentes	1
1.2 Planteamiento del problema	3
1.3 Objetivos	4
1.3.1 Objetivo general	4
1.3.2 Objetivos específicos	4
1.4 Justificación	5
2 Marco Conceptual y Estado del Arte	6
2.1 Robot móvil terrestre	6
2.2 Sistemas embebidos	15
2.3 Comunicación en Red	16
2.4 Procesamiento digital de imágenes	20
2.5 Algoritmos de navegación robótica	33
2.6 Estado del arte	36
3 Análisis y Diseño del sistema	37
3.1 Metodología del diseño	37
3.2 Análisis y requerimientos del sistema	39
3.3 Diseño global	51
3.4 Diseño en detalle	57
4 Desarrollo y Pruebas	63
4.1 Adquisición de la imagen digital	63
4.2 Módulo de Procesamiento Digital de Imagen	66
4.3 Módulo de Motores	72
4.4 Módulo de Navegación Autónoma	84
4.5 Módulo de Aplicación Web	93
4.6 Integración de Módulo de Navegación y Motores	97
5 Resultados y Conclusiones	104
5.1 Resultados: Prueba final del sistema	104
5.2 Conclusiones	105
5.3 Trabajo a futuro	107

Índice de figuras

2.1	Ejemplo de rueda fija [6].	7
2.2	Ejemplo de rueda tipo castor [6].	7
2.3	Modelo de tracción y dirección en diferentes ejes [7].	8
2.4	Modelo de tracción y dirección en el mismo eje [7].	8
2.5	Modelo de tracción y dirección en todos los ejes [7].	9
2.6	Sensor CNY70 [8].	9
2.7	Sensor de Imagen.	10
2.8	Sensor Ultrasónico [8].	11
2.9	Acelerómetro en chip [8].	12
2.10	Giroscopio en chip [8].	12
2.11	Puente H [15].	14
2.12	Funcionamiento del Puente H [15].	14
2.13	Imagen con 256 niveles de intensidad y representación numérica de un fragmento de 8x8 [25]	21
2.14	Etapas involucradas en el procesamiento de imágenes [28]	22
2.15	Planos de color RGB representados como tres matrices bidimensionales [25] . .	24
2.16	Modelo HSV	24
2.17	Representacion grafica del modelo HSV	25
2.18	Operaciones lógicas sobre imágenes binarias [28]	29
2.19	Representación gráfica de la composición de procesos [26]	31
2.20	Ejemplo de segmentación de una imagen basada en textura [32].	32
2.21	Estructura de navegación basada en mapa [33]	34
2.22	Estructura de navegación libre de mapa [33]	35
3.1	Metodología en V [37]	38
3.2	Raspberry Pi Cámara V1.	45
3.3	Chásis, ruedas y motores del robot móvil terrestre.	46
3.4	Controlador de motores L298N.	47
3.5	Powerbank.	48
3.6	Escenario ideal	51
3.7	Diagrama a bloques del sistema	52
3.8	Diagrama del proceso general del sistema	53
3.9	Diagrama de arquitectura de prototipo	54
3.10	Diagrama de componentes	55
3.11	Diagrama de bloques	55
3.12	Diagrama de caso de uso general	56
3.13	Diagrama de proceso del módulo de Procesamiento Digital de Imagen	57

3.14 Diagrama de proceso del módulo de Navegación Autónoma	58
3.15 Diagrama de proceso del módulo de App Web	60
3.16 Interfaz web de aplicación de visualización de imagen	61
3.17 Diagrama de flujo del algoritmo de navegación propuesto	62
4.1 Conexión de la cámara.	64
4.2 Raspberry Pi con la cámara equipada.	64
4.3 Imagen de prueba tomada con la cámara utilizando OpenCV.	65
4.4 Interaccion del programa de PDI	66
4.5 Robot en posición en la pista	67
4.6 Programa de afinamiento de parámetros de umbralizado	68
4.7 Programa de afinamiento de parámetros de umbralizado después de haber encontrado los valores óptimos	69
4.8 Programa de afinamiento de puntos de perspectiva.	70
4.9 Ajuste de perspectiva logrado después de afinar los valores de los puntos.	71
4.10 Chasis del robot	72
4.11 Circuito integrado del módulo L298N	73
4.12 Módulo L298N con entradas y salidas especificadas	74
4.13 GPIO de Raspberry Pi	75
4.14 Conexión de módulo L298N, Raspberry Pi, motores y fuente de alimentación. .	76
4.15 Vistas del módulo L298N (a) lado del motor izquierdo, (b) lado del motor derecho	76
4.16 Vistas del módulo L298N (a) frontal, (b) superior	77
4.17 Giro constante de un motor	77
4.18 Vista de armado para giro constante de un motor	77
4.19 Vistas del armado del robot para prueba de motores.	82
4.20 Representación de una curva del camino	84
4.21 Sumarización de pixeles	85
4.22 Casos singulares de un camino recto	85
4.23 Ajuste de línea central del camino	86
4.24 Búsqueda de curvatura de camino	87
4.25 Búsqueda de curvatura de camino	88
4.26 Ejecución del algoritmo con <i>display</i> =1, mostrando solo la imagen resultante .	90
4.27 Ejecución del algoritmo con <i>display</i> =2, es decir mostrando cada paso del algoritmo	92
4.28 Editando el archivo de configuración de Apache	94
4.29 Estado del servidor Apache	94
4.30 Probando instalación correcta de Apache Web Server	95
4.31 Estructura del programa cliente	95
4.32 Programa web cliente en donde se puede visualizar la imagen de monitoreo que comprende las imágenes tomadas y procesadas por la Raspberry Pi.	96
4.33 Base de la cámara y distribución de peso	99
4.34 Circuito de pruebas	100
4.35 Valores de curvatura derecha y recto	101
4.36 Robot con chasis de 4 ruedas	102

Índice de tablas

2.1 Profundidades de bits de tipos de imágenes comunes y dominios de aplicación típicos [30]	26
2.2 Comparativa entre formatos de imagen	27
2.3 Vecindad 4 entre píxeles [28]	28
2.4 Vecindad 8 entre píxeles [28]	28
3.1 Requerimientos básicos	39
3.2 Requerimientos funcionales	40
3.3 Requerimientos no funcionales	41
3.4 Comparativa de los Sistemas en Chip (SoC) [38], [39], [40]	43
3.5 Comparativa entre Raspberry Pi 3 [38], [41], [42]	44
3.6 Comparativa de las Cámaras para Raspberry Pi [43], [44]	45
4.1 Modos de operación del módulo L298N	74

1

Introducción

1.1. Antecedentes

Debido a la continua demanda de la industria de hardware y software, el desarrollo de nuevas tecnologías en la cuarta revolución industrial (la revolución tecnológica) y la necesidad de procesos automatizados se ha modificado fundamentalmente la forma en la que viven, trabajan y se relacionan los seres humanos.

En [1] se habla de cómo el costo decreciente de la potencia de procesamiento, de la memoria y de la capacidad de diseñar sistemas de bajo costo en chip, ha llevado al desarrollo y despliegue de sistemas embebidos (sistemas de computación que tienen una función dedicada dentro de un sistema mecánico o eléctrico más grande), en una amplia gama de entornos de aplicaciones. Los ejemplos incluyen máquinas industriales, electrónica de consumo, dispositivos de la industria agrícola y de procesos, automóviles, equipos médicos, cámaras, electrodomésticos, aviones, máquinas expendedoras y juguetes, así como dispositivos móviles.

Como se menciona en [2], la importancia de los sistemas embebidos aumenta continuamente teniendo en cuenta la amplitud de los campos de aplicación donde son empleados, esto ha impulsado al surgimiento de una industria fuerte que los desarrolla y utiliza. Su importancia para los servicios en todos los frentes y para el crecimiento tecnológico y, por lo tanto, económico ha llevado a esfuerzos significativos para abordar los desafíos planteados por el desarrollo y despliegue de sistemas embebidos.

Es indudable su relevancia por el impacto que tienen en la economía de los países, ya que favorecen su competitividad, hacen más eficientes los procesos productivos, aportan valor agregado a los bienes y servicios que los integran, así como a los distintos sectores económicos que los producen. Tienen especial importancia en las industrias electrónica y tecnologías de la información.

En [3] se explica cómo normalmente los sistemas embebidos están relacionados con la robótica, la cual se define como la técnica que se utiliza en el diseño y la construcción de robots o aparatos que realizan operaciones o trabajos, generalmente en instalaciones industriales. Esta técnica, o conexión, viene de la necesidad de la automatización industrial caracterizada

durante los periodos de cambios bruscos en los métodos populares. Dichos periodos de cambio en las técnicas de automatización parecen estar estrechamente ligados con la economía mundial. La robótica se relaciona en sí con el deseo de sintetizar algunos aspectos de la función humana mediante el uso de mecanismos, sensores, actuadores y computadoras.

En [4] explica que los tipos de robot según su locomoción y cinemática se clasifican en: estacionarios, móviles terrestres, con piernas, acuáticos y aéreos. Hoy en día, el robot móvil terrestre se ha vuelto popular en el ejército; ya que se puede usar como robot detector de bombas; en casa, se puede usar como aspiradora o jardinería y en la industria, para ayudar a las personas con fines de fábrica.

Para cualquier dispositivo móvil (entiéndase cualquier dispositivo capaz de trasladarse por cualquier medio), la capacidad de navegar en su entorno de manera segura es importante. La navegación inteligente de un dispositivo móvil trata con tres problemas principales: el mapeo del ambiente, la localización y navegación del mismo. Esto significa que el móvil requiere de una representación del entorno en el que se encuentra (mapeo), de la capacidad para determinar su propia posición en su marco de referencia (localización) y de un método para planificar un camino hacia alguna ubicación objetivo (navegación).

La navegación basada en la visión o la navegación óptica utiliza algoritmos de visión por computadora y sensores ópticos, algunos incluyen un telémetro basado en láser y cámaras fotométricas que utilizan matrices CCD, para extraer las características visuales requeridas para la localización en el entorno circundante. Sin embargo, hay una variedad de técnicas para la navegación y localización que utilizan información de visión.

A pesar de lo mencionado anteriormente, la implementación de algoritmos de cómputo pesado comúnmente utilizados en la navegación autónoma o el procesamiento de imágenes sigue representando un reto para los programadores de sistemas embebidos; al contar con recursos inferiores a los normalmente empleados en computadoras de escritorio, se debe tener especial cuidado en implementar una solución eficiente en el uso de los recursos como la memoria y la capacidad de procesamiento. Por lo tanto, los lenguajes y técnicas de programación se escogen de manera que se aprovechen al máximo los recursos ofrecidos por el sistema, mención especial tienen las técnicas de Cómputo de Alto Desempeño (HPC) por sus siglas en inglés.

1.2. Planteamiento del problema

Como ya se ha mencionado, la importancia de los sistemas embebidos y la robótica crece constantemente en el desarrollo de las actividades humanas, la investigación de algoritmos de navegación basados en visión y las técnicas de procesamiento digital de imágenes han emergido a su vez impulsando dicha evolución tecnológica.

Los sistemas de navegación autónoma por visión suelen ser implementados con una combinación de sensores: infrarrojos, Unidades de Medición Inercial (IMU), cámaras montadas en el robot o externas. En el diseño de dichos algoritmos se suelen contemplar dos enfoques; en ambiente controlado y no controlado. En ambientes no controlados la necesidad de múltiples sensores aunado a algoritmos de procesamiento pesado incrementan en gran medida el uso de recursos. En ambientes controlados se definen ciertas características del entorno en donde se colocará el dispositivo móvil que permiten el diseño de un sistema de navegación que requiere menos recursos con mejor desempeño.

Múltiples propuestas en el estado del arte optan por diseñar sistemas sobre el enfoque de ambientes no controlados, cada una de ellas aportando alguna técnica, diferente arreglo de sensores, diversos algoritmos de navegación, Sistemas en Chip (SoC), etc. Sin embargo, pocas son las propuestas bajo un ambiente controlado utilizando una sola cámara como sensor y un sistema embebido que no haga uso de entornos de desarrollo específico como suele ser el *Robot Operating System* (ROS).

El llevar a cabo el desarrollo de algunos de estos sistemas significa invertir una cantidad considerable de dinero en los componentes a utilizar. Aunado a ello, existen otras opciones como kits de desarrollo que proporcionan al usuario componentes de propiedad intelectual, lo que limita al usuario desde muchas perspectivas una de ellas es el costo, por otra parte, los desarrollos de propiedad intelectual no suelen ofrecer documentación más allá de los manuales de uso y cuando lo hacen suelen ser pobremente documentados. Lo anterior deriva en un total desconocimiento de lo que se está utilizando incluso posibles errores a la hora de implementar, además al ser arquitecturas cerradas no se puede o es difícil hacer modificaciones.

1.3. Objetivos

1.3.1. Objetivo general

Desarrollar un sistema embebido prototípico para la navegación autónoma de un robot móvil terrestre en un escenario controlado empleando técnicas de procesamiento de imágenes, un sistema basado en un computador de placa única, una cámara, y comunicación Wi-Fi con una aplicación web.

1.3.2. Objetivos específicos

1. Desarrollar un módulo de captura y procesamiento digital de imágenes utilizando la cámara conectada al sistema embebido como sensor.
2. Desarrollar un módulo de navegación autónoma.
3. Desarrollar un módulo de manipulación de los motores del robot.
4. Desarrollar módulo de servidor-cliente para la transmisión de las imágenes procesadas hacia una computadora personal por medio de Wi-Fi.
5. Desarrollar una aplicación web que permita monitorear las imágenes procesadas.
6. Documentar el trabajo realizado.

1.4. Justificación

El contar con un dispositivo móvil autónomo en un ambiente controlado puede ser la base de un campo de aplicaciones tan extenso como: procesos industriales, sistemas de vigilancia, automóviles, equipos médicos, dispositivos de trabajo doméstico, aviones, drones, juguetes, etc.

En la industria muchas empresas recurren a la automatización de tareas en el área de producción, sin embargo la manipulación de materiales y el transporte interno a menudo se realiza de forma manual. Con robots móviles de colaboración estas actividades pueden automatizarse.

Usualmente las escuelas e instituciones suelen invertir recursos financieros para comprar robots móviles terrestres comerciales para su programación y, posteriormente, pruebas de algoritmos de navegación los cuales están limitados a las librerías y funciones de propiedad intelectual que el proveedor ofrece con el equipo, además, en ocasiones el código fuente de dichas funciones no está disponible al usuario o se encuentra pobemente documentado.

Este trabajo propone una arquitectura modular y de código abierto con el fin de servir como material didáctico y de apoyo hacia futuros trabajos de investigación para la implementación y pruebas de distintos algoritmos de navegación basados en visión sobre un robot móvil terrestre.

Este trabajo propone una arquitectura modular y de código abierto con el fin de servir como material didáctico y de apoyo hacia futuros trabajos de investigación basados en visión sobre un robot móvil terrestre.

Marco Conceptual y Estado del Arte

2.1. Robot móvil terrestre

El desarrollo de los robots móviles (RM), responde a la necesidad de expandir el campo de aplicación de la robótica; inicialmente los robots estaban restringidos al alcance de su estructura mecánica anclada en una base, y para incrementar la autonomía de los robots, buscando campos en donde la intervención humana es complicada.

Como se menciona en [5] los RM terrestres pueden clasificarse de acuerdo al tipo de locomoción que utilizan, entre los tipos más ampliamente empleados se encuentran por: patas, ruedas y orugas. Aunque las locomociones por patas u orugas han sido ampliamente estudiadas, la mayoría de los RM que se han reportado, construido y evaluado en trabajos de investigación utilizan ruedas para desplazarse. Esto se debe a que los RM terrestres con ruedas presentan las siguientes características:

- Más eficientes en energía, debido a que generalmente se desplazan en superficies lisas y firmes.
- Ampliamente utilizados en aplicaciones de la industria.
- Requieren menor número de partes y menos complejas, haciendo más fácil su construcción.
- El control de las ruedas es menos complejo.
- Ocasionan menor desgaste en la superficie donde se mueven en comparación con las bandas de las orugas.
- Los problemas de balance no presentan gran dificultad, ya que el robot siempre está en contacto con una superficie.

Conforme a [5] un RM terrestre se conforma de tres partes fundamentales; ruedas y su disposición, sistema de actuadores o motores y los sensores. La disposición de las ruedas o arreglo cinemático es aquel que se genera de las configuraciones relativas a la disposición de las ruedas en la estructura del robot. Un sistema de actuadores es el recurso que proporciona

el movimiento del móvil y los sensores son dispositivos que obtienen información del área de trabajo y la procesan en señales que pueda comprender el robot.

Ruedas y Disposición de las mismas en un RM Terrestre

Las ruedas proporcionan la habilidad de trasladarse de un punto a otro al robot, dentro del tipo de ruedas a emplear en un RM terrestre destacan las ruedas fijas y tipo castor (u orientable no centrada). Las ruedas fijas tienen dicho nombre porque se encuentran fijadas a la estructura del móvil, su propósito es el de proporcionar la locomoción necesaria al robot a través de los motores conectados a ellas, además de la maniobrabilidad (capacidad de cambiar de rumbo). Su forma suele ser como se muestra en la Figura 2.1.

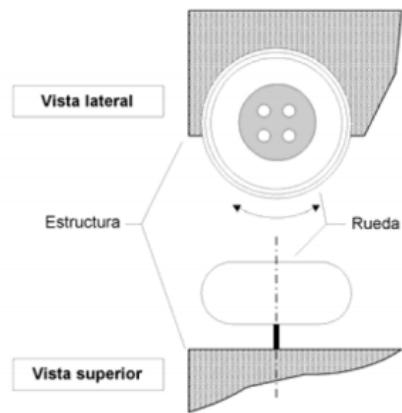


Figura 2.1: Ejemplo de rueda fija [6].

Las ruedas de tipo castor proporcionan estabilidad además de servir como rueda de dirección. Un diagrama se aprecia en la Figura 2.2.

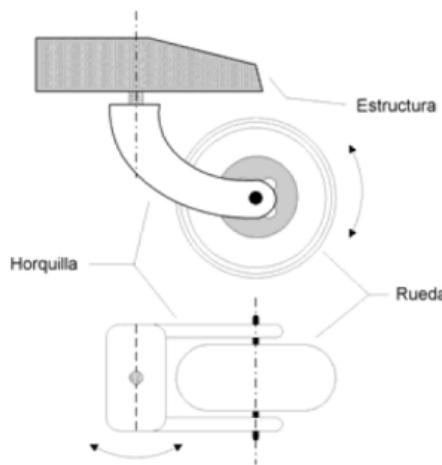


Figura 2.2: Ejemplo de rueda tipo castor [6].

Dentro de los arreglos cinemáticos se encuentran diversas configuraciones que han sido de interés en múltiples estudios, las diferentes combinaciones de ruedas y ejes afectan en mayor

o menor medida el grado de maniobrabilidad y, a su vez, complejidad de control. A continuación, se presentan brevemente las principales características de los tres modelos básicos a partir de los cuales se pueden obtener diversas configuraciones, de acuerdo a [7].

Tracción y dirección en ejes independientes.

En este modelo la tracción se genera en las ruedas traseras y la dirección en las delanteras, véase Figura 2.3. En este caso el control de dirección es más sencillo, pero la precisión en la dirección depende de la adherencia de las ruedas correspondientes, una característica importante es que posee un radio de giro bastante elevado en relación a otros sistemas, lo cual causa que no se puedan efectuar giros muy bruscos.

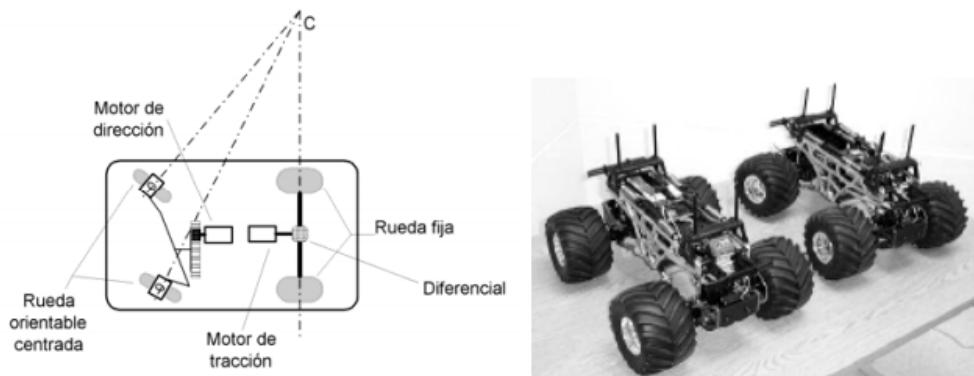


Figura 2.3: Modelo de tracción y dirección en diferentes ejes [7].

Tracción y dirección en el mismo eje (Tracción diferencial).

En la Figura 2.4 se muestra un ejemplo de este tipo de arreglo, el cual involucra ruedas fijas con motores independientes entre ellos ubicados en un mismo eje los cuales proporcionan la tracción y la dirección, con el resto de las ruedas de tipo castor en los demás ejes. Este modelo es de construcción sencilla y permite giros sobre el eje central del robot, sin embargo para lograr un control sencillo se requiere que todos los motores sean de características idénticas, de lo contrario se podría complicar el manejo.

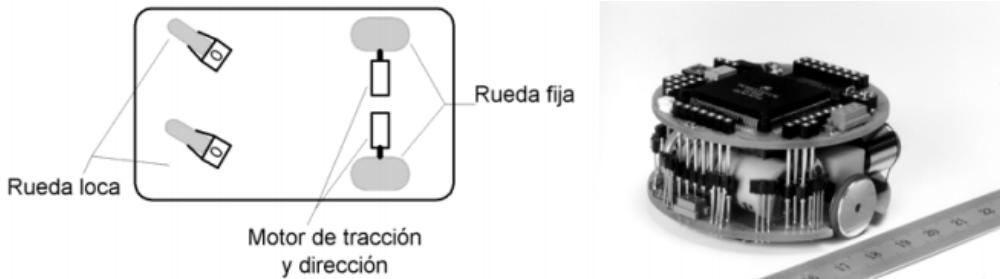


Figura 2.4: Modelo de tracción y dirección en el mismo eje [7].

Tracción y dirección en todos los ejes.

En este caso se requiere de un sistema odométrico para calcular la incertidumbre de los radios de giro asociados a este tipo de modelo en particular, su ventaja se presenta en los terrenos hostiles, donde la velocidad de traslación es menos importante que una buena adherencia al terreno. Un diagrama de este tipo de modelo se muestra a continuación Figura 2.5.

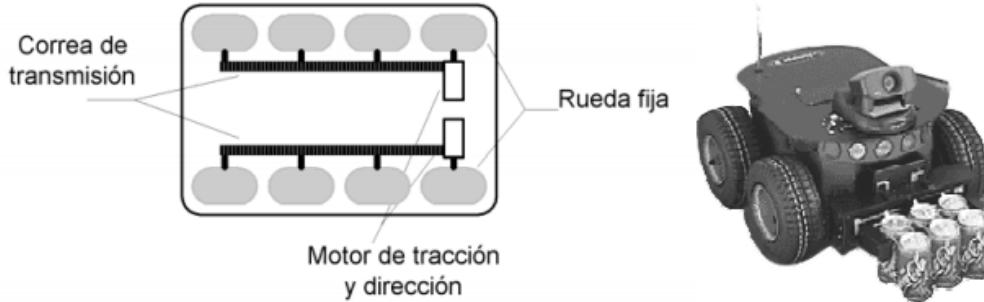


Figura 2.5: Modelo de tracción y dirección en todos los ejes [7].

Sensores de un RM Terrestre

Los sensores del robot proporcionan autonomía al robot para realizar su desplazamiento en un determinado espacio de trabajo. El uso de sensores, permite al robot percibir el espacio de trabajo, y navegar de acuerdo a las señales percibidas, a diferencia de seguir una secuencia de instrucciones predeterminadas. Algunos de los sensores comúnmente utilizados son: de distancia, posición, velocidad, luz, infrarrojos o ultrasónicos.

Sensores de luz.

Dentro de los sensores de luz podemos encontrar sensores como los reflectivos, de los cuales se habla en [8], estos se basan en una fuente de luz (lámpara, diodo LED o diodo láser) y una célula receptora del reflejo de esta luz, que puede ser un fotodiodo o un fotoresistor. Dentro de estos existen de diferentes sensibilidades desde los que detectan un objeto a 5mm de distancia hasta los que apoyados de un infrarrojo pueden hacerlo a más de un metro. Un ejemplo muy usado en los robots de tipo seguidor de línea es el sensor de reflexión CNY70 (2.6), el cual consta de un diodo emisor de infrarrojos y un fotoresistor.



Figura 2.6: Sensor CNY70 [8].

Existen otro tipo de sensores conformados de un arreglo de millones de celdas sensibles a la luz que detectan y transmiten la información que se utiliza para crear una imagen. Lo hacen convirtiendo la atenuación variable de las ondas de luz en señales, pequeñas ráfagas de corriente eléctrica que transmiten la información. Las ondas pueden ser luz u otra radiación electromagnética. Como se menciona en [9] los sensores de imagen se utilizan en dispositivos de imágenes electrónicos que incluyen cámaras digitales, módulos de cámara, teléfonos con cámara, dispositivos de ratón óptico, equipos de imágenes médicas, equipos de visión nocturna, radares o sonares. A continuación, se muestra un ejemplo de sensor de imagen (Figura 2.7), usado en sistemas embebidos de la marca Raspberry, comúnmente utilizado en proyectos de visión por computadora, que utiliza la interfaz *Camera Serial Interface* (CSI) de comunicación.



Figura 2.7: Sensor de Imagen.

La interfaz de comunicación CSI, sirve para comunicarse con un modulo de cámara en sistemas en chip como lo son las tarjetas de la familia Raspberry, esta interfaz es una especificación de la alianza de interfaces de procesador de la industria móvil por sus siglas en inglés MIPI. Las últimas especificaciones mencionadas en [10] de interfaz activas son CSI-2 v3.0, CSI-3 v1.1 y CCS v1.0, que se lanzaron en 2019, 2014 y 2017 respectivamente.

El propósito de esta interfaz es la de estandarizar la conexión de módulos de cámaras a procesadores para la industria de la telefonía móvil entre otros dispositivos. El conector CSI consta de dos interfaces pequeñas; la primera interfaz es para la transferencia de señales de reloj y datos desde la cámara al procesador en una sola dirección, la segunda interfaz consta de líneas de control bidireccional.

Sensores de distancia.

En este rubro nos encontramos con sensores como los medidores de distancia ultrasónicos los cuales consisten de un emisor de pulsos ultrasónicos (en el rango de 38 KHz a 50 KHz de acuerdo a [8]), y un módulo de medición que espera a que los pulsos vuelvan al sensor al rebotar con alguna superficie, el tiempo de espera es medido y se hace un cálculo el cual da como resultado la distancia entre el emisor y el objeto o superficie que produjo el rebote del pulso. El alcance de este tipo de sensores puede ser de hasta 6 metros. Un ejemplo se puede apreciar en la Figura 2.8.



Figura 2.8: Sensor Ultrasónico [8].

Otro tipo de sensores de este tipo son los de haz infrarrojos, su funcionamiento es similar al de los ultrasónicos, no obstante, en este caso se utiliza un emisor de haces infrarrojos que rebotan e inciden en un arreglo de dispositivos de carga acoplada (CCD) los cuales son capaces de detectar las señales infrarrojas para así determinar la distancia hacia objetos en el campo de visión del robot.

Sensores de posición.

Dentro de los sensores de posición podemos encontrar los acelerómetros; dispositivos con la capacidad de medir el movimiento, los cuales actualmente y de acuerdo a [8], están fabricados en forma de chips integrados con los elementos sensibles sobre el propio microcircuito, pues antaño se utilizaban imanes, resortes y bobinas para su construcción. Uno de los acelerómetros más conocidos es el ADXL202 (Figura 2.9) de dos ejes, de bajo consumo y salida digital, integrado en un chip monolítico el cual puede medir aceleración dinámica, por ejemplo una vibración y también aceleración estática, por ejemplo, la atracción de la gravedad.

Existen también dispositivos utilizados para medir la orientación y la velocidad angular de un dispositivo como un robot móvil, llamados giroscopios. Dichos dispositivos tradicionalmente se basan en una rueda o disco giratorio en el que el eje de rotación (eje de giro) es libre de asumir cualquier orientación por sí mismo. Al girar, la orientación de este eje no se ve afectada por la inclinación o rotación del soporte, de acuerdo con la conservación del momento angular.

Actualmente se han desarrollado giroscopios de tamaño muy pequeño similar al de un chip de montaje superficial cuyo principio operativo son pequeñas lenguetas vibratorias construidas sobre un chip de silicio los cuales vienen empaquetados en microchips que se encuentran en dispositivos electrónicos, estos giroscopios son capaces de medir la rotación en uno, dos o hasta tres ejes. Como se menciona en [11] los giroscopios similares al anterior se pueden utilizar para determinar la orientación y se encuentran en la mayoría de los sistemas de navegación autónoma. Por ejemplo, si se quiere balancear un robot, el giroscopio se puede utilizar para medir la rotación de la posición balanceada en tres ejes (x, y, z) y enviar las correcciones a un motor. En la Figura 2.10 podemos apreciar un diagrama de dicho tipo de giroscopio.

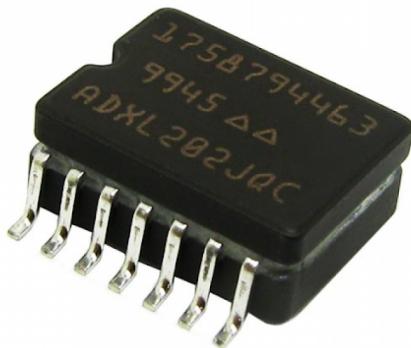


Figura 2.9: Acelerómetro en chip [8].

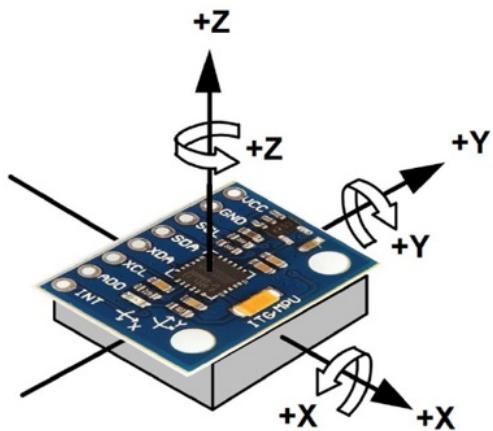


Figura 2.10: Giroscopio en chip [8].

De acuerdo a [12] forma principal para usar un giroscopio se efectúa mediante una interfaz de comunicación, esta puede ser digital o analógica. Los giroscopios con una interfaz digital por lo general usan la Interfaz Periférica Serial (SPI) o Circuito Inter-Integrado (I2C), el uso de estas interfaces permite una conexión fácil a un microcontrolador o sistema embebido. Una de las limitaciones de una interfaz digital es la velocidad de muestreo máxima. I2C tiene una frecuencia de muestreo máxima de 400 Hz, por otro lado, SPI puede tener una frecuencia de muestreo mucho más alta.

Actuadores o motores de un RM Terrestre

En relación al sistema de actuadores, a tenor de [5], este tiene como tarea generar el movimiento de los elementos del robot según las órdenes dadas por la unidad de control. Existen diversos tipos de motores entre los cuales destacan los siguientes: de corriente directa (CD), de corriente alterna (CA), a pasos, de inducción, servo motor; siendo el motor de CD el de mayor uso. El uso de los motores de CD se debe principalmente a su mayor grado de flexibi-

lidad para el control de la velocidad, debido a que su modelo asociado tiene la característica de ser lineal, lo que permite que se facilite su manejo en comparación con los de CA donde su control es más complejo.

En la práctica y como se discute en [13] los motores de CD suelen ser acoplados con reductores en un solo artefacto llamándose estos motorreductores. Los motorreductores tienen la capacidad de entregar un torque elevado a bajas velocidades, ya que el reductor funciona como un multiplicador de torque y puede permitir que motores pequeños generen velocidades más altas.

Una forma muy común de manejar un motor de corriente directa es a través de algo que se conoce como “Modulación por Ancho de Pulso” o PWM por sus siglas en inglés, dicho método consiste en aplicar pulsos eléctricos de distintos anchos hacia el motor en frecuencias muy elevadas, con la finalidad de emular un voltaje constante. Como se menciona en [14] el tiempo que un pulso está en un estado dado, ya sea alto o bajo, es el ancho de una onda de pulso, si el pulso es elevado el 50% del tiempo, se le llama ciclo de trabajo del 50%. Un dispositivo manejado de esta forma acaba comportándose como si recibiera de manera constante el promedio en voltaje de los pulsos.

Tomando como ejemplo un motor de ventilador, si se sabe que en alto el voltaje es 24 V, en bajo es de 0 V y el ciclo de trabajo es de 50%, entonces podemos determinar el voltaje promedio multiplicando el ciclo de trabajo por el nivel alto del pulso. Si se desea que el motor vaya más rápido, se incrementa el ciclo de trabajo. Cuanto mayor sea la frecuencia de los pulsos altos, mayor será el voltaje promedio y más rápido girará el motor del ventilador. Si se activa el pulso en alto y en bajo a intervalos iguales de 1 segundo, se obtiene una salida digital que impulsa el ventilador a un promedio constante de 12 V.

