PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

Escuela de Ingeniería Eléctrica
PONTIFICIA UNIVERSIDAD CATOLICA DE VALPARAISO

Enrique Andrés Torres Abarca

Sistema de visión para la navegación de robots móviles avanzados

Informe Proyecto de Título de Ingeniero Civil Electrónico

Escuela de Ingeniería Eléctrica
Facultad de Ingeniería

Valparaíso, 28 de diciembre de 2018

Sistema de visión para la navegación de robots móviles avanzados

Enrique Andrés Torres Abarca

Informe Final para optar al título de Ingeniero Civil Electrónico,

aprobada por la comisión de la

Escuela de Ingeniería Eléctrica de la

Facultad de Ingeniería de la

Pontificia Universidad Católica de Valparaíso

conformada por

Sr. Gonzalo Alberto Farías Castro

Profesor Guía

Sr. Héctor Renato Vargas Oyarzún

Segundo Revisor

Sr. Sebastián Fingerhuth Massmann

Secretario Académico

Valparaíso, 28 de diciembre de 2018

A mi familia y amigos.

Agradecimientos

Principalmente quiero agradecer a mi madre Paola, por darme el apoyo incondicional en todas las aventuras que he tenido, dándome fuerzas suficientes para poder seguir adelante y no dar marcha atrás. También con sus grandes consejos, tomando las mejores decisiones en el estudio y en la vida para ser cada vez una persona mejor.

A mis hermanos Carlos y Néstor por darme siempre alegrías y preocupaciones, proporcionándome la oportunidad de poder ser una parte importante en sus vidas, también apoyándome y estando siempre conmigo, lo cual no muchos hermanos son así, sintiéndome orgulloso de ser su hermano.

Quiero agradecer a Victoria por estar en todos los momentos importantes de mi vida, desde mí enseñanza media, donde di mis primeros pasos en el área de electrónica hasta ahora. Toda esa paciencia que has tenido para una mente cuadrada como la mía y ese cariño infinito que lograste mantener para hacerme una persona de bien.

A su vez, desde que empecé mis estudios universitarios, conocí a Roberto, Osvaldo, Miguel y Sebastián, que fueron una parte muy importante para el desarrollo de mi carrera. Con ellos compartimos muchos momentos, entre ellos, las juntas de estudio, trabajos de proyectos, salidas y siempre animarme con risas y consejos para poder seguir surgiendo.

Valparaíso, 28 de diciembre de 2018

Enrique Torres

Resumen

La robótica móvil ha evolucionado a escala exponencial estos últimos años, debido a que las tecnologías y las necesidades de esta época ha permitido su gran avance, por lo que este proyecto propone un nuevo sistema para la navegación en robótica móvil utilizando visión artificial. En primer lugar, se presentan los antecedentes generales, que involucran las problemáticas existentes en la robótica móvil y como estas han sido abordadas. Por lo tanto, se plasmó un estado del arte de las contribuciones que se han realizado este último tiempo, para luego formular una propuesta al problema.

Se planteó un nuevo sistema de visión artificial para el robot Khepera IV, el cual permitirá realizar pruebas y corroborar los resultados de manera real. Por lo tanto, se efectuó una solución a la propuesta con herramientas que se están utilizando hoy en día en la visión por computadora, dando así un marco teórico completo.

Luego del marco teórico, para la realización de experimentaciones, fue necesario realizar un proceso de preparación y desarrollo del sistema, el cual involucra en donde se desempeñará el entorno de trabajo, que en este caso será de manera simulada y real. Gracias al simulador de robots móviles V-REP y la plataforma de experimentación ubicada en la Escuela de Ingeniería Eléctrica, es posible la realización de experimentos donde se obtengan resultados concretos. Además, para la detección de objetos fue necesario utilizar la impresora 3D del laboratorio de robótica, que permitió la construcción de estos objetos (señales de tránsito) para que la cámara del robot los detectara de buena manera. Para clasificar los objetos se utilizó el clasificador en cascada de tipo HAAR, el cual la librería de visión artificial OpenCV realiza tanto la creación de muestras positivas y el entrenamiento mismo.

Finalizando, en la etapa de experimentación, se presentaron las pruebas realizadas que afirman la efectividad del algoritmo. En primera instancia se realizó la detección de objetos simples tanto en forma simulada como real, luego se realizaron las pruebas de los clasificadores en donde se entrenaron 10 objetos distintos, para luego realizar escenarios donde el robot debiera llegar a un punto de destino. Esto es producido ya que el clasificador determina el significado de la señal, que conlleva a realizar un movimiento inteligente y cumplir el objetivo planteado del proyecto.

Palabras claves: Visión artificial, Khepera IV, Espacio de colores, Clasificadores en cascada, Señales de tránsito.

Abstract

Mobile robotics has evolved exponentially in recent years, because the technologies and needs of this time has allowed its breakthrough, so this project proposes a new system for navigation in mobile robotics using artificial vision. First, the general background is presented, which involves the existing problems in mobile robotics and how they have been addressed. Therefore, a state of the art of the contributions that have been made this last time was expressed, and then formulate a proposal to the problem.

A new artificial vision system was proposed for the Khepera IV robot, which will allow testing and corroborating the results in a real way. Therefore, a solution to the proposal was made with tools that are being used today in computer vision, thus giving a complete theoretical framework.

After the theoretical framework, for the realization of experiments, it was necessary to carry out a process of preparation and development of the system, which involves where the work environment will be performed, which in this case will be simulated and real. Thanks to the mobile robot simulator V-REP and the experimentation platform located in the School of Electrical Engineering, it is possible to carry out experiments where concrete results are obtained. In addition, for the detection of objects it was necessary to use the 3D printer of the robotic laboratory that allowed the construction of these objects (traffic signals) so that the robot's camera could detect them in a good way. To classify the objects, the HAAR type cascade classifier was used, which the OpenCV artificial vision library performs both the creation of positive samples and the training itself.

Finalizing, in the experimentation stage, the tests that affirm the effectiveness of the algorithm were presented. In the first instance, the detection of simple objects was carried out, both simulated and real, then the tests of the classifiers where 10 different objects were trained were carried out, in order to then perform scenarios where the robot should reach a destination point. This is produced because the classifier determines the meaning of the signal, which leads to an intelligent movement and fulfill the objective set in the project.

Key words: Artificial vision, Khepera IV, Color space, Cascade classifiers, Traffic signs.

Índice general

Introducción

En el mundo que se está viviendo hoy en día, ha estado en constantes cambios debido a que la tecnología avanza a escala exponencial y está afectando fuertemente en el área de la robótica. Cada vez son más las áreas donde las máquinas apoyan a los humanos en realizar procesos autónomos, en donde se necesite una gran eficacia y eficiencia. Este efecto se puede observar en el área medicinal, la cual, al momento de realizar una cirugía, existen brazos robóticos que permiten realizar un excelente compromiso, por ejemplo, otorgando una ayuda a los doctores a realizar procesos curativos, dando así una mejor calidad de vida hacia las personas [1]. En el área industrial, se ofrece una labor de mayor productividad y eficiencia, debido a que las maquinas al realizar procesos sistemáticos, ayudan a disminuir los tiempos de proceso, donde es posible supervisarla y controlarla; este es el caso de una empresa de ensamblado de automóviles con sistemas automáticos [2]. A través de la robótica educativa, esta se puede dividir en dos aspectos, investigación y enseñanza, donde la primera permite el análisis y el estudio de nuevos métodos y alternativas para solucionar distintos tipos de problemas que hayan surgido. Estos métodos pueden ser experimentados para así poder calcular errores en relación a pruebas reales y pruebas simuladas. En el ámbito educacional, permite apoyar procesos pedagógicos (por ejemplo, asignaturas de física y matemática), además permite motivar a que los estudiantes se integren desde muy jóvenes en el ámbito de la ciencia y la tecnología [3]. Por último, está el área de la navegación, donde los robots se convierten en máquinas móviles que permiten realizar operaciones de transporte, vigilancia, limpieza y descubrimiento de lugares desconocidos, donde el costo de llevar a una persona seria muy alto, tanto monetario como peligroso, por ejemplo, llevar una persona a otro planeta [4].

Para que la robótica móvil pueda interactuar con el entono, necesita varios sistemas que permitan su funcionamiento, entre ellas está el sistema mecánico, alimentación, control y sensorial. Este último es muy importante debido a que es necesario medir a gran y corta escala los elementos que nos rodean. Además de los sensores tradicionales, existe uno muy particular, llamada visión por computadora, ya que permite medir el entorno con la utilización de una cámara digital, el cual consiste en adquirir, procesar, analizar y comprender las imágenes que se rodean con el fin de producir información de tipo numérica para ser analizada por un computador. La herramienta visión artificial modela la visión que tiene una persona, lo que significa que con ella se permite realizar el mismo efecto de percibir y actuar bajo ciertas circunstancias, entre ellas está el

reconocimiento de objetos, detección de eventos, reconstrucción de escenarios, entre otros, dando así un mejoramiento al sistema sensorial [5].

Dentro del mundo de la visión artificial, una imagen es representada por una función bidimensional el cual posee coordenadas x e y, donde cada pixel representa el brillo o intensidad de la luz de un punto cualquiera. Existen diversas aplicaciones que permiten utilizar estos pixeles para realizar una operación inteligente, como es el caso de procesamiento en imágenes de rayos X, donde se elevan los niveles de intensidad para una mejor interpretación y por ende una correcta decisión, o como es el caso de los Geógrafos que estudian los patrones de polución a partir de imágenes aéreas o satelitales, y la restauración de imágenes borrosas para las áreas de la arqueología. Además, en el área de ingeniería, permite la clasificación y detección de objetos, produciendo que la maquina actué de manera autónoma sin la necesidad de presencia humana, lo que equivale a un sistema inteligente que permite la disminución de costos y la optimización del tiempo.