Incluso cuando se habla de un pequeño motor de 5 V de corriente directa, el pico inicial de corriente que necesita a la hora de ser accionado suele encontrarse entre los 300 mA a 400 mA, que disminuye hasta los 200 mA a medida que el motor gana velocidad. Esta es una corriente muy alta para dispositivos de bajo consumo como lo son microcontroladores o tarjetas Raspberry Pi, es por ello que existen controladores de motores; éstos toman las señales provenientes de un dispositivo como un microcontrolador y se encargan de proveer la corriente necesaria para accionar el motor.

Para accionar el giro de un motor en un sentido u otro es necesario un tipo especial y muy conocido de circuito llamado “Puente H”, el cual tiene dicho nombre por la forma que toma, como se aprecia en la Figura 2.11. Los controladores de motores en la actualidad se encuentran como circuitos integrados encapsulados que cuentan con un circuito de tipo Puente H.

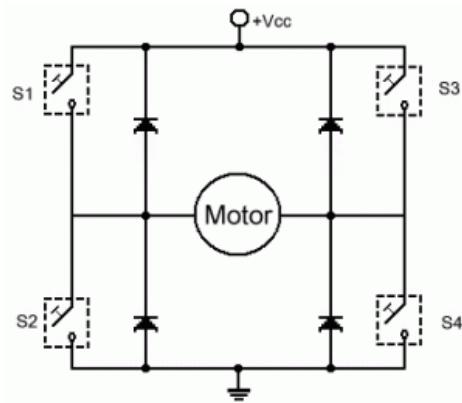


Figura 2.11: Puente H [15].

El motor colocado en el medio del circuito gira en una dirección cuando los interruptores S1 y S4 se encuentran cerrados y S2 junto con S3 abiertos, permitiendo a la corriente fluir a través del motor. Para hacer girar al motor en el sentido contrario se cierran S2 y S3, mientras que S1 y S4 se abren, como se aprecia en la Figura 2.12.

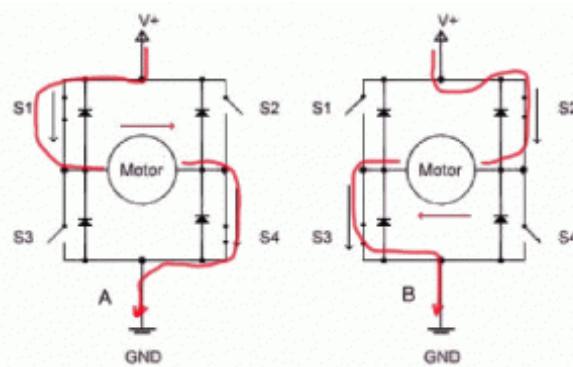


Figura 2.12: Funcionamiento del Puente H [15].

2.2. Sistemas embebidos

Un sistema embebido es un sistema computacional conformado por hardware y software, pero con restricciones de recursos. Pueden ser un producto final o pueden ser solo una parte incrustada dentro de un sistema mecánico o eléctrico más complejo. Utilizan típicamente un procesador general o microcontrolador junto con diversos periféricos y memoria en un solo circuito integrado o chip, a todo este conjunto se le conoce como Sistema en Chip (SoC). [16].

Las complejidades van desde un solo microcontrolador hasta un conjunto de procesadores con periféricos conectados y su complejidad varía significativamente según la tarea para la que está diseñado. La estructura básica de un sistema embebido incluye los siguientes componentes, de acuerdo a [17].

- **Sensor.** El sensor mide y convierte la cantidad física deseada en una señal eléctrica, que luego puede ser leída por cualquier instrumento electrónico.
- **Convertidor Analógico Digital.** Un convertidor de analógico a digital convierte la señal analógica enviada por el sensor en una señal digital.
- **Convertidor Digital Analógico.** Un convertidor de digital a analógico cambia los datos digitales alimentados por el procesador a datos analógicos.
- **Actuador.** Los actuadores son dispositivos que llevan incorporado un motor eléctrico y un reductor que permite accionar cualquier dispositivo para llevar a cabo determinado movimiento u acción.

Los rasgos que caracterizan a un sistema embebido son mencionados a continuación.

- **Funcionalidad dedicada.** Los sistemas embebidos suelen ejecutar un único programa repetidamente o un conjunto de programas para una aplicación específica.
- **Restricciones de diseño.** Son pequeños en tamaño, consumen poca energía y no son caros, comparados con una computadora convencional.
- **Reactivos en su entorno.** Los sistemas embebidos deben reaccionar continuamente a los cambios en el entorno del sistema.
- **Fiables.** Están diseñados para funcionar durante un tiempo prolongado por lo cual la fiabilidad es una clave fundamental a la hora de su construcción.

2.3. Comunicación en Red

La comunicación en red, o internetworking, define un conjunto de protocolos (es decir, reglas y estándares) que permiten que los programas de aplicación se comuniquen entre sí sin importar el hardware y los sistemas operativos donde se ejecutan. En [18] se menciona como la interconexión permite que los programas de aplicación se comuniquen independientemente de sus conexiones de red físicas, una característica altamente deseada en sistemas computacionales que pueden llegar a ser altamente heterogéneos, debido al surgimiento de diversas empresas de tecnología.

Wi-Fi

Según la Real Academia Española es un sistema de conexión inalámbrica, dentro de un área determinada, entre dispositivos electrónicos, y frecuentemente para acceso a internet. En un sentido literal Wi-Fi no significa nada, es una marca comercial que también es usada para designar a las tecnologías de red sin cable.

En la actualidad es la que ofrece la mayor cantidad de beneficios al costo más bajo entre las tecnologías inalámbricas. Es económica, ínter-operable con equipos de diferentes fabricantes y puede ser extendida para ofrecer funcionalidades mucho más allá de las previstas originalmente por los fabricantes.

Los estándares inalámbricos son la base de muchos productos inalámbricos, lo que asegura su interoperabilidad y su usabilidad por parte de los que desarrollan, instalan y gestionan redes inalámbricas. Los estándares usados en la mayoría de las redes, mencionados en [19], fueron establecidos por los grupos de trabajo 802 del Instituto de Ingernieros Eleéctricos y Electrónicos (IEEE). Los equipos de la familia de estándares 802.11 son los más fabricados e instalados para comunicaciones inalámbricas.

Las redes Wi-Fi poseen una serie de ventajas, entre las cuales podemos destacar:

- Al ser redes inalámbricas, la comodidad que ofrecen es muy superior a las redes cableadas porque cualquiera que tenga acceso a la red puede conectarse desde distintos puntos dentro de un espacio lo bastante amplio.
- Una vez configuradas, las redes Wi-Fi permiten el acceso de múltiples dispositivos sin ningún problema ni gasto en infraestructura, ni gran cantidad de cables.
- Asegura que la compatibilidad entre dispositivos es total, con lo que en cualquier parte del mundo podremos utilizar la tecnología Wi-Fi con una compatibilidad absoluta.

Protocolo TCP/IP

Un protocolo de comunicación en red es un conjunto de reglas o estándares que cada miembro dentro de una red debe seguir para permitir que otros miembros reciban e interpreten los mensajes que se les envían. Hay dos tipos generales de protocolos de transporte los orientados y no orientados a conexión.

Un protocolo no orientado a conexión es un protocolo que trata cada datagrama como independiente de todos los demás en la que cada datagrama debe contener toda la información requerida para su entrega. Un ejemplo de dicho protocolo es el Protocolo de Datagramas de Usuario o UDP por sus siglas en inglés. UDP es un protocolo construido directamente sobre la capa de Protocolo de Internet (IP) y utilizado para programas de aplicación a aplicación. UDP no garantiza la entrega de datos; cualquier error en la comunicación como datagramas que no llegaron a su destino o datagramas duplicados debe ser manejado por el programador y, por lo tanto, no se le suele considerar confiable a menos que se implementen los mecanismos necesarios.

Por otro lado, un protocolo orientado a la conexión requiere que los miembros establezcan una conexión lógica entre sí antes de que pueda tener lugar la comunicación, esta conexión a veces se denomina circuito virtual, aunque el flujo de datos real utiliza una red de commutación de paquetes. Un intercambio orientado a la conexión incluye tres fases: iniciar la conexión, transferir datos y terminar la conexión.

Un ejemplo de tal protocolo es el Protocolo de Control de Transmisión o TCP, el cual proporciona un vehículo confiable para entregar paquetes entre hosts en Internet. TCP divide un flujo de datos en datagramas, envía cada uno individualmente usando IP y vuelve a ensamblar los datagramas en el nodo de destino, si algún datagrama se pierde o se daña durante la transmisión TCP lo detecta y retransmite los datagramas faltantes automáticamente sin intervención del programador.

Arquitectura Cliente - Servidor

En [20] se expone la arquitectura cliente-servidor la cual es una estructura de aplicación distribuida que divide las tareas o cargas de trabajo entre los proveedores de un recurso o servicio, llamados servidores, y los solicitantes de servicios, llamados clientes. Es común que los clientes se encuentren en sectores de red diferentes a los servidores, aunque tampoco es extraño encontrarlos juntos en una red local. Un servidor ejecuta uno o más programas que ofrecen recursos a los clientes que soliciten acceso a ellos. Los clientes, por lo tanto, inicián sesiones de comunicación con los servidores, que esperan las solicitudes entrantes, algunos ejemplos de aplicaciones que utilizan el modelo cliente-servidor son el correo electrónico, la impresión en red y la World Wide Web.

Aplicación Web

Una aplicación web es un programa que utiliza un navegador web para realizar una función en particular. Las aplicaciones web están presentes en muchos sitios web. Un ejemplo simple es un formulario de registro para crear una cuenta en un sitio web. Como se discute en [21] una aplicación web utiliza la arquitectura cliente-servidor, en donde el término cliente se refiere al programa que el individuo usa para ejecutar la aplicación, por ejemplo; en el caso de un formulario de registro, el cliente es el programa a través del cual el usuario ingresa datos, comúnmente un buscador web como Google Chrome, y el servidor es el programa que almacena la información en una base de datos.

El programa del lado del servidor que se ocupa de almacenar y recuperar la información requiere lenguajes como Python o Java para su programación aunque existen muchos más, todos con distintas características. El programa del lado del cliente requiere lenguajes como; JavaScript para la creación de páginas web dinámicas y hojas de estilo en cascada (CSS) para describir la presentación de un documento escrito en un lenguaje de marcado como HTML5.

Protocolo HTTP

El Protocolo de Transferencia de Hipertexto (HTTP), es un protocolo que permite la obtención de recursos a través de la Web, es la base de cualquier intercambio de datos en la Web bajo una arquitectura cliente-servidor, lo que significa que las solicitudes las inicia el cliente que al hacer estas solicitudes comúnmente se reconstruye un documento completo a partir de los diferentes subdocumentos requisitados por las múltiples solicitudes, por ejemplo: texto, imágenes, videos y más. Los clientes y los servidores se comunican mediante el intercambio de mensajes individuales, con un flujo de datos orientado a conexión subyacente llevado a cabo en la mayoría de los casos por TCP/IP. Los mensajes enviados por el cliente, generalmente un navegador web, se denominan *solicitudes* y los mensajes enviados por el servidor como respuesta se denominan *respuestas*.

Diseñado a principios de la década de 1990, HTTP es un protocolo extensible que ha evolucionado con el tiempo. Es un protocolo de capa de aplicación que se envía por TCP o por una conexión TCP encriptada con Seguridad de la Capa de Transporte (TLS), aunque teóricamente se podría utilizar cualquier protocolo de transporte confiable. Debido a su extensibilidad, se utiliza no solo para buscar documentos de hipertexto, sino también imágenes y videos o para publicar contenido en servidores.

Los aspectos generales de HTTP discutidos en [22], suelen ser los siguientes:

- **Simple.** HTTP está diseñado para ser simple y legible por humanos, incluso con la complejidad adicional introducida en HTTP/2 al encapsular los mensajes HTTP en marcos. Los mensajes HTTP pueden ser leídos y comprendidos por humanos, lo que facilita las pruebas para los desarrolladores y reduce la complejidad para los que lo están aprendiendo.
- **Extensible.** Introducido en HTTP/1.0, los encabezados HTTP hacen que este protocolo sea fácil de extender y experimentar. Incluso se puede introducir una nueva funcio-

nalidad mediante un simple acuerdo entre un cliente y un servidor sobre la semántica de un nuevo encabezado.

- **HTTP no tiene estado.** No existe un vínculo entre dos solicitudes que se llevan a cabo sucesivamente en la misma conexión. Esto inmediatamente tiene la perspectiva de ser problemático para los usuarios que intentan interactuar con ciertas páginas de manera coherente, por ejemplo, utilizando sitios de comercio electrónico. Pero mientras que el núcleo de HTTP en sí mismo no tiene estado, las cookies HTTP permiten el uso de sesiones con estado. Mediante la extensibilidad del encabezado, las cookies HTTP se agregan al flujo de trabajo, lo que permite la creación de sesiones en cada solicitud HTTP para compartir el mismo contexto o el mismo estado.
- **HTTP y conexiones.** Antes de que un cliente y un servidor puedan intercambiar mensajes de solicitud/respuesta HTTP, deben establecer una conexión TCP, un proceso que requiere de varios pasos. El comportamiento predeterminado de HTTP/1.0 es abrir una conexión TCP separada para cada par de mensajes solicitud/respuesta HTTP. Esto es menos eficaz que compartir una sola conexión TCP cuando se envían varias solicitudes en sucesión cercana. Para mitigar esta falla, HTTP/1.1 introdujo *pipelining* y conexiones persistentes; la conexión TCP subyacente se puede controlar parcialmente mediante el encabezado *Connection*. HTTP/2 fue un paso más allá al multiplexar mensajes a través de una sola conexión, lo que ayudó a mantener una conexión más eficiente.

2.4. Procesamiento digital de imágenes

El hombre y sus sentidos

De los cinco sentidos - vista, oído, tacto, olfato y gusto - la visión es indudablemente del que el ser humano ha llegado a depender por encima de todos los demás, y de hecho el que proporciona la mayoría de los datos que recibe. La vía de entrada de los ojos no sólo proporcionan mega-bits de información en cada mirada, sino que las velocidades de datos para la visión continua probablemente superen los 10 mega-bits por segundo ($\frac{mbits}{s}$). Sin embargo, mucha de esta información es redundante y está comprimida por las diversas capas de la corteza visual, de modo que los centros superior del cerebro tiene que interpretar de forma abstracta sólo una pequeña fracción de los datos.

No obstante, la cantidad de información que los centros superiores del cerebro reciben de los ojos debe ser al menos dos órdenes de magnitud más grandes que toda la información que obtiene los otros sentido. El sistema visual humano tiene una increíble facilidad de interpretación, ya que se percibe la estructura tridimensional del mundo con aparente sencillez. No se necesitan deducciones obvias ni se requiere un esfuerzo manifiesto para interpretar cada escena: las respuestas son efectivamente inmediatas y normalmente están disponibles en una décima de segundo [23].

Desde el comienzo de la ciencia, la observación ha jugado un papel importante, ya que anteriormente la única manera de documentar los resultados de un experimento era mediante la descripción verbal y dibujos manuales. Conforme fue avanzando el tiempo y la tecnología, uno de los grandes pasos de la humanidad fue **la invención de la fotografía** que permitió documentar los resultados de forma objetiva.

Imagen digital

Una fotografía o imagen puede ser definida como una función bidimensional, $I(x, y)$, donde x e y son las coordenadas de un plano que contiene todos los puntos de la misma, cuyos elementos se le conocen como *puntos elementales de la imagen*, pels o *píxeles* siendo este último el término utilizado comúnmente para denotar **la unidad mínima de medida de una imagen digital**. Cuando x , y y los valores de intensidad de la función I son todos discretos y finitos, le llamamos a la imagen *una imagen digital* [24], la cual se representa a continuación:

$$I(x, y) = \begin{bmatrix} I(0, 0) & I(0, 1) & \cdots & I(0, N - 1) \\ I(1, 0) & I(1, 1) & \cdots & I(1, N - 1) \\ \vdots & \vdots & \ddots & \vdots \\ I(M - 1, 0) & I(M - 1, 1) & \cdots & I(M - 1, N - 1) \end{bmatrix} \quad (2.1)$$

donde M son las filas y N son las columnas, cuyos valores de $x = 0, 1, 2, \dots, M - 1$ e $y = 0, 1, 2, \dots, N - 1$.

En la figura 2.13 se presenta una imagen con 256 niveles de intensidad. En ella, cada uno de los píxeles está representado por un número entero que es interpretado como el nivel de intensidad luminosa en la escala de grises.

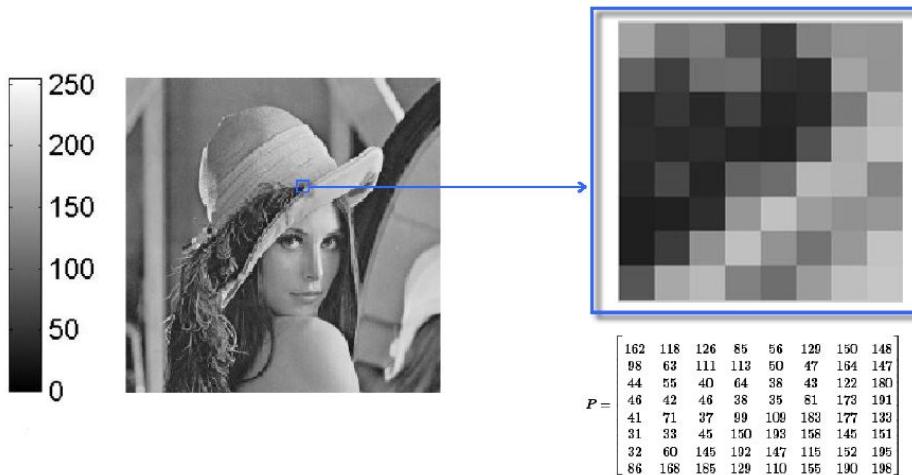


Figura 2.13: Imagen con 256 niveles de intensidad y representación numérica de un fragmento de 8x8 [25]

Análisis y procesamiento de imágenes

El manejo de las imágenes digitales se ha convertido en las últimas décadas en un tema de interés extendido en diferentes áreas de las ciencias naturales, las ciencias médicas y las aplicaciones tecnológicas, entre otras. Anteriormente, las posibilidades de los equipos de captura y procesamiento digital eran bastante limitadas y, los costos y tiempos de procesamiento prohibitivos. Ante lo cual en muy pocas áreas se prestaba atención al potencial que las herramientas para el manejo de imágenes digitales ofrecían, como lo menciona [26]. El crecimiento en el poder de cómputo, las capacidades de almacenamiento y los nuevos sistemas de despliegado, captura e impresión de bajo costo han facilitado el desarrollo de ésta disciplina.

En la actualidad es posible explotar plataformas de bajo costo y obtener resultados de gran calidad para crear aplicaciones de gran utilidad, versátiles y flexibles, así como aplicaciones de software de propósito específico para atender las diversas necesidades de los especialistas para utilizar el **Análisis Digital de Imágenes**.

El **Análisis Digital de Imágenes** es el área de la ingeniería que se encarga de la extracción de mediciones, datos o información contenida en una imagen. Incluye aquellas técnicas cuyo principal objetivo es facilitar la búsqueda e interpretación de la información contenida en ellas, así lo describe [25]. El análisis y procesamiento de imágenes ha sido desarrollado en respuesta a los tres mayores problemas concernientes con imágenes [27]:

- Digitalización y codificación de imágenes para facilitar la transmisión, la impresión y el almacenamiento de las mismas.
- Mejora y restauración de imágenes para, por ejemplo, interpretar más fácilmente las imágenes de la superficie de otros planetas tomadas por diversas sondas.

- La segmentación y descripción de imágenes para aplicaciones de visión robótica y visión artificial.

Componentes de un sistema de procesamiento de imágenes

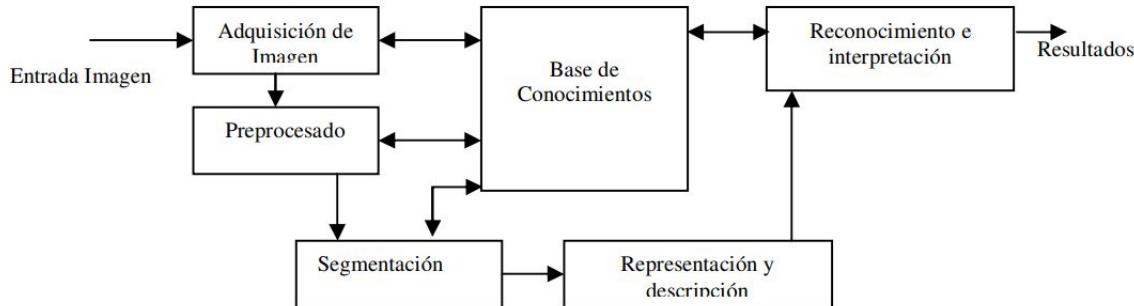


Figura 2.14: Etapas involucradas en el procesamiento de imágenes [28]

Un sistema de procesamiento digital de imágenes esta compuesto por dos etapas fundamentales: etapa de adquisición y etapa de procesamiento de imágenes. La primera de ellas consta de un lente, un sensor y una tarjeta o microcontrolador.

1. Lente: En el caso más simple, este podría ser una cámara CCD, un escáner de cama plana o una grabadora de vídeo. requiere que las imágenes sean obtenidas en forma de señales eléctricas. Estas señales pueden ser digitalizadas dentro de secuencias de números que luego pueden ser procesadas por una computadora.
2. Sensor: Un dispositivo capaz de convertir la señal eléctrica (normalmente una señal de vídeo analógica) obtenido del lente, en una imagen digital que puede ser procesada y almacenada, permitiendo así una visualización de imágenes en escala de grises con hasta 256 sombras (8 bits) e imágenes en color verdadero con hasta 16,7 millones de colores (3 canales con 8 bits cada uno).
3. Tarjeta: Es una estación de trabajo (*workstation*) que proporcione el poder de procesamiento para almacenar múltiples imágenes, suficiente para manejar tareas complejas de procesamiento de imágenes.

Durante la segunda etapa, se realiza el procesamiento digital de imágenes, cuyas tareas son la caracterización e interpretación de información de imágenes tomadas de un mundo tridimensional. Estos procesos a su vez están subdivididos en cinco áreas principales como se observa en la Figura 2.14: **Adquisición, Preprocesado, Segmentación, Representación y descripción, y Reconocimiento e interpretación**.

Existe una descripción clara con respecto a cada una de estas áreas:

La **adquisición** es el proceso a través del cuál se obtiene una imagen. El **preprocesado** incluye técnicas tales como el suavizado y realce de detalles y/o bordes. La **segmentación** es el proceso que divide una imagen en objetos de interés.

Mediante los procesos de **descripción** se obtienen características (tamaño, perímetro, etc) convenientes para diferenciar un objeto de otros. El **reconocimiento** es el proceso que identifica los objetos (i.e. llave inglesa, retén, arandela, etc). Finalmente, la **interpretación** le asocia un significado a un conjunto de objetos reconocidos. Generalmente es conveniente agrupar estas áreas de acuerdo con la complicación y el grado de detalle que lleva aparejada su implementación [28].

El conjunto de métodos de procesamiento de imágenes está dividido en tres grandes grupos, los cuales se listan a continuación:

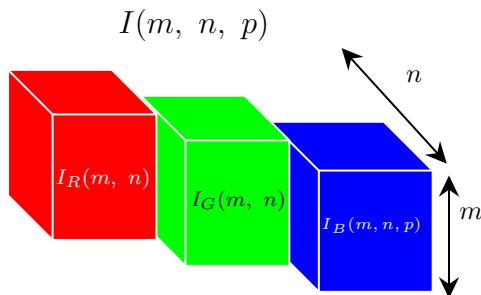
- **Algoritmos en el dominio espacial:** Se refiere a métodos que procesan una imagen píxel por píxel, o también tomando en cuenta un conjunto de píxeles vecinos.
- **Algoritmos en el dominio de la frecuencia:** Frecuentemente, estos métodos son aplicados sobre los coeficientes resultantes de la *Transformada de Fourier* de una imagen.
- **Algoritmo de extracción de características:** A diferencia de los dos grupos anteriores, estos algoritmos están enfocados al análisis de imágenes para la extracción de atributos y regiones de interés, separación de objetos del fondo, detección de bordes o formas, entre otros.

Es importante mencionar que el desarrollo de este trabajo se dará, en su mayoría, utilizando algoritmos en el dominio espacial como los implementados para la binarización y la segmentación de la imagen, así como el ajuste de perspectiva de la misma, cuyas operaciones se realizan en los píxeles y regiones específicas de la imagen. Dichas implementaciones se realizan sobre un sistema en chip.

Adquisición

Imágenes a color

El fundamento para describir una imagen digital a color es el mismo que el expuesto en la sección 2.4, con la salvedad de que cada elemento o píxel es descrito y codificado de otra forma, según el espacio de color que esté utilizando. De esta forma, para un espacio de color RGB (generalmente el más usado para representar imágenes), se representa cada píxel como un color creado a partir de ciertas cantidades de los colores rojo, verde y azul [29]. Esta representación se puede interpretar como una matriz de tres niveles de intensidad, donde cada nivel corresponde a la intensidad de color de las componentes rojo, verde y azul, como es mostrado en la Figura 2.15.



$$I_R(m,n,1) = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \dots & r_{mn} \end{bmatrix} I_G(m,n,2) = \begin{bmatrix} g_{11} & g_{12} & \dots & g_{1n} \\ g_{21} & g_{22} & \dots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{m1} & g_{m2} & \dots & g_{mn} \end{bmatrix} I_B(m,n,3) = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix}$$

Figura 2.15: Planos de color RGB representados como tres matrices bidimensionales [25]

El color resultante del píxel vendrá por tanto definido por la *cantidad* de intensidad que tenga cada componente. Así, el color blanco estará compuesto de la máxima intensidad de color para los tres componentes. De forma contraria, el color negro será el resultado de reducir al mínimo la intensidad de los componentes.

Espacio de color HSV

El modelo HSV (del inglés Hue, Saturation, Value, esto es, Tonalidad, Saturación, Valor), establece un modelo de color en términos de sus componentes constituyentes. La representación de dicho modelo coordenado es cilíndrico, y el subconjunto de este espacio donde se define el color es una pirámide de base hexagonal.

En el modelo HSV los colores más brillantes están contenidos en el área hexagonal correspondiente a $V=1$. Para medir el tono, se usa el ángulo alrededor del eje S. El rojo se sitúa a 0° , el verde a los 120° y el azul a los 240° . Los colores complementarios se encuentran a 180° de su color primario. El rango de S se extiende desde 0, situado en el eje de la pirámide, donde se sitúan los colores más oscuros, hasta 1, coincidiendo con el final del área hexagonal de la pirámide.

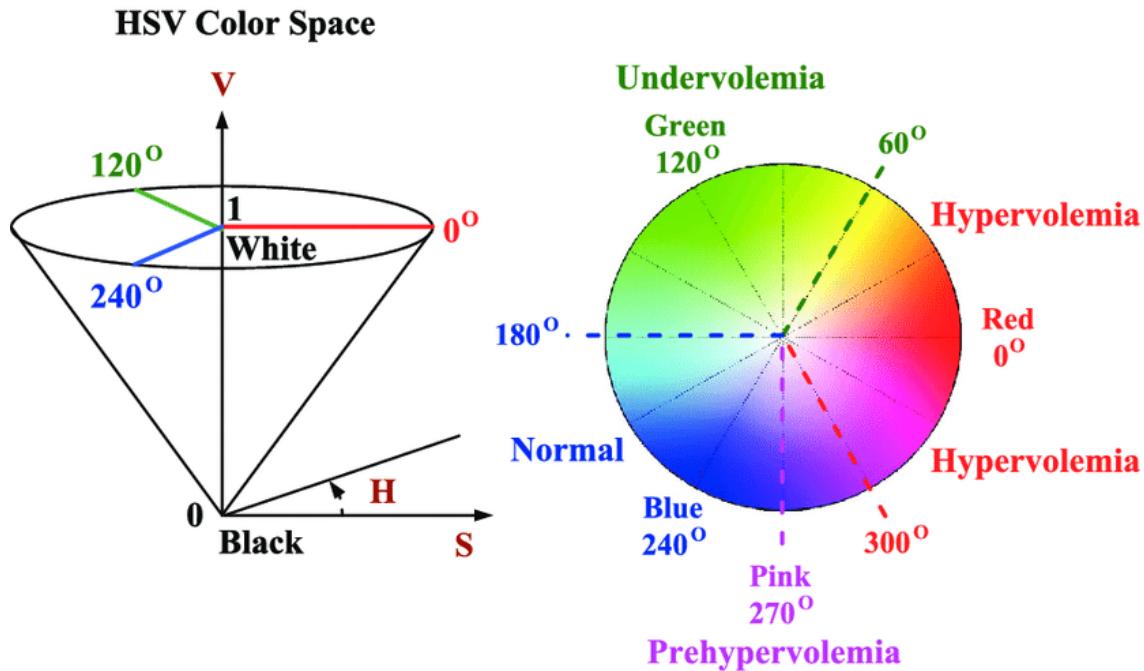


Figura 2.16: Modelo HSV

El vértice corresponde al negro con coordenadas $S=0$ y $V=0$. El blanco corresponde a $S=0$ y $V=1$. Los valores que se encuentran en el eje de la pirámide son los grises. Cuando $S=0$ el valor de H no está definido. Sin embargo, a medida que S va creciendo, el valor de H comienza a tener importancia. Por ejemplo, el rojo puro se sitúa a $H=0$, $S=1$ y $V=1$. Si se añade blanco disminuye S , pero no cambia el valor de V . Las sombras se crean manteniendo $S=1$ y disminuyendo V . De esta forma, el espacio HSV tiene la ventaja de ser invariante a las condiciones de luz.

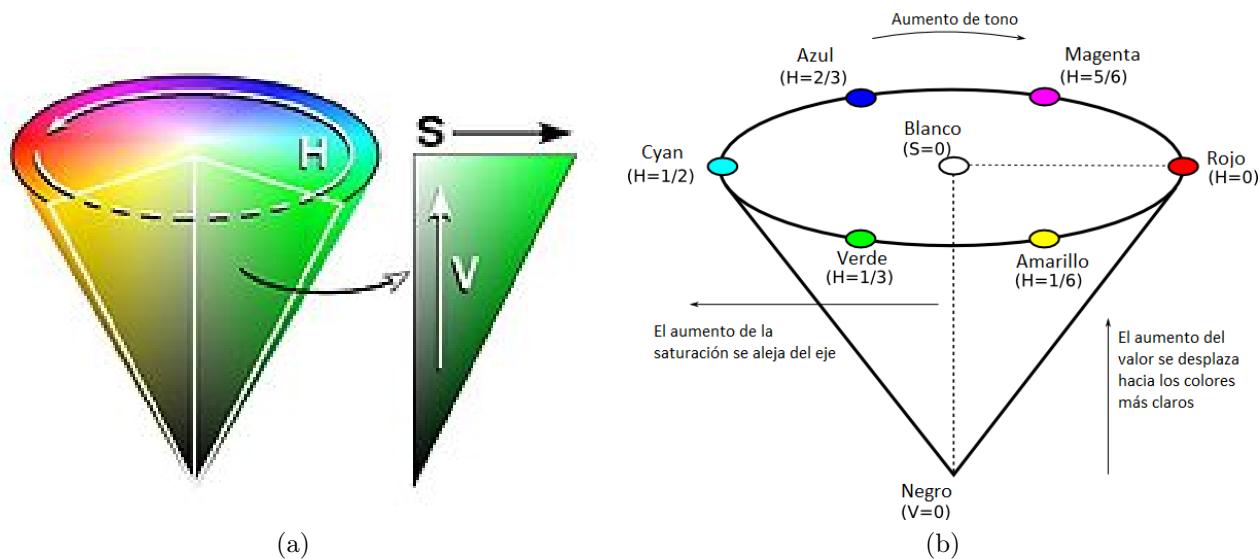


Figura 2.17: Representacion grafica del modelo HSV

Resolución y tamaño

Como se menciona en [30], el tamaño de una imagen es determinada directamente por el *ancho N* (número de columnas) y el *alto M* (número de filas) de la matriz imagen I . La resolución de una imagen especifica las dimensiones espaciales de la imagen en el mundo real y es dada como el número de elementos de la imagen por medición. En la mayoría de los casos, la resolución de una imagen es la misma en las direcciones vertical y horizontal, lo cuál significa que los elementos de la imagen son cuadrados.

Los valores de los píxeles son prácticamente siempre elementos binarios de longitud k tal que un píxel puede representar cualquiera valor diferente de 2^k . El valor de k es llamado el bit de profundidad (o solamente “profundidad”) de la imagen. La disposición exacta del nivel de bits de un píxel individual depende del tipo de imagen, por ejemplo: binaria, a escala de grises o color RGB. Las propiedades de algunos tipos de imágenes comunes se resumen en la Tabla 2.1

Escala de grises (Intensidad):

Cadena	Bits/Pix.	Rango	Uso
1	1	[0, 1]	Imágenes binarias: documentos, ilustración, fax.
1	8	[0, 255]	Universal: Fotos, escáner, impresión.
1	12	[0, 4095]	Alta calidad: Foto, escáner, impresión.
1	14	[0, 16383]	Profesional: Foto, escáner, impresión.
1	16	[0, 65535]	Mayor alta calidad: Medicina, Astronomía

Imágenes a color:

Cadena	Bits/Pix.	Rango	Uso
3	24	[0, 255] ³	RGB, universal: Foto, escáner, impresión.
3	36	[0, 4095] ³	RGB, alta calidad: Foto, escáner, impresión.
3	42	[0, 16383] ³	RGB, profesional: Foto, escáner, impresión.
3	32	[0, 255] ⁴	CMYK, preimpresión digital

Imágenes especiales:

Cadena	Bits/Pix.	Rango	Uso
1	16	[-32768, 32767]	Valores enteros pos./neg., rango incrementado
1	32	$[\pm 3.4 \cdot 10^{38}]$	Valores de punto flotante: medicina, astronomía
1	64	$[\pm 1.8 \cdot 10^{308}]$	Valores de punto flotante: procesamiento interno

Tabla 2.1: Profundidades de bits de tipos de imágenes comunes y dominios de aplicación típicos [30]

Compresión y formatos de imágenes

La compresión de información consiste en la reducción del volumen de información tratable. En un principio, con la compresión se pretende transportar la misma información, pero empleando la menor cantidad de espacio [31].

Formato de imagen	Características
BMP (Bitmap - Mapa de bits)	<ul style="list-style-type: none"> - No sufre pérdidas de calidad (compresión pobre) - No comprime el color real de la imagen - Alberga gran cantidad de información de la imagen - En ocasiones tiende a ser un archivo muy grande - Se forma mediante una parrilla de píxeles - Fue desarrollado por Microsoft
JPEG-JPEG (Joint Photographic Experts Group - Grupo de Expertos Fotográficos Unidos)	<ul style="list-style-type: none"> - Admite una paleta de colores mayores a la de GIF - Es el formato más común para publicar imágenes en la web - Implica una gran compresión de la imagen - Pierde la calidad de la imagen dependiendo del factor de compresión definida - Reduce el tamaño del archivo de la imagen - Permite una visualización aceptable
TIF - TIFF (Tagged Image File Format - Formato de Archivo de Imagen Etiquetada)	<ul style="list-style-type: none"> - Almacena imágenes de calidad excelente - Utiliza cualquier profundidad de color de 1-32 bits - Formato ideal para editar o imprimir una imagen - Ideal para archivar archivos originales - Produce archivos muy grandes

Tabla 2.2: Comparativa entre formatos de imagen

Preprocesamiento

Relación entre píxeles vecinos

Considerando lo que se menciona en [28], cuando se haga referencia a un píxel en particular se emplearán letras minúsculas, como p y q . Un píxel p de coordenadas (x, y) tiene cuatro vecinos, horizontales y verticales, cuyas coordenadas vienen dadas por:

$$(x + 1, y); (x - 1, y); (x, y + 1); (x, y - 1) \quad (2.2)$$

De forma gráfica, podemos visualizarlo en la Tabla 2.3

	$(x - 1, y)$	
$(x, y - 1)$	(x, y)	$(x, y + 1)$
	$(x + 1, y)$	

Tabla 2.3: Vecindad 4 entre píxeles [28]

Este conjunto de píxeles, denominado los 4-vecinos de p , se representa por $N_4(p)$. Cada píxel está a una unidad de distancia de (x, y) , y algunos de los vecinos de p caen fuera de la imagen digital si (x, y) está en el borde de la imagen. Mientras que los cuatro vecinos en diagonal de p tiene las siguientes coordenadas:

$$(x + 1, y + 1); (x + 1, y - 1); (x - 1, y + 1); (x - 1, y - 1) \quad (2.3)$$

y se representan por $N_D(p)$. Estos puntos, junto a los 4-vecinos, se denominan los 8-vecinos de p (Tabla 2.4), y se presenta por $N_8(p)$.

$(x - 1, y - 1)$	$(x - 1, y)$	$(x - 1, y + 1)$
$(x, y - 1)$	(x, y)	$(x, y + 1)$
$(x + 1, y - 1)$	$(x + 1, y)$	$(x + 1, y + 1)$

Tabla 2.4: Vecindad 8 entre píxeles [28]

El conjunto de píxeles vecinos al píxel actual, en (x, y) se le conoce comúnmente como *ventana* o *plantilla*. Las ventanas usadas en las operaciones locales no están limitadas solamente a los píxeles adyacentes ($N_4(p), N_D(p)$), es decir, no tiene por qué tener un tamaño de 3×3 forzosamente.

Tipos de transformaciones

A partir de la relación del píxel de salida con los vecinos del píxel actual, las transformaciones de una imagen de entrada en una imagen procesada pueden clasificarse de la siguiente manera:

- **Transformaciones puntuales:** Son aquellas en las cuales el píxel resultante de la operación depende sólo del valor del píxel de entrada. Las operaciones puntuales típicas incluyen la manipulación de los píxeles uno a uno, por ejemplo la binarización, la segmentación, la corrección de color, tono, saturación, gamma, etc.

- **Transformaciones locales:** En este caso, para obtener el píxel de salida, se utilizan las contribuciones de los píxeles vecinos en la operación. Muchas operaciones son locales, tales como: suavizado, media, operaciones morfológicas, realce de bordes. Se clasifican en filtros lineales, como la media, y los no lineales, como la mediana.
- **Transformaciones globales:** El píxel de salida como resultado de la operación, se obtiene a partir del total de datos de la imagen como valor de entrada. Las operaciones globales se realizan a menudo en el dominio de la frecuencia. Un ejemplo es la compresión de imágenes que tomando el total de una imagen de entrada obtiene una imagen comprimida de salida.
- **Transformaciones geométricas:** Se realizan tomando en cuenta las posiciones de los píxeles en la imagen, y se les aplica operaciones de translación y/o rotación, por ejemplo: translación, rotación, cambios de escala, rectificación y transformaciones radio-métricas de los píxeles.

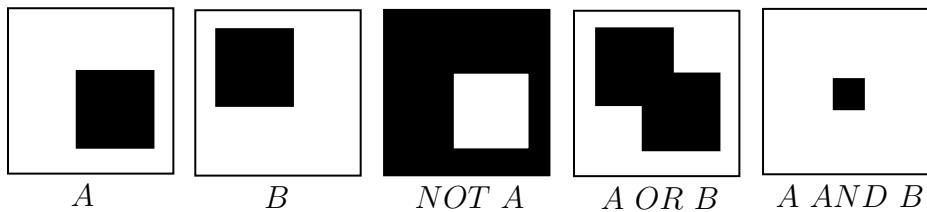


Figura 2.18: Operaciones lógicas sobre imágenes binarias [28]

Asimismo, si la imagen la cuál se va a tratar es binaria (en donde los píxeles toman los valores de '0' o '1' según pertenezcan al fondo o a objetos en primer plano), se define las transformaciones lógicas, en dónde el píxel de salida es el resultado de aplicar operadores lógicos (AND, OR, NOT, XOR) sobre dos imágenes binarias, como se muestra en la Figura 2.18.

Transformaciones espaciales

Como se menciona en [26], las operaciones espaciales se aplican directamente sobre píxeles de una imagen, éstas pueden ser operaciones orientadas al punto, las cuáles transforman a la imagen modificando un píxel a la vez, en general sin importar el estado de los píxeles vecinos, o pueden ser orientadas a la región, quiénes transforman a la imagen modificando un píxel a la vez y toman en cuenta, para dicha transformación, a los píxeles vecinos. La transformación se puede aplicar a toda la imagen o a una región de ella.

Transformación orientada al punto y a la región

Sea p un píxel de una imagen $I(x, y)$, es decir: $p \in I(x, y)$. Supóngase que la función $g(x, y) = T(I(x, y))$ transforma a una píxel p mediante la regla T , generando un nuevo valor para él, digamos q . Entonces diremos que la nueva imagen $g(x, y)$, donde $q \in g(x, y)$ es el producto de aplicar T sobre $I(x, y)$. Simbólicamente diremos que:

- Orientadas al punto

$$g(x, y) = T(I(x, y)) \quad (2.4)$$

- Orientadas a la región

$$g(x, y) = T(I(x + \alpha, y + \beta)), \text{ donde } \alpha, \beta \in \{-1, 0, 1\} \quad (2.5)$$

El proceso de transformación en la mayor parte de los casos será tal que:

$$\text{Si } p = I(i, j) \Rightarrow q = g(i, j), \text{ donde } q = T(p) \quad (2.6)$$

Al cambiar T la transformación será diferente. Si definimos la composición de transformaciones de la manera habitual, tendremos que:

$$T_1 \circ T_2(I(x, y)) = T_1(T_2(I(x, y))) \quad (2.7)$$

En general al aplicar dos transformaciones a una imagen en diferente orden, no se debe esperar que imagen resultante sea la misma, es decir, *la composición de transformaciones no es conmutativa*, simbólicamente tendremos que:

$$T_1 \circ T_2(I(x, y)) \neq T_2 \circ T_1(I(x, y)) \quad (2.8)$$

Batería de transformaciones

Se define una *batería* o *serie* de transformaciones T_k mediante la composición de ellas. Muchas de las operaciones de mejora de la imagen: detección de bordes, suavizado, etc., se definen como una batería. El sentido de ésta es similar a la composición de las funciones que generan cada transformación.

Sea T_1, T_2, \dots, T_n las funciones que definen cada proceso sobre la imagen, entonces la transformación compuesta o batería será:

$$\begin{aligned} B(I(x, y)) &= T_1 \circ T_2 \circ \dots \circ T_n(I(x, y)) \\ &= T_1(T_2(\dots T_{n-1}(T_n(I(x, y)))))) \end{aligned}$$

Gráficamente se puede representar el proceso de transformación múltiple mediante celdas, donde cada celda representa una transformación o *filtro*. La figura siguiente (Figura 2.19) ilustra la situación.

Segmentación

El proceso de segmentación se encarga de evaluar cada píxel de la imagen y decidir si contiene o no las características de interés. Como resultado, este método genera una imagen

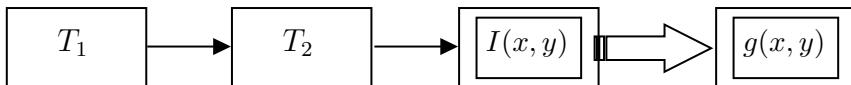


Figura 2.19: Representación gráfica de la composición de procesos [26]

binaria, donde los píxeles que pertenecen al objeto se representan con un '1' (objeto en primer plano), mientras que los que no pertenecen al mismo se representan con un '0' (fondo). La decisión de pertenencia de un píxel a uno u otro segmento se basa en el análisis de alguna característica de la imagen, como por ejemplo los niveles de intensidad o la textura [24].

Citando a [25], la mayoría de los algoritmos de segmentación están basados en dos propiedades básicas de intensidad de la imagen: la discontinuidad y la similitud. En la categoría de segmentación mediante discontinuidad, el proceso se realiza dividiendo la imagen por cambios abruptos en intensidad, como es el caso de la detección de bordes en una imagen. Con respecto a la segmentación con base en la similitud, ésta se logra mediante la partición de una imagen en regiones que son similares de acuerdo a un conjunto de criterios predefinidos.

Las técnicas de segmentación dependen fuertemente del objetivo que persigue la aplicación en particular, así como del tipo de imagen a analizar y sus características. Por lo tanto, en una etapa previa a la segmentación, es preciso tener claro qué objetos interesan y qué características poseen. También es común realizar operaciones de filtrado una vez terminada la etapa de segmentación, así como determinar las características que permitan separar y clasificar los objetos encontrados.

Segmentación basada en características del píxel

Se evalúa cada píxel en función de las características locales de la imagen en el píxel (y usualmente también sus vecinos), y se decide a qué región (también conocido como segmento) pertenece. Este tipo de segmentación se usa comúnmente cuando se requiere separar objetos con similares características de color o intensidad de un fondo heterogéneo. El caso ideal es aquel en el cual los objetos poseen un rango de colores o intensidad de gris muy estrecho, siendo el fondo uniforme. En tal caso se puede definir un umbral de segmentación para separar objetos del fondo. A esta técnica de asignación de un umbral se la conoce como thresholding (literalmente «umbralización») [25].

Segmentación basada en textura

En este enfoque se definen modelos de texturas para pensar en una imagen no como en una colección de pixeles, sino para tratarla como una función $I(x,y)$. El propósito del modelo es transformar una ventana de una imagen en una colección de valores que constituyen un vector de características. Este vector será un punto de un espacio n-dimensional. La representación corresponderá a la textura si ventanas tomadas de la misma muestra de textura están "cercaas" en el espacio de características, y si ventanas de la imagen con diferentes patrones de texturas quedan "alejadas" en el espacio de características considerado.

Los modelos de textura se dividen, a grandes rasgos, en tres categorías: basados en estructuras piramidales, que tratan de capturar las frecuencias espaciales a distintos niveles de resolución; basados en campos aleatorios, que asumen que los valores de un pixel son seleccionados mediante un proceso estocástico bidimensional; y los basados en métodos estadísticos, que utilizan matrices de co-ocurrencias construidas a partir de las imágenes y de estas ma-

trices se extraen una serie de medidas como la media, la varianza, la entropía, la energía y la correlación entre los pixeles. En la figura 2.20 podemos apreciar un ejemplo.

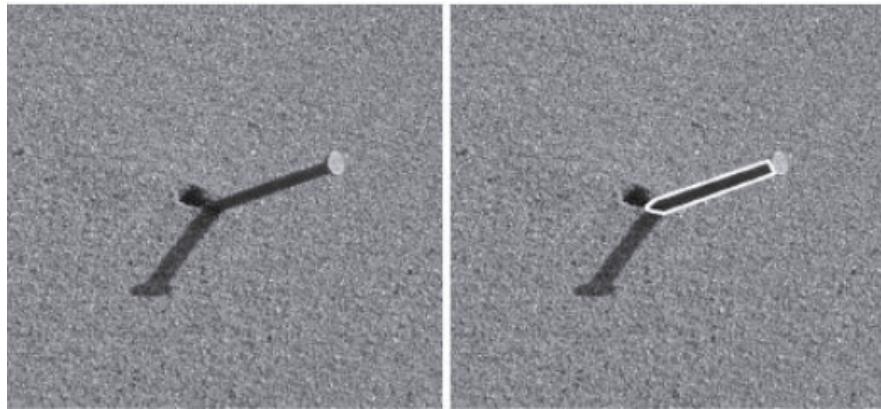


Figura 2.20: Ejemplo de segmentación de una imagen basada en textura [32].

Segmentación basada en movimiento

El movimiento puede constituir una potente herramienta para la segmentación de objetos animados sobre fondos estáticos. Las técnicas básicas consisten en el estudio de la imagen resultante de la resta de dos imágenes consecutivas de una secuencia animada. Esta técnica se conoce con el nombre de substracción de fondo. Los objetos que se desplazan entre estas dos imágenes producen en la imagen resta un conjunto de pixeles con valores distintos a cero. Mientras, los elementos estáticos de la imagen, por no variar, producen cero tras la resta.

2.5. Algoritmos de navegación robótica

Como se menciona en [33], la utilización de robots móviles en aplicaciones industriales y técnicas es amplia y continúa creciendo. La preferencia para emplear este tipo de dispositivos en el desempeño de una determinada función radica en dos puntos principalmente:

1. La fiabilidad: los robots móviles pueden ejecutar de forma confiable y continua labores monótonas como la vigilancia.
2. La accesibilidad: En ambiente peligrosos o en espacios estrechos son útiles para sustituir a los humanos; como es el caso del rescate de víctimas en un área de incendio.

Para interactuar de una manera óptima y correcta con el entorno, estos dispositivos autónomos pueden utilizar un conocimiento previo de éste o procesar la información entregada por sus sensores.

Descripción de la navegación

La navegación en un robot es el proceso de guiarlo a través de una trayectoria desde una posición a otra en una área de trabajo. Surgen entonces tres problemas por solucionar en relación a la navegación:

1. Auto-localización del robot: El robot recurre al conocimiento previamente adquirido y a sus observaciones locales, (por medio de *encoders* acoplados en sus llantas) y externas, para determinar la localización con relación a su entorno.
2. Identificación de la posición del objetivo con relación al entorno.
3. Planificación de la trayectoria que permita establecer un camino posible desde la posición inicial a la final.

Un robot puede navegar en dos tipos de ambientes. Los **entornos estructurados** que permanecen estáticos y no sufren cambios con el transcurso del tiempo, y los **entornos no estructurados** que son dinámicos y pueden presentar cambios inesperados en los elementos que lo conforman. Con lo anterior, dentro de ambientes desconocidos o no estructurados, el robot deberá desarrollar capacidad para reaccionar ante la presencia de imprevistos en su trayectoria.

Etapas de la navegación

Tomando como referencia a [33], para que un robot pueda ejecutar una tarea de navegación, que establezca una metodología para comandar su curso a lo largo de un entorno de obstáculos alcanzando posiciones intermedias hasta llegar a la posición de meta u objetivo, se demanda el cumplimiento de algunas etapas que al interrelacionarse se efectúa el **control de navegación** en el robot. Estas etapas se describen a continuación.

- **Percepción del entorno.** Esta etapa se lleva a cabo utilizando sensores, un mapa o un modelo del ambiente que se emplea para la navegación.

- **Planificación de la trayectoria.** Esta fase se basa en la descripción del entorno para establecer una secuencia ordenada de posiciones cartesianas intermedias hasta llegar a la posición objetivo o meta. La ruta diseñada debe garantizar que evada obstáculos cumpliendo con los requisitos de la tarea de navegación.
- **Generación de la trayectoria.** Ésta es la encargada de elaborar una función que interpola las posiciones establecidas por la planificación de trayectoria, y de discretizar a la misma para generar la trayectoria. En otras palabras se diseña una referencia que utiliza el seguidor de caminos.
- **Seguimiento de la trayectoria:** Es la encargada de efectuar el desplazamiento por medio de comandos de direccionamiento y velocidad, que actúan sobre los servo-controladores del robot, con base a la trayectoria generada.

Enfoques de la navegación

Para la navegación robótica, sus soluciones pueden estar enfocadas en dos categorías principalmente detalladas a continuación.

- **Navegación basada en mapa.** Esta navegación se lleva a cabo utilizando un mapa, que se define como una representación entendible por el robot del entorno. El mapa puede ser construido por el mismo robot o basado en los planos del área de trabajo. Al contar con un mapa del entorno, el robot puede planificar y generar una ruta posible desde la posición inicial a la posición objetivo. La ruta planificada se considera como una aproximación a la trayectoria que el robot deberá seguir, pues no considera los aspectos cambiantes del entorno. A esta planificación se le denomina **Planificación Global**. La estructura de navegación basada en mapa se describe en la Figura 2.21.

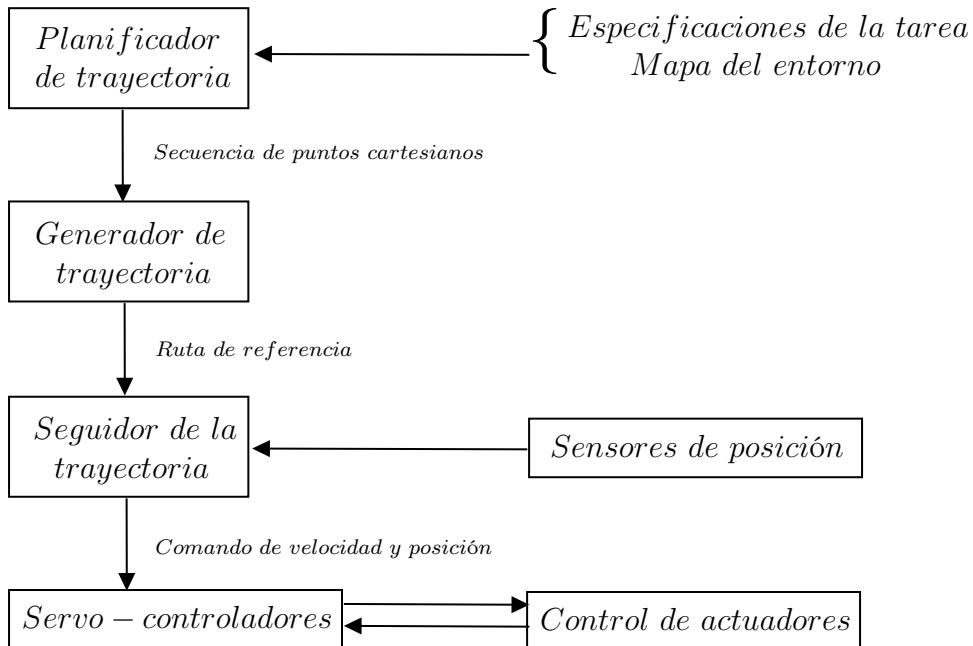


Figura 2.21: Estructura de navegación basada en mapa [33]

Como el robot se desplaza a lo largo del camino planificado en función de un conocimiento a priori del entorno, es necesario implementar una rutina adicional que evite colisiones entre el robot y objetos que no estén especificados en el mapa. El control en este tipo de navegación corresponde al sistema deliberativo en su mayor parte, pues las acciones que debe ejecutar el robot para alcanzar el objetivo están en función al modelo del entorno. Para determinar su propia localización, y la del objetivo con respecto al mapa, el robot puede valerse tanto de las mediciones hechas por sus sensores como del uso de la información previamente almacenada.

- **Navegación libre de mapa.** Este tipo de navegación consiste en conducir al robot desde una posición inicial a una posición destino esquivando obstáculos durante su trayecto. Entonces, el robot debe de ser capaz de navegar en ambientes donde los obstáculos puedan aparecer de repente o no estén previstos en el mapa. Por tal motivo, esta navegación se caracteriza por dotar al robot con la capacidad de reaccionar rápidamente ante imprevistos.

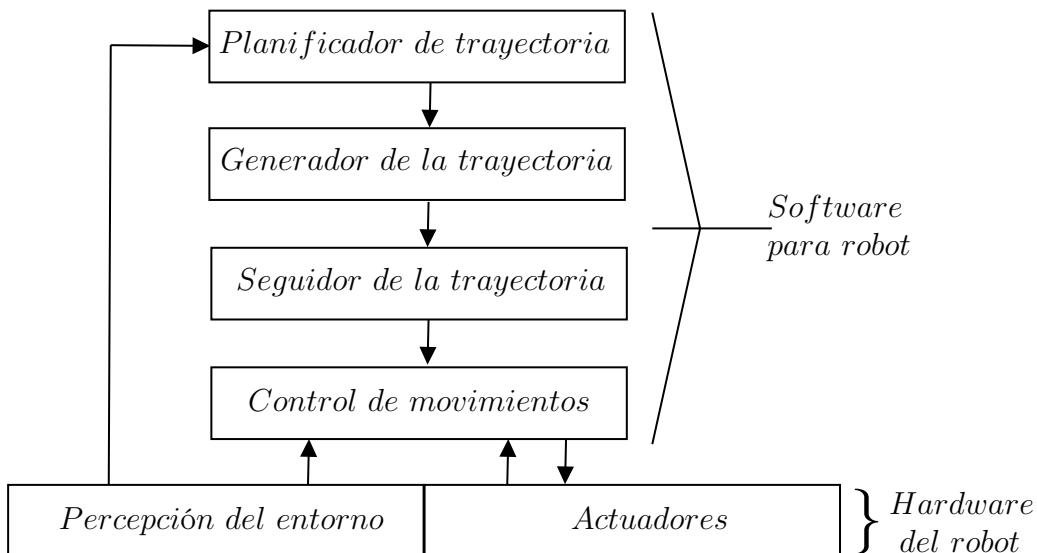


Figura 2.22: Estructura de navegación libre de mapa [33]

A diferencia de la navegación basada en mapa, el robot recurre a la Planificación Local para determinar la ruta que deberá seguir. La planificación local consiste en resolver obstrucciones que pueden presentarse en una ruta establecida para determinar la trayectoria real que el robot deberá seguir recurriendo a la información del entorno proporcionada por sus sensores. La estructura de navegación libre de mapa está esquematizada en la Figura 2.22. El control que necesita la navegación libre de mapa es de tipo reactivo, porque permite responder ante estímulos percibidos por sus sensores en tiempo real y no demanda de un modelo preciso del entorno para que el robot se comporte de manera inteligente. Sin embargo, esta navegación posee una limitada cobertura en ambientes amplios, por lo cual es considerada como una navegación de rango corto debido a que no utiliza un mapa para su consecución.

2.6. Estado del arte

El avance tecnológico acelerado de los últimos años ha permitido que distintas industrias abran las puertas a la innovación. Hablando concretamente de la robótica, el aumento en capacidad de cómputo, la disminución del tamaño y costo de los componentes de hardware así como la investigación en inteligencia artificial y redes neuronales ha permitido a dicha rama de la ciencia crecer en capacidades exponencialmente. En efecto, compañías como Boston Dynamics o Tesla son prueba de esto. De las múltiples ramas que tiene la robótica, una a destacar es la de robots de navegación autónoma, los cuales aprovechan mucho la cantidad de poder de cómputo disponible así como el avance en algoritmos de visión. A continuación se presentan algunos ejemplos y el trabajo expuesto en este documento:

Nombre	Aplicación	Características
Features Image Analysis for Road Following Algorithm Using Neural Networks [34]	Identificación de terreno y obstáculos usando una red neuronal de perceptrón multicapa.	Se hacen pruebas experimentales usando un automóvil y una cámara de video en escenarios reales.
Mobile Robots Navigation in Indoor Environments Using Kinect Sensor [35]	Sistema de percepción en ambientes interiores para robots de vigilancia.	Un sistema de navegación reactivo para evitar obstáculos en el ambiente usando un Kinect como sensor de distancia y una red neuronal para reconocer diferentes configuraciones del ambiente (e. g. camino hacia adelante, camino a la izquierda).
Embedded Mobile ROS Platform for SLAM Application with RGB-D Cameras [36]	Vehículo rastreado con capacidad de navegación autónoma para atravesar terreno impredecible.	Se procesa la imagen provista por la cámara de profundidad Intel RealSense D435i usando un NVIDIA Jetson Nano con capacidades CUDA. Se usa el sistema operativo ROS (Robot Operating System) y algoritmos SLAM (Simultaneous Location and Mapping).
Sistema para la navegación de un robot terrestre utilizando procesamiento de imágenes	Navegación autónoma sobre un ambiente controlado por medio de procesamiento de imágenes.	Se usa un chasis común de 4 ruedas con motor eléctrico manejados por un driver L289N a través de una Raspberry Pi la cual captura y analiza imágenes usando OpenCV para la aplicación de un algoritmo de navegación basado en un área de interés de código abierto.

Análisis y Diseño del sistema

3.1. Metodología del diseño

Para el desarrollo del sistema se tomará como base la **metodología en V**. Se ha elegido esta metodología debido a que **el sistema se compone tanto de software como hardware**, por lo que las etapas que la conforman se adaptan perfectamente para el desarrollo de esta clase de sistemas, como a continuación se muestra.

La metodología indica que se debe de partir de la especificación de requisitos, tanto para software como para hardware. En esta etapa se definen los requerimientos funcionales, técnicos y no funcionales, trazando un plan para el diseño del sistema mediante casos de uso, comprendiendo las limitaciones, identificando el impacto del mismo y previendo posibles cambios. En la siguiente etapa se realiza un diseño de alto nivel con base en la información recogida sobre requisitos y análisis, permitiendo obtener un diseño y una visión general del sistema. Posteriormente, se realiza un diseño en detalle, donde se ve al sistema de una forma modular, en este caso se cuenta con tres módulos, teniendo así la ventaja de rediseñar un módulo específico, hacer cambios de manera efectiva y reducir costos por modificaciones futuras. A continuación, el diseño es implementado con el lenguaje de programación elegido por cada módulo, obteniendo programas capaces de ofrecer la funcionalidad esperada. A su vez, cada módulo va a ser sometido a pruebas con el fin de validar el análisis y diseño previamente realizado [37].

Las pruebas de cada uno de los niveles se detallan a continuación:

- **Pruebas unitarias/modulares:** Estas pruebas comprueban que todas las funcionalidades especificadas en el diseño del componente sean correctas y cubiertas, estas pruebas son realizadas por el desarrollador del componente.
- **Pruebas de integración/interfaz:** Una vez que los módulos han sido probados de manera unitaria, se procede a probar su funcionalidad de manera conjunta, estas pruebas están enfocadas a resolver los siguientes supuestos:
 - Que espera un componente de otro componente, en término de servicios.
 - Como es que estos servicios serán solicitados.
 - Como es que estos servicios serán entregados.

- El manejo de errores por condiciones inesperadas.

Las pruebas deben corroborar el correcto funcionamiento de todas las interfaces entre componentes, al tener todos los componentes y sus interfaces terminados, tendremos como resultado el sistema completo, estas pruebas pueden ser realizadas por el desarrollador del componente.

- **Pruebas del sistema:** Una vez el sistema ha sido construido, debe ser probado contra las especificaciones del sistema para comprobar que cubra las funcionalidades requeridas. Esta parte se enfoca en probar el sistema de manera monolítica y debe incluir las pruebas para validar los requerimientos no funcionales.
- **Pruebas de validación:** similar a las pruebas del sistema, pero hay un cambio en el enfoque, ya que éstas comprueban que el sistema cumple con lo requerido, además las pruebas son realizadas por el usuario final del sistema.

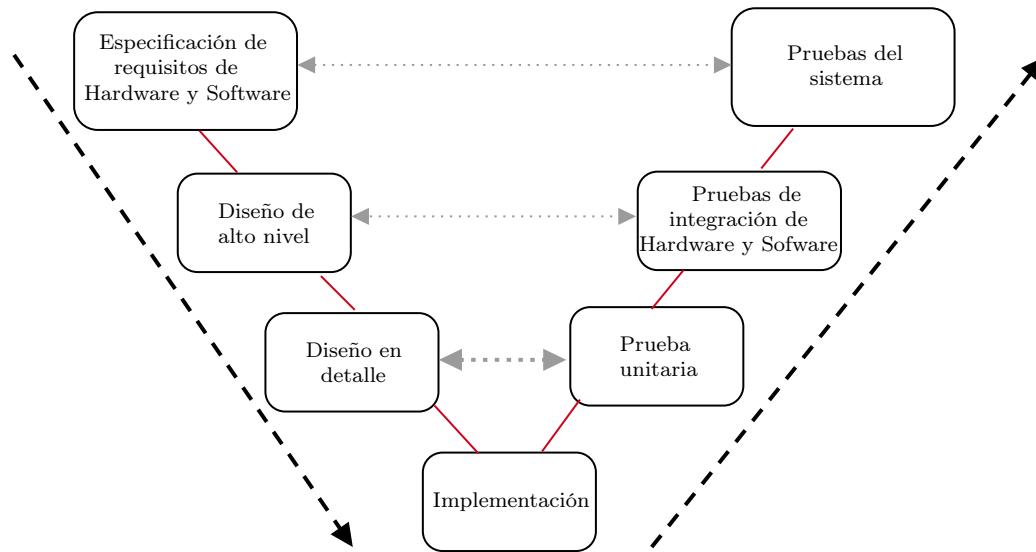


Figura 3.1: Metodología en V [37]

3.2. Análisis y requerimientos del sistema

Requerimientos

Esta sección pretende establecer aquellos requerimientos básicos, funcionales y no funcionales necesarios para el correcto funcionamiento del sistema.

Requerimientos básicos

ID	Nombre	Descripción	Prioridad	Origen
RB01	Adquisición de imagen digital	Obtención de imagen a través de la cámara conectada a la Raspberry Pi	Alta	Definición
RB02	Procesamiento digital de la imagen	Tratamiento de la imagen para la interpretación de la ruta a seguir por el robot	Alta	Definición
RB03	Desarrollo de algoritmo de navegación	Desarrollar algoritmo para generar los movimientos a través del escenario	Alta	Definición
RB04	Navegación del robot terrestre	Envío de señales a controlador para el movimiento del robot	Alta	Definición
RB05	Aplicación web	Despliegue de información en la aplicación web para monitoreo	Media	Definición

Tabla 3.1: Requerimientos básicos

Requerimientos funcionales

Los requerimientos funcionales describen las actividades o los comportamientos que el sistema debe realizar cuando se cumplen ciertas condiciones. A continuación, se muestra un listado con los requerimientos funcionales que se han obtenido del sistema.

ID	Nombre	Descripción	Prioridad	Origen
RF01	Adquisición de imagen digital	Captura de imagen digital haciendo uso de la cámara conectada a la Raspberry Pi 3	Alta	RB01
RF02	Lectura de la imagen digital	Almacenamiento de la imagen obtenida por la cámara	Media	RB01
RF03	Segmentacion de la imagen	Segmentacion de la imagen utilizando el espacio de color HSV	Alta	RB02
RF04	Ajuste de perspectiva de la imagen	Ajuste de perspectiva del trayecto obtenido a partir de la imagen capturada	Alta	RB02
RF05	Cálculo de región de interés	Calculo de la region de interes utilizando summarizacion de pixeles	Alta	RB03
RF06	Cálculo de histograma de región de interés	Obtencion del histograma para valor de curvatura	Alta	RB03
RF07	Movimiento de motores del robot móvil	Manipulacion de motores para movimiento del robot móvil terrestre	Alta	RB04
RF08	Aplicación web	Implementación de una aplicación web cliente-servidor para el envío de imágenes hacia una computadora personal	Media	RB05

Tabla 3.2: Requerimientos funcionales

Requerimientos no funcionales

Los requerimientos no funcionales describen características o propiedades del sistema. Especifican criterios para evaluar la operación de un servicio.