Dado una imagen de cualquier tipo, existen distintas maneras de procesarla, el cual pueden ser la detección de elementos de un solo color, por ejemplo, flechas, triángulos, etc. aunque en un entorno real es más complicado, debido a que una imagen posee distinta iluminación, ruido, producirá a que esta sea conformada por múltiples tonos lo cual es necesario una especie de aprendizaje. Por ejemplo, en la vida cotidiana para desplazarse en una vía pública, uno debe seguir ciertas normas que involucran acciones a seguir, con el fin de tener un orden para controlar la seguridad de las personas, estos indicadores son representados por señales de tránsito. Por lo general estas señales se ubican a un costado de la pista con el objetivo de que el conductor las visualice y tome una decisión al respecto. En la actualidad, se han presentado distintos casos de accidentes donde en general las personas no respetan estas normas, la cual se produce por los siguientes casos: conductor distraído, problemas en el auto o de manera a propósito. Para que estos accidentes sean los menos posibles, es necesario de la presencia de un sistema inteligente que permita tomar la acción de comunicar al conductor o actuar propiamente antes la presencia de una señal. Independientemente de la decisión, es necesario que el sistema tenga la capacidad de reconocer indicadores, un método que se sigue utilizando hoy en día, son los clasificadores en cascada, que permiten obtener características de una imagen que hasta una persona normal no es capaz de detectar, con el fin de obtener una información útil y actuar bajo esas circunstancias.

Los clasificadores hoy en día están muy familiarizados con la inteligencia artificial, el cual necesitan una serie de elementos de prueba para poder diseñar uno, como una imagen posee múltiples características, es necesario utilizar el método de la imagen integral para obtener la información de interés lo más rápida posible. Esta técnica permite extraer de manera rápida características a diferentes escalas ya que no se trabajan directamente con valores de intensidad sino con una imagen acumulativa, de tal operación, se pueden caracterizar con un tiempo menor que al utilizar la imagen real, con la sola utilización de operaciones básicas para su construcción [6]. Al tener un clasificador de imágenes, producirá un mayor costo computacional para el procesamiento, el cual un sistema podría no tener las capacidades físicas para realizar el procesamiento. Si se desea clasificar una imagen donde se necesite ejecutar una acción en tiempo real, es necesario llevar esta imagen a un sistema que si se pueda procesar la información.

El desarrollo de la computación y su unión con las redes de telecomunicaciones, han dado grandes posibilidades a las comunicaciones inalámbricas, lo que permiten dar distintas oportunidades para que dos o más computadoras sean capaces de interactuar entre ellas. Una de esas redes son las LAN, el cual permiten conectar distintos computadores con un cierto protocolo para que puedan realizar tareas en conjunto, como es el caso de que una computador no sea capaz de resolver un problema debido a sus capacidades, lo que resulta necesario pedir ayuda a otro que si las tenga, con el fin de mejorar la eficacia del proceso [7]. Por lo tanto, es indispensable conocer el sistema que uno trata, ya que, si no se conoce bien, esta tendrá problemas desde la comunicación, hasta el proceso de datos.

En aspectos técnicos, antes de llevar el sistema a entornos reales, es necesario que este pase por un proceso de experimentación, por lo que la Escuela de Ingeniería Eléctrica posee una plataforma a escala de laboratorio que permite la realización de pruebas en robots móviles. Además, la escuela cuenta con robots de última generación llamados Khepera IV, los cuales son robots no holonómicos con múltiples características, dando así una gran posibilidad de realizar pruebas de cualquier tipo, desde controles de posición, hasta clasificación de objetos utilizando sus sensores.

1 Antecedentes generales

Para todo sistema que por alguna razón este afectado por un problema en cualquier tipo de área, es necesario tener muy claro de cómo este es producido, que variables interactúan con él y que elementos lo componen, con el fin de detallar el problema y realizar una correcta solución. Es por ello, que un estudio detallado siempre es recomendable, ya que se establece una compilación de resultados sobre otras investigaciones que hayan solucionado de otra manera el problema planteado. En este capítulo se presentan distintas problemáticas que han surgido a través de los años y como estas han sido solucionadas por terceros en el ámbito de la robótica y visión artificial.

1.1 Problemas en la robótica móvil

En lo general, las personas no le dan importancia a su sistema interno, por ejemplo, si se toma una taza de té, automáticamente se agarra y se bebe su contenido, pero realmente este proceso posee una serie de mecanismos que actúan en conjunto, como es el mover el brazo hacia la taza, sentir el objeto con la mano, interpretar el tipo de objeto y la coordinación para ejecutar esos pasos de manera adecuada. Para que un robot pueda percibir e imitar el comportamiento mencionado, se debe tener una cierta cantidad de mecanismos, como son los sensores, el cual se posicionen de manera adecuada, para su correcto funcionamiento. Otro aspecto es el tema de la localización, donde sin duda, la posición y orientación es uno de los aspectos más importantes en la navegación autónoma de robots móviles, donde permite planificar caminos, generar trayectorias y monitorizar la ejecución de movimientos. Si no existe un sistema estable de localización, la funcionalidad de este puede ser afectada severamente provocando un riesgo para las personas y aumento de costos.

Este es el caso del robot Curiosity (Figura 1-1), el cual es un robot que realiza investigación en la superficie del planeta Marte, por lo que en ella pueden existir rocas llevando a la probabilidad de que el robot pueda ser volcado, produciendo de que los sensores realicen una mala lectura. De igual manera si el robot se resbala por la arena, provocará que su posición no sea la que se esperaba, llevando a que las ecuaciones de control de posición no sean confiables. Hoy en día se están ocupando herramientas tales como el uso del GPS y cámaras digitales, por lo que solamente utilizando procesamiento de señales, es posible la determinación de la posición del robot.

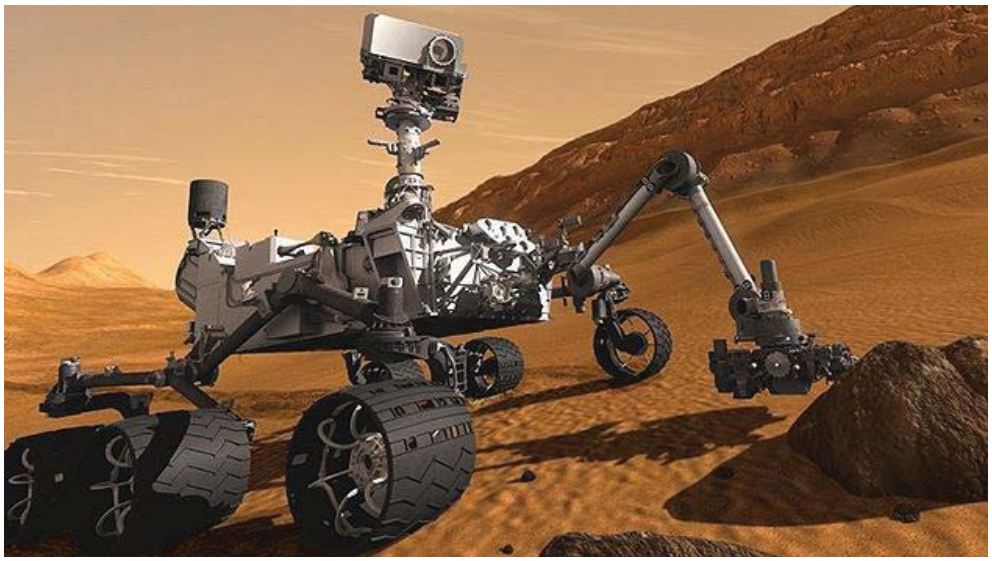

Figura 1-1: Robot Curiosity (fuente: https://www.nasa.gov).

Para la robótica móvil, es muy importante la navegación controlada, si un robot debe moverse hacia un lado debe preguntarse: ¿Cómo me dirijo hacia ese lugar?, Esto conlleva a que el robot debe saber cómo planificarse, ya que, al estar realizando navegación, no sabe exactamente como realizar ese movimiento. Una solución a este sistema es la integración de algoritmos de navegación, en donde se realice el cálculo de movimiento llevando al robot al punto objetivo de una manera óptima, donde también se ve reflejada la inteligencia y autonomía, el cual en un robot es un tema trascendental en el momento que este tome una decisión. Para el aprendizaje automático, existen dos maneras, una es por la ayuda de humanos y la otra por cuenta propia, esta última no es difícil si se tiene un conocimiento inicial, pero no tan sencillo si se debe adquirir conocimiento procedimental. Por ello existen desarrollos dirigidos a facilitar el aprendizaje procedimental, en el que constan de varios agentes externos que pueden ayudar, esto incluye por interacción, imitación y demostración. Esto último se aplica en sistema de control basado en inteligencia artificial, como redes neuronales, lógica difusa o algoritmos genéticos [8].

En un entorno de simulación y/o real, los sensores que posea un robot, ya sean sensores tradicionales (infrarrojos y ultrasónicos), estas solo entregarían datos de la distancia de separación entre el robot y el objetivo, pero no entregaría información de que objeto es, por lo que existiría un grado de incertidumbre al momento de realizar la detección. Al igual que los humanos ocupan sus ojos para comprender su entorno, la visión artificial permite modelar el mismo efecto llevándolo a percibir y entender una imagen o secuencia de imágenes y actuar según convenga en una determinada situación [9].

1.2 Estado del arte

En la robótica móvil, es necesario tener un sistema robusto, el cual permita sobrepasar las dificultades que presentan al experimentar en un estado real, ya sea por la adquisición de información, fluctuaciones en el lugar, condiciones lumínicas, entre otros; por lo que los sensores tradicionales no son lo suficientemente estables para tener el control en todo el proceso. Un

sensor que ayuda a estos detectores es una cámara digital, el cual brinda muchas características, por lo que aumenta la probabilidad de detección de obstáculos, lo que equivale un mayor cuidado al robot. En la literatura se han realizado experimentos con visión artificial en robótica móvil, en [10] se desarrolla un sistema de control la cual realiza la detección de objetos con visión estereoscópica, por lo que envía la imagen captada por medio de un router WiFi para que sea recibida por una aplicación en un computador externo. Este sistema utiliza la librería AForget.NET el cual es un fragmento en C# de código abierto diseñada para desarrolladores en el campo de la visión por computadora y la inteligencia artificial, por lo cual permite operaciones tales como procesamiento de imágenes, redes neuronales y algoritmos genéticos.