ID	Nombre	Descripción	Prioridad	Origen
RNF01	Escalabilidad	La arquitectura modular bajo la que es construido el sistema permite que éste sea escalable y pueda expandirse hacia más funcionalidades.	Alta	Definición
RNF02	Sistema embebido	El sistema de procesamiento, segmentación y algoritmo de navegación se encontrará alojado sobre una tarjeta de desarrollo	Alta	Definición
RNF03	Sistema operativo	El sistema operativo donde funcionará el sistema, el servidor y la aplicación web será una distribución de Linux	Alta	Definición
RNF04	Portabilidad	La aplicación web recibirá información del servidor, siempre y cuando se tenga una conexión WiFi	Alta	Definición

Tabla 3.3: Requerimientos no funcionales

Análisis de componentes

En esta sección se realiza el análisis de los distintos componentes que el sistema pretende usar. Para ello se muestran tablas comparativas de los distintos sistemas en chip, información relevante de los sensores de imagen, chasis del robot móvil terrestre y controlador de los motores.

Sistema Embebido

Para llevar a cabo este trabajo se requiere de una tarjeta de desarrollo de bajo costo, bajo consumo de energía comparado a una computadora personal, comunicación Wi-Fi, una interfaz de comunicación con cámara (CSI) y dos salidas PWM para controlar los motores.

Actualmente, se encuentran en el mercado una amplia variedad de tarjetas de desarrollo con SoC, cada una con distintas características. De acuerdo a cada proyecto a realizar, es necesario elegir la que mejor se ajuste a nuestras necesidades; a continuación se muestra una tabla comparativa (Tabla 3.4) de 5 de las tarjetas de desarrollo más populares con SoC en el 2019 en el mercado, según el sitio EE Times, revista online de la industria de la electrónica, publicada por AspenCore Media.

Para este proyecto nos decantamos por una tarjeta de la familia Raspberry Pi 3 por su bajo costo, bajo consumo de energía comparado a una computadora personal, comunicación Wi-Fi, interfaz para cámara CSI y dos salidas PWM para control de los motores. Además, cuenta con un procesador Cortex de la serie A, específicamente el modelo A53 de 64bits con 4 núcleos, el cuál está diseñado para aplicaciones de alto rendimiento; como será nuestro caso, al utilizar algoritmos de procesamiento de imagen y navegación.

A continuación una tabla (3.5) comparativa de los 3 modelos pertenecientes a la familia Raspberry Pi 3.

Características	Raspberry Pi 3	BeagleBone Black	Intel Joule570x	nVidia Jetson nano	Dragon board
SoC	Broadcom BCM2837B0	TI AM3358	Intel Atom T5700	Quad-core ARM A57	Qualcom Snapdragon
Procesador	Cortex-A53 64bits	RISC 32bits	64bits x86		RISC 64-bits
Núcleos	4	1	4	4	4
Frecuencia	1.2 GHz - 1.4GHz	1 GHz	1.7 GHz	1.43 GHz	1.2 GHz
GPU	Broadcom VideoCore	PowerVR SGX530	Intel HD	128-core Maxwell	Qualcomm Adreno 306
Tipo de memoria	LPDDR2	DDR3	LPDDR4	LPDDR4	LPDDR3
RAM	512 MB - 1GB	512MB	4GB	4GB	1GB
Almacenamiento	Micro SD	4GB eMMC Flash	16GB eMMC Flash	No	8GB eMMC Flash
S.O soportados	Linux	Android, Linux	Linux, Windows 10	Linux	Linux, Android
WiFi	Sí	No	Sí	Sí	Sí
Bluetooth	Sí	No	Sí	Sí	Sí
GPIO	40	69	8		12
Voltaje GPIO	3.3V	3.3V	1.8V	3.3V	1.8V
PWM	2	8	4	0	0
USB	1 - 4 Tipo A	1x Tipo A-mini	1 Tipo C	4 USB 3.0	1 micro USB Tipo A
Interfaz para cámara	Sí	Sí	Sí	Sí	Sí
Salida HDMI	1080p60	1080p24	1080p60	Sí	1080p30
Interfaz para display	Sí	No	No	No	HDMI MIPI-DSI
Consumo voltaje	300mA - 1.3A 5V	210-460mA 5V	130-600 mA	1-2A - 5V	50-400 mA 12V
Precio (USD)	\$31.28 - \$54	\$70	\$209	\$100	\$86

Tabla 3.4: Comparativa de los Sistemas en Chip (SoC) [38], [39], [40]

Modelo de Raspberry Pi 3 a elegir

Característica	Raspberry Pi 3B+	Raspberry Pi 3B	Raspberry Pi 3A+
Arquitectura	Broadcom BCM2837B0 (ARMv8) Cortex-A53 64-bits	Broadcom BCM2837 Quad-core	Broadcom BCM2837B0 (ARMv8) Cortex-A53 64-bits
Frecuencia	1.4 GHz	1.2 GHz	1.4 GHz
GPU	400 MHz VideoCore IV Multimedia	400 MHz VideoCore IV Multimedia	VideoCore IV
Tipo de memoria	LPDDR2	LPDDR2	LPDDR2
RAM	SDRAM 1GB	SDRAM 1GB	SDRAM 512MB
Almacenamiento	MicroSD	Micro SD	Micro SD
Tarjeta de red	2.4 GHz y 5 GHz IEEE 802.11.b/g/n/c wireless LAN Gigabit Ethernet sobre USB 2.0 (300Mbps)	BCM43438 wireless LAN 100 Base Ethernet	2.4 GHz y 5 GHz IEEE 802.11.b/g/n/c wireless LAN
Salida de vídeo	Full-size HDMI	Full-size HDMI	Full-size HDMI
Salida de Audio	HDMI Jack 3.5mm estéreo	HDMI Jack 3.5mm estéreo	HDMI
USB	4 Tipo A (2.0)	4 Tipo A (2.0)	1 Tipo (2.0)
S.O soportados	Linux	Linux	Linux
Precio (USD)	\$54	\$44.39	\$31.28

Tabla 3.5: Comparativa entre Raspberry Pi 3 [38], [41], [42]

Para el proyecto se selecciona la tarjeta Raspberry Pi 3A+, debido a que cuenta con todas las especificaciones necesarias para el proyecto mencionadas anteriormente, además de tratarse de una tarjeta con menos consumo de energía que las otras dos y más especializada para aplicaciones como la nuestra; donde no es necesario contar con diversos puertos USB ni puerto Ethernet, pues se utilizará comunicación inalámbrica Wi-Fi.

Sensor de Imagen

El bus CSI es un bus en serie para cámara, estándar definido por la *Mobile Industry Processor Interface* (MIPI), que agrupa a empresas que intervienen en el dominio de los terminales móviles. MIPI define el bus que vincula los diferentes componentes embebidos en los dispositivos móviles. La Raspberry Pi 3A+ dispone de un bus CSI para manejo de cámara. Actualmente se encuentran en el mercado tres modelos de cámaras diferentes, como se aprecia en la Tabla 3.6:

Características	Cámara v1	Cámara v2	Cámara HQ
Precio (USD)	\$29.90	\$29.95	\$50.00
Tamaño	25 x 24 x 9 mm	25 x 23 x 9mm	38 x 38 x 18.4mm
Resolución	5 Megapixeles	8 Megapixeles	12.3 Megapixeles
Modos de vídeo	1080p@30fps 720p@60fps 640x480@60/90fps	1080p@30fps 720p@60fps 640x480@60/90fps	1080p@30fps 720p@60fps 640x480@60/90fps
Integración con Linux	V4L2 driver disponible	V4L2 driver disponible	V4L2 driver disponible
Interfaz de programación en C	OpenMAX IL y otros disponibles	OpenMAX IL y otros disponibles	Información no disponible
Sensor	OmniVision OV5647	Sony IMX219	Sony IMX477
Área de imagen del sensor	3.76 x 2.74 mm	3.68 x 2.76 mm (4.6 mm diagonal)	6.287mm x 4.712 mm (7.9mm diagonal)

Tabla 3.6: Comparativa de las Cámaras para Raspberry Pi [43], [44]

Para este proyecto se seleccionó la Cámara V1 (Figura 3.2), por tener una resolución adecuada para la aplicación además, de su costo menor comparado con las otras dos.



Figura 3.2: Raspberry Pi Cámara V1.

Robot móvil terrestre

El kit de trabajo seleccionado para implementar el robot móvil terrestre se puede apreciar en la Figura 3.3. Fue elegido de acuerdo a su bajo costo (\$200 MXN) y características convenientes para el desarrollo del trabajo, como lo son el contar con cuatro motorreductores de CD de 200 mA de 5 V, además de un chasis de dos niveles; el superior para colocar el controlador de los motores junto con la tarjeta Raspberry Pi y la cámara, mientras en el inferior se colocará la fuente de energía.



Figura 3.3: Chasis, ruedas y motores del robot móvil terrestre.

Controlador de Motores

El controlador de motores o driver será el componente encargado de controlar los motores a través de las señales PWM que recibirá por parte del sistema embebido, en este caso seleccionamos el driver L298N (Figura 3.4), pues tiene capacidad de corriente de 2 A y potencia máxima 25 W lo cual es suficiente para controlar cuatro motores de corriente directa de 250 mA como los que se utilizarán y, además, es funcional para sistemas de bajo consumo de voltaje ya que su voltaje de potencia es desde 5 V a 35 V. Su precio es de \$65 MXN.

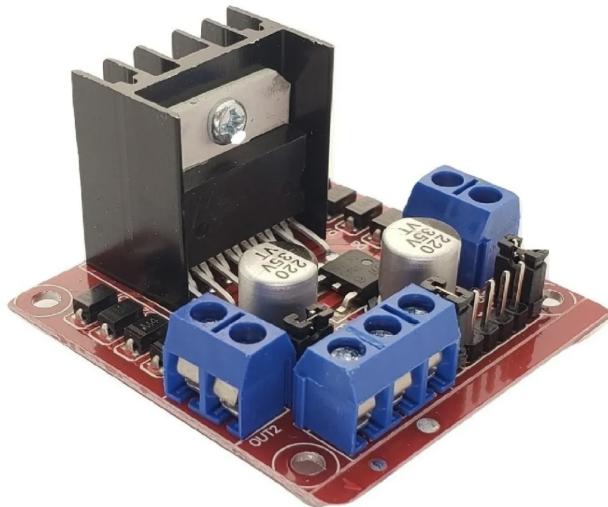


Figura 3.4: Controlador de motores L298N.

Fuente de energía

La fuente de energía seleccionada para alimentar el driver de los motores además de la tarjeta de desarrollo fue una powerbank de la marca Sedary (Figura 3.5). Sus especificaciones la hicieron una atractiva elección ya que cuenta con una capacidad nominal de 10,000 mAh por hora, dos salidas USB de 5 V con 2 A suficientes para alimentar ambos componentes por más de una hora, además se escogió por su tamaño reducido en comparación a otras powerbank con dimensiones de 10.1 cm de largo por 6.5 cm de ancho y un peso de 230 g.



Figura 3.5: Powerbank.

Herramientas de desarrollo

- **Apache Web Server.** Apache HTTP Server es un software de servidor web gratuito y de código abierto para plataformas Unix con el cual se ejecutan el 46 % de los sitios web de todo el mundo. Es mantenido y desarrollado por la Apache Software Foundation. Se trata de uno de los servidores web más antiguos y confiables, con la primera versión lanzada hace más de 20 años, en 1995.

Apache es altamente personalizable, ya que tiene una estructura basada en módulos. Los módulos le permiten a los administradores del servidor activar y desactivar funcionalidades adicionales. Apache tiene módulos de seguridad, almacenamiento en caché, reescritura de URL, autenticación de contraseña y más. También se pueden ajustar configuraciones del servidor a través de un archivo a preferencia del programador.

- **OpenCV.** OpenCV es una biblioteca de funciones de programación dirigida principalmente a la visión por computadora en tiempo real y aprendizaje máquina. Cuenta con tres aspectos muy llamativos, los cuáles son:

- **Código abierto.** OpenCV es de código abierto y se publica bajo la licencia BSD 3ra Cláusula. Es gratis para uso comercial.
- **Optimizado.** OpenCV es una biblioteca altamente optimizada que se enfoca en aplicaciones en tiempo real.
- **Multiplataforma.** Las interfaces C++, Python y Java son compatibles con Linux, MacOS, Windows, iOS y Android.

Costo de desarrollo

Cantidad	Componente	Precio (USD)	Detalles
1	Raspberry Pi 3A+	\$ 31.28	
1	Cámara V1	\$ 29.90	
1	Base de robot móvil terrestre	\$ 15.10	
1	Driver	\$ 1.66	
1	Power Bank 2000 mA	\$ 17.62	
Total de hardware		\$ 95.56	
Total de software		\$ 20,148	Desarrollo de sistema embebido y desarrollo de aplicación web Salario promedio mensual de Ingeniero en Sistemas en México (841 USD)[45] Tiempo de desarrollo (8 meses) Número de desarrolladores (3)
Total (USD)		\$ 20,243.56	

3.3. Diseño global

Definición del ambiente controlado

La especificación y las restricciones del ambiente son importantes para llevar a cabo las pruebas necesarias con el fin de realizar el sistema de navegación.

■ Escenario

- El escenario está limitado por una superficie lisa hecha de mica adherible sin obstáculos (Figura 3.6).
- La superficie tiene un carril delimitado diseñado con cartulina negra con curvas.
- El escenario se encuentra dentro de un ambiente cerrado con iluminación controlada.

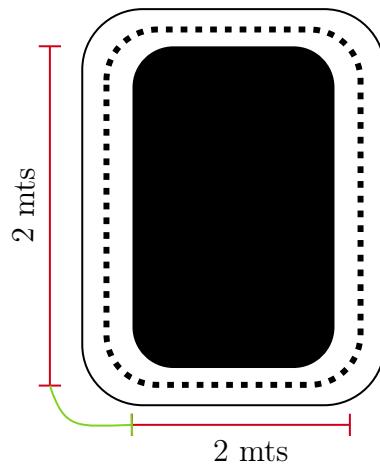


Figura 3.6: Escenario ideal

Diagrama a bloques del sistema

En la Figura 3.7, se muestra el sistema con sus distintos bloques y módulos, así como una pequeña descripción general.

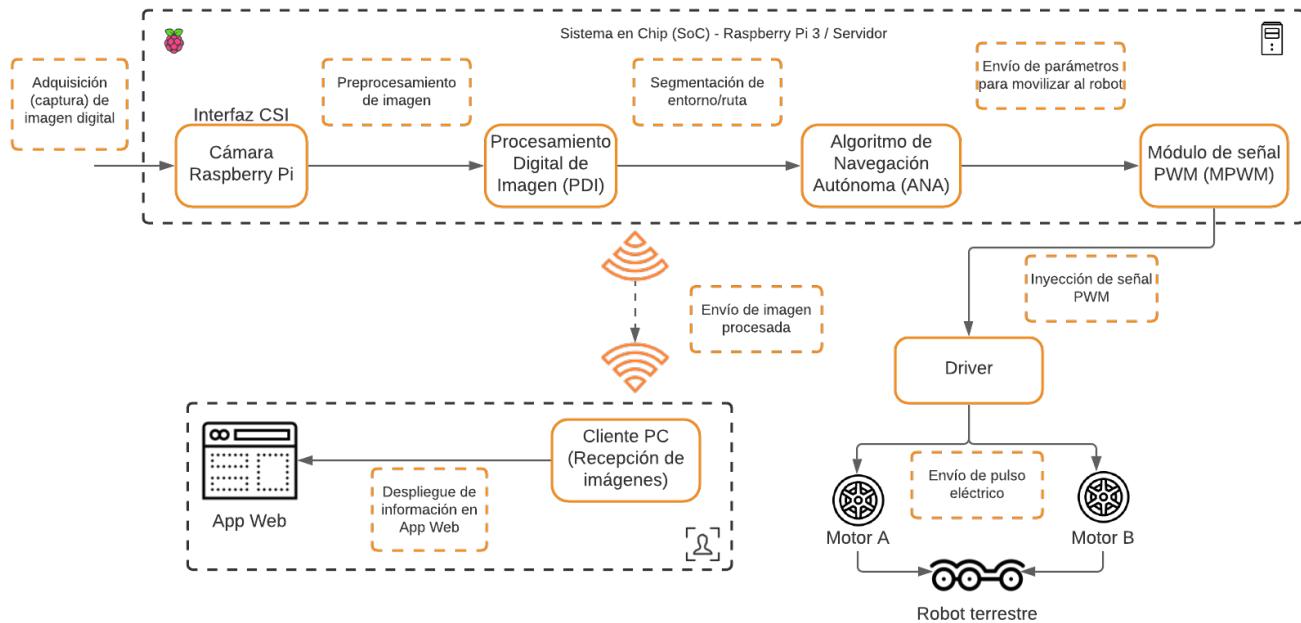


Figura 3.7: Diagrama a bloques del sistema

El prototipo se conforma por tres bloques, los cuales son: el sistema embebido, el chasis (la parte mecánica) junto con el driver de los motores, y una computadora personal. El sistema embebido será conectado a una cámara para poder realizar la adquisición de la imágenes, mismas que serán procesadas por el módulo de PDI, seguido de ello el módulo del algoritmo de navegación se encargará de analizar la imagen para enviar el comando adecuado al módulo de los motores quien lo interpretará y transformará en una señal PWM que hará llegar al driver. El driver accionará los motores del móvil de acuerdo a la señal recibida. Por otro lado, una computadora personal se encontrará programada para enviar peticiones al sistema embebido, quien contará con una implementación de un servidor web que regresará las imágenes procesadas por el módulo de PDI.

Diagrama del proceso general del sistema

Antes de continuar con las siguientes etapas, se realiza una breve presentación del diagrama del proceso general del sistema, diseñado para entender cómo funcionará el sistema y las interacciones que existen para que se tenga un correcto comportamiento y funcionamiento.

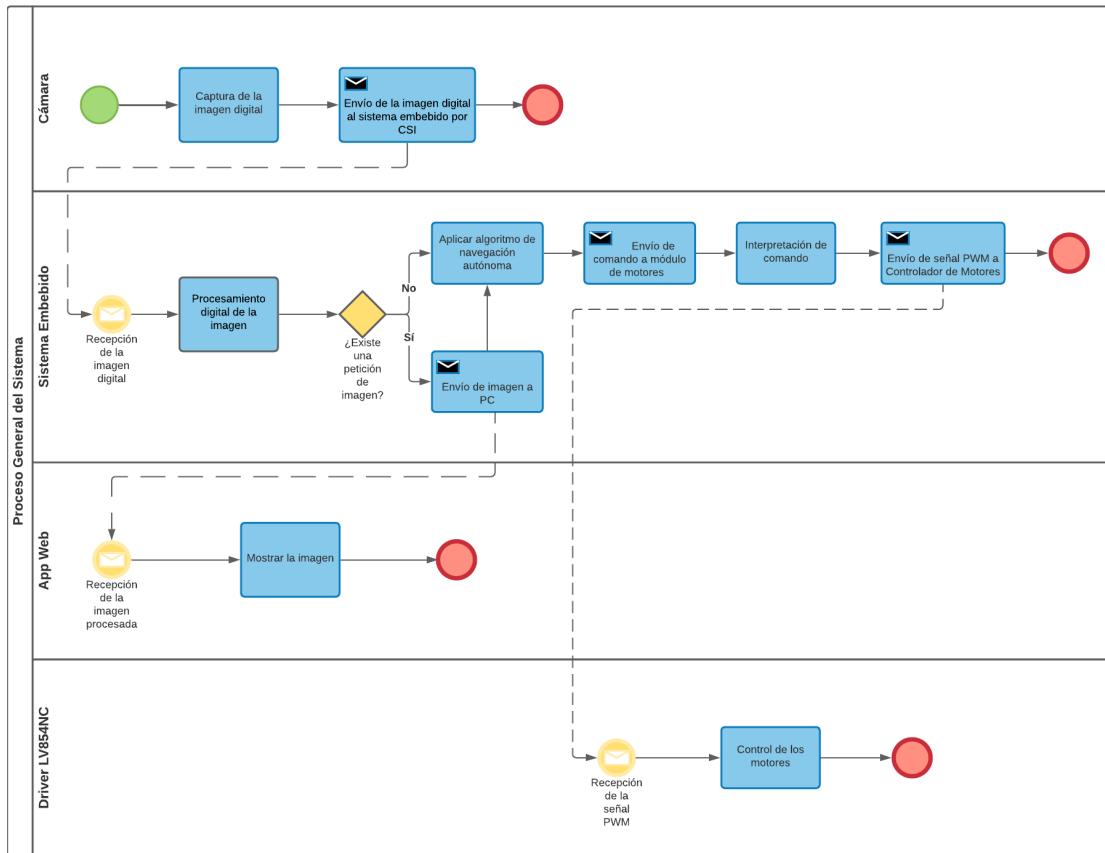


Figura 3.8: Diagrama del proceso general del sistema

Como se puede apreciar en la Figura 3.8, el proceso comienza por la captura de la imagen digital por parte de la cámara, que después envía al sistema embebido por medio de la interfaz CSI. Posterior a ello se aplica el procesamiento digital a la imagen para que el algoritmo de navegación trabaje correctamente, pasado este punto el programa se encontrará monitoreando si existe alguna petición por parte del cliente web que, de existir, accionará el procedimiento de envío de la imagen procesada hacia el cliente web quien entonces desplegará la imagen para su monitoreo. A continuación se ejecuta el algoritmo de navegación quien recibe como entrada la imagen procesada, para devolver como salida el comando requerido para mover el robot en la posición deseada; este comando será una abstracción que luego el módulo de los motores interpretará y enviará a manera de señal PWM al driver quien, finalmente, accionará cada uno de los motores del robot.

Arquitectura del prototipo

En la Figura 3.9 se tiene la arquitectura del prototipo. En el diagrama se muestra una representación del sistema y la correlación de sus componentes de hardware y software.

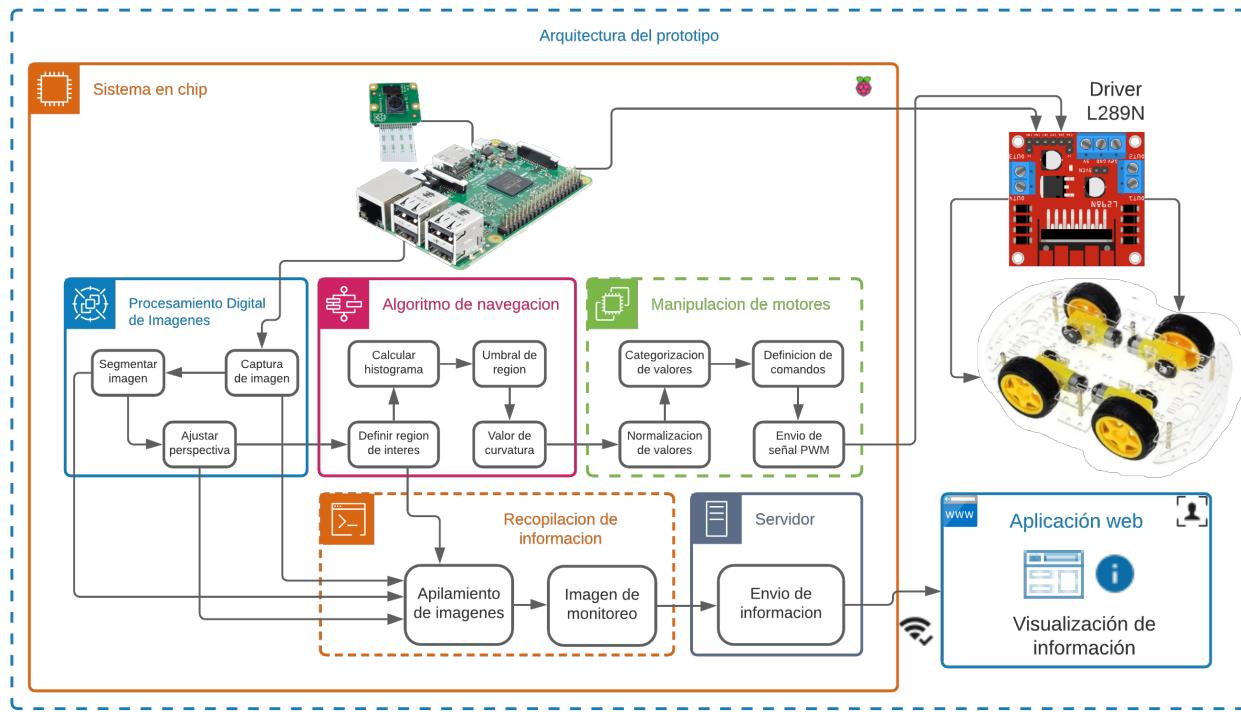


Figura 3.9: Diagrama de arquitectura de prototipo

Diagrama de Componentes

La Figura 3.10 muestra los componentes a utilizar para desarrollar el sistema, así como la interacción entre ellos. La cámara y los motores se conectarán al sistema embebido que los va a procesar mediante sus GPIOS, y éste a su vez se comunicará por Wi-Fi hacia la aplicación web, que funcionará como visualizador de información. Una vez obtenga la imagen a ser procesada, el siguiente paso será segmentarla, para posteriormente definir un polígono regular que permitirá el ajuste de perspectiva del camino. Posteriormente se determinará el valor de curvatura de la región obtenida y se determinará qué tanto a la derecha, al centro o a la izquierda el robot se encuentra. Finalmente, se normalizan y categorizan los valores para establecer los comandos de movimientos, los cuales serán enviados al driver que controla los motores del robot.

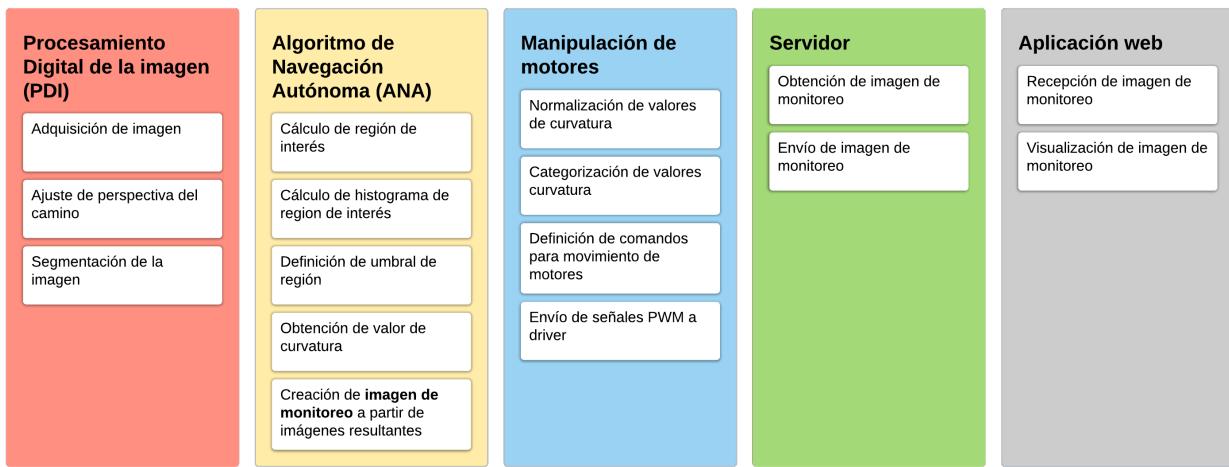


Figura 3.10: Diagrama de componentes

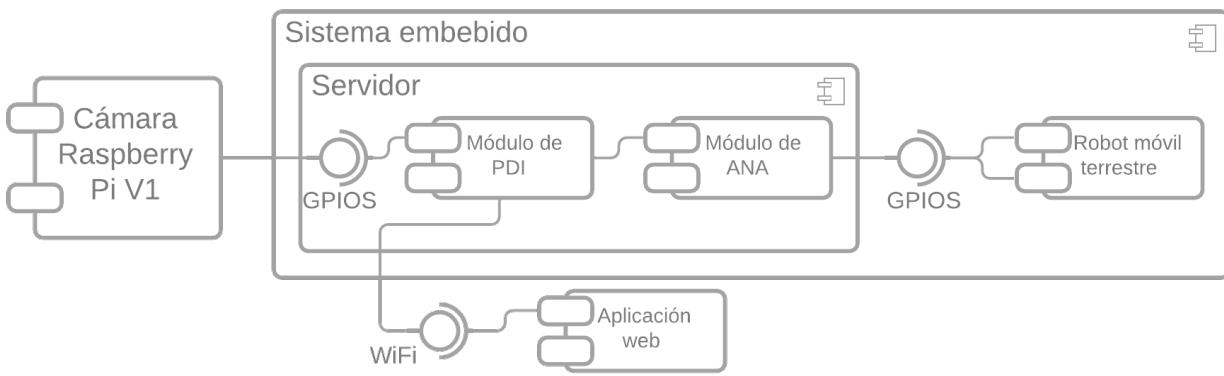


Figura 3.11: Diagrama de bloques

Diagrama de caso de uso general

Como se sustenta en [46], los diagramas de caso de uso permiten identificar a los actores implicados en un interacción, y nombra el tipo de interacción. Éstas interacciones se complementa con información adicional que describe la interacción con el sistema. La información adicional puede ser una descripción textual, o bien, uno o más modelos gráficos. Además, hace hincapié que el conjunto de casos de uso representa todas las interacciones posibles que se describen en los requerimientos del sistema. Los actores en el proceso, que pueden ser individuos u otros sistemas, se representan como figuras sencillas. Cada clase de interacción se constituye como una elipse con etiqueta. Líneas vinculan a los actores con la interacción. De manera opcional, se agregan puntas de flecha a las líneas para mostrar cómo se inicia la interacción.

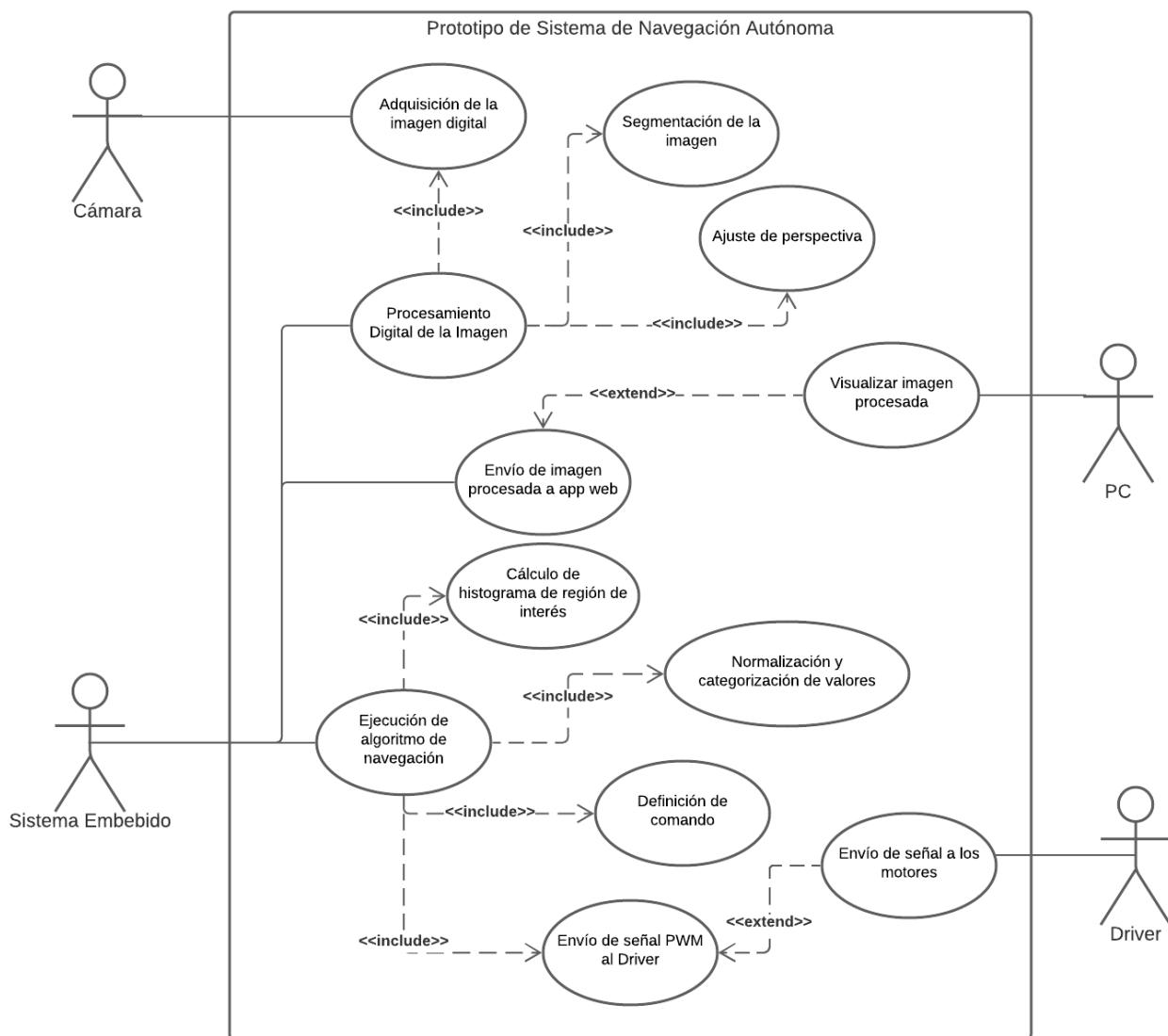


Figura 3.12: Diagrama de caso de uso general

3.4. Diseño en detalle

A continuación se describirá el diseño en detalle de cada uno de los módulos que compone el sistema, éstos son: Procesamiento Digital de Imagen, Algoritmo de Navegación Autónoma y App Web.

Módulo de Procesamiento Digital de Imagen

Este módulo será el encargado de capturar la imagen digital y aplicarle las transformaciones necesarias para que, tanto el módulo de navegación autónoma pueda trabajar correctamente, como el módulo de aplicación web pueda mostrar dicho procesamiento.