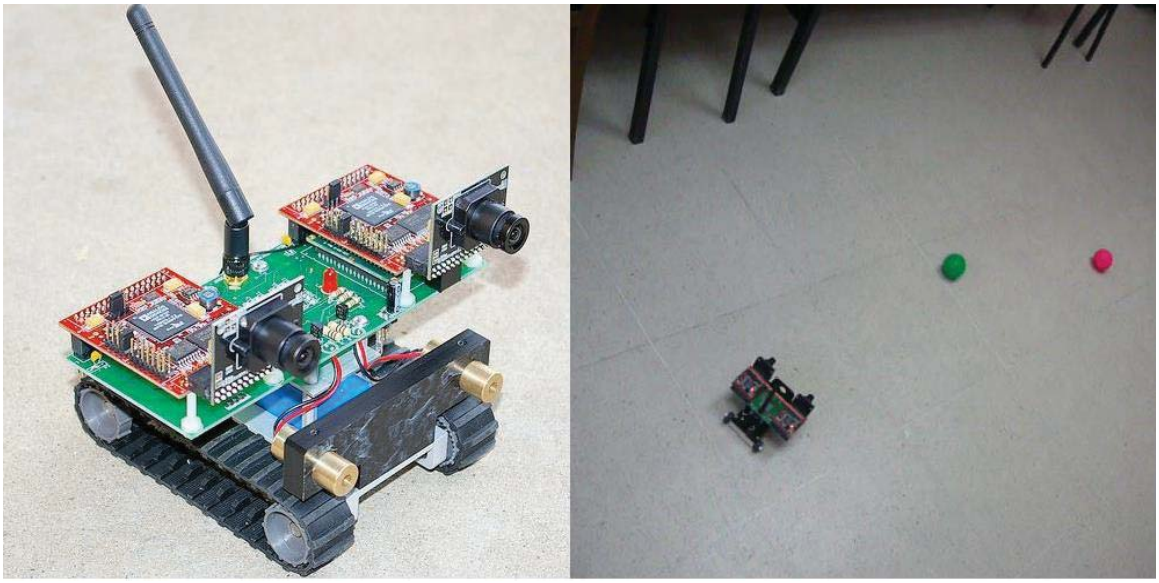

Figura 1-2: Robot móvil Surveyor SRV-1 con módulo para la visión estereoscópica [10].

El robot móvil Surveyor (Figura 1-2) debe como objetivo crear una ruta con su cámara el cual permita la detección de objetos y evitación de obstáculos. En [11] se propone llevar la robótica al entorno de agricultura, donde se realiza la automatización de la pulverización de productos fitosanitarios en el interior de invernaderos.

Figura 1-3: Robot Fitorrobot utilizado en invernaderos [11].

En la Figura 1-3, se observa el robot Fitorrobot, este tiene una cámara que permite observar el entorno y un sistema que realiza todo el procesamiento. Al ser recibida la imagen, esta realiza una

umbralización con un enmascaramiento del color del camino, lo cual permite sacar un promedio del centro de ella y crear una ruta. Por lo tanto, se propone un algoritmo que ayude al robot a recorrer de manera eficaz los pasillos del mismo, teniendo en cuenta que deberá recorrerlos lo más centrado posible para tener una uniformidad del producto fitosanitario en las plantas. En [12] presenta una estrategia para coordinar múltiples robots de forma cooperativa, el cual se utiliza la visión artificial para establecer la posición y orientación relativa de los robots para mantener una formación especificada. Dado que un robot líder se mueve por una trayectoria desconocida para el robot esclavo, este debe seguir como propósito al robot líder a la misma velocidad, el cual también es desconocida, con la sola utilización de la cámara. En la Figura 1-4 muestra al robot líder y esclavo (Robot Poineer 2DX), en la parte trasera del robot principal existe un código que permite al robot secundario determinar ciertas condiciones para poder guiarse y con la técnica de la umbralización permite la detección de los códigos, el cual con ello puede determinar factores como el tamaño y la posición.

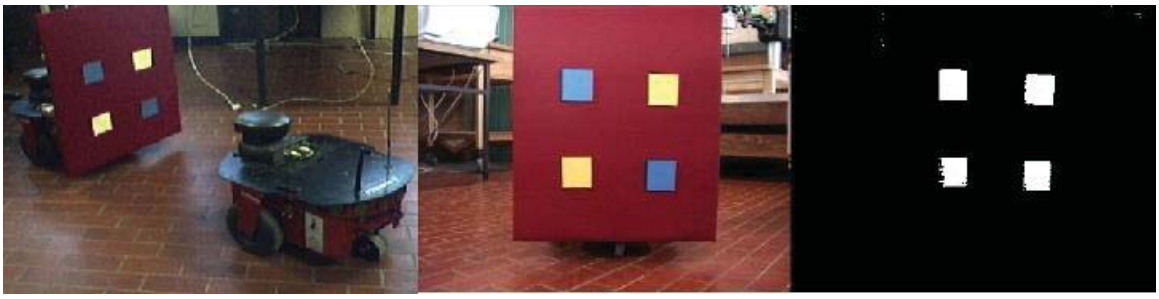

Figura 1-4: Robot Poineer 2DX [12].

En [13] se presenta un sistema de tiempo real para la detección y clasificación de objetos dinámicos utilizando un buscador de rango laser 2D y una cámara móvil (Figura 1-5), se desarrolla un sistema de clasificación utilizando las características HOG y adaBoost como proceso de aprendizaje, es decir, clasificadores en cascada para la detección de personas.

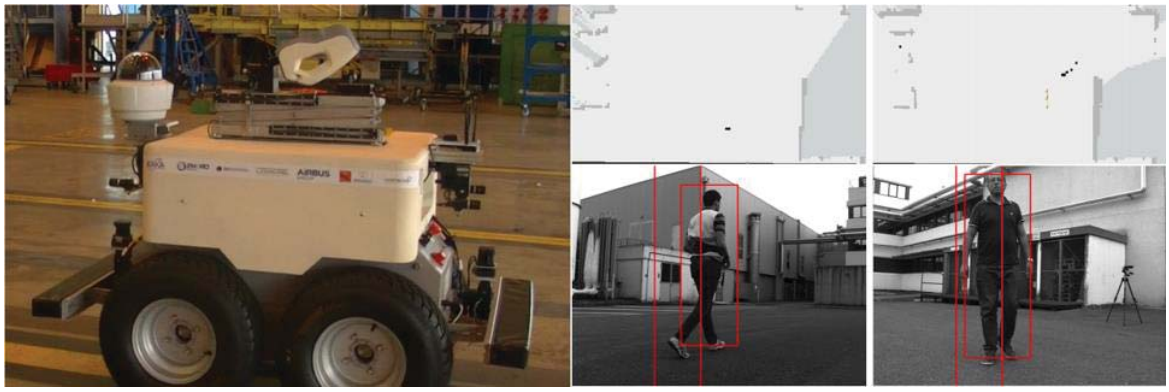

Figura 1-5: Robot Air-Cobot y detección de personas [13].

Utilizando otro tipo de características, en [14] se presenta un sistema de control de alineación basado en la visión propuesta para un vehículo autónomo de carretilla elevada de carga pesada. Esta máquina se desarrolla en recoger y colocar carretes, el cual se dividió en dos procesos:

inspección de la bobina y alineación. La inspección de la bobina utiliza una cámara de visión para realizar el marcado del centroide de la bobina utilizando cascada de clasificadores del tipo HAAR, y la alineación toma ese centroide para centrarse en el carrete gracias a sus motores.

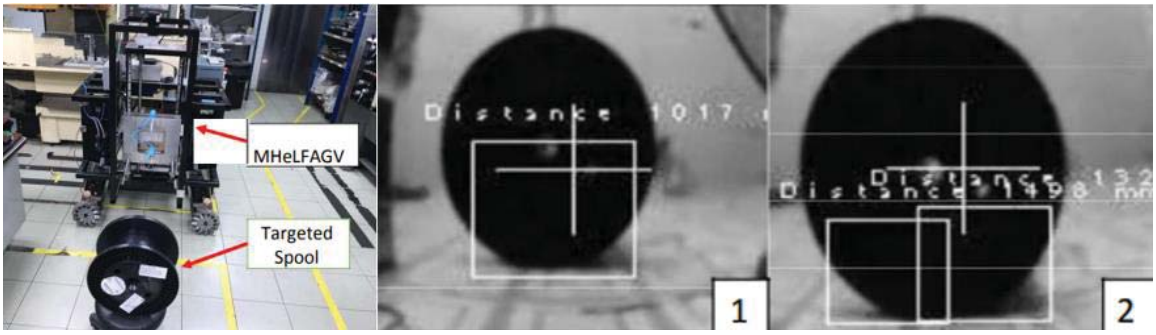

Figura 1-6: Implementación del control basado en la visión propuesto en el sistema MHeLFAGV [14].

Las imágenes son capturadas en tiempo real y se analizan para marcar el carrete seleccionado en un tiempo de 1000 [ms], el cual los datos de visión envían continuamente la coordinación entre la cámara y el carrete seleccionado (Figura 1-6). Por otro lado, para la detección de objetos más robustos (como son las señales de tránsito), en [15] se presenta un sistema de reconocimiento de señales, el cual posee tres etapas: en primer caso es la segmentación, el cual extrae las regiones rojas con un umbral adaptable, luego en la detección se utiliza una máquina de vector de soporte lineal y eficiente con características de histograma de gradientes orientados, y por ultimo una clasificación de tipo árbol K-d tree y Random Forest el cual identifica el contenido de las señales de tráfico encontradas.

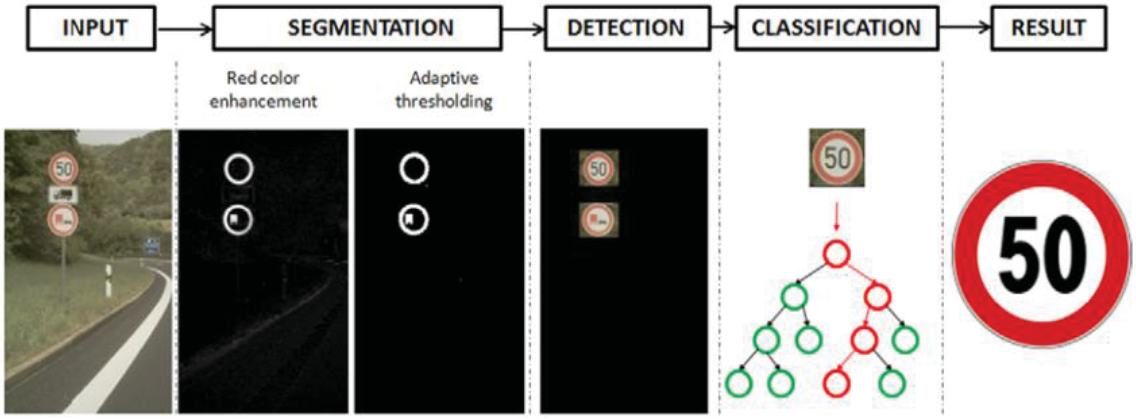

Figura 1-7: Reconocimiento de señales de tránsito utilizando detección mediante HOG y SVM [15].

En la Figura 1-7 se muestra el procedimiento de la detección de la señal, desde la adquisición de la imagen, hasta la clasificación de la señal detectada. En [16] se expone otro método de la detección de señales de tránsito, donde en primera instancia realiza un umbral de color para segmentar el análisis de la imagen y forma para detectar los signos; y en segundo lugar una clasificación utilizando una red neuronal, aunque esa solo es válida para la detección de objetos rojos. A su vez, en [17] se propone un modelo de detección y reconocimiento para señales de

advertencia de carreteras con un sistema de notificación de voz para vehículos autónomos en tiempo real, el cual, la clasificación se realizó utilizando cascadas de clasificadores de tipo HAAR. Por último, en [18] se utiliza un algoritmo genético para el paso de detección, lo que permite una localización de invariancia a los cambios de posición, escala, rotación y la presencia de otros objetos del mismo color. En la Figura 1-8 se muestra la eficiencia del algoritmo, donde la imagen (a), (b), (c) y (d) se realiza la detección de varias señales de tránsito.

Figura 1-8: Detección de señales de un contorno del mismo color [18].

A su vez, los robots móviles autónomos necesitan rastrear caminos para poder orientarse en su entorno, es por ello que en [19] se detalla un sistema económico que consiste en la detección de carreteras basados en color y un detector de línea de textura, el cual están diseñados por separado para luego ser fusionados para rastrear el objetivo. El área media superior del resultado de detección de carretera se considera el objetivo de seguimiento del camino y se entrega al sistema de seguimiento de la carretera para el robot. Por otro lado, la detección de señales también está reflejada en la robótica móvil. En [20] trata de un sistema de visión aplicado en un robot móvil generado con energía eléctrica el cual puede reconocer los signos de giro hacia la derecha, izquierda y hacia arriba usando los algoritmos de Harris corner, donde en la Figura 1-9 se realiza un movimiento inteligente con visión artificial en la robótica móvil.

Figura 1-9: Robot móvil utilizado para la detección de señales de giro [20].

1.3 Propuesta al problema

Realizando un estudio exhaustivo a través del estado del arte, se deduce que no existe documentación en la que se realicen escenarios con pruebas reales en un entorno de laboratorio para el procesamiento de imágenes de la detección de objetos robustos en un robot móvil. Por lo tanto, la propuesta planteada es algo novedoso, ya que permitirá dar otra solución al comportamiento inteligente de un vehículo autónomo, cumpliendo los siguientes objetivos:

1.3.1 Objetivos generales

- Diseñar e implementar algoritmos de navegación en robots móviles mediante visión artificial.

1.3.2 Objetivos específicos

- Estudiar los problemas y soluciones de control de navegación en robótica móvil.
- Modelar y simular un escenario de navegación de un robot móvil.
- Diseñar algoritmos de navegación mediante visión artificial en un entorno simulado.
- Implementar un sistema de control de navegación de un robot móvil avanzado en un escenario real a escala de laboratorio.

En trabajos anteriores, relacionados con la robótica móvil en el laboratorio de robótica de la Escuela de Ingeniería Eléctrica [21], se realizó todo un desafío a la incorporación de robots de última generación en un escenario donde se pueda realizar experimentaciones.

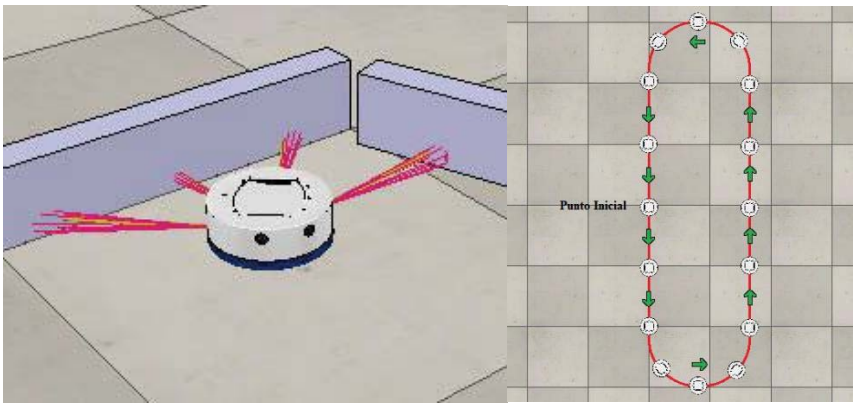

Figura 1-10: Primeras pruebas simuladas utilizando robótica móvil en el robot Khepera IV [21].

Las primeras pruebas fueron la realización de implementos de algoritmos tales como: control de posición, evitación de obstáculos, multiagentes, entre otros, tanto de manera virtual y real (Figura 1-10). Ahora se aspira a realizar un diseño y aplicación de un sistema de navegación para robots móviles no holonómicos utilizando visión artificial, la cual se realice detección de señales de tránsito a escala de laboratorio con el robot Khepera IV, el cual generará una solución desde otro punto de vista a la problemática de detección de señales en robótica móvil enlazado a la visión artificial.

2 Marco teórico

Los grandes avances tecnológicos que se han desarrollado en este último siglo, han logrado dar soluciones a varios problemas que se han presentado en la robótica móvil. En ella se explayan muchas teorías que permiten su funcionamiento autónomo e inteligente, en esta sección se detallan distintos aspectos que son necesarios para su labor. Dado el objetivo de este proyecto, es necesario tener conocimiento del sistema interno y externo del robot, técnicas de procesamiento de imágenes, simuladores de robótica móvil y estudio de características del robot a utilizar.

2.1 Sistemas del robot

En general, un robot es un sistema compuesto por varios sistemas internos y externos, por lo que tienen distintos funcionamientos que permiten realizar un movimiento, entre ellos está el sistema mecánico, sensorial, de control y alimentación.

2.1.1 Sistema mecánico

Un robot móvil debe tener una manera de interactuar con su entorno, en el que pueda moverse definitivamente y apropiadamente para su aplicación. En el diseño es necesario considerar algunas características, por ejemplo, su peso, tamaño, modo de transporte y estabilidad. En cuanto al peso, es una variable muy importante, ya que afecta su movilidad y define la potencia necesaria que necesitan tener los actuadores para su desplazamiento, a su vez, el diseño debe estar bien definido, ya que el peso definirá probablemente su material de construcción. En la estabilidad, en el robot es necesario calcular el centro de balance horizontal y vertical, es decir, hay que diseñar al robot ajustando el peso de forma equitativa por todos los lados. Por último, en el sistema de engranajes utilizados para la transmisión de fuerza, estos deben ser desarrollados en cuanto a lo que se requiere, ya que, si se necesita fuerza o velocidad, la caja reductora tendrá menos o más engranajes.

2.1.2 Sistema sensorial

El sistema que da la información sobre el entorno del robot es el sistema sensorial. Los sensores se pueden dividir en dos categorías, los sensores propioceptivos, el cual determinan la velocidad y la posición del robot, y los sensores exteroceptivos, que se encargan de establecer la interacción del entorno con el mismo. Todo este sistema esta abarcado sobre un principio, en como los

humanos perciben variables para procesarlas y ejecutar una función, como es la detección de objetos, donde se requiere datos como la posición y la distancia del robot, por lo que los sensores tradicionales (capacitivos, inductivos, infrarrojos, ultrasónicos) ayudan como herramienta a la detección. Este sistema debe ser diseñado de manera que exista un criterio de comparación confiable, así para cumplirlo, es necesario emplear sensores idénticos que utilicen el mismo principio de detección, para luego aplicar una estadística e interpolación, con el fin de mejorar el sistema sensorial.

2.1.3 Sistema de control

El sistema más importante de todos es el sistema de control, ya que este se encarga de recibir la información (sensores) y transmitirla (motores) como una nueva salida, es decir, realiza todo el procesamiento en un robot. Este sistema emplea el manejo de motores, manejo de información del sistema sensorial, sistema de alimentación, manejo de algoritmos de navegación y procesamiento de información. En otras palabras, es el cerebro del robot, ya que esta toma todas las decisiones de todo el sistema completo.

2.1.4 Sistema de alimentación

Uno de los elementos más importantes para un sistema autónomo, es la fuente de alimentación, la importancia radica en varios segmentos, por ejemplo, el peso, si el robot tiene una carga muy elevada, provocará una fuerza mayor al sacarlo de la inercia, lo que producirá un mayor gasto de energía para su movilidad. Además, este sistema debe poder suministrar un nivel de tensión y corriente adecuado para los componentes del robot, de lo contrario, llevará al robot a un estado de mal funcionamiento, lo que producirá desconexiones con las bases transmisoras, desconocimiento de las rutinas a realizar, perdidas de ruta, entre otros.