Diagrama de proceso del módulo de Procesamiento Digital de la Imagen

En seguida se muestra el diagrama del proceso que se lleva a cabo en este módulo (Figura 3.13). El primer paso es la captura de la imagen por parte de la cámara que es enviada por el bus CSI a la tarjeta de desarrollo, posteriormente se aplica sobre la imagen capturada una segmentación binaria por umbralización. Para terminar, la imagen es preparada con un *warping*, ajuste de perspectiva, y enviada al módulo de navegación autónoma para su evaluación.

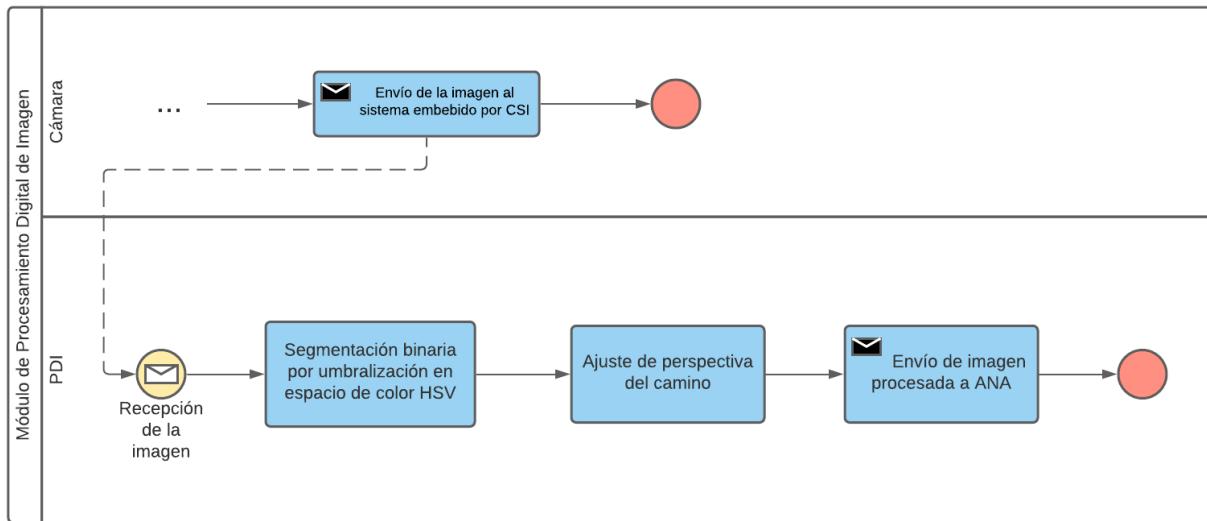


Figura 3.13: Diagrama de proceso del módulo de Procesamiento Digital de Imagen

Módulo de Navegación Autónoma

Este módulo será el encargado de obtener el valor de curvatura del camino recorrido utilizando una técnica de sumarización usando el cálculo del histograma para, posteriormente, llevar a cabo un algoritmo de votación para encontrar el punto medio del circuito transitado, usado para determinar los comandos utilizados para la manipulación de los motores del robot móvil terrestre.

Diagrama de proceso del módulo de Navegación Autónoma

A continuación se muestra el diagrama del proceso que se lleva a cabo en este módulo (Figura 3.14). Una vez segmentada la imagen del camino a transitar, el siguiente paso es definir una región de interés utilizando un polígono regular. Para realizarlo es necesario realizar un ajuste de perspectiva de la imagen capturada utilizando un programa de ajuste manual (con apoyo de una *barra de seguimiento*) para obtener una mejor perspectiva del camino a transitar. Posteriormente, se utiliza una técnica de sumarización de pixles, es decir, se obtiene el histograma de la imagen, posteriormente se indexan cada uno de los valores sumarizados por columna para posteriormente establecer un umbral para definir si la región ubicada pertenece o no al camino.

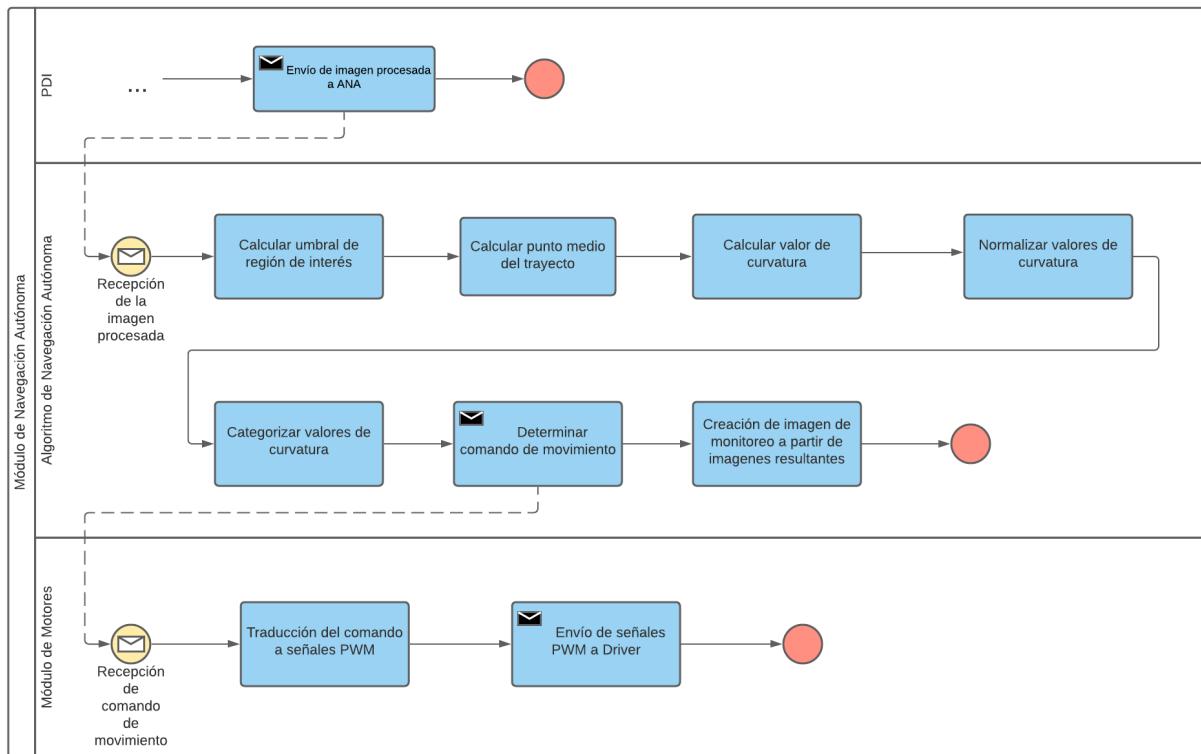


Figura 3.14: Diagrama de proceso del módulo de Navegación Autónoma

Posteriormente, se calculó el punto medio de la región perteneciente al camino, el representa un valor de curvatura, o sea, este valor representa cuán a la derecha, al centro o a la

izquierda se encuentra el robot. Luego, se normalizan los valores en un intervalo delimitado: -1, 0 y 1. Dichos valores son categorizados para determinar qué comando se debe de utilizar para manipular los motores del robot móvil terrestre. Para poder llevar a cabo esto, se envían señales PWM a cada uno de los motores y se establece, en cada comando, una cierta velocidad, grado de giro y tiempo de ejecución.

En cada una de las etapas del algoritmo de navegación se guardan las imágenes del camino para, finalmente, conjuntar una imagen de monitoreo, la cuál contiene imágenes de: Circuito original, circuito segmentado, circuito con la región de interés, circuito con los valores normalizados.

Módulo de App Web

Este módulo será el encargado de permitir el correcto despliegue de la imagen de monitoreo en una PC.

Diagrama del proceso de App Web

A continuación se muestra el diagrama del proceso que se lleva a cabo en este módulo (Figura 3.15). En este caso, son tres actores lo que interactúan para completar la tarea requerida. Por un lado, el servidor web inicia su ejecución junto con el sistema operativo de la tarjeta Raspberry, y corre como un proceso en segundo plano que se encuentra monitoreando activamente cualquier petición HTTP entrante. El módulo de Navegación Autónoma se encarga de la creación y almacenamiento en el sistema de una imagen de monitoreo, como se mencionó en el diseño de dicho módulo. Esta imagen es obtenida por el servidor web cuando detecta una petición HTTP entrante, para luego enviarla hacia el cliente web. El cliente web es un programa que se ejecuta desde una computadora personal en un navegador web, y que será programado para realizar una petición cada determinado tiempo hacia el servidor web corriendo en la tarjeta Raspberry, para así obtener y desplegar la imagen de monitoreo.

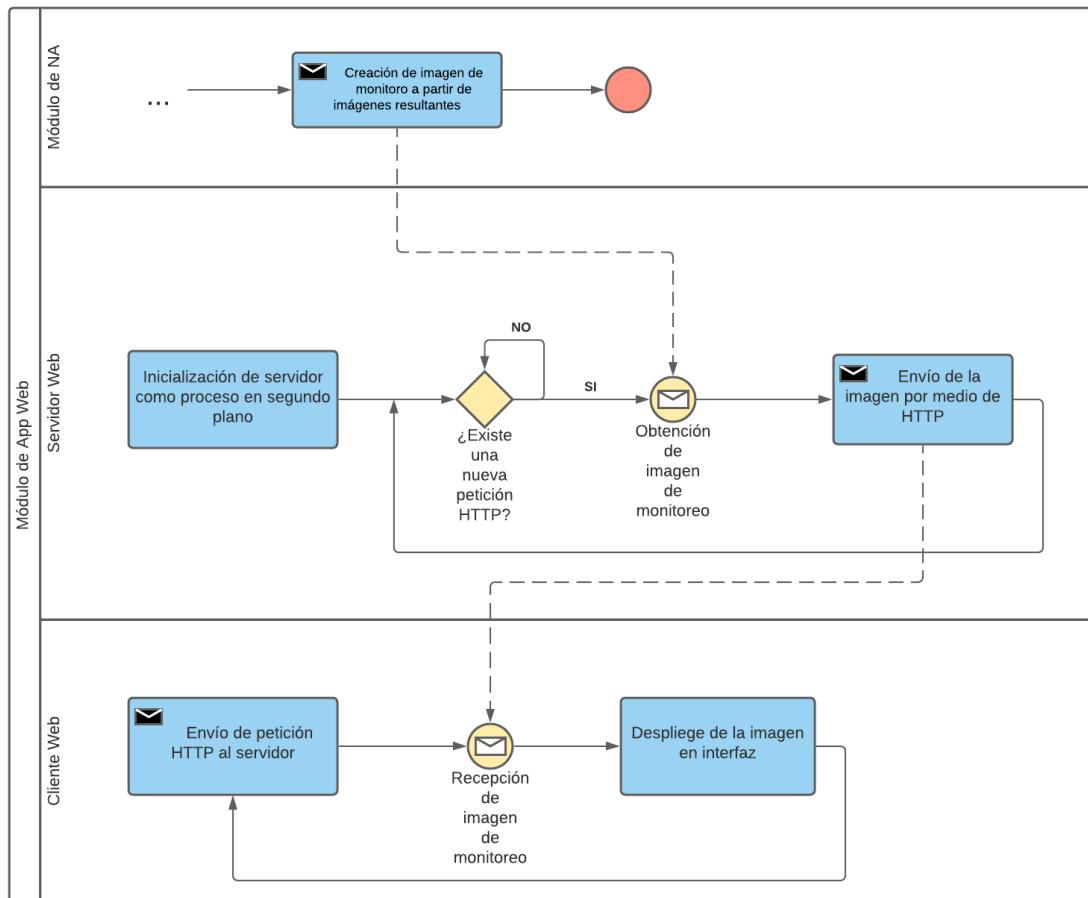


Figura 3.15: Diagrama de proceso del módulo de App Web

Diseño de App Web

En seguida se muestra el maquetado de la página web desde donde se puede monitorear el progreso del algoritmo de navegación en divisiones: la imagen capturada, la imagen mostrando los puntos de perspectiva sobre la imagen capturada, la imagen después de la umbralización y warping(ajuste de perspectiva), la imagen procesada con el punto medio calculado, el área de interés y por último la imagen resultante.

Interfaz de Usuario

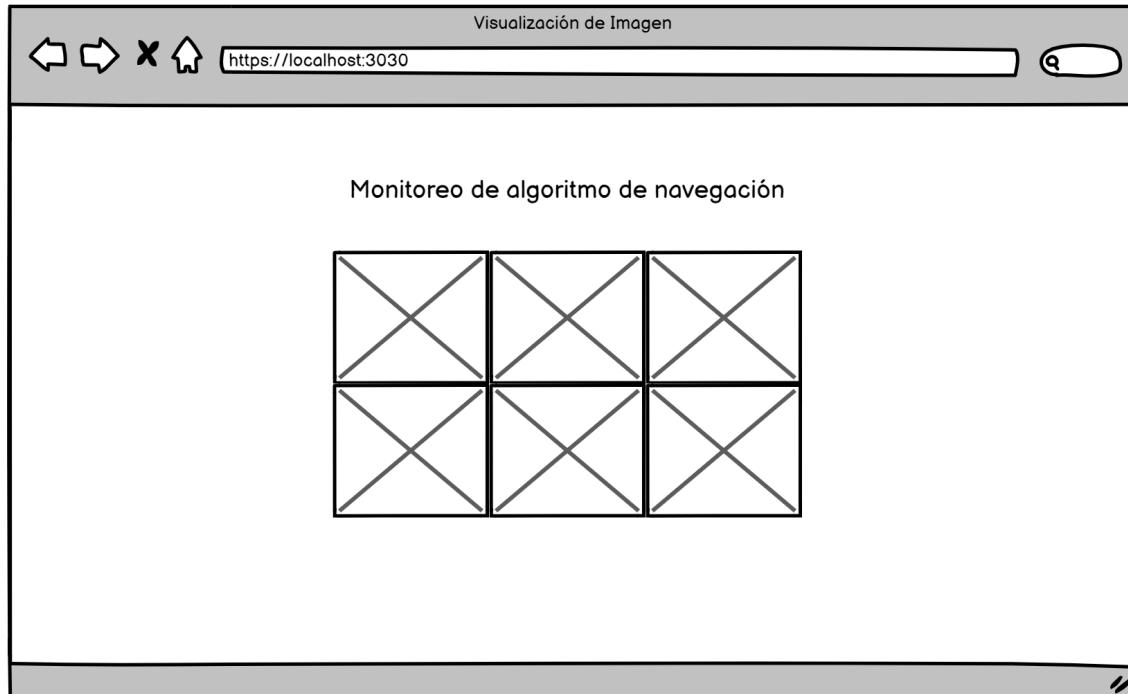


Figura 3.16: Interfaz web de aplicación de visualización de imagen

Diagrama de flujo de Algoritmo de Navegación

En la Figura 3.17 se presenta el diagrama de flujo del algoritmo de navegación que se utiliza para el sistema de navegación.

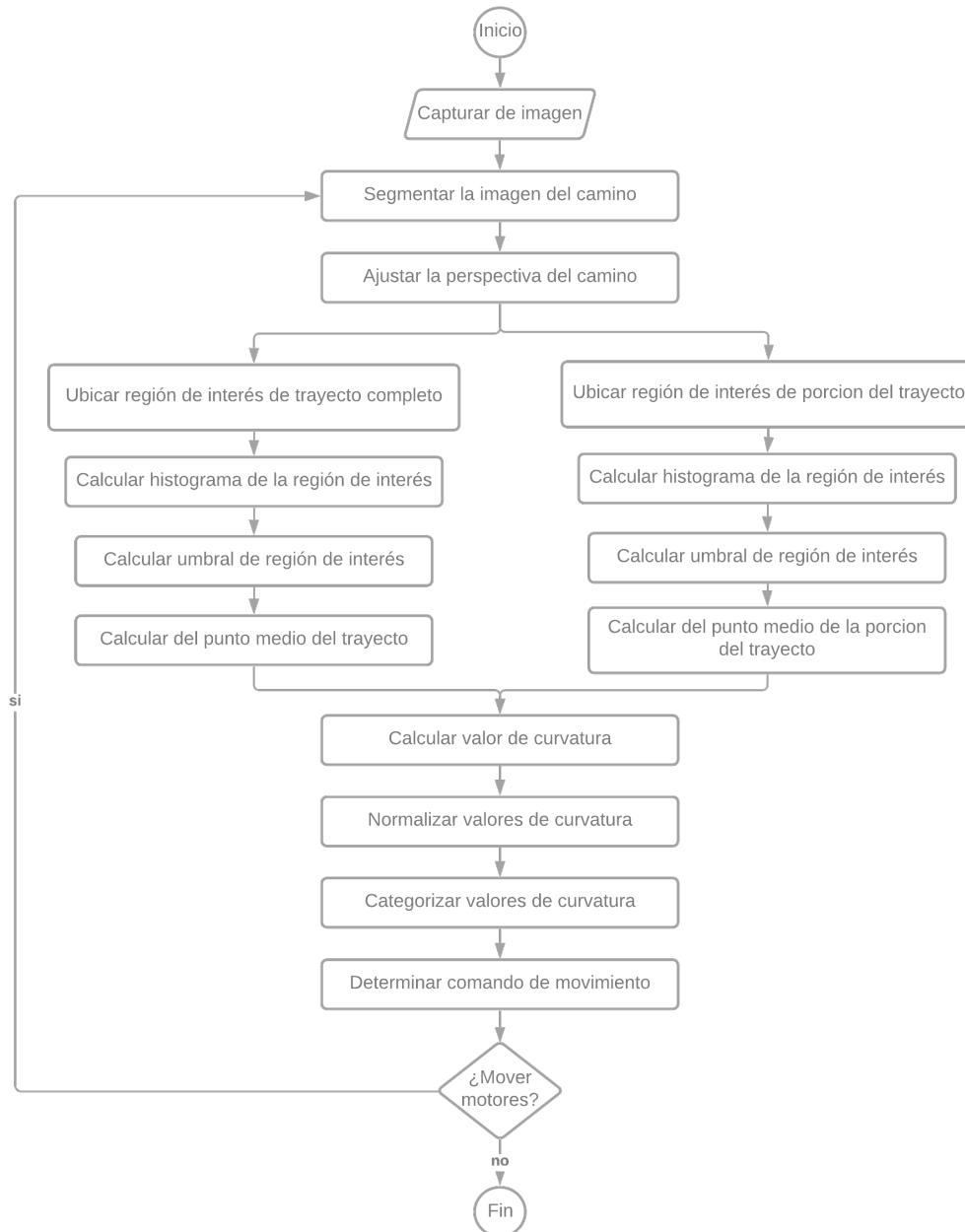


Figura 3.17: Diagrama de flujo del algoritmo de navegación propuesto

4

Desarrollo y Pruebas

4.1. Adquisición de la imagen digital

Objetivo

Implementar la funcionalidad descrita en el requerimiento funcional RF01 (Sección 3.2) , donde se indica que el sistema deberá ser capaz de obtener una imagen digital, a través de la cámara conectada, para su posterior procesamiento.

Desarrollo

Para realizar la captura de imágenes con la Raspberry Pi se necesitan hacer varios pasos previos y que se pueden consultar en la documentación oficial [47]. A continuación, una descripción de dichos pasos.

Primero se debe habilitar el bus CSI, para esto se editó el archivo “/boot/config.txt” agregando las siguientes líneas al final del archivo:

```
1 start_x=1  
2 gpu_mem=128
```

Posterior a ello, se debe reiniciar la Raspberry Pi utilizando el siguiente comando en una terminal:

```
1 $sudo reboot
```

A continuación, se conecta la cámara a la tarjeta, esto se hace con la tarjeta apagada y se debe conectar en el conector CSI; recuadro encerrado en rojo que se aprecia en la siguiente figura 4.1.



Figura 4.1: Conexión de la cámara.

Continuando con el desarrollo, se conectó la cámara a la Raspberry Pi, como se indicó anteriormente y, como se observa en la Figura 4.2.



Figura 4.2: Raspberry Pi con la cámara equipada.

Para hacer la captura de una imagen digital usando OpenCV se desarrolló un módulo que se puede ejecutar de manera independiente o ser exportado a otro archivo, para utilizar su funcionalidad. En este caso, el módulo se divide en dos funciones: la principal “main” y “getImage()”. La función principal solo es llamada cuando se ejecuta directamente el módulo, esta hace la tarea de llamar indefinidamente a la función “getImage()” hasta que se presione la tecla “q”.

Por otro lado, la función “getImage()” ejecuta los siguientes pasos; utiliza la función “VideoCapture()” de OpenCV para obtener acceso a la cámara; luego utiliza la función “read()” que hace la captura de la imagen, dicha imagen es reajustada en tamaño de acuerdo al segundo parámetro que recibe y, a continuación, despliega la imagen en pantalla en caso de haberle pasado un valor “True” al primer parámetro, finalmente regresa la imagen capturada a la función que llamó a esta función. El código antes descrito se puede ver a continuación.

Nota: El tiempo que se le da a la cámara para tomar una imagen es de 500ms, pues la documentación oficial [47], indica que pueden ocurrir problemas si se utilizan valores menores a ese.

```
1 def getImage(display = False, size = (480,240)):  
2     cap = cv2.VideoCapture(0)  
3     # Capturar frame-por-frame  
4     _, img = cap.read()  
5  
6     img = cv2.resize(img, size)  
7  
8     if (display):  
9         cv2.imshow('Imagen', img)  
10  
11    return img
```

Prueba Unitaria

Se conectó la cámara a la raspberry de igual forma que en la Figura 4.2, y se procedió a tomar una imagen de prueba como se ve en la siguiente Figura 4.3.



Figura 4.3: Imagen de prueba tomada con la cámara utilizando OpenCV.

4.2. Módulo de Procesamiento Digital de Imagen

Objetivo

Implementar la funcionalidad descrita en los requerimientos funcionales: RF02, RF03 y RF03 (Sección 3.2), que se requiere para cumplir con el diseño de este módulo.

Desarrollo

En esta segunda aproximación el desarrollo se hizo con OpenCV en lenguaje de programación Python. Se pensó dividir el módulo en dos programas, uno principal llamado “LaneDetectionModule.py” y uno auxiliar llamado “utils.py”, este último contiene funciones de ayuda para ejecutar los algoritmos del programa principal de detección del camino y de navegación. Una imagen más clara se observa en la siguiente figura 4.4.

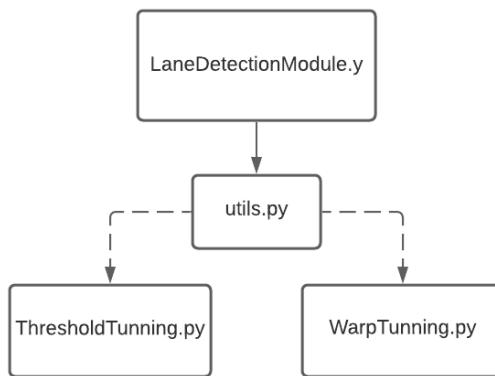


Figura 4.4: Interacción del programa de PDI

Segmentación binaria por umbralización

La función principal del programa “LaneDetectionModule.py” comienza por hacer una detección del camino, a través de una segmentación binaria por umbralización, dicha umbralización se realiza en espacio de color HSV, pues el canal de matices de este modelo permite separar de manera más sencilla los objetos por su color; cuando en RGB los tres canales influyen en la detección de un color, esto nos permite separar de manera más sencilla el camino de otros objetos. Para esto se creó una función llamada “thresholding()” y cuyo código se observa a continuación.

```

1 def thresholding(img):
2     imgHsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
3     lowerWhite = np.array([60, 0, 0])
4     upperWhite = np.array([179, 65, 255])
5     maskWhite = cv2.inRange(imgHsv, lowerWhite, upperWhite)
6
7     return maskWhite
  
```

Esta función, como todas las demás nombradas a continuación se encuentran en el programa “utils.py”. La función recibe como parámetro la imagen original capturada, cambia su espacio de color de RGB a HSV utilizando la función de OpenCV “cvtColor()” luego define los límites inferiores y superiores de umbralizado para el maticz, saturación y brillo o valor, respectivamente. Por último, se emplea la función “inRange()” que nos devuelve la imagen umbralizada de acuerdo a los límites asignados. La función termina por devolver la imagen umbralizada.

Sin embargo, la definición de los límites superiores e inferiores para los matices no es cosa trivial, y se necesita de un afinamiento empírico de acuerdo al nivel de luz con el que se esté trabajando en el momento. Para hacer esta tarea más sencilla se desarrolló un programa llamado “ThresholdTunning.py” que nos ayuda a afinar en tiempo real los valores de los límites inferior y superior de umbralizado, es decir, los de los tres canales del espacio HSV. Con la cámara conectada a la Raspberry Pi, el programa cargado en la misma, y el robot puesto en posición (Figura 4.5) en la pista.



Figura 4.5: Robot en posición en la pista

Se procede correr el programa obteniendo la siguiente vista 4.6.

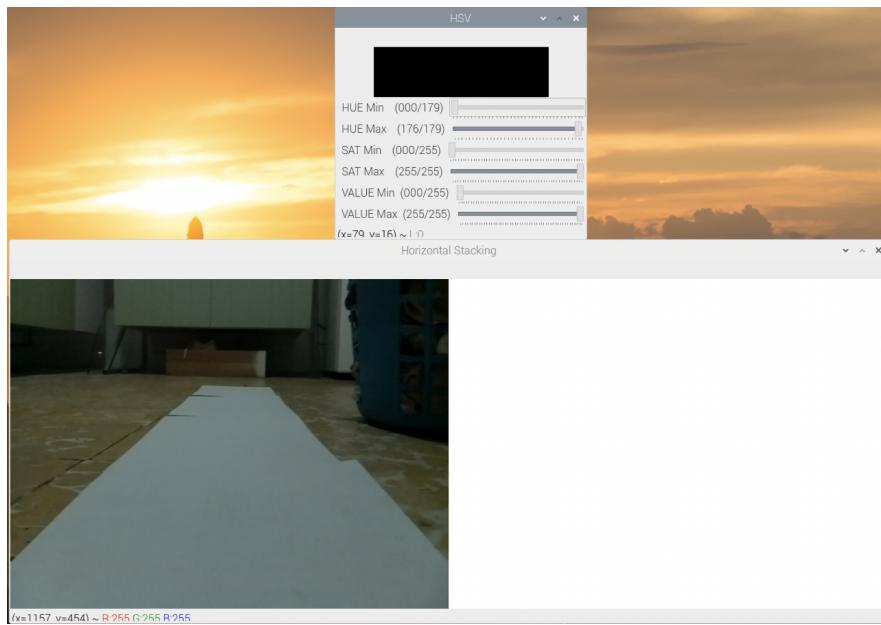


Figura 4.6: Programa de afinamiento de parámetros de umbralizado

Dentro del programa se crean dos trackbars por cada canal del espacio de color HSV, uno correspondiente al límite inferior y otro al superior, y que son mostrados en pantalla junto con las vistas de la imagen original y la umbralizada. Del lado izquierdo se encuentra la imagen capturada por la cámara mientras que del derecho la imagen resultante del umbralizado. De esta manera se logra ensayar con los valores de los trackbars de cada canal y ver en tiempo real el efecto que estos tienen en el resultado final del umbralizado, así es como se logra obtener los valores óptimos para segmentación del camino por color. Posterior a ello, se escriben dichos valores en las líneas 6 y 7 del código de la función “thresholding()” mostrado anteriormente. En el ejemplo, con las condiciones de luz del momento se afinaron los parámetros resultando en la imagen que se puede apreciar en la figura 4.7

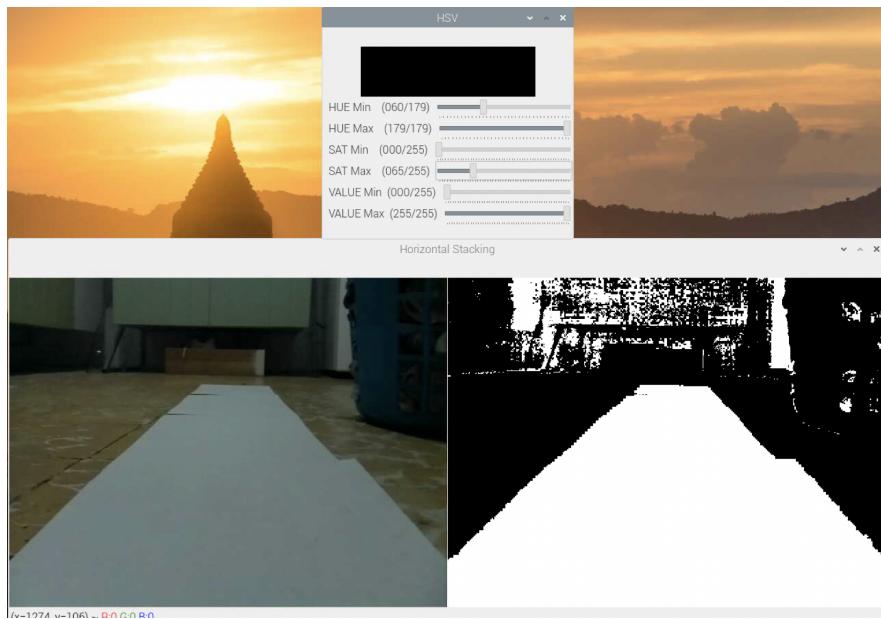


Figura 4.7: Programa de afinamiento de parámetros de umbralizado después de haber encontrado los valores óptimos

Ajuste de Perspectiva

La siguiente etapa en el proceso de PDI es el ajuste de perspectiva, esto se hace dado que en realidad el robot no necesita saber cuánto camino sobra por seguir avanzado, lo que importa es la situación del camino en el momento inmediato y la cantidad de curvatura que presenta el mismo. Para esto se ajusta la perspectiva a manera de una “vista de águila”, para ver el camino como si se estuviese tomando una captura desde arriba y poder medir la magnitud de curvatura del camino de manera más certera, pues es mucho más difícil hacerlo cuando la imagen del camino está inclinada.

Para lograr esto se creó una función llamada “warpImg()” que recibe como parámetros: la imagen original, puntos desde los cuales se va a ajustar la perspectiva de la imagen original, el ancho y alto de la imagen que se desea obtener, y por último si el proceso de ajuste va a ser inverso o normal. El código de dicha función es el siguiente.

```

1 def warpImg(img, points, width, height, inverse = False):
2     points1 = np.float32(points)
3     points2 = np.float32([[0, 0], [width, 0], [0, height], [width, height]])
4     if inverse:
5         matrix = cv2.getPerspectiveTransform(points2, points1)
6     else:
7         matrix = cv2.getPerspectiveTransform(points1, points2)
8
9     imgWarp = cv2.warpPerspective(img, matrix, (width, height))
10
11    return imgWarp

```

Lo primero que hace es un casteo de los puntos pasados como argumentos al tipo “float32”, para después checar si se desea hacer un proceso inverso, es decir, de perspectiva ajustada a original o viceversa. Seguido de eso el programa genera una matriz de transformación de perspectiva, según sea el caso y la aplica a la imagen original utilizando la función “warpPerspective()” de OpenCV, de esta manera se obtiene una nueva imagen con la perspectiva deseada la cual se retorna a la función que llamó a ésta función.

Así como con la definición de los parámetros de umbralizado, la definición de los puntos de ajuste de perspectiva no es cosa trivial, y se necesita de un afinamiento empírico de acuerdo al ángulo de la cámara respecto al camino con el que se esté trabajando en el momento. Para hacer esta tarea más sencilla se desarrolló un programa llamado “WarpTunning.py” que nos ayuda a afinar en tiempo real los valores de dichos puntos, a través de la creación de unos trackbars para ajustar los valores.

El programa toma una imagen, aplica la función de segmentación binaria por umbralización que se desarrolló previamente, utiliza la imagen resultante para pasarla como parámetro a la función que hace el ajuste de perspectiva junto con los valores de los puntos de perspectiva obtenidos de los trackbars y finalmente despliega el resultado. Este proceso se ejecuta indefinidamente mientras el usuario tiene la posibilidad de ajustar los valores de los trackbars. Para hacer el proceso más visual, se desarrolló una pequeña función que dibuja los puntos de perspectiva que se están tomando sobre la imagen original. El programa se aprecia en la figura 4.8.

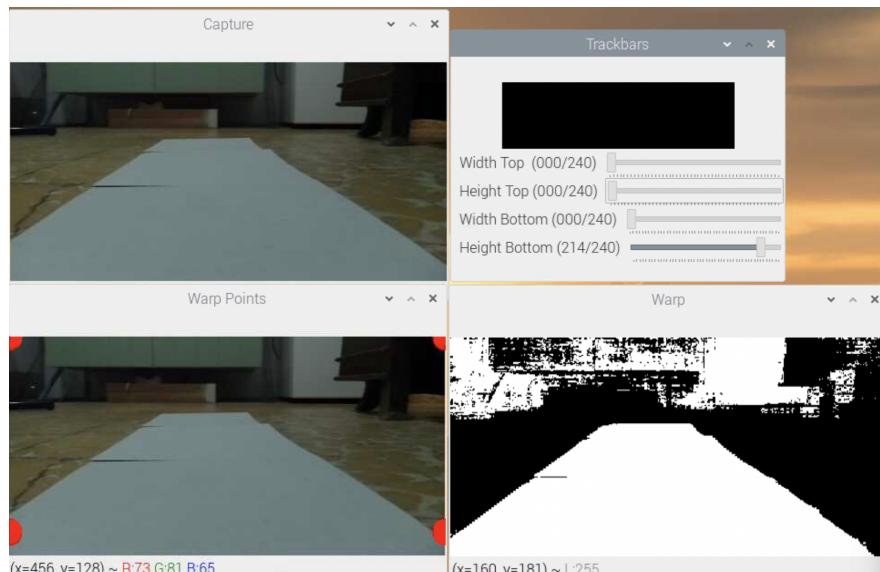


Figura 4.8: Programa de afinamiento de puntos de perspectiva.