2.2 Librerías de visión artificial

Para realizar el procesamiento de una imagen, existen librerías que permiten la manipulación de ellas de una manera más sencilla. Dentro de los años se han diseñado distintas librerías el cual poseen distintas especificaciones, por lo que la librería utilizada dependerá de lo que uno realice. A continuación, se presentan algunas de ellas:

2.2.1 Torch3vision

La librería Torch3vision es una librería escrita en lenguaje C++, el cual dispone de un procesamiento básico de imágenes, por ejemplo, detección de rostros, de bordes, cascadas de clasificadores, implementación de características y algoritmos basados en la iluminación. También posee otras características como leer y escribir formatos de imágenes básicos, manipulación y dibujo de objetos en dos dimensiones, decodificación y codificación de archivos de video, capturas de imágenes desde un capturador de fotogramas o desde un USB, control de giro, inclinación, entre otros. Por ejemplo, en [22] se realiza un reconocimiento de rostros, donde el sistema es invariante al nivel de iluminación, lo que lo convierte en un detector confiable.

2.2.2 VXL

VXL (Vision something Libraries) es una librería que fue creada para la investigación e implementación de imágenes, con el objetivo de ser un sistema más ligero, rápido y consistente. Entre ella, las librerías más importantes de VXL son:

- VNL (numéricos): Contenedores numéricos y algoritmos, por ejemplo, matrices, vectores, descomposiciones, optimizadores.
- VIL (imágenes): Carga, guarda y manipula imágenes en muchos formatos de archivos comunes, incluidas las imágenes de gran tamaño.
- VGL (geometría): Geometría para puntos, curvas y otros objetos elementales en uno, dos o tres dimensiones.
- VSL (transmisión de I/O), VBL (plantillas básicas), VUL (Utilidades): Funcionalidad independiente de la plataforma.

Además de las bibliotecas principales, existen librerías que cubren algoritmos numéricos, procesamiento de imágenes, sistema de coordenadas, geometría de cámara, manipulación de video, recuperación de estructura de movimientos, modelado de probabilidad, clasificación, entre otros [23].

2.2.3 JavaVIS

JavaVIS es una librería escrita en lenguaje java que permite portabilidad, fácil ampliación, interfaz gráfico y formato de imágenes especiales que permite almacenar secuencias de imágenes y datos de tipo geométrico. Esta tiene disponible algunas funciones, entre ellas la conversión de formato, ajuste de brillo, contraste de intensidad de las imágenes, suavizado, realce, convolución, entre otros [24].

2.2.4 LTI-lib

El LTI-lib es una librería orientada a objetos con algoritmos y estructuras de datos utilizados frecuentemente en el procesamiento de imágenes y la visión por computadora. El objetivo principal es proporcionar una biblioteca orientada a objetos en C++, lo que simplifica el uso compartido y el mantenimiento de códigos, pero sigue proporcionando algoritmos rápidos que se pueden usar en aplicaciones reales. Esta librería posee más de 500 funciones distintas en el que se destacan en los siguientes aspectos:

- Algebra lineal: Se proporcionan matrices, vectores, tensores y funtores para extraer valores propios, vectores propios, soluciones de ecuaciones lineales, estadísticas, etc.
- Clasificación y agrupación: Clasificadores de funciones de base radial, máquinas de vectores de soporte, medidos k-medias, medidos difusos C-medias, estadística de clasificación son solo algunos ejemplos de lo que pueden hacer con LTI-lib.

- Procesamiento de imágenes: La mayoría de las clases se ocupan de los problemas de procesamiento de imágenes, diferentes enfoques de segmentación, filtros lineales, wavelets, filtros orientables y muchos más ya están disponibles.
- Visualización y herramienta de dibujo: La parte más difícil al desarrollar algoritmos de procesamiento de imágenes en C++ es mostrar imágenes temporales durante la depuración. Debido a la arquitectura orientada a objetos de LTI-lib, solo necesita crear un objeto de visor y darle la imagen que necesita mostrar, y si además se necesitara dibujar alguna información adicional sobre la imagen (algunos textos, elipses, cuadros, líneas o puntos) pueden usar uno de los objetos de dibujo [25].

2.2.5 OpenCV

OpenCV (Open source Computer Vision library) es una reconocida librería de visión artificial, el cual posee interfaces de C++, Python y Java. Además, es compatible con sistema operativo Windows, Linux, Mac OS, IOS y Android, lo que lo convierte una librería muy accesible. La librería fue diseñada para una eficiencia computacional y con un fuerte enfoque en aplicaciones en tiempo real. OpenCV cuenta con más de 2500 algoritmos optimizados, que incluyen un complejo de algoritmo de visión artificial y de aprendizajes automáticos, tanto clásicos como avanzados. Estos algoritmos se pueden usar para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en videos, rastrear movimientos de la cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, encontrar imágenes similares en una base de datos de imágenes, seguir los movimientos oculares, reconocer el escenario y establecer marcadores para superponerlo con realidad aumentada [26].

2.3 Segmentación

Existen muchas maneras de obtener características de una imagen, entre ellas está la técnica de la segmentación, el cual subdivide una imagen, con el fin de separar las partes de interés del resto de la imagen, por lo tanto, el nivel que lleva la subdivisión depende del problema a resolver. De otra manera la segmentación clasifica los puntos (pixeles) de la imagen indicando que clase pertenecen.

2.4 Umbralización

Otra técnica es la umbralización (enmascaramiento), es uno de los métodos más importantes para la segmentación de imágenes. Se define el umbral como una función que convierte una imagen con diferentes tonalidades en una imagen en blanco y negro. Si la imagen original es (x, y), la imagen segmentada será (x', y') el cual será determinado por un umbral $U(0 < U < 255)$. La operación de enmascaramiento se define como:

$$(x', y') = 255 \text{ Si } (x, y) > Umbral$$
$$(x', y') = 0 \text{ Si } (x, y) <= Umbral$$

Se selecciona un umbral que permita agrupar los pixeles de una imagen perteneciente a los diversos objetos de la misma imagen diferenciándolos del fondo.

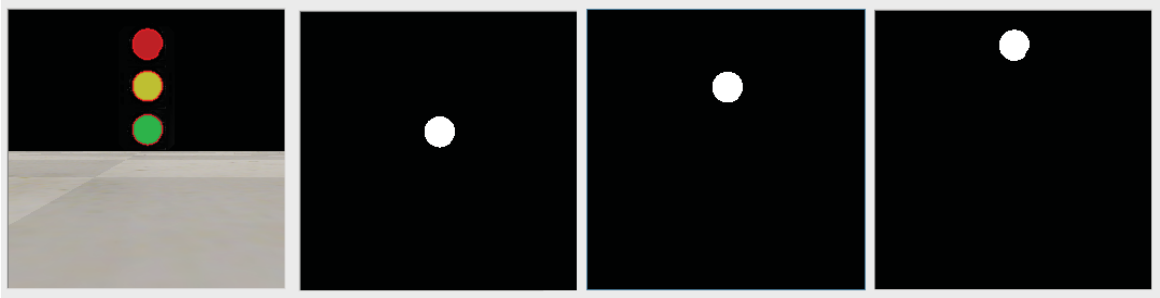

Figura 2-1: Imagen Segmentada.

La Figura 2-1 representa la umbralización de los colores de un semáforo. La imagen de la izquierda representa la figura original y las tres imágenes a la derecha su enmascaramiento, lo cual existe una segmentación por cada color, ósea, se umbraliza para el color verde, amarillo y rojo. Para la realización de la umbralización de la imagen, es necesario saber en qué espacio o código de colores se utilizará, por lo que a continuación se describen los más importantes.

2.4.1 Espacio de color RGB

Es un espacio de color aditivo donde los colores se obtienen mediante una combinación lineal de valores, además los canales esta correlacionados por la cantidad de luz que llega a la superficie.

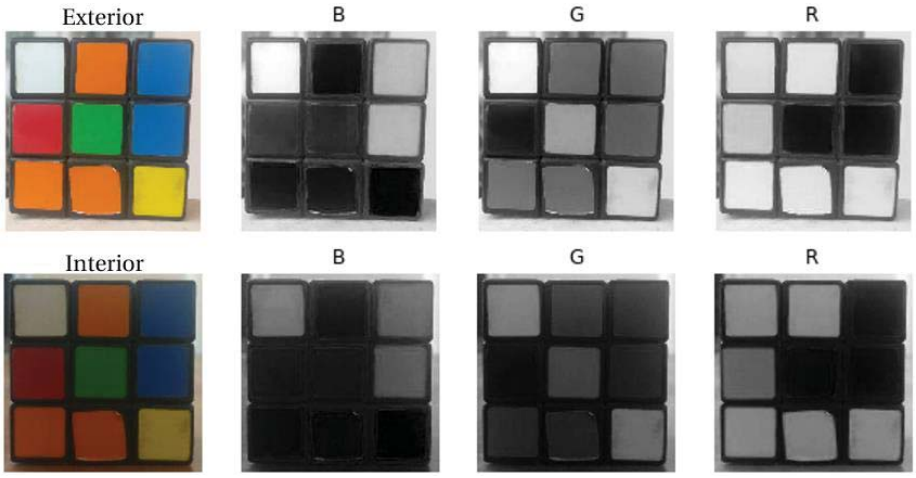

Figura 2-2: Componentes RGB del cubo rubik (fuente: http://www.learnopencv.com).

Si se observa la Figura 2-2 en el canal B, las piezas blancas y azules se ven similares pero existe una clara diferencia por el asunto de la iluminación, este tipo de falta de uniformidad hace que la segmentación basada en el color sea muy difícil en ese espacio, ósea, se mezcla la información de datos de crominancia (información relacionada con el color) y de la luminancia (información relacionada con la intensidad).