En la parte superior izquierda se visualiza la imagen capturada original, en la inferior izquierda la posición de los puntos de perspectiva, en la superior derecha los trackbars para el ajuste de dichos puntos y finalmente en la inferior derecha el resultado de aplicar umbralización y el ajuste de perspectiva. Después de mover los valores al punto deseado se obtiene una perspectiva de "vista de águila" como la que se aprecia en la figura 4.9.

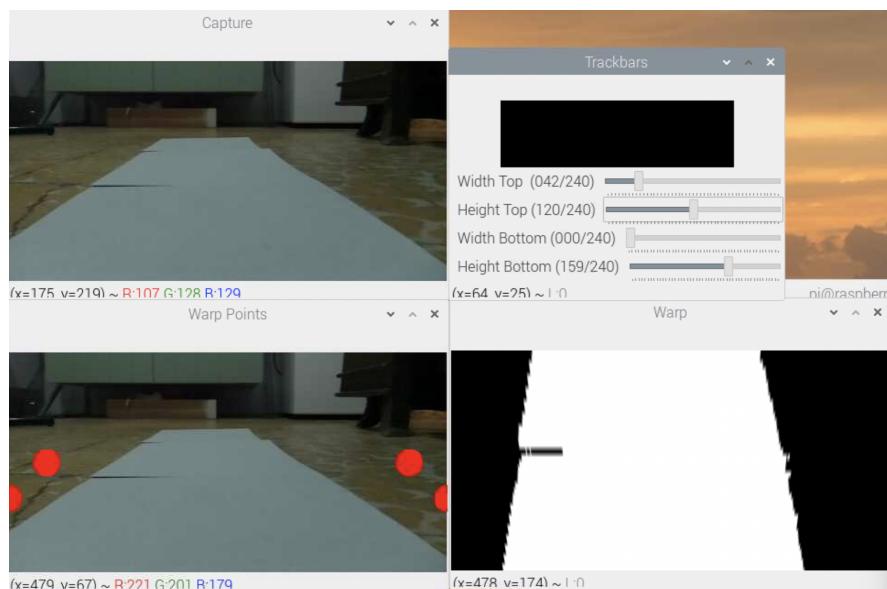


Figura 4.9: Ajuste de perspectiva logrado después de afinar los valores de los puntos.

4.3. Módulo de Motores

Objetivo

Implementar la funcionalidad descrita en el requerimiento funcional RF05 (Sección 3.2), que se requiere para cumplir con el diseño del módulo de motores.

Desarrollo

Para realizar el desarrollo relacionado con el robot móvil terrestre se ha utilizado en primer instancia un chasis de cuatro ruedas, como el que se mencionó en la etapa de Diseño y se muestra en la Figura 4.10.



Figura 4.10: Chasis del robot

Con el fin de tener el control del robot móvil se ha incorporado el módulo L298N cuyo circuito integrado contiene un puente H y la circuitería de potencia necesaria para el control de los dos motores a partir de señales PWM y, los pines de entrada y salida de propósito general (GPIOs) de la tarjeta Raspberry Pi. Este circuito integrado se puede observar en la Figura 4.11.

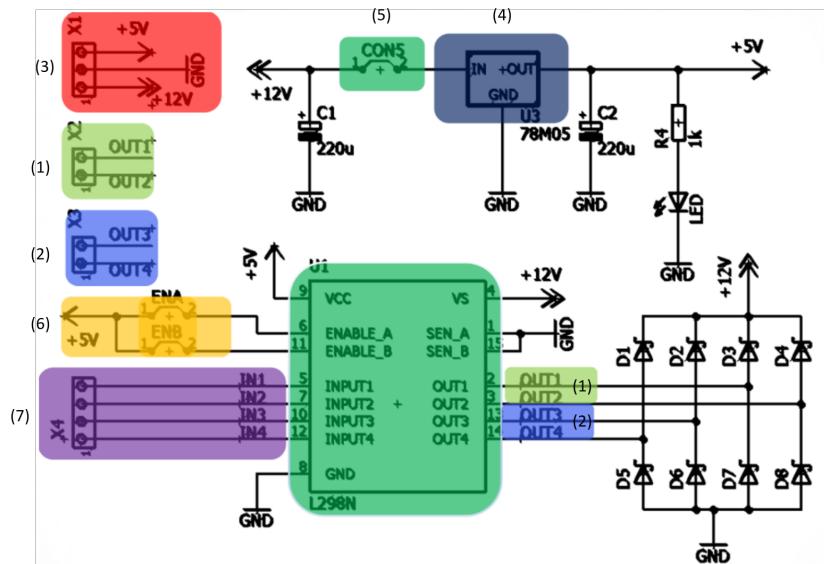


Figura 4.11: Circuito integrado del módulo L298N

El diagrama de la Figura 4.12 muestra el circuito integrado del módulo L298N de forma física, el cuál tiene 4 salidas, desde OUT1 hasta OUT4, donde OUT1 y OUT2 corresponden al motor A, mientras que OUT3 y OUT4 al motor B. Además, la corriente de salida nominal es de 2A, con picos de corriente de hasta 4A. Dicho módulo necesita dos fuentes de alimentación: La primera puede ir desde 5V a 32V, según las necesidades del proyecto, para suministrar la energía para los circuitos de potencia de los puentes H y de los motores, mientras que la segunda es una fuente que suministra 5V para la alimentación de las compuertas lógicas que se usan para la lógica de control del puente H. Esta fuente de 5V puede ser suministrada a partir del voltaje que alimenta los motores usando un regulador de 5V como se observa en la Figura 4.12 (índice 4), siempre y cuando el jumper “CON5” esté conectado en las terminales, habilitando el regulador 78M05. Para el desarrollo del proyecto se ha mantenido conectado el jumper de habilitación.

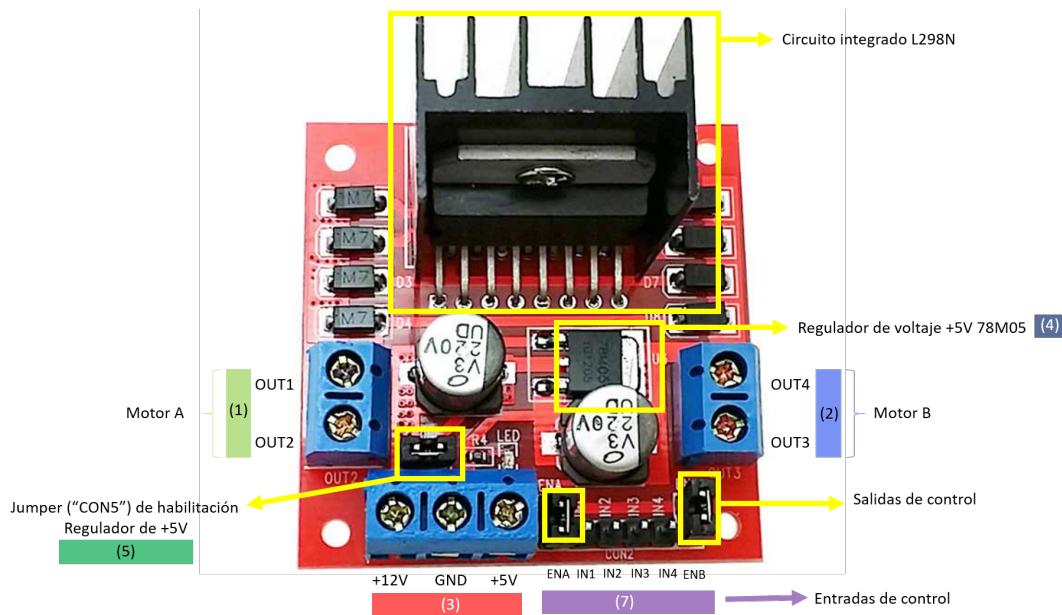


Figura 4.12: Módulo L298N con entradas y salidas especificadas

Asimismo, el módulo contiene salidas y entradas de control las cuales se conectan directamente a las GPIO de la Raspberry Pi. Las salidas del índice 6 de la Figura 4.12 son salidas de 5V que deben habilitarse para ser conectadas. Los modos de operación del módulo L298N se describen en la Tabla 4.1.

		ESTADO MOTOR A	
ENA	IN1	IN2	
ENB	IN3	IN4	ESTADO MOTOR B
1	0	0	DETENIDO
1	0	1	SENTIDO CONTRA RELOJ
1	1	0	SENTIDO A FAVOR DEL RELOJ
1	1	1	DETENIDO
0	X	X	DESHABILITADO

Tabla 4.1: Modos de operación del módulo L298N

El siguiente paso es conocer las entradas y salidas de propósito general (GPIOs) de la Raspberry Pi que ayudarán a conectarse al módulo para poder manipularlo. En la Figura 4.13 se muestran las GPIOs del modelo de Raspberry usado.

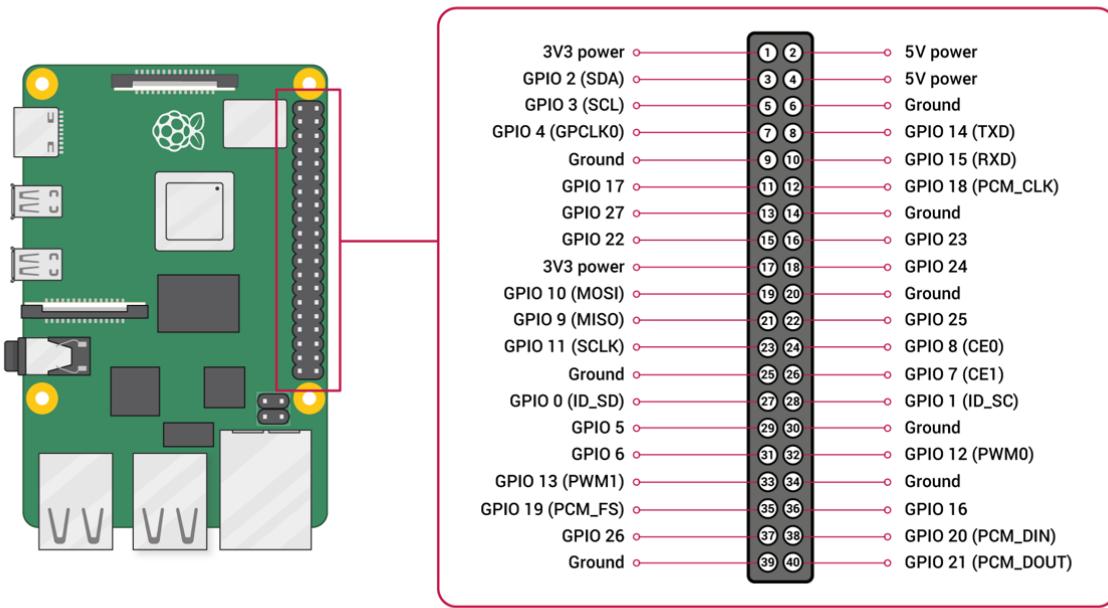


Figura 4.13: GPIO de Raspberry Pi

Posteriormente se realiza la conexión entre los motores, los GPIOs de la Raspberry Pi y las fuentes de alimentación. Las entradas y salidas de control ENA, IN1 e IN2, las cuales gestionan al motor A, son conectadas a los GPIOs 18, 23 y 24 respectivamente, mientras que las entradas y salidas ENB, IN3, IN4, que dominan al motor B, son conectadas a los GPIOs 17, 27 y 22 respectivamente. Las salidas OUT1 y OUT2 se conectan al motor A, mientras que OUT3 y OUT4 al motor B. Finalmente, la fuente de alimentación del módulo se conecta a las entradas de +12V y GND. Además, una de las salidas de GND va conectada al pin 14 de la Raspberry Pi. Dicha fuente de alimentación puede ser un portapilas o una powerbank de 10000 mAh de carga con una salida de 2A aproximadamente. Esta conexión se puede visualizar en la Figura 4.14.

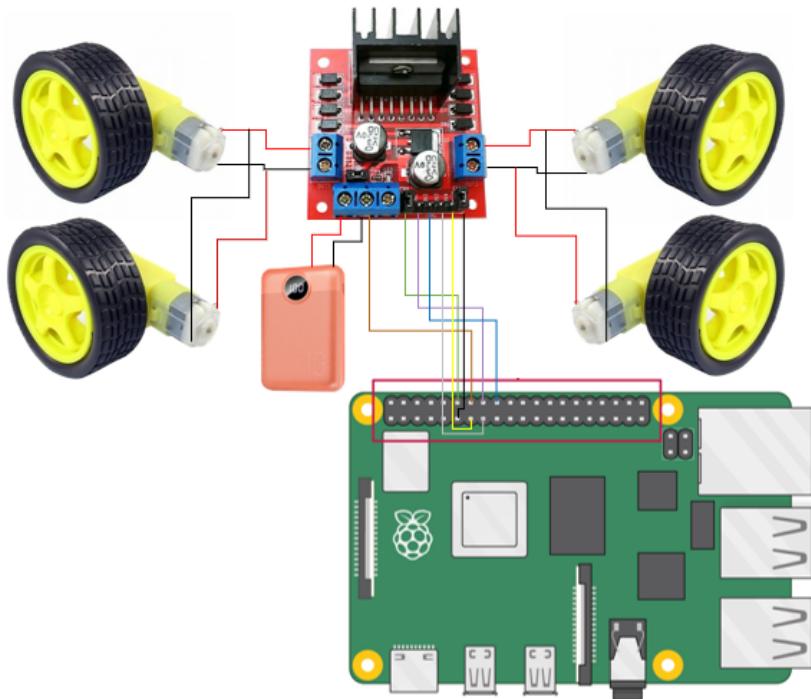


Figura 4.14: Conexión de módulo L298N, Raspberry Pi, motores y fuente de alimentación.

A continuación se muestra el proceso de armado y conexión realizado de forma física utilizando el módulo L298N, la Raspberry Pi y una powerbank de 10,000 mAh como fuente de alimentación, usando un cable USB.

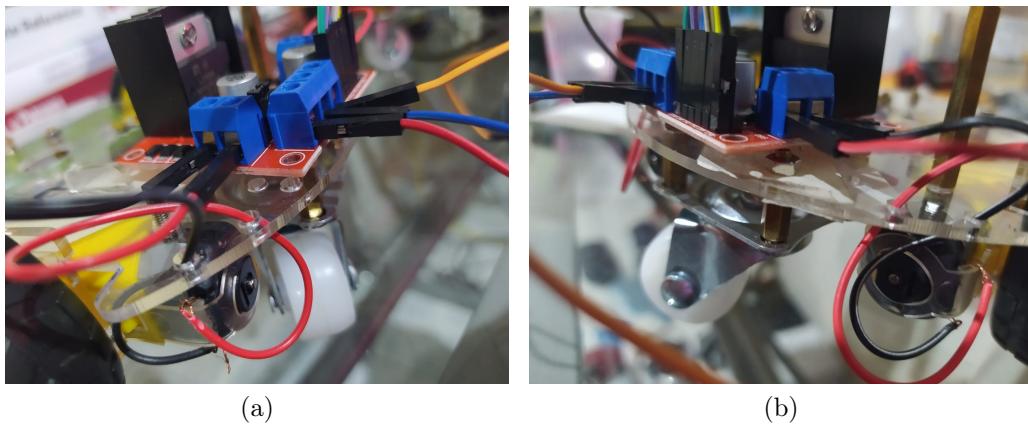


Figura 4.15: Vistas del módulo L298N (a) lado del motor izquierdo, (b) lado del motor derecho

Para poder hacer que uno de los motores giren con una velocidad constante, es suficiente conectar IN1 a la tierra (GND) de la fuente, e IN2 al voltaje, tal, deshabilitando la salida ENA, como se muestra en la Figura 4.17. De forma física, el proceso luce como en la Figura 4.18

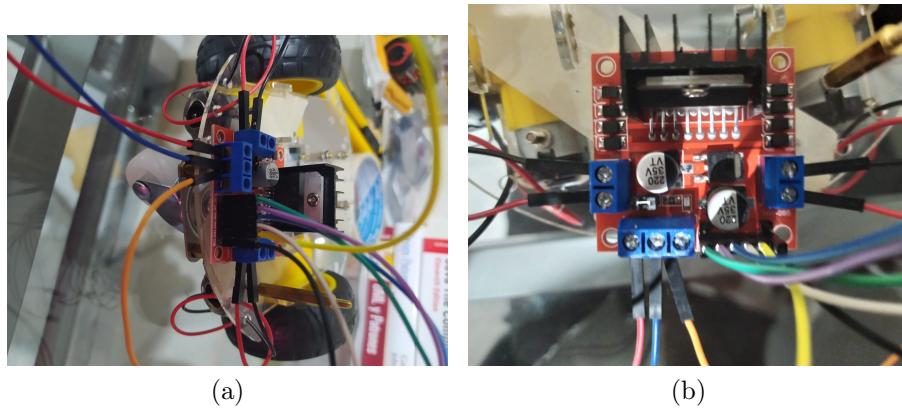


Figura 4.16: Vistas del módulo L298N (a) frontal, (b) superior



Figura 4.17: Giro constante de un motor

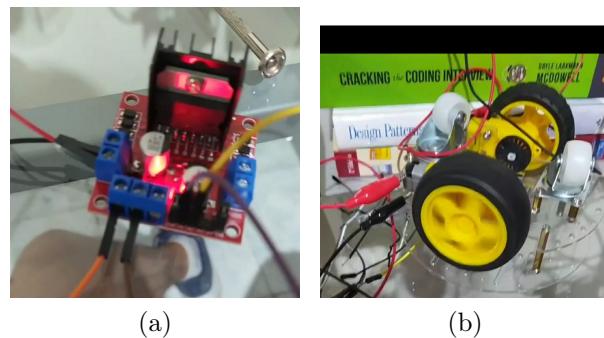


Figura 4.18: Vista de armado para giro constante de un motor

El video de la demostración del funcionamiento de los motores se puede visualizar [aquí](#).

Prueba Unitaria 02

Usando los componentes discutidos anteriormente al igual que las conexiones establecidas se realizaron pruebas individuales de los motores conectados al driver L298N. Estas pruebas se ejecutaron usando 4 funciones, 2 por cada motor. Cada par de funciones representan un giro a favor y en contra de las manecillas del reloj, es decir, una vuelta a la derecha e izquierda respectivamente, durante la cual un motor se mantiene estacionario, mientras el otro se mueve en la dirección correspondiente (hacia adelante o hacia atrás).

A continuación se muestra la definición de las 4 funciones mencionadas:

```

1 def Giro_Favor_Reloj_MotorA():
2     GPIO.output(in1, False)
3     GPIO.output(in2, True)
4
5 def Giro_Contra_Reloj_MotorA():
6     GPIO.output(in1, True)
7     GPIO.output(in2, False)
8
9 def Giro_Favor_Reloj_MotorB():
10    GPIO.output(in3, False)
11    GPIO.output(in4, True)
12
13 def Giro_Contra_Reloj_MotorB():
14    GPIO.output(in3, True)
15    GPIO.output(in4, False)
```

Para cerciorarse del funcionamiento de cada motor en ambas direcciones el programa solicita comandos pre-establecidos de la siguiente manera: **af80**, donde la primer letra representa el motor a mover (a o b), la segunda letra representa la dirección (adelante o *forward* y atrás o *backward*) y por último, la cifra del final representa la potencia con la que se ejecutara la acción en el motor. El comando anterior realiza un movimiento hacia adelante del motor a con una potencia de 80. La instrucción dada se ejecutara hasta el paro del programa o hasta que se de una instrucción diferente:

```

1 while True:
2     cmd = input("inserte el comando ")
3     cmd = cmd.lower()
4     motor = cmd[0]
5     direccion = cmd[1]
6     velocidad = cmd[2:5]
7
8     if motor == "a":
9         if direccion == "f":
10             Giro_Favor_Reloj_MotorA()
11             print("motor A, CW, vel="+velocidad)
12         elif direccion== "r":
13             Giro_Contra_Reloj_MotorA()
14             print("motor A, CCW, vel="+velocidad)
15         else:
16             print("comando no reconocido")
17             pwm_a.ChangeDutyCycle(int(velocidad))
18             print
```

```

20     elif motor == "b":
21         if direccion == "f":
22             Giro_Favor_Reloj_MotorB()
23             print("motor B, CW, vel="+velocidad)
24         elif direccion == "r":
25             Giro_Contra_Reloj_MotorB()
26         else:
27             print("comando no reconocido")
28             pwm_b.ChangeDutyCycle(int(velocidad))
29             print
30     else:
31         print
32         print("comando no reconocido")
33         print

```

Para poder detener los motores al final de las pruebas se ejecuta el código dentro de un bloque *try-catch*, de esta manera el usuario puede detener el programa con la señal del sistema operativo Ctrl+C.

```

1 except KeyboardInterrupt:
2     pwm_a.stop()
3     pwm_b.stop()
4     GPIO.cleanup()
5     os.system('clear')
6     print
7     print("Programa Terminado por el usuario")
8     print
9     exit()

```

Un par de videos de la demostración del funcionamiento del programa se encuentran en los siguientes enlaces: **Explicación breve de la conexión**, **Funcionamiento**.

Prueba Unitaria 03

Una vez logrado lo anterior se procedió a moldear el código con una aproximación más orientada a objetos, con el fin de convertir el código de manejo de los motores en un único módulo que pudiera ser fácilmente importado en el programa principal. Además, se redujo el número de funciones a solo dos: “move()” y “stop()”. El módulo de motores toma como parámetros para creación de la instancia “motor” los pines GPIO a usar para cada uno de los dos motores, respectivamente: ENA, IN1, IN2, IN3, IN4 y ENB.

La función “move()” es la encargada de mover los motores en la dirección deseada, esta toma tres parámetros los cuales son: velocidad, giro y tiempo. La velocidad es un valor entre -1.0 y 1.0 que asigna la cantidad de ciclo de trabajo de la señal PWM enviada hacia los motores, en caso de ser un valor negativo indica a los motores moverse en reversa. El giro es un valor entre -1.0 y 1.0 que influye en la cantidad de activación de cada motor, es decir, cuando el valor es negativo se activa más el motor derecho permitiendo un giro a la izquierda, y viceversa cuando se trata de un valor positivo. Por último, el tiempo es un valor mayor a 0, en segundos, que indica cuánto tiempo deben moverse los motores con los parámetros de velocidad y giro correspondientes. El código de dicha función se puede observar a continuación.

```

1  def move(self, speed = 50, turn = 0, time = 0):
2      # Normalizing the values
3      speed *= 100
4      turn *= 100
5
6      leftSpeed = speed - turn
7      rightSpeed = speed + turn
8
9      if leftSpeed > 100:
10         leftSpeed = 100
11     elif leftSpeed < -100:
12         leftSpeed = -100
13     if rightSpeed > 100:
14         rightSpeed = 100
15     elif rightSpeed < -100:
16         rightSpeed = -100
17
18     self.pwmA.ChangeDutyCycle(abs(leftSpeed))
19     self.pwmB.ChangeDutyCycle(abs(rightSpeed))
20
21     if leftSpeed > 0:
22         GPIO.output(self.In1A,GPIO.HIGH)
23         GPIO.output(self.In2A,GPIO.LOW)
24     else:
25         GPIO.output(self.In1A,GPIO.LOW)
26         GPIO.output(self.In2A,GPIO.HIGH)
27
28     if rightSpeed > 0:
29         GPIO.output(self.In1B,GPIO.HIGH)
30         GPIO.output(self.In2B,GPIO.LOW)
31     else:
32         GPIO.output(self.In1B,GPIO.LOW)
```

```

33         GPIO.output(self.In2B,GPIO.HIGH)
34
35     sleep(time)

```

La función “stop()” recibe un solo parámetro y este es la cantidad de tiempo que los motores estarán sin moverse, definido en segundos. El código de dicha función se puede observar a continuación.

```

1 def stop(self, time = 0):
2     self.pwmA.ChangeDutyCycle(0)
3     self.pwmB.ChangeDutyCycle(0)
4     sleep(time)

```

Para hacer la prueba más dinámica del módulo de los motores, se desarrolló un módulo que es capaz de detectar cuando una tecla se presiona en el teclado, de esta manera se programó un código de ejemplo en el cual se detectan las teclas arriba, abajo, derecha e izquierda y se mueve al robot de acuerdo a las teclas. La función principal del módulo de detección de teclas se encuentra a continuación, dicha función es llamada “isKeyPressed()” y toma como único parámetro el nombre de la tecla de la cual nos interesa saber si está siendo presionada.

```

1 def isKeyPressed(keyName):
2     wasPressed = False
3     for event in pygame.event.get():
4         pass
5
6     keyInput = pygame.key.get_pressed()
7
8     myKey = getattr(pygame, 'K_{}'.format(keyName))
9     if keyInput[myKey]:
10        wasPressed = True
11
12     pygame.display.update()
13     return wasPressed

```

La función principal del código de prueba para el movimiento del robot utilizando teclas se muestra a continuación.

```

1 def main():
2     ##### TO TEST WITH KEYBOARD
3     if kp.isKeyPressed('UP'):
4         motor.move(0.8, 0, 0.1)
5     elif kp.isKeyPressed('DOWN'):
6         motor.move(-0.8, 0, 0.1)
7     elif kp.isKeyPressed('LEFT'):
8         motor.move(0.8, -0.5, 0.1)
9     elif kp.isKeyPressed('RIGHT'):
10        motor.move(0.8, 0.5, 0.1)
11    else:
12        motor.stop(0.1)

```

La función anterior es ejecutada indefinidamente; se mantiene checando si cualquiera de las teclas de interés: arriba, abajo, derecha e izquierda, es presionada y, de ser el caso, se

mueven los motores hacia adelante, atrás, derecha o izquierda, respectivamente.

El programa anterior se cargó en la tarjeta Raspberry Pi, luego se procedió a controlar la tarjeta a través de VNC para ejecutar el programa y, de esta manera, cuando una tecla se presionó en la computadora que se usó para conectarse por medio de VNC el robot se movió acorde a. La conexión de la circuitería se describió previamente en 4.14. El armado luce como se muestra a continuación; utilizando una powerbank de capacidad nominal de 10,000mAh con dos salidas de 2A: una para alimentar la tarjeta Raspberry Pi y otra para el driver LN298N.

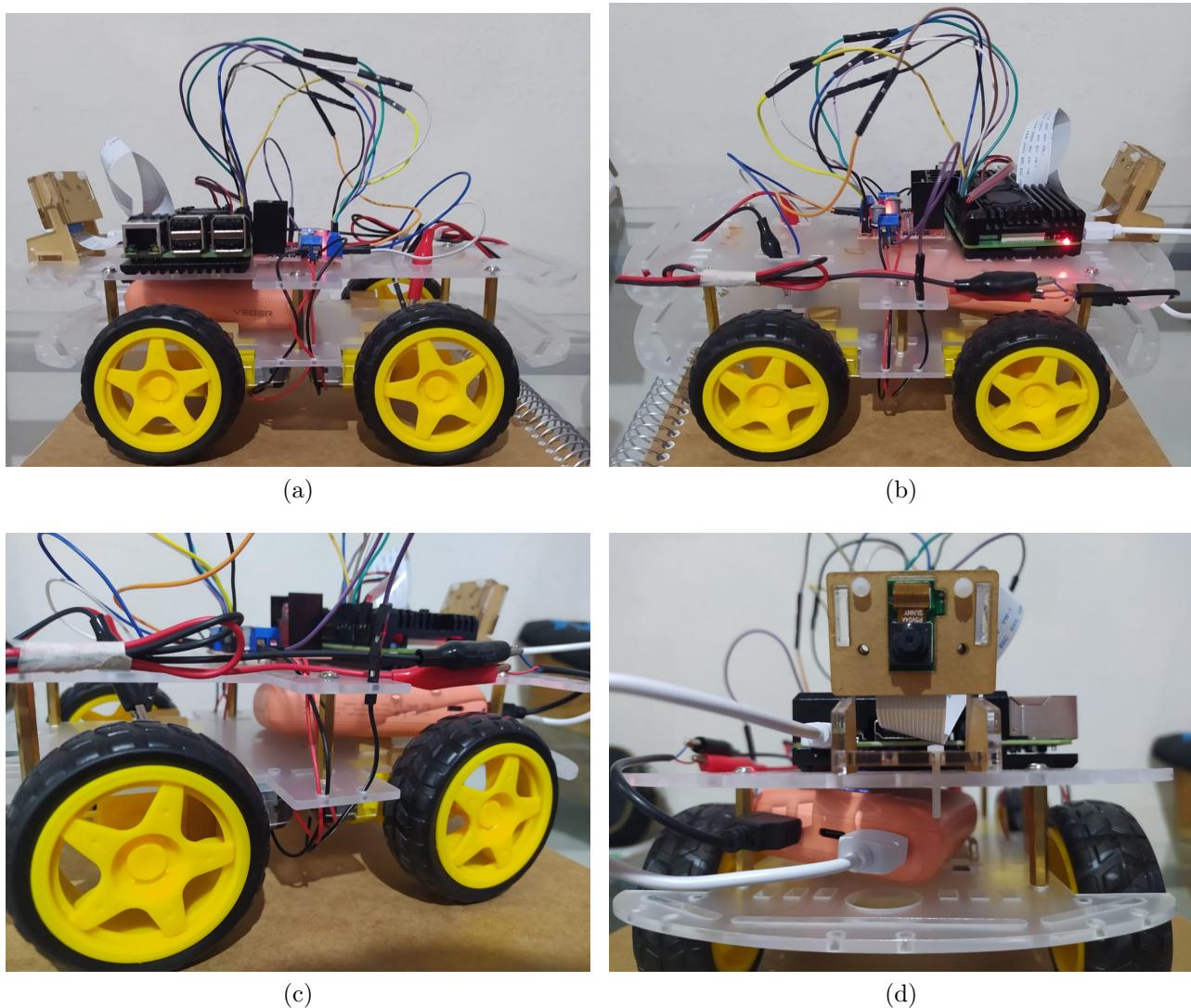


Figura 4.19: Vistas del armado del robot para prueba de motores.

A continuación se muestra un vídeo de demostración del funcionamiento del robot con el chasis de 4 ruedas:

- **Funcionamiento del robot moviéndose con teclas.**

4.4. Módulo de Navegación Autónoma

Objetivo

Implementar la funcionalidad descrita en los requerimientos funcionales RF04 y RF05 (Sección 3.2), que se requiere para cumplir con el diseño del módulo de navegación autónoma y visualización de imágenes.

Desarrollo

Una vez que se han realizado las transformaciones sobre la imagen, para proceder con el algoritmo de navegación ha sido necesario segmentar el camino a partir de su matiz, saturación y brillo (HSV), y luego ajustar la perspectiva del camino utilizando un polígono regular.

Posteriormente, uno de los aspectos más relevantes del algoritmo de navegación es encontrar las curvas a través del camino que se recorre. Para realizar esto, se ha recurrido a una técnica llamada “Sumarización”. Considerando la imagen que se tiene, con el ajuste de perspectiva y binarizada, es decir, tiene píxeles blancos o negros, se pueden sumar los valores de los píxeles en la dirección vertical, tal y como se muestra en la Figura 4.20

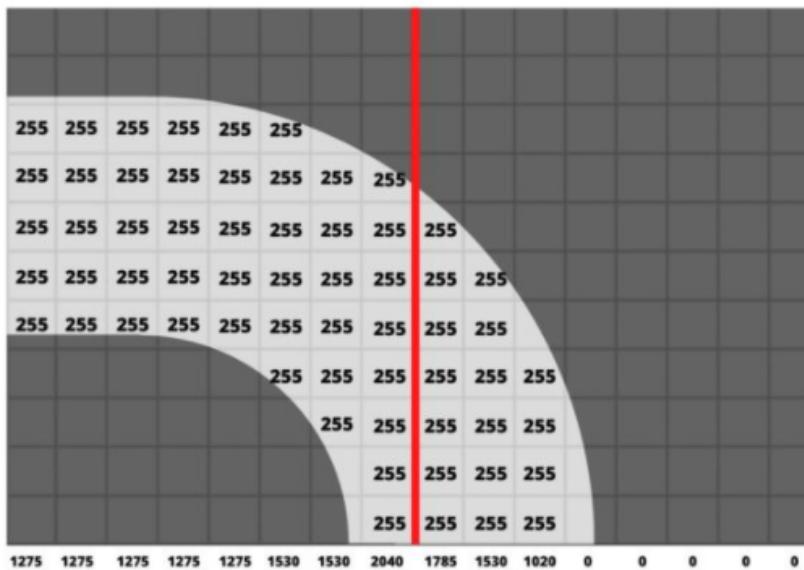


Figura 4.20: Representación de una curva del camino

La imagen apreciada arriba muestra que los píxeles blancos son representados con un valor de 255, mientras que los negros con 0. Si se realiza una suma de forma vertical, o sea por cada una de las columnas, se puede obtener un valor representativo por cada columna, para la primera columna se tiene lo siguiente:

$$255 + 255 + 255 + 255 + 255 = 1275 \quad (4.1)$$

Si este proceso es llevado por cada una de las columnas, considerando que la imagen utilizada es de 240x480, se obtienen 480 valores. Se puede considerar la siguiente representación matemática generalizada para obtener el j-ésimo valor:

$$Valor_j = \sum_{i=0}^{H_T=240} y_i = y_0 + y_1 + \dots + y_{H_T-1} \quad (4.2)$$

Una vez que obtienen dichos valores, el siguiente paso es ver cuántos de ellos están por encima de un determinado umbral, supongamos 1000 de cada lado de la línea roja (en el centro de la imagen). En el ejemplo anterior, se tiene un total de 16 columnas de las cuales 8 columnas de la izquierda contiene un valor mayor al umbral y 3 del lado derecho. Por tal motivo, esto indica que la curva está hacia la izquierda. Este es el concepto básico con el cual el algoritmo de navegación trabaja. Sin embargo, si se profundiza un poco más en esta idea se observan los siguientes problemas.



Figura 4.21: Sumarización de píxeles

En la Figura 4.21 se aprecian los tres escenarios en los que el método anteriormente descrito funcionaría; Se observa claramente que cuando la curva es derecha, el número de píxeles en el lado derecho es mayor que en el izquierdo y viceversa. Y cuando el camino es recto, el número de píxeles es aproximadamente el mismo en ambos lados. Todo es risas y diversión hasta que se presentan dos casos en los que el método fallaría:

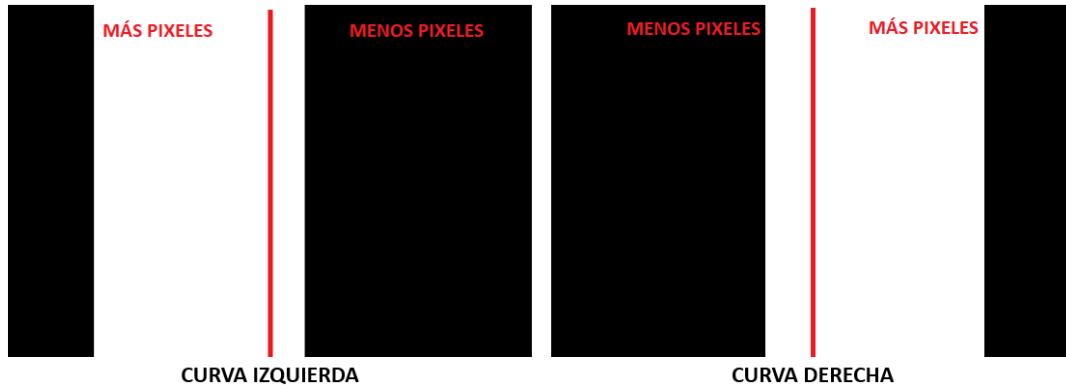


Figura 4.22: Casos singulares de un camino recto

El problema se presenta cuando el método detecta más píxeles de un lado que del otro, entonces determinaría que se trata de una curva izquierda o una curva derecha, como se muestra en la Figura 4.22 . Para solucionar esta situación se debe de **ajustar la línea central** con respecto al camino. Si se realiza este proceso entonces el resultado se vería como en la Figura 4.23:

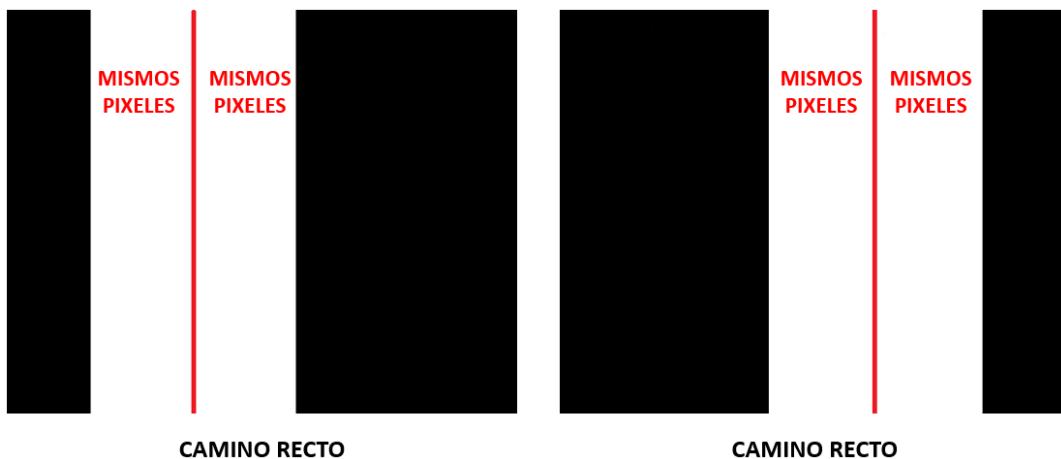


Figura 4.23: Ajuste de línea central del camino

Ahora el algoritmo se enfoca primero en encontrar el centro base que provea la línea central del camino conforme se vaya avanzando a través del circuito para posteriormente comparar los píxeles de ambos lados, es decir: El lado que *tenga mas votos* determina el trayecto. Afortunadamente, ambos procesos pueden ser calculados en una sola función, ya que mediante la sumarización de los píxeles se está encontrando básicamente el histograma y posteriormente se obtiene el valor promedio de dicho histograma previamente indexado.

El primer paso es declarar una función que tome como parámetros la imagen o frame del camino obtenido, así como un valor mínimo que servirá como umbral para considerar al pixel parte del camino. Después se procede a realizar la sumarización, utilizando la función sum() que proporciona el paquete de Numpy, para guardar los valores del histograma.

```
1 def getHistogram(img, minPer=0.1, display= False):
2
3     histValues = np.sum(img, axis=0)
```

En este punto ya se tiene los 480 valores que contiene cada una de las columnas de la imagen. Sin embargo, algunos de los píxeles de la imagen podrían ser ruido, los cuáles no se desean utilizar en el cálculo. Por tal motivo, se procede a utilizar un umbral que será el valor mínimo requerido para que cualquier columna sea considerada como parte de la ruta y no como ruido. Así que se encontrará el valor máximo de la suma y se multiplicará por el porcentaje definido por el usuario (minPer) para crear un valor de umbral según se requiera.

```
1 maxValue = np.max(histValues)
2 minValue = minPer*maxValue
```

A continuación, se podría sumar todo el número de píxeles de cada lado y encontrar la dirección: izquierda, derecha o recta. Pero esto no es lo que realmente se quiere, sino que si la curva es a la derecha se desea saber cuánto a la derecha. Para **obtener el valor de la curvatura** se deben de encontrar los índices de todas las columnas que tienen un valor mayor que el umbral establecido y luego promediar los índices. Ejemplificando esta idea: Si los índices de los píxeles empiezan en 50 y terminan en 280, la media sería: $\frac{(280-50)}{2} + 50 = 165$.

```
1 indexArray = np.where(histValues >= minValue)
2 basePoint = int(np.average(indexArray))
```

De esta forma, el valor base (basePoint) es ahora el punto base de la imagen procesada. Posteriormente, se podría dibujar este punto base para visualizarlo mejor y finalmente regresamos ese punto base, es decir, cuán curvado está el camino.

```

1  if display:
2      imgHist = np.zeros((img.shape[0],img.shape[1],3), np.uint8)
3      for x, intensity in enumerate(histValues):
4          cv2.line(imgHist, (x,img.shape[0]), (x, img.shape[0]-intensity
//255), (255,0,255), 1)
5          cv2.circle(imgHist, (basePoint, img.shape[0]), 20, (0, 255,
255), cv2.FILLED)
6      return basePoint, imgHist
7
8  return basePoint

```

A continuación, en la Figura 4.24 se muestra el histograma de la imagen previamente modificada con el ajuste de perspectiva y binarizada. El circulo amarillo representa **el valor medio**. El valor medio que se obtiene es 235.

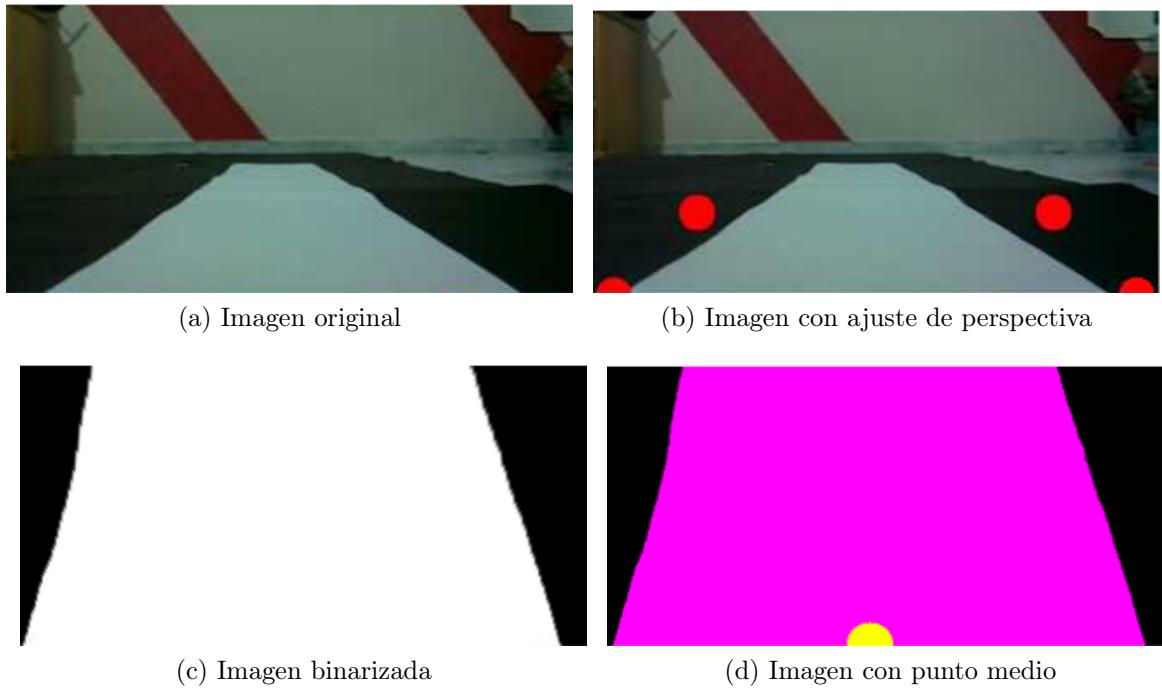


Figura 4.24: Búsqueda de curvatura de camino

Se puede optimizar este método restando el valor del centro para obtener el valor de la curva real. Suponiendo que el centro está en 240 entonces el valor de la curva es $235 - 240 = -5$. Aquí el signo menos indica que la curva está hacia el lado izquierdo (así como los positivos hacia la derecha) y el valor 5 indica la intensidad de la curva. Pero como se ha comentado anteriormente este método no es preciso ya que el centro puede cambiar.

Así que para encontrar el punto medio se puede utilizar la función creada previamente (getHistogram), pero esta vez en lugar de aplicar la técnica del histograma a la imagen completa, se aplica sólo en la parte inferior 1/4 de la imagen.

Esto se debe a que se está interesado en la media de la base por lo que no queremos promediar los píxeles por encima de 1/4 de la imagen. Para lograr esto se puede añadir un argumento de entrada de región y con base al valor de entrada de ésta se puede decidir si promediar toda la imagen o parte de ella.

```

1 def getHistogram(img, minPer=0.1, display= False, region=1):
2
3     height, width = img.shape [:2]
4
5     if region == 1:
6         histValues = np.sum(img, axis=0)
7     else:
8         histValues = np.sum(img[height//region: , :], axis=0)
9
10    maxValue = np.max(histValues)
11    minValue = minPer*maxValue
12
13    indexArray = np.where(histValues >= minValue)
14    basePoint = int(np.average(indexArray))
15
16    if display:
17        imgHist = np.zeros((height, width,3), np.uint8)
18        for x, intensity in enumerate(histValues):
19            cv2.line(imgHist, (x,height), (x, height-intensity//255//region
19 ), (255,0,255), 1)
20            cv2.circle(imgHist, (basePoint, height), 20, (0, 255, 255), cv2
21 .FILLED)
22        return basePoint, imgHist
23
24    return basePoint

```

Así, si la región es 1, se promediará toda la imagen y si la región es 4, se promediará la cuarta parte de la parte inferior. La Figura 4.25 muestra el promedio de la parte inferior.

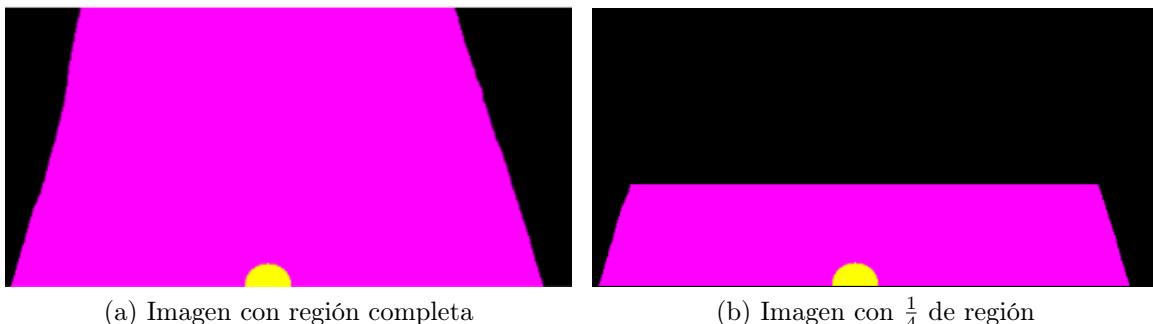


Figura 4.25: Búsqueda de curvatura de camino

Ahora el valor medio que se obtuvo es 228. Esto significa que el centro real de la imagen es 228 en lugar de 240. Ahora se puede restar el valor medio que se obtuvo antes de este punto central, así que $235 - 228 = 7$. Antes el valor era -5 y ahora se tiene 7. Mirando la imagen deformada se puede apreciar que la intensidad de la curva es mínima ya que se trata de un trayecto recto, por lo tanto esto confirma que el segundo método da un mejor resultado.

El archivo que controla y orquesta las transformaciones que se realizan sobre la imagen, tales como: segmentación, ajuste de perspectiva, obtención de histograma, etc., es el archivo **LaneDetectionModule.py**. Dentro de él, se ha creado una función llamada **getLane**, la cual obtiene los valores correspondientes que se ajusten al camino para poder realizar dichas transformaciones.

El primer paso es obtener la imagen segmentada para posteriormente ajustar la perspectiva con puntos superiores e inferiores que forman un polígono regular.

```

1 def getLane(img, display = 2):
2
3     imgResult = img.copy()
4     imgCopy = img.copy()
5     hT, wT = img.shape [:2]
6
7     # Segmentamos la imagen del camino
8     imgThres = util.thresholding(img)
9
10    # Obtenemos los puntos de la perspectiva transformada
11    widthTop = 136
12    heightTop = 134
13    widthBottom = 20
14    heightBottom = 186
15    points = np.float32([(widthTop, heightTop), (wT - widthTop, heightTop),
16                          (widthBottom, heightBottom), (wT - widthBottom,
17                          heightBottom)])
18
19    # Transformamos la perspectiva del camino
20    imgWarp = util.warpImage(imgThres, points, wT, hT)
21
22    # Dibujamos los puntos del polígono usado para
23    # la transformacion de perspectiva del camino
24    imgWarpPoints = util.drawPoints(imgCopy, points)
```

El siguiente paso es obtener el punto medio para una región del camino, en este caso de $\frac{1}{4}$, así como obtener el punto medio de toda la región del camino. Todo esto para obtener un mejor valor del punto base.

```

1     # Obtenemos el punto medio central para una region del camino
2     middlePoint, imgHist = util.getHistogram(imgWarp, minPer = 0.5, display=
3     True, region = 4)
4
5     # Obtenemos el punto medio central de la imagen completa
6     curveAveragePoint, imgHist = util.getHistogram(imgWarp, minPer = 0.9,
7     display=True)
8
8     # Determinamos el valor de la curvatura de nuestro camino
9     realCurve = curveAveragePoint - middlePoint
```

Una vez que se tiene el valor de la curva, se añade en una lista para poder promediar este valor. Con esto reducimos una oscilación del ruido al momento de detectar curvas, para evitar valores muy altos o muy pequeños. Es decir, el promediado permitirá un movimiento suave y evitará cualquier movimiento dramático.

```

1 # Reducimos la oscilacion del ruido al momento de detectar curvas
2 # para evitar valores muy altos o muy pequeños para obtener
3 # transiciones suaves
4 curveList.append(realCurve)
5 if (len(curveList) > avgVal):
6     curveList.pop(0)
7 curve = int(sum(curveList) / len(curveList))

```

Posteriormente, se normalizarán los valores del punto base, es decir, los valores de la curvatura con el fin de que éstos vayan en un rango de -1 a 1 para detectar curvas izquierdas, caminos rectos y curvas derechas. Esto con el fin de poder utilizar dicho valor de curvatura en la función de movimiento del robot, descrito en la Sección 4.3.

```

1 # Es una normalizacion de los datos para poder
2 # detectar las curvas izquierdas o derechas
3 curve /= 100
4 if (curve > 1):
5     crurve = 1
6 if (curve < -1):
7     curve = -1

```

Por último, para su uso en depuración y ajuste de los distintos parámetros del algoritmo, se necesita visualizar la imagen resultado así como también las salidas de los distintos procesos por los que se pasa la imagen capturada para llegar a la imagen final. Sin embargo, una vez establecidos los parámetro correctos, se necesita la mayor cantidad de velocidad de procesamiento disponible para la ejecución del algoritmo, por lo que se estableció la variable *display* que actúa como bandera de opciones con los valores 0, 1 y 2. 0 se ejecuta el algoritmo sin mostrar nada, 1 se muestra la imagen resultante del proceso y 2 se muestra todo el *pipeline* de procesamiento.

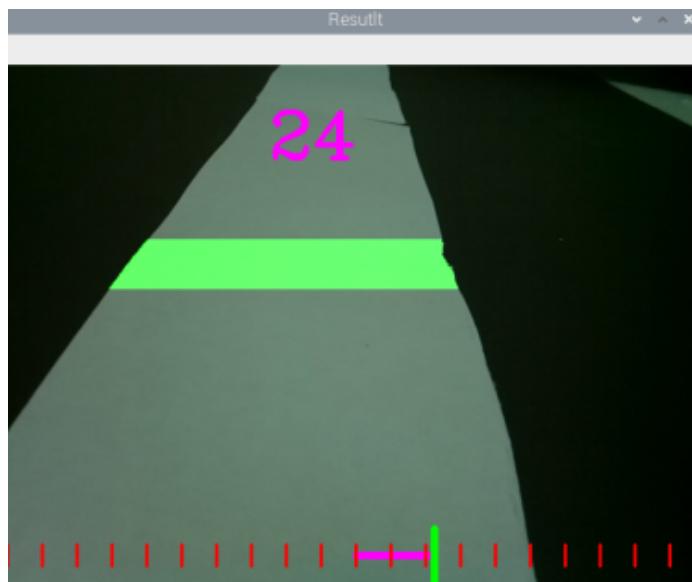


Figura 4.26: Ejecución del algoritmo con *display*=1, mostrando solo la imagen resultante

Para mostrar de manera más concisa, en lugar de tener una ventana por cada salida de un proceso del algoritmo, se juntan las imágenes resultantes en una única ventana por medio de la función *stackImages* la cual recibe como parámetro una valor de escala (*scale*) para las imágenes y un arreglo bidimensional (3x2) que contendrá las imágenes resultantes de cada uno de los 6 procedimientos del algoritmo para mostrarse 3 arriba y 3 abajo. El orden de estas imágenes dentro de la ventana contenedora de izquierda a derecha es: imagen capturada, imagen mostrando los puntos de perspectiva sobre la imagen capturada, la imagen después de la umbralización y *warping* (ajuste de perspectiva), la imagen procesada con el punto medio calculado, el área de interés y por último la imagen resultante.

```

1 # Visualizamos el trayecto del robot
2 if display != 0:
3     # Obtenemos la region ubicada a partir del
4     # cambio de perspectiva en la imagen original
5     imgInvWarp = util.warpImage(imgWarp, points, wT, hT, inv = True)
6     imgInvWarp = cv2.cvtColor(imgInvWarp, cv2.COLOR_GRAY2BGR)
7     imgInvWarp[0:hT//3,0:wT] = 0,0,0
8
9     imgLaneColor = np.zeros_like(img)
10    imgLaneColor[:, :] = 0, 255, 0
11    imgLaneColor = cv2.bitwise_and(imgInvWarp, imgLaneColor)
12
13    imgResult = cv2.addWeighted(imgResult, 1, imgLaneColor, 1, 0)
14    midY = 450
15    cv2.putText(imgResult, str(curve), (wT//2-80, 85), cv2.
16    FONT_HERSHEY_COMPLEX, 2, (255, 0, 255), 3)
17    cv2.line(imgResult, (wT//2, midY), (wT//2+(curve*3), midY), (255, 0, 255)
18    , 5)
19    cv2.line(imgResult, ((wT // 2 + (curve * 3)), midY-25), (wT // 2 +
20    (curve * 3), midY+25), (0, 255, 0), 5)
21    for x in range(-30, 30):
22        w = wT // 20
23        cv2.line(imgResult, (w * x + int(curve//50 ), midY-10),
24                  (w * x + int(curve//50 ), midY+10), (0, 0,
25      255), 2)
26
27 if display == 2:
28     imgStacked = util.stackImages(0.7, ([img, imgWarpPoints, imgWarp], [
29     imgHist, imgLaneColor, imgResult]))
30     cv2.imshow('ImageStack', imgStacked)
31 elif display == 1:
32     #cv2.imshow('Threshold', imgThres)
33     #cv2.imshow('Warp', imgWarp)
34     #cv2.imshow('Warp Points', imgWarpPoints)
35     cv2.imshow('Histograma', imgHist)
36     cv2.imshow('Lane Color', imgLaneColor)
37     cv2.imshow('Img Result', imgResult)
38
39 return curve

```

El despliegue del apilado de imágenes se puede apreciar en la Figura 4.27

A continuación, se muestra un video donde se aprecia el la vista del funcionamiento del

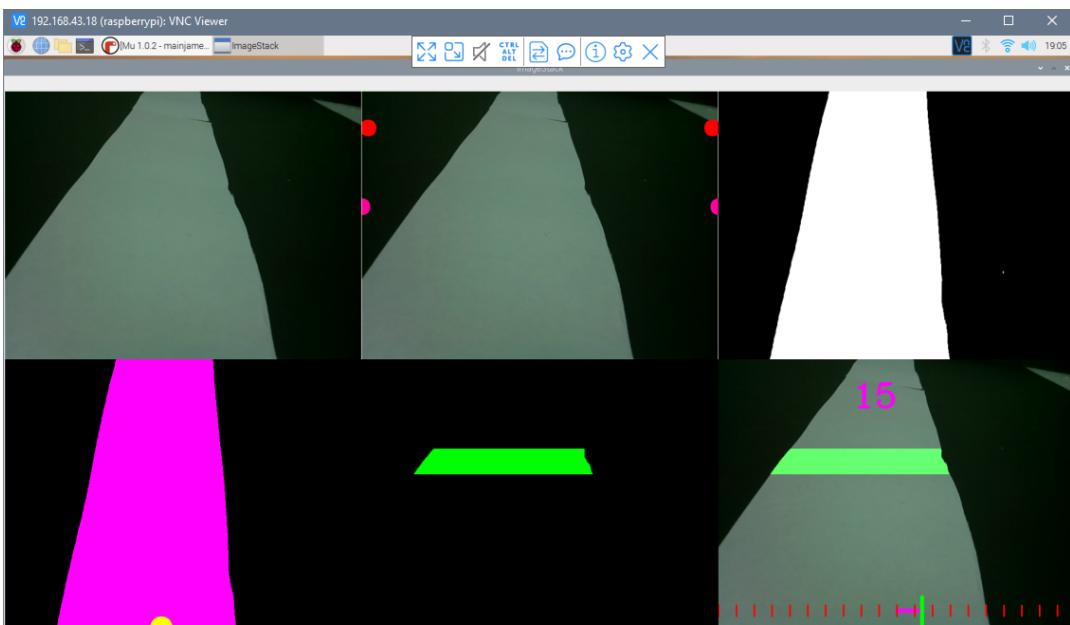


Figura 4.27: Ejecución del algoritmo con $display=2$, es decir mostrando cada paso del algoritmo

algoritmo de navegación:

- **Vista del algoritmo de navegación**

4.5. Módulo de Aplicación Web

Objetivo

Implementar la funcionalidad descrita en los requerimientos funcionales RF08 (Sección 3.2), que se requiere para cumplir con el diseño del módulo de la aplicación web.

Desarrollo

Servidor

Para el desarrollo de este modulo y como se mencionó en Herramientas de Desarrollo, se optó por utilizar el servidor HTTP gratis y de código abierto más popular en la actualidad: Apache Web Server, y que esta disponible en una gran cantidad de sistemas operativos, incluyendo Raspberry OS. Apache permite servir archivos a través de la web utilizando el protocolo HTTP, en este caso se utilizará para el envío de las imágenes procesadas hacia el cliente.

Para la instalación y configuración de Apache en Raspberry OS se consultó la documentación oficial de Raspberry Foundation, el artículo que se revisó se puede encontrar en [48]. A continuación una descripción de los pasos que se tomaron para dicha configuración.

Lo primero fue abrir una terminal y ejecutar el siguiente comando, el cual actualiza todos los paquetes disponibles en el sistema:

```
1 $sudo apt update
```

A continuación se ejecuta el siguiente comando, el cual realiza la instalación de Apache en el sistema:

```
1 $sudo apt install apache2 -y
```

Hecho lo anterior Apache ha sido instalado como un servicio en el sistema operativo; esto quiere decir que se ejecuta como un programa en segundo plano que escucha peticiones HTTP dirigidas a la Raspberry Pi, sin embargo, hay que realizar una configuración especial para que las imágenes enviadas hacia el cliente se visualizan en su navegador web sin problema. Esta configuración tiene que ver con una política de seguridad de los navegadores web modernos que limita el despliegue de archivos provenientes de un equipo externo (en este caso la Raspberry Pi), esta política es conocida como "Política de uso compartido de recursos de origen cruzado"(CORS por sus siglas en inglés). Bien, para realizar lo anterior se edita el archivo de configuración de Apache que se encuentra en `/etc/apache2/apache2.conf`, y se agrega la línea “Header set Access-Control-Allow-Origin “*”” en la línea 174, como se aprecia en la siguiente imagen 4.28.

```
<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
    Header set Access-Control-Allow-Origin "*"
</Directory>
```

Figura 4.28: Editando el archivo de configuración de Apache

Ahora, para que esta configuración tenga efecto se debe instalar un módulo de Apache llamado “headers” el cual permite personalizar las cabeceras de respuesta HTTP (paso anterior), que se envían hacia el navegador del programa cliente cuando este solicita una imagen. Para instalar dicho módulo se ejecuta el siguiente comando:

```
1 $sudo a2enmod headers
```

Y se procede a reiniciar Apache con el comando:

```
1 $sudo service apache2 restart
```

Para comprobar que la configuración ha tomado efecto y Apache se encuentra corriendo correctamente se ejecuta el siguiente comando, el cual devuelve el estado del servicio:

```
1 $sudo service apache2 status
```

En la terminal se debe observar en verde “active (running)”, lo cual quiere decir que la configuración fue realizada con éxito, como se aprecia en la siguiente imagen 4.29.

```
[pi@raspberrypi:~] $ sudo service apache2 status
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2021-03-13 22:10:23 CST; 2 days ago
     Docs: https://httpd.apache.org/docs/2.4/
 Process: 464 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
 Process: 1018 ExecReload=/usr/sbin/apachectl graceful (code=exited, status=0/SUCCESS)
 Main PID: 497 (apache2)
    Tasks: 55 (limit: 1939)
   CGroup: /system.slice/apache2.service
           └─ 497 /usr/sbin/apache2 -k start
               ├─ 1017 /usr/sbin/apache2 -k start
               ├─ 1018 /usr/sbin/apache2 -k start
```

Figura 4.29: Estado del servidor Apache

Una vez hecho lo anterior, se procedió a crear una carpeta llamda “images” en el directorio especial de Apache `/var/www/html/`, el módulo de Navegación Web se programa de tal forma que las imágenes de monitoreo creadas se coloquen en este directorio, pues solo los archivos que se encuentran en él pueden ser servidos por Apache ante alguna petición.

Para comprobar su funcionamiento se accedió desde una computadora cliente a la página por defecto que Apache muestra tras una instalación correcta, como se puede apreciar en la siguiente imagen 4.30. Nótese que en el navegador se escribió la dirección IP de la Raspberry Pi, es decir; no se está accediendo a localhost.

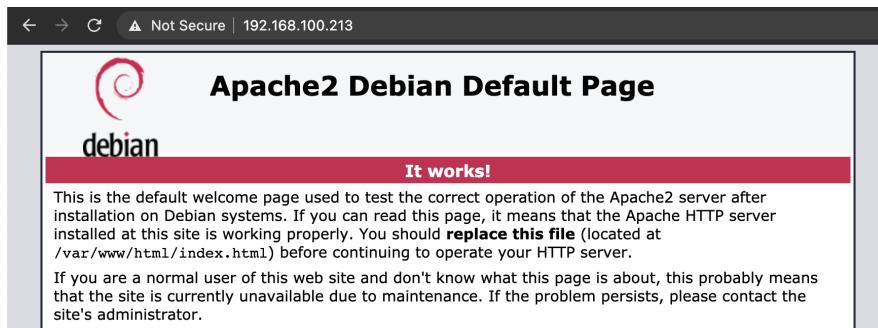


Figura 4.30: Probando instalación correcta de Apache Web Server

Cliente

El programa cliente consta de dos archivos: “index.html” y “style.css”. El primero es un archivo html que contiene un script programado en Javascript, este realiza una petición HTTP al servidor para obtener la imagen procesada por la Raspberry Pi y la despliega en el navegador web. El segundo es una hoja de estilos que le da un aspecto más amigable a la interfaz web. La estructura del programa cliente se muestra a continuación 4.31

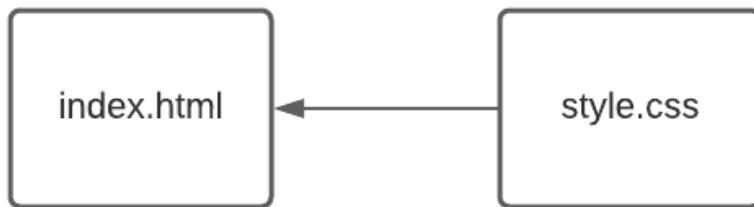


Figura 4.31: Estructura del programa cliente

El programa cliente se desarrolló en lenguaje Javascript, por ser este el lenguaje predilecto para los navegadores web. El programa se ejecuta de manera automática cuando se abre desde un navegador web, pues se trata de una página web con un script que ejecuta una petición HTTP hacia el servidor cada 2 segundos, de esta manera se refresca la imagen presentada.

Dicho código se encuentra en el archivo “index.html”, a continuación se muestra la función que hace la petición y refresca la imagen presentada en el navegador:

```

1 let xhr = new XMLHttpRequest();
2
3 xhr.open('GET', 'http://192.168.100.213:80/images/sobel.bmp', true);
4 xhr.responseType = "arraybuffer";
5
6 xhr.onload = function()
7 {
8     if(this.status == 200)
9     {
10         let img_bytes = new Uint8Array(this.response);
  
```

```

11     let decoder = new TextDecoder();
12     let img_base64 = bytesToBase64(img_bytes);
13     document.getElementById("sobelimage").src = "data:image/bmp;base64,
14     " + img_base64;
15     console.log(document.getElementById("sobelimage"));
16   }
17 }

```

Para darle un aspecto más amigable a la interfaz de la página web se desarrolló una hoja de estilos, dicho archivo se llama “style.css” y al cual se referencia desde “index.html”.

Prueba Unitaria

Se procedió a realizar una prueba con la tarjeta Raspberry Pi equipada con la cámara y siendo accedida desde una terminal en una laptop utilizando el protocolo SSH. Una vez se accedió utilizando SSH, se procedió a ejecutar el programa principal. Hecho esto, la tarjeta se encuentra tomando imágenes, aplicando el procesamiento necesario y creando las imágenes de monitoreo que se guardan en la carpeta “images”.

A continuación, y desde una laptop se procedió a abrir el programa web cliente en el navegador Google Chrome para visualizar las imágenes procesadas, dicha prueba se puede apreciar en la siguiente imagen 4.32.

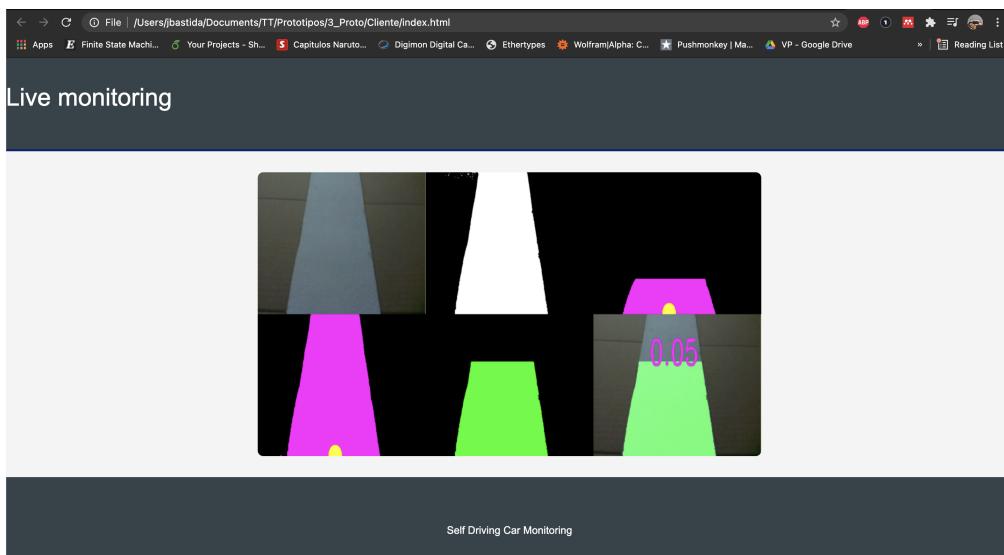


Figura 4.32: Programa web cliente en donde se puede visualizar la imagen de monitoreo que comprende las imágenes tomadas y procesadas por la Raspberry Pi.