2.4.2 Espacio de color LAB

Es un espacio de color cromático usado normalmente para describir todos los colores que pueden percibir el ojo humano. Los tres parámetros del modelo representan la luminosidad de color, su posición entre rojo-verde y su posición entre amarillo y azul.

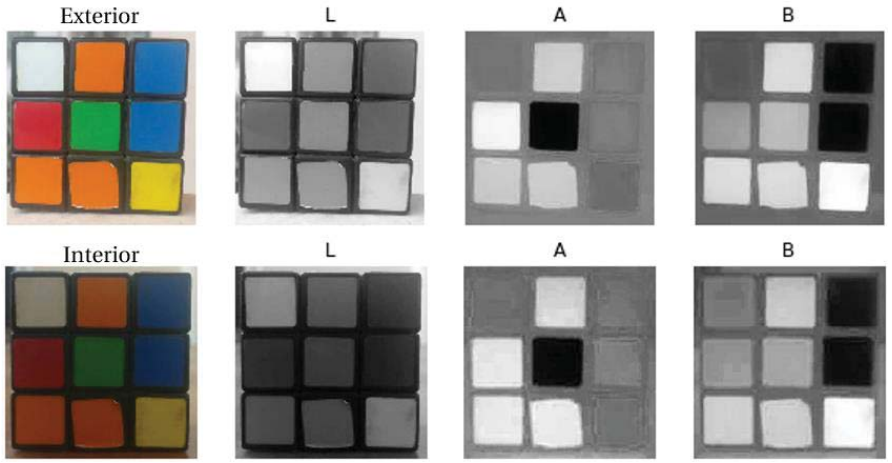

Figura 2-3: Componentes LAB del cubo rubik (fuente: http://www.learnopencv.com).

El canal L es independiente de la información de color y solo codifica el brillo, además este espacio de color es perceptualmente uniforme que se aproxima a la forma que se percibe el color, independientemente del dispositivo. En la Figura 2-3 se observa el cambio de iluminación que afecta a la componente L, pero las componentes A y B no sufrieron muchos cambios.

2.4.3 Espacio de color YCrCb

YCrCb es derivativo del espacio de color RGB, el parámetro Y pertenece a la luminancia obtenida después de la corrección gamma, Cr es R-Y el cual significa que tan lejos está el componente rojo de Luma y Cb el cual es B-Y que trata sobre qué tan lejos es la componente azul de Luma.

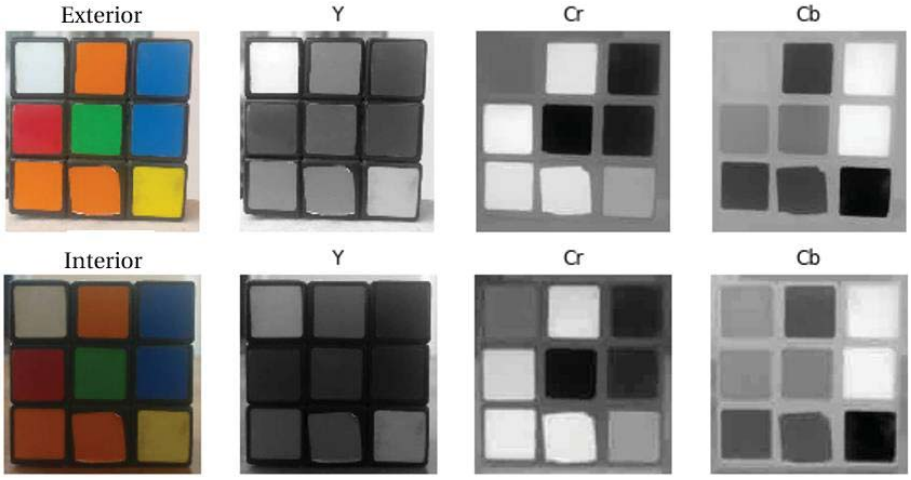

Figura 2-4: Componentes YCrCb del cubo rubik (fuente: http://www.learnopencv.com).

En este espacio de color mostrado en la Figura 2-4 se hace similar al espacio de colores LAB para los componentes de intensidad y color con respecto a los cambios de iluminación, la diferencia radica entre el color rojo y naranja, ya que estos son menos uniformes en el exterior que en LAB, además el color blanco ha sufrido cambios en los tres componentes.

2.4.4 Espacio de color HSV

Este espacio está definido por H (pureza/tonos de color), S (saturación) y V (intensidad).

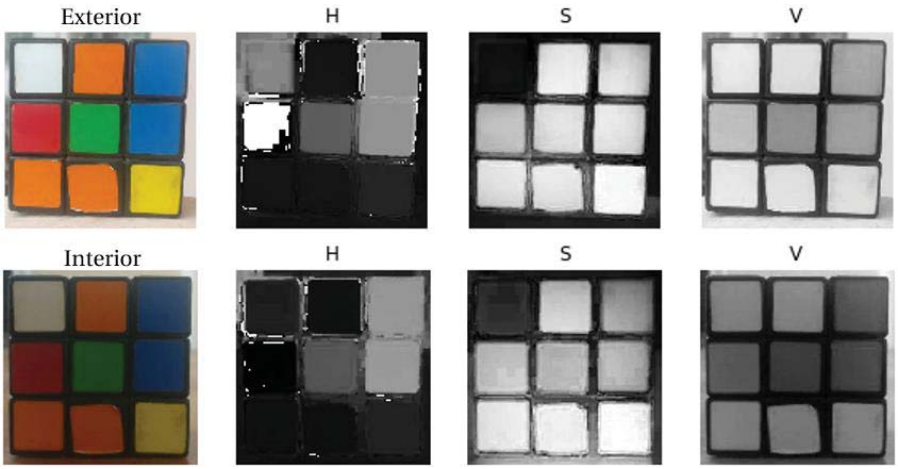

Figura 2-5: Componentes HSV del cubo rubik (fuente: http://www.learnopencv.com).

La componente H en la Figura 2-5 es muy similar en ambas imágenes, lo que indica que la información de color está intacta incluso bajo cambios de iluminación, de la misma manera sucede en la componente S, el componente V captura la cantidad de luz que cae sobre él, por lo que cambia debido a los cambios de iluminación. Existe una diferencia drástica entre los valores de la pieza roja de la imagen exterior e interior, esto se debe a que H se representa como un círculo y el color rojo está en el ángulo de inicio. Por lo tanto, puede tomar valores entre [300,360] y nuevamente [0,60]. Todo lo relacionado con Espacio de colores, se extrajo de la referencia [27].

2.5 Cascadas de clasificadores

Cuando los clasificadores son débiles, estos se pueden combinar en una suma ponderada que representa el resultado final del clasificador potenciado, en el que se mejora excepcionalmente la clasificación produciendo buenos resultados, esta herramienta viene dada por el algoritmo Boosting. Este es un meta-algoritmo de aprendizaje automático que reduce la desviación y la varianza en un contexto de aprendizaje supervisado [28] [29], el cual está basado por el siguiente cuestionamiento: ¿Puede un conjunto de clasificadores débiles crear un clasificador robusto? [30] [31]. Un clasificador débil está definido para ser un clasificador que esta débilmente correlacionado con la clasificación correcta (el mismo clasifica mejor que un clasificador aleatorio), por otro lado, un clasificador que tiene un mejor desempeño que el de un clasificador débil, es uno que trabaja en conjunto con otro, ya que sus clasificadores se aproximan más a las verdaderas clases. De esta manera se puede crear una jerarquía de clasificadores, ya que el

algoritmo provee una mejor hipótesis sobre las instancias en donde los demás se equivocan, creando así una suposición conjunta que es fuerte, dando así precisión y confianza. De todos los algoritmos Boosting uno género un debate importante en la comunidad de Machine Learning, debido a que sus resultados superaban las predicciones, este método se llamó Adaboost.

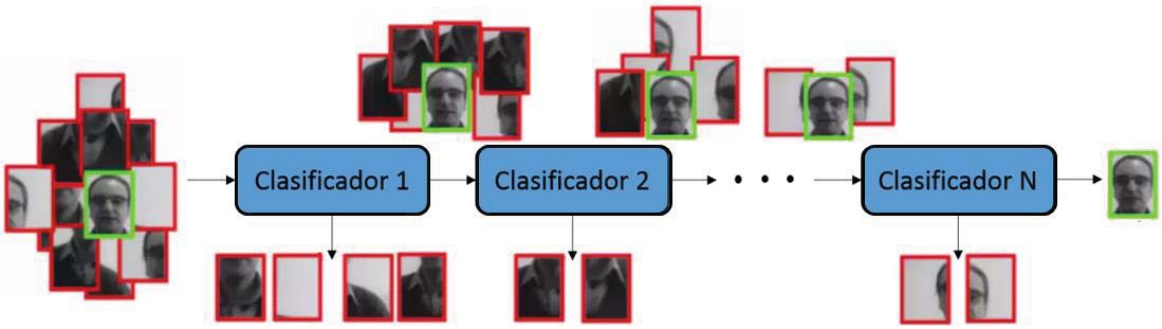

Figura 2-6: Detección de rostros con clasificadores en cascada (fuente: https://es.coursera.org).

En la Figura 2-6 se observa una cascada de clasificadores en la detección de rostros, el cual dada una imagen de entrada esta es clasificada por una serie de clasificadores. Esta cascada permitirá alcanzar el objetivo, mediante una combinación secuencial de clasificadores, de tal manera que una imagen será detectada como positiva si es reconocida de manera correcta por todos los clasificadores de la cascada, además si solo uno de estos clasificadores no admite la imagen como positiva, esta será rechazada [32]. Para la extracción de características, existen distintos métodos, entre ellos esta HAAR, LBP y HOG.