4.6. Integración de Módulo de Navegación y Motores

Objetivo

Integrar el módulo de Navegación Autónoma junto con el módulo de Motores para determinar los comandos necesarios para el movimiento robot.

Desarrollo

Una vez se ha obtenido el valor de curvatura del camino, es decir, el valor que indica cuán a la derecha, izquierda o al centro se encuentra el robot, se envía como parámetro a la función de **move()** explicada en la Sección 4.3 - Prueba Unitaria 03.

Para llevar a cabo ello, es necesario crear un archivo que permite comunicarse con los distintos módulos desarrollados, dicho archivo es llamado **main.py**. En el, se importan los módulos de adquisición de imagen, modulo de navegación y modulo de motores, para posteriormente inicializar un objeto de la clase **ModuleMotor** para especificar aquellos GPIOs de la Raspberry Pi que se utilizan para comunicarse con los motores del robot, haciendo uso del modulo L298N.

```

1 from MotorModule import Motor
2 import CameraModule
3 from LaneDetectionModule import getLaneCurve
4 import cv2
5
6 ######
7 motor = Motor(2, 3, 4, 17, 22, 27)

```

Además, se utilizan dos variables: **firstTime** y **imgCounter** que permiten ubicar el comienzo de la aplicación para capturar la imagen y la segunda mantiene el conteo de la imagen del trayecto capturada para guardarla y mostrarla, posteriormente, en la aplicación web.

```

1 imgCounter = 0
2 firstTime = 1

```

Dentro de la función principal, el primer paso es obtener la imagen del trayecto, a partir de la cámara conectada a la Raspberry Pi, utilizando la función **getImage()**. Luego, se realiza el procesamiento digital de imágenes (PDI) (segmentación y ajuste de perspectiva) para aplicar el algoritmo de navegación autónoma. Tanto el PDI como el algoritmo de navegación están contenidos en la función **getLane()**, cuyos parámetros son dos: la imagen previamente obtenida (**img**), y la opción de despliegue de las imágenes procesadas. En este caso, el segundo parámetro es un **1** ya que no se quiere mostrar las imágenes, solamente se necesitan el valor de curvatura (almacenado en la variable **curveVal**) y la imagen resultante (véase la Figura 4.27). Posteriormente, se guarda la imagen obtenida en la ruta del servidor montando, utilizando la variable contador **imgCounter** y la función **imwrite** del modulo Open CV.

```

1 def main():
2     global imgCounter
3     global firstTime
4
5     ##### PDI y ANA
6     img = CameraModule.getImage()
7     curveVal, imgResult = getLaneCurve(img, 1)
8
9     ##### IMG WRITING
10    cv2.imwrite(f'/var/www/html/images/warp{imgCounter}.jpg', imgResult)

```

A partir de la segunda iteración, se establece un valor de sensibilidad (**sen**) el cual permite percibir y determinar, junto con el valor de curvatura, cuanto girar cada motor y hacia que dirección. Además, es necesario establecer un ciclo de trabajo (**dutyCycle**) para establecer la velocidad en la que los motores se moverán, así como un tiempo de movimiento (**time**).

Cuando se ha obtenido el valor de curvatura, se debe de categorizar para determinar la dirección en la que el robot debe de moverse y/o alinearse. Para hacer esto, se ha establecido la siguiente categorización:

curveVal	Comando	Sen
(-0.90, -0.28]	Curva izquierda	3
(-0.28, -0.10]	Alinear izquierda	1
(-0.10, 0.10)	Recto	1
[0.10, 0.28)	Alinear a la derecha	1
[0.28, 0.90)	Curva derecha	3

Al utilizar esta categorización, es mas sencillo determinar el comando a utilizar y la dirección en la cual se mueve el robot. Finalmente, se utiliza la función **move()** para mandar la velocidad, el tiempo y el valor de giro, representado por el producto de la sensibilidad y el valor de curvatura.

```

1     if (curveVal > 0):
2         print(f"{imgCounter} DERECHA {dutyCicle}, {-curveVal * sen}, {
3             time}")
4     elif (curveVal < 0):
5         print(f"{imgCounter} IZQUIERDA {dutyCicle}, {-curveVal * sen},
6             {time}")
7     else:
8         print(f"{imgCounter} RECTO {dutyCicle}, {-curveVal * sen}, {
9             time}")
10
11    motor.move(dutyCicle, curveVal * sen, time)
12    imgCounter += 1
13    cv2.waitKey(1)
14    motor.stop(0.01)

```

Prueba de Integración

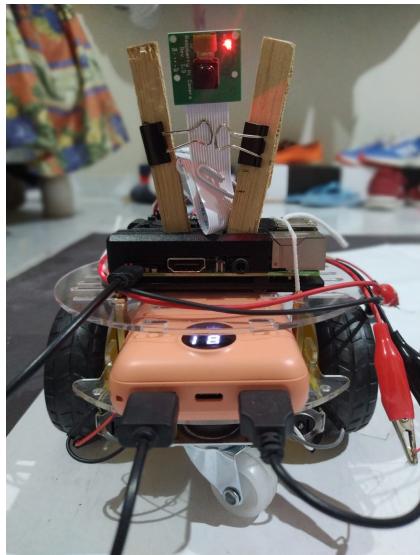
Se utilizaron tres distintos tipos de chasis para las pruebas de integración con el fin de encontrar el mejor desempeño del algoritmo de navegación.

Chasis diferencial circular y rectangular de dos ruedas

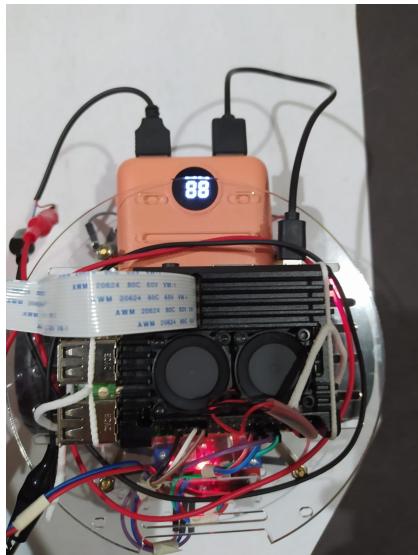
Al utilizar este tipo de chasis, el desempeño del algoritmo de navegación fue deficiente, ya que el robot perdía el control y giraba en su mismo eje durante repetidas ocasiones.

Se realizaron las siguientes modificaciones para comprobar el desempeño:

- **Ajuste de posición de la cámara:** Se consideró construir una base de la cámara para poder percibir de una mejor forma el camino. Sin embargo, la perspectiva que se obtenía necesitaba un ajuste más preciso. Por esta razón se decidió comprar una base para la cámara para obtener una mejor imagen del trayecto
- **Cambio de materiales de entorno:** Anteriormente se hacia uso de cartulinas negras y hojas de papel, pero la tracción de las ruedas no era la mejor, ya que se quedaba varado girando sus ruedas pero no avanzaba el chasis.
- **Variación en distribución de peso:** Se reposición la fuente de poner, la Raspberry Pi y el modulo utilizado para probar si había variación alguna, pero los percances se mantuvieron.



(a) Base casera de cámara Versión 01



(b) Distribución de peso



(c) Base casera de cámara Versión 02

Figura 4.33: Base de la cámara y distribución de peso

En los siguientes videos se puede apreciar la vista del algoritmo de navegación y el problema de tracción mencionado anteriormente.

- **Vídeo de vista del algoritmo de navegación**

- **Vídeo de problemas de tracción**



Figura 4.34: Circuito de pruebas

Al realizar el cambio de chasis y ajustar los valores de giro, el comportamiento del robot mejoró. Para determinar los primeros valores categóricos que permiten realizar giros de una forma más suave, se realizó una variación manual moviendo el robot utilizando el teclado.

En la Figura 4.35 se aprecia que cuando el carro se mueve de forma relativamente recta, los valores de curvatura se mantienen entre 0.1, mientras que si el carro se acerca a una curvatura derecha, el valor incrementaba a 0.5. Esto ha permitido establecer posibles rangos para determinar los comandos de movimiento. Una vez que se obtuvo este acercamiento, el siguiente paso fue realizar variaciones empíricas para encontrar los valores que mejores respondían a las curvaturas.

A continuación se muestran dos vídeos, el primero de ellos es antes de la categorización de los valores, y el segundo una vez aplicados los cambios con los valores categorizados, considerando la Figura 4.35

- **Vídeo del trayecto del robot con valores de curvatura no categorizados**
- **Vídeo del trayecto del robot con valores de curvatura categorizados**

Al realizar la categorización de valores, la distancia que el robot recorrió fue aproximadamente de 5 hojas de papel tamaño carta (55cm). El siguiente paso fue alcanzar una mayor distancia en un espacio completamente recto, para ello se ajustaron los valores de curvatura para que se categorizaran como rectos en el intervalo [-0.085, 0.085]. Los resultados se presentan a continuación.

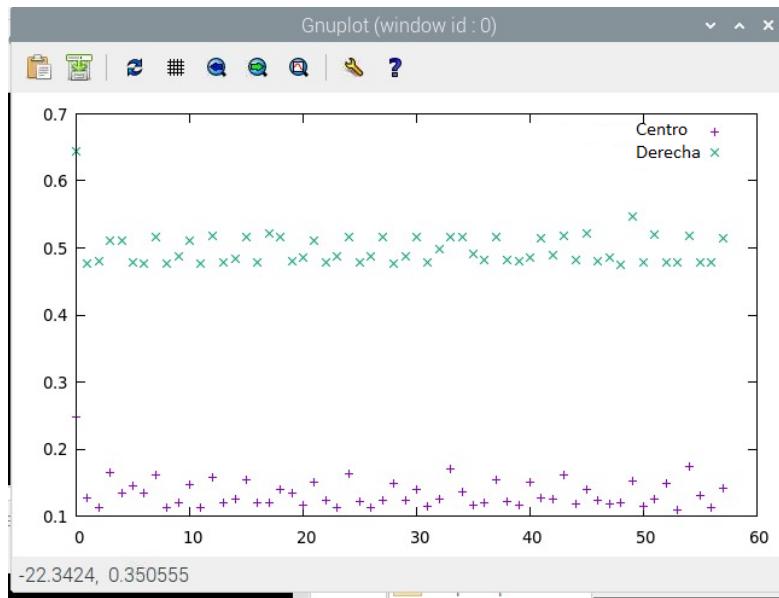


Figura 4.35: Valores de curvatura derecha y recto

- **Vídeo del trayecto del robot con valores de curvatura categorizados en un camino completamente recto**

Los resultados fueron buenos en este punto, ya que la distancia que recorrió, de forma autónoma, el robot fue de 197cm. Para ello, también se modificó la fuente de alimentación, ya que se utilizó un eliminador de 5 Voltios a 3 Amperios, usando una velocidad del 85 %, y alimentando el driver L298N con un porta-pilas para cuatro pilas AA.

Chasis diferencial rectangular de cuatro ruedas

Posterior a ello se decidió utilizar un chasis rectangular de cuatro ruedas para solventar los problemas de tracción e inestabilidad que se presentaban con el chasis anterior. Lo primero fue realizar el armado y comprobación de que el módulo de motores funcionaba correctamente, para ello se realizó una prueba en la que probamos el robot con el programa que permite mover los motores con las teclas de la computadora. El a Figura 4.36 se aprecia el robot con el nuevo chasis.

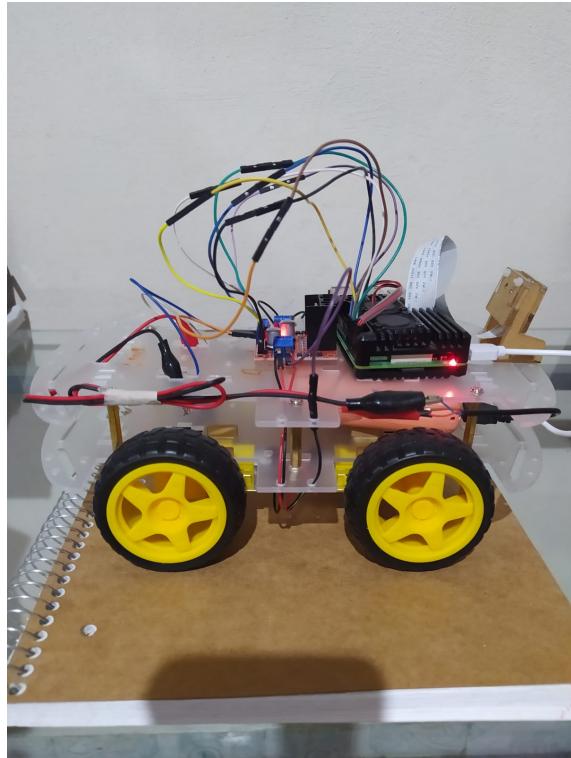


Figura 4.36: Robot con chasis de 4 ruedas

- **Vídeo del robot con el nuevo chasis moviéndose con las teclas de la computadora.**

A continuación, se decidió continuar con una serie de pruebas del robot en un camino recto, para verificar que el comportamiento del módulo de navegación integrado con el de motores, fuera el correcto en el caso más básico, que es cuando se habla de un trayecto sin curvas.

- **Vídeo del robot puesto a prueba en un trayecto recto.**

Siendo exitosa la prueba anterior, se procedió a comprobar el funcionamiento modificando el programa solo mostrando en consola el valor de curvatura obtenido en cada imagen junto con el comando respectivo que el módulo de navegación envía al de motores, esto para permitir un movimiento más fluido del robot, pues cuando se muestra la imagen de monitorio el procesamiento es más tardado.

- **Vídeo del robot puesto a prueba en un trayecto recto con una mejora de fluidez.**

En la siguiente prueba se decidió extender el camino a uno de cerca de 280 cm, para comprobar el correcto funcionamiento del módulo de navegación aún cuando se trate de trayectos más largos.

- **Vídeo del robot puesto a prueba en un trayecto recto más largo.**

Habiendo probado un correcto funcionamiento en los casos anteriores, se procedió a realizar una prueba del robot en un camino con más complejidad, es decir, con una curva. Además, y debido a diversos problemas encontrados con el piso en el que se estaba trabajando, se decidió construir el piso de la pista con mica adherible, por ser un material liso que permite una mejor respuesta por parte de las ruedas al momento de girar, al hacer este cambio se construyó el camino con cartulina negra para un mejor contraste entre el piso y el camino de la pista.

- **Vídeo del robot puesto a prueba en un trayecto con una curva.**

Resultados y Conclusiones

5.1. Resultados: Prueba final del sistema

Habiendo concluido el desarrollo y pruebas de integración de todos los módulos, se procedió a probar el sistema completo; para empezar se tiene todo el módulo de PDI haciendo el procesamiento de la imagen, luego el módulo de navegación procesando la misma y mandando los comandos respectivos al módulo de motores para moverse en el entorno controlado, y finalmente la aplicación web mostrando de manera continua la imagen de monitoreo.

- **Vídeo de prueba final del sistema.**

Los valores obtenidos que permitieron el correcto funcionamiento del sistema fueron los siguientes. Por parte del PDI en las condiciones de luz de la prueba anteriormente presentada, se encontró que los valores de umbralización óptimos en el espacio HSV para el límite inferior son [0, 0, 0] mientras que para el límite superior son [179, 255, 50]. Para el ajuste de perspectiva se utilizaron los siguientes puntos: (0, 80), (480, 80), (0, 150) y (480, 150).

Para la categorización de los valores de las curvas se encontraron que los siguientes fueron los más óptimos:

curveVal	Comando	Sen
(-0.90, -0.28]	Curva izquierda	3
(-0.28, -0.10]	Alinear izquierda	1
(-0.10, 0.10)	Recto	1
[0.10, 0.28)	Alinear a la derecha	1
[0.28, 0.90)	Curva derecha	3

Respecto al movimiento de los motores se encontró que el ciclo de trabajo, es decir, el valor de velocidad más óptimo es de 0.45, en una escala de 0.0 a 1.0. Para el tiempo de movimiento de los motores, por cada vez que se llama a la función “move()”, se encontró que 0.01 segundos es el valor más adecuado.

Se concluye gracias a todas las pruebas realizadas que el chasis de 4 ruedas fue, en efecto, el que mejor respuesta presentó a la hora de realizar los giros y moverse por la pista sin salirse del camino.

Respecto a los materiales para la pista, se encontró que los más favorables fueron: mica adherible para el suelo, por sus característica lisa que permite al robot girar adecuadamente, y cartulina de color negro para el camino, por su color para contrastar con el suelo.

5.2. Conclusiones

- **Módulo de captura de imágenes:** Se ha logrado desarrollar un módulo de captura de imágenes usando la cámara de la Raspberry a través de comunicación CSI y la librería OpenCV. Este módulo nos permite indicar el tamaño de la imagen capturada.

- **Módulo de procesamiento digital de imágenes:** Se ha desarrollado el archivo “utils.py” el cual contiene las funciones para realizar el umbralizado y ajuste de perspectiva de la imagen capturada en el módulo anterior, las cuales son esenciales para la correcta detección del camino que el robot va a recorrer.

Se ha concluido que es mejor utilizar la umbralización en el espacio de color HSV, ya que usar el canal RGB implica el manejo de los 3 que influyen en el color final de una imagen. Es por eso que se usa el canal de matiz del modelo HSV para separar de manera más sencilla el camino de cualquier otro objeto de la imagen.

También se pudo observar que no tiene utilidad o mejora en el desempeño el mostrar una cantidad de camino restante considerable al robot para su procesamiento sino partes más pequeñas que representan el momento inmediato. De esta manera existe una mejor evaluación del algoritmo de navegación en la detección de la curvatura del camino.

Dado que las condiciones de luz y el entorno, así como también el ángulo y posición de la cámara afectan el resultado del algoritmo, se han desarrollado dos programas para el ajuste de los parámetros de los algoritmos. Dicho ajuste se hace de manera manual para un resultado más acorde al deseado por el usuario.

- **Módulo de navegación autónoma:** Se puede afirmar que gran parte de la complejidad de navegación autónoma en escenarios planos está en el correcto manejo de las curvas ya que implican la correcta detección, evaluación y ejecución sobre ellas para que el recorrido sea exitoso. En un primer acercamiento, ha resultado lógico dividir la imagen por el centro y contar la cantidad de pixeles pertenecientes al camino en ambos lados del centro. Si existen más de un lado, significa que existe una curva hacia ese lado. De igual manera, mientras mayor sea la diferencia entre la cantidad de pixeles por lado, la curva es más pronunciada. Sin embargo esto conllevaba una asunción que era muy difícil de cumplir en pruebas reales, esto es, que el centro de la imagen siempre correspondiera al centro del camino y por ende, que el robot siempre se mantuviera, de manera casi perfecta, centrado sobre el camino. Esto definitivamente no se puede cumplir para cualquier escenario, es por eso que se realizó una modificación al algoritmo. La solución fue, en lugar de partir la imagen por la mitad, se encontraría el punto medio del camino, que muchas veces no coincidía con el punto medio de la imagen. De esta manera, sin importar si el robot no está centrado perfectamente, se podrá partir

en dos partes la imagen, las cuales contendrán una mitad del camino, a partir de la base, respectivamente. El resto se mantiene igual, se suman los pixeles y donde exista mayor cantidad, hacia allá es la curvatura.

Al haber detectado hacia dónde va el camino, es decir la curva, se necesita conocer qué tan pronunciada está la curva, para actuar de manera acorde. Para esto se desarrolló una función que permite obtener el centro de la imagen, pero solamente considerando $\frac{1}{4}$ parte. De esta forma se centra solamente en el momento inmediato. Cuando se obtienen ambos puntos (punto centro del camino y punto centro de la imagen en $\frac{1}{4}$ de la región) se pueden restar. El valor absoluto de esa resta, indica qué tan pronunciada es la curva, mientras que el signo resultante, establece la dirección de dicha curva (negativo izquierda, positivo derecha).

Lo último que se ha necesitado es normalizar los valores de curvaturas para que vayan desde -1 a 1 , para su correcto uso en el módulo de los motores. También es requerido visualizar cada uno de los procesos comentados, por lo que existe la comunicación con el módulo web para mostrar cada uno de los pasos mencionados de ser necesario.

- **Módulo de manipulación de los motores del robot:** Para poder controlar los motores del robot se ha usado el módulo L298N cuyo circuito integrado contiene un puente H y la circuitería de potencia necesaria para el control de los dos motores a partir de señales PWM ya que nos facilita la integración con la Raspberry por medio de sus pines de entrada y salida de propósito general (GPIOs). Para realizar pruebas inalámbricas, tanto el módulo como las Raspberry están siendo alimentadas por medio de una *powerbank*. Se desarrolló, además, un módulo llamado "MotorModule.py", donde se realiza la abstracción de un motor por medio de la clase "Motor". La cual contiene las definiciones de los pines necesarios y también una función general de movimiento, que tiene como parámetros, la velocidad, dirección y tiempo de movimiento.
- **Módulo de transmisión de imágenes hacia computador por medio de Wi-Fi y visualización de imágenes en web:** Se usó un servidor HTTP ampliamente utilizado: Apache Web Server. Esto por ser gratis y de código abierto. Este servidor se ejecuta como un programa en segundo plano que escucha peticiones HTTP. Se ha configurado para que se pudiera visualizar en la página web sin problema. Se editó el archivo de configuración de Apache que se encuentra en /etc/apache2/apache2.conf, y se agregó la línea "Header set Access-Control-Allow-Origin "*" " en la línea 174. También se instaló un módulo de Apache llamado "headers" el cual permite personalizar las cabeceras de respuesta HTTP. Esto permitió mostrar los resultados del algoritmo correctamente. El programa cliente se desarrolló en lenguaje Javascript, por ser este el lenguaje predilecto para los navegadores web, este programa refresca las imágenes cada 1.5 segundos para un monitoreo más preciso.
- **Respositorio en Github:** El código presentado en este documento se encuentra alojado en un repositorio público en la plataforma Github, el cual se puede acceder desde [aquí](#).
- **Documentación:** En el presente documento se han documentado cada uno de los módulos desarrollados, así como el uso del hardware y herramientas externas.

5.3. Trabajo a futuro

Como trabajo a futuro proponemos las siguientes mejoras:

- **Chasis del robot:** Como se observó en las pruebas realizadas, los resultados del algoritmo de la navegación tuvieron un resultado más favorable cuando se hizo el cambio a un chasis de 4 ruedas en lugar de uno de 2. Una clara mejora del sistema sería utilizar un chasis de mejor calidad con mayor estabilidad así como unas ruedas de un material con más fricción para responder de manera más precisa a las órdenes del módulo de navegación.
- **Protocolo de reingreso a camino:** Un problema recurrente en las pruebas del algoritmo fue el descarrilamiento del móvil al recorrer un camino. Una mejora no trivial es la implementación de un protocolo de búsqueda y reingreso al camino después de la detección de un descarrilamiento.
- **Ajuste de parámetros del algoritmo de navegación en ejecución:** Para cada escenario de prueba se deben ajustar parámetros como, por ejemplo, la saturación y la perspectiva de la imagen, para que el algoritmo este lo mejor adaptado a las condiciones del escenario para una ejecución más correcta, sin embargo sería de gran ayuda poder modificar estos parámetros al mismo tiempo que se ejecuta el algoritmo para ver de manera inmediata los efectos y repercusiones que tienen los cambios realizados.

Bibliografía

- [1] D. Serpanos and T. Wolf, “Chapter 1 - architecture of network systems overview,” in *Architecture of Network Systems* (D. Serpanos and T. Wolf, eds.), The Morgan Kaufmann Series in Computer Architecture and Design, pp. 1 – 9, Boston: Morgan Kaufmann, 2011.
- [2] M. Rouse, “Embedded systems.” *IoT Agenda*, 2020. [En línea]. Disponible en: <https://cutt.ly/DhqEUvv>. [Accedido: 27-sep-2020].
- [3] “Robótica.” *Lexico*, 2020. [En línea]. Disponible en: <https://www.lexico.com/es/definicion/robotica>. [Accedido: 01-oct-2020].
- [4] A. R. Ismail, R. Desia, M. F. R. Zuhri, and R. M. Daniel, “Implementation of cognitive mapping algorithm for mobile robot navigation system,” in *2014 4th World Congress on Information and Communication Technologies (WICT 2014)*, pp. 280–284, 2014.
- [5] C. Márquez *et al*, “Robótica.” *UPIITA - IPN*, 2014. [En línea]. Disponible en: <http://www.boletin.upiita.ipn.mx/index.php/ciencia/593-cyt-numero-45/1081-robots-moviles-de-ruedas-generalidades>. [Accedido: 01-oct-2020].
- [6] V. T. Mora, “Tipos de locomoción de robots móviles con ruedas.” *UAEM Zumpango*, 2015. [En línea]. Disponible en: <http://ri.uaemex.mx/oca/view/20.500.11799/35197/1/secme-21956.pdf>. [Accedido: 14-dic-2020].
- [7] I. Bambino, *Una introducción a los robots móviles*. June 2008.
- [8] “Sensores conceptos generales.” *Robots Argentina*, 2020. [En línea]. Disponible en: <https://cutt.ly/ihGn2hQ>. [Accedido: 15-dic-2020].
- [9] “Sensor de imagen.” *Wikipedia*, 2014. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Image_sensor. [Accedido: 01-oct-2020].
- [10] P. Vis, “Raspberry pi csi camera interface.” *Peter Vis*, 2019. [En línea]. Disponible en: <https://cutt.ly/whnaY89>. [Accedido: 05-oct-2020].

- [11] “Giroscopio.” *Enciclopedia de Ingeniería*, 2017. [En línea]. Disponible en: <https://cutt.ly/1hna2KT>. [Accedido: 05-oct-2020].
- [12] “Introducción al giroscopio.” *5Hertz*, 2016. [En línea]. Disponible en: <https://cutt.ly/IhnswGq>. [Accedido: 06-oct-2020].
- [13] “Geared motor.” *Premium Transmission*, 2020. [En línea]. Disponible en: <https://premium-transmission.com/blog/geared-motor-types-and-their-importance/>. [Accedido: 06-oct-2020].
- [14] J. Heath, “Pwm.” *Premium Transmission*, 2017. [En línea]. Disponible en: <https://www.analogictips.com/pulse-width-modulation-pwm/>. [Accedido: 10-oct-2020].
- [15] V. Garcia, “Pwm.” *Electrónica Aplicada Práctica*, 2014. [En línea]. Disponible en: <https://www.diarioelectronicohoy.com/blog/el-puente-h-h-bridge>. [Accedido: 16-dic-2020].
- [16] F. Vahid and T. Givargis, *Embedded System Design: A unified Hardware/Software Approach*. Wiley, 1999.
- [17] “Sistemas embebidos.” *OmniSci*, 2018. [En línea]. Disponible en: <https://cutt.ly/jhqEExP>. [Accedido: 10-oct-2020].
- [18] “Network communications under unix system services.” *IBM Knowledge Center*, 2017. [En línea]. Disponible en: <https://cutt.ly/ThqRvcb>. [Accedido: 12-oct-2020].
- [19] “Estándares inálambricos.” *Enciclopedia Cubana*, 2019. [En línea]. Disponible en: <https://cutt.ly/hhtwrVv>. [Accedido: 12-oct-2020].
- [20] “Client server model.” *Wikipedia*, 2020. [En línea]. Disponible en: <https://cutt.ly/RhrlSP4>. [Accedido: 12-oct-2020].
- [21] “Web application.” *Indeed*, 2020. [En línea]. Disponible en: <https://www.indeed.com/career-advice/career-development/what-is-web-application>. [Accedido: 15-oct-2020].
- [22] “An overview of http.” *MDN Web Docs*, 2019. [En línea]. Disponible en: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>. [Accedido: 15-oct-2020].
- [23] E. Davies, *Computer and Machine Vision: Theory, Algorithms, Practicalities*. Elsevier, fourth ed., 2012.
- [24] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Pearson, fourth ed., 2017.
- [25] N. A. Dobernack, “Implementación de un sistema de detección de señales de tráfico mediante visión artificial.” Master’s thesis, Universidad de Sevilla - Departamento de Ingeniería Electrónica, Abril 2013.
- [26] O. M. Manuel, *Procesamiento Digital de Imágenes*. Jhon Wiley and Sons Ltd, Enero 2013.

- [27] M. Petrou and P. Bosdogianni, *Image Processing: The fundamentals*. Jhon Wiley and Sons Ltd, 1999.
- [28] C. y Franco Ana, *Programa de segmentación de regiones en imágenes médicas en MATLAB*. PhD thesis, Universidad de las Américas Puebla, Enero 2004.
- [29] M. Echenique, “Consideraciones generales de la teoría cromática.” *Calameo*, 2011. [En línea]. Disponible en: <https://es.calameo.com/read/00204736273f7a25a93af>. [Accedido: 02-nov-2020].
- [30] W. Burger and M. Burge, *Digital Image Processing: An Algorithmic Introduction Using Java*. Texts in Computer Science, Springer London, 2016.
- [31] J. Perez, J. Penagos, J. Useche, and D. Bonaventura, eds., *Formatos de compresión de datos*, (<http://www.unitec.edu.ve/materiasenlinea/upload/T1178-11-1.pdf>), 2009.
- [32] S.-H. Y. Bustamante, “Algoritmos de procesamiento de imagen aplicados a la detección de figuras geométricas y sus propiedades espaciales.,” Master’s thesis, Pontificia Universidad Católica de Valparaíso - Facultad de Ingeniería, Marzo 2014.
- [33] L. M. Álvarez Romero and J. E. F. Montenegro, eds., *Implementación de algoritmo de navegación utilizando la plataforma del IRobot Create y módulos de comunicación inalámbrica Xbee*, (<http://repositorio.espe.edu.ec/xmlui/handle/21000/2643>), Departamento de eléctrica y electrónica, 2011.
- [34] P. Yuri Shinzato and D. Fernando Wolf, “Features image analysis for road following algorithm using neural networks,” in *IFAC Proceedings Volumes Volume 43*, pp. 306–311, 2010.
- [35] D. S. O. Correa, D. F. Sciotti, M. G. Prado, D. O. Sales, D. F. Wolf, and F. S. Osorio, “Mobile robots navigation in indoor environments using kinect sensor,” in *2012 Second Brazilian Conference on Critical Embedded Systems*, pp. 36–41, 2012.
- [36] A. Newman, G. Yang, B. Wang, D. Arnold, and J. Saniie, “Embedded mobile ros platform for slam application with rgb-d cameras,” in *2020 IEEE International Conference on Electro Information Technology (EIT)*, pp. 449–453, 2020.
- [37] C. Consulting, ed., *Type of testing in the V-Model*, (<https://www.coleyconsulting.co.uk/testtype.htm>), Consultado: 2020-11-29.
- [38] “Raspberry pi 3 b+.” *Raspberry Pi Foundation*, 2020. [En línea]. Disponible en: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/?resellerType=home>. [Accedido: 23-nov-2020].
- [39] “Beaglebone.” *BeagleBoard*, 2020. [En línea]. Disponible en: <https://beagleboard.org/bone>. [Accedido: 23-nov-2020].
- [40] “Jetson nano developer kit.” *Nvidia*, 2020. [En línea]. Disponible en: <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>. [Accedido: 23-nov-2020].

- [41] "Raspberry pi 3 b." *Raspberry Pi Foundation*, 2020. [En línea]. Disponible en: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/?resellerType=home>. [Accedido: 01-dic-2020].
- [42] "Raspberry pi 2 b." *Raspberry Pi Foundation*, 2020. [En línea]. Disponible en: <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/?resellerType=industry>. [Accedido: 01-dic-2020].
- [43] "Raspberry pi camera comparison." *SEEED Studio*, 2020. [En línea]. Disponible en: <https://www.seeedstudio.com/blog/2020/05/14/raspberry-pi-camera-comparison-of-high-quality-camera-with-camera-module-v2/>. [Accedido: 01-dic-2020].
- [44] "Raspberry pi camera module v2." *SEEED Studio*, 2020. [En línea]. Disponible en: <https://www.seeedstudio.com/Raspberry-Pi-Camera-Module-V2.html>. [Accedido: 01-dic-2020].
- [45] "Salario promedio mensual del ingeniero en sistemas en méxico." *Glassdoor*, 2020. [En línea]. Disponible en: https://www.glassdoor.com.mx/Sueldos/ingeniero-en-sistemas-computacionales-sueldo-SRCH_KO0,37.htm. [Accedido: 01-dic-2020].
- [46] I. Sommerville, *Ingeniería de Sofware*. Addison Wesley, 2011.
- [47] "Camera module documentation." *Raspberry Pi Foundation*, 2020. [En línea]. Disponible en: <https://cutt.ly/4zc811o>. [Accedido: 23-nov-2020].
- [48] "Setting up an apache web server on a raspberry pi." *Raspberry Pi Foundation*, 2020. [En línea]. Disponible en: <https://bit.ly/2OEV2w2>. [Accedido: 10-dic-2020].