2.5.1 Clasificador tipo HAAR

Cada característica es un valor único que se obtiene al restar la suma de pixeles debajo del rectángulo blanco, de la suma de pixeles debajo del rectángulo negro. En la Figura 2-7 se puede observar tres maneras de obtener características, entre ellas está la característica de borde, de línea y de cuatro rectángulos. Ahora todos los tamaños y ubicaciones posibles de cada imagen se utilizan para calcular muchas características. Para cada cálculo de características, es necesario encontrar la suma de los píxeles debajo de los rectángulos blanco y negro. Para solucionarlo, se introduce la imagen integral, ya por grande que sea la imagen, los cálculos serán reducidos para un pixel dado a una operación que involucra solo cuatro píxeles.

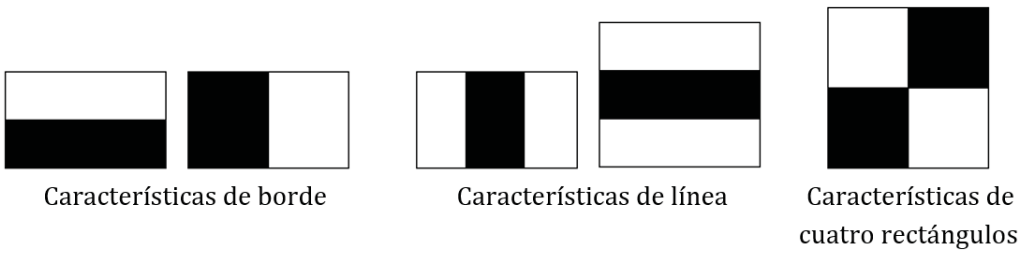

Figura 2-7: Características tipo HAAR (fuente: https://docs.opencv.org).

2.5.2 Clasificador tipo LBP

Local Binary Pattern (LBP) es un operador de textura simple pero muy eficiente, que etiqueta los píxeles de una imagen mediante el umbral de la vecindad de cada píxel y considera el resultado como un número binario. El primer paso para construir el descriptor de textura LBP es convertir la imagen a escala de grises, para cada pixel se selecciona una vecindad que rodea al píxel central, luego se calcula un valor LBP para ese pixel central y se almacena en una matriz bidimensional de salida con el mismo tamaño de la imagen de entrada. A continuación, en la Figura 2-8 se ilustra un ejemplo de esta técnica:

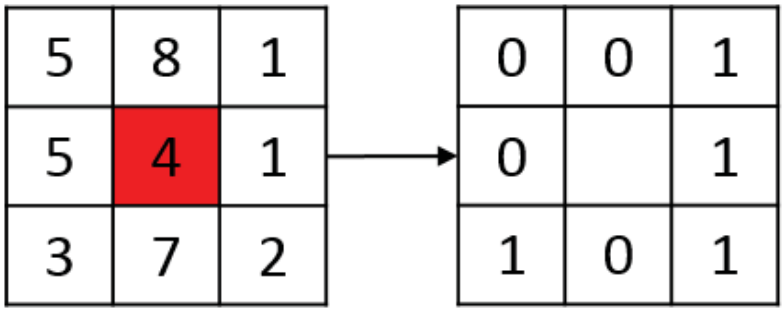

Figura 2-8: Matriz de pixeles de la imagen a LBP (fuente: https://www.pyimagesearch.com).

El primer paso es tomar la vecindad de ocho pixeles que rodea al pixel central y establecer un umbral para construir un conjunto de ocho dígitos binarios. Si la intensidad del pixel es mayor que la de su vecino, esos tomaran el valor de 1, de lo contrario tomara el valor de 0. Con 8 píxeles circulantes, se obtiene un total de 256 (2^8) combinaciones posibles de códigos LBP. Para calcular el valor de LBP para el píxel central, se puede comenzar por cualquier pixel vecino y trabajar en el sentido de las agujas del reloj o en el sentido contrario, pero manteniendo el orden para todos los pixeles de la imagen. Dado un vecindario de 3x3, se tiene 8 vecinos en los que se deben realizar una prueba binaria. Los resultados de esta prueba binaria se almacenan en una matriz de 8 bits que luego se convierte en decimal.

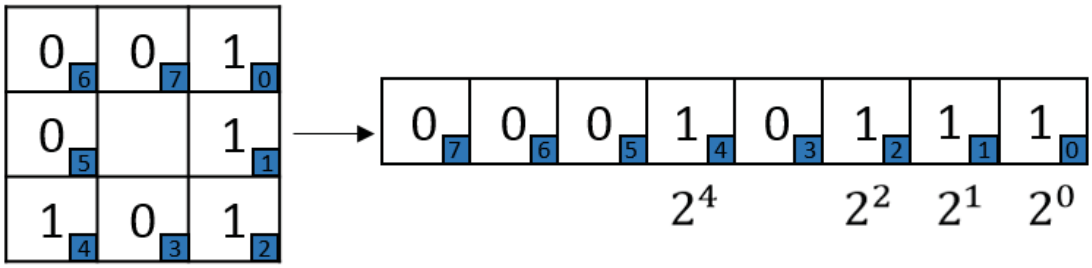

Figura 2-9: Suma de datos binarios (fuente: https://www.pyimagesearch.com).

Tomando la vecindad binaria de 8 bits del pixel central y convirtiéndola en una representación decimal, se obtiene el valor LBP del pixel. En el ejemplo de la Figura 2-9, se comenzó con el punto superior derecho y siguiendo el camino de la aguja del reloj, se va acumulando la secuencia binaria hasta completar el ciclo. Por lo tanto, el valor final del pixel es de 23 ($2^4 + 2^2 + 2^1 + 2^0$), el cual es almacenado en la matriz bidimensional LBP. En la Figura 2-10 se observa la salida de la

imagen en LBP y para completar toda la matriz, todo el procedimiento se repite para cada pixel de la imagen [33].

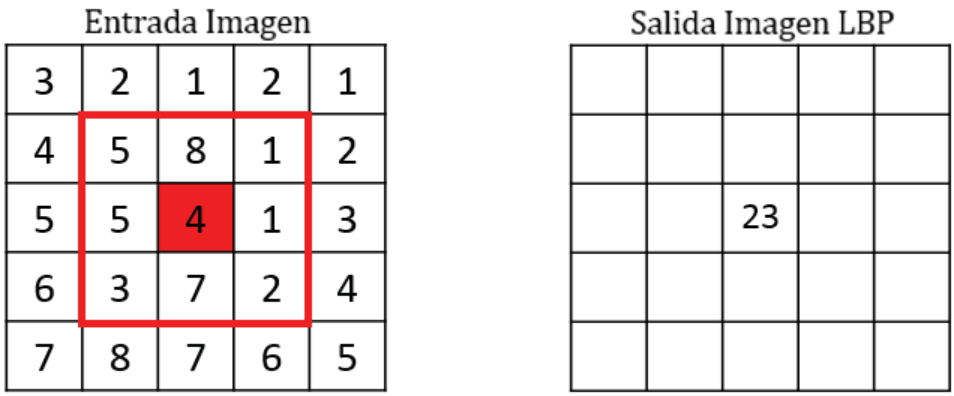

Figura 2-10: Imagen de entrada versus Imagen de salida LBP (fuente: https://www.pyimagesearch.com).

2.5.3 Clasificador tipo HOG

Histogram of Oriented Gradients (HOG) es un descriptor de características utilizado para detectar objetos en la visión por computadora y el procesamiento de imágenes. La técnica cuenta las ocurrencias de la orientación del gradiente en porciones localizadas de una ventana de detección de imágenes o región de interés [34].

HOG se obtiene determinando los gradientes de la imagen que calculan información de contorno y silueta de imágenes en escalas de grises. La información del gradiente se agrupa en un histograma de una dimensión de orientaciones, transformando así una imagen en dos dimensiones en un vector de una dimensión mucho más pequeño que forma la entrada para los algoritmos de aprendizaje automático tales como bosques aleatorios o máquinas de vectores de soportes. En la Figura 2-11 se puede observar un ejemplo de una imagen procesada en HOG, la imagen de la derecha seria la entrada y la izquierda pertenece a la procesada [35].

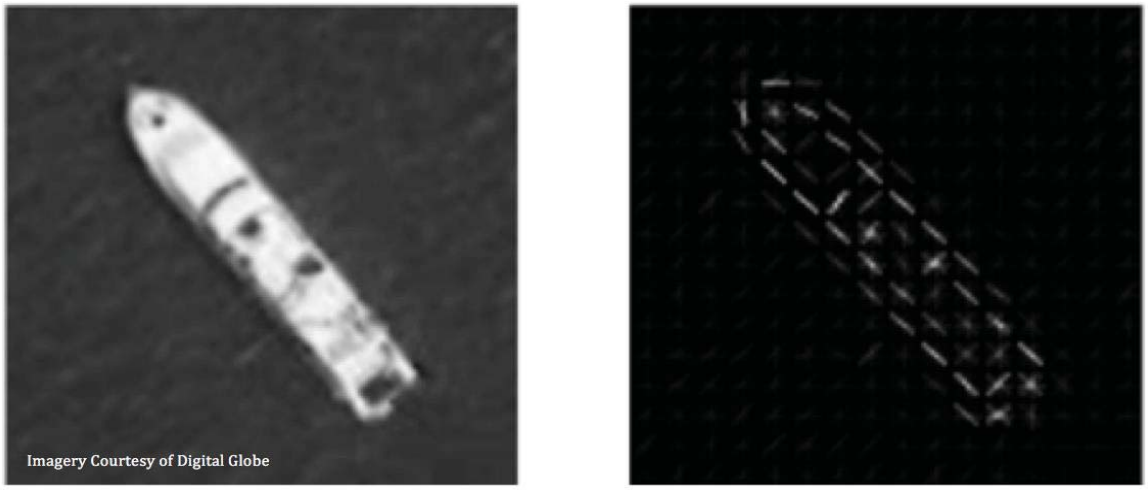

Figura 2-11: Imagen de entrada versus Imagen de salida HOG (fuente: https://medium.com).

2.6 Simuladores Robots móviles

Un simulador en la robótica es algo fascinante, ya que permite una experimentación controlada debido a las posibles causas que puedan suceder especialmente en un entorno de investigación, donde se realizan pruebas de nuevos algoritmos, provocando un entono seguro a la incertidumbre. Actualmente, existen varios simuladores de robot móviles que serán mencionados a continuación:

2.6.1 ARGoS

El simulador ARGoS permite proporcionar características para admitir enjambres de robots heterogéneos a gran escala, lo que permite buenos resultados en simulaciones con gran cantidad de robots y sin degradamiento (Figura 2-12). A su vez, la precisión de los resultados está estrechamente relacionada con la biblioteca de física empleada y no con una biblioteca de física especifica como otros simuladores. Su visión es proporcionar soluciones eficientes para una variedad cada vez mayor de aplicaciones diferentes, como la exploración de entornos hostiles, la recuperación de desastres y la medicina a nanoescala [36].

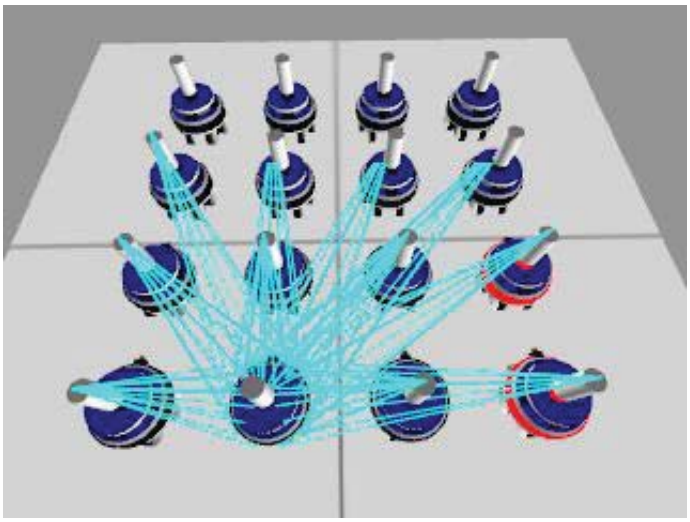

Figura 2-12: Experimentación en la plataforma de ARGoS [36].

2.6.2 Webots

Este simulador de robots proporciona un entorno de desarrollo completo para modelar, programar y simular robots. Se basa en una versión extendida del motor de física ODE que consiste en mejoramientos de cálculo de puntos de contactos para colisiones y estabilidad más precisas, capacidad dinámica de fluidos simple para facilitar la simulación de drones y robots submarinos, por último, implementación de varias optimizaciones incluyendo subprocesos múltiples (Figura 2-13). Además, cuenta con una biblioteca de robots, sensores, actuadores y objetos, importación y exportación de modelos y una amplia multiplataforma como Windows, Linux, Mac, entre otras [37].

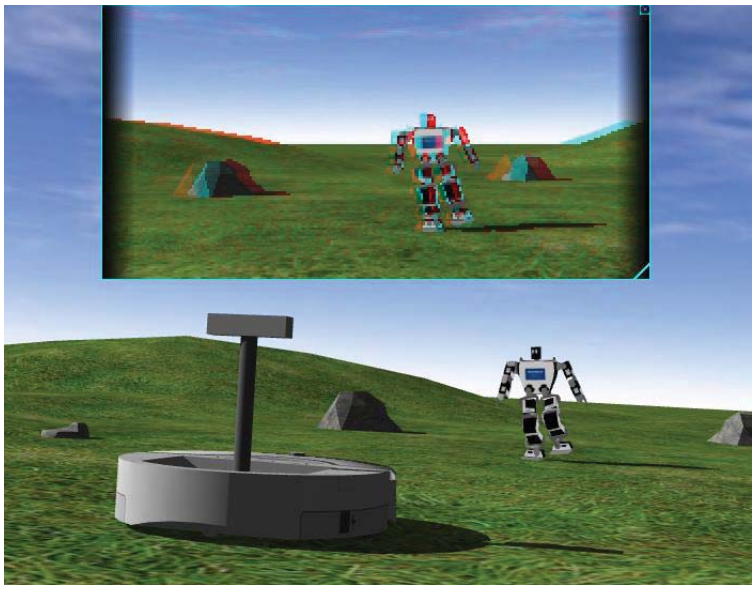

Figura 2-13: Experimentación en la plataforma de Webots [37].

2.6.3 RFCSIM

El RFCSIM es un simulador de robótica móvil para experimentos de control de posición y control de formación de un sistema multi-robots con evitación de obstáculos de altas prestaciones graficas e interactivas desarrolladas con Easy Java Simulations. Este simulador permite demostrar varias leyes y criterios de control así incluyendo obstáculos estáticos y dinámicos (Figura 2-14). También es de gran aporte a la docencia e investigación ya que permite demostrar conceptos asociados a la robótica [38].

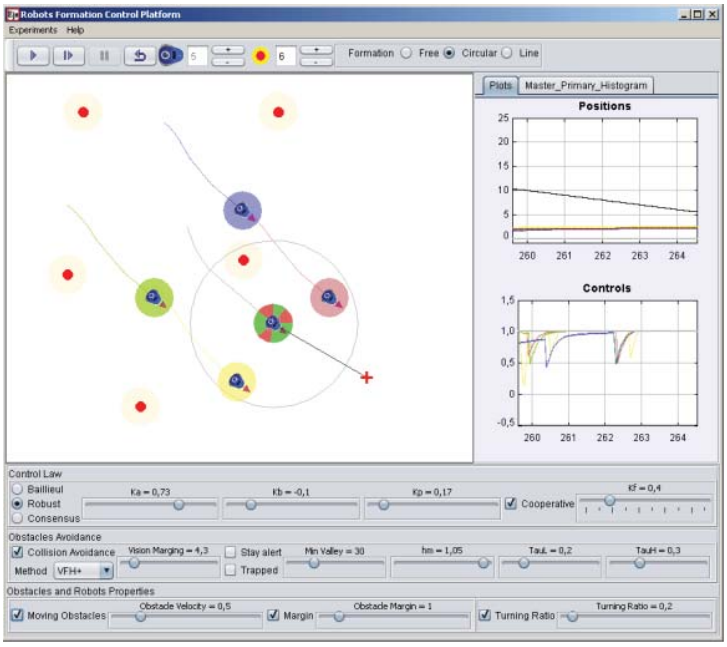

Figura 2-14: Experimentación en la plataforma RFCSIM [38].

2.6.4 V-REP

El simulador de robots V-REP, se basa en una arquitectura de control distribuido: cada objeto/modelo puede controlarse individualmente a través de un script incrustado. V-REP hace simulaciones realistas de cada una de las piezas que forman un robot, como patas, hélices o motores. El usuario puede crear scripts para controlar el movimiento del robot y configurar sus sensores como si se tratase de una máquina real. Este simulador ofrece muchas características, entre ellas están: Distintos sistemas operativos a utilizar, programación en distintos lenguajes, dinámica y física, detección de colisiones, cálculo de distancia mínima, simulación de sensores de proximidad, simulación de sensores de visión, grabación y visualizador de datos, interfaces de usuario personalizada y muchas otras características(Figura 2-15) [39].

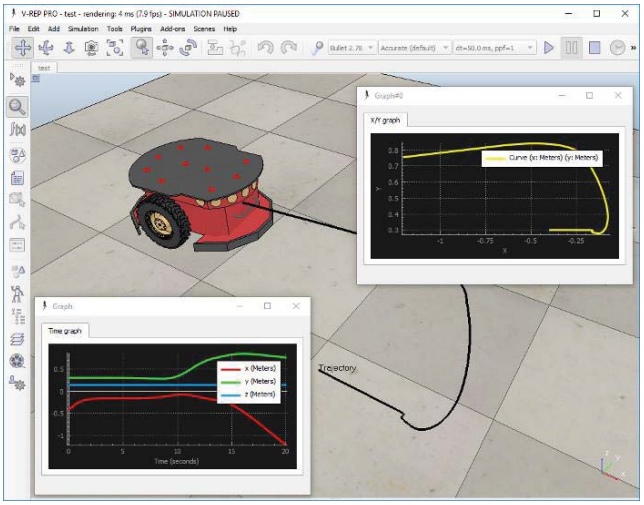

Figura 2-15: Experimentación en la plataforma V-REP [39].

2.7 Khepera IV

La Universidad cuenta con robots móviles de última generación llamados Khepera IV, creado por la Empresa Suiza K-Team, el cual es dedicado al desarrollo, fabricación y comercialización de robots móviles de alta calidad, el cual ha logrado ser una gran fuente en la educación e investigación avanzada. Además, con su experiencia, ha logrado brindar las mejores soluciones, en donde se han implementado en más de 600 universidades y centros de investigación industrial alrededor del mundo [40].

Figura 2-16: Robot Khepera IV [40].

Este robot (Figura 2-16) está diseñado para cualquier aplicación de laboratorio en interiores, como navegación, enjambre, inteligencia artificial, computación, demostración, etc. Este trae consigo una gran cantidad de funciones, como un núcleo Linux, cámara a color, WiFi, Bluetooth, giroscopio, entre muchas más.

2.7.1 Características

Con un núcleo de Linux, el robot incorpora un sistema operativo Linux completo y estándar. Proporciona un entorno C/C++ para el desarrollo de aplicaciones, lo que permite el desarrollo de algoritmos y aplicaciones integradas portátiles. Por otro lado, los componentes externos del robot pueden ser observados a simple vista, por lo que en la Tabla 2-1 se confeccionan estos elementos, además de la cantidad y una pequeña descripción:

Tabla 2-1: Características externas del robot [41].

Elemento	Cantidad	Características
Ruedas Locas	2	Robot estable en desplazamiento por superficie lisa.
Interruptor I/O	1	Encender y apagar el robot.
LED de estado	1	Simboliza el estado del robot.
Conector Mini USB B	1	Comunicación alámbrica del robot.
Conector USB-A	1	Conexión de elementos externos del fabricante.
Conector de alimentación	1	Conector para cargar la batería del robot.
LED de carga	1	Simboliza si la batería se está cargando.
Botón de Reinicio	1	Reinicio total del robot.
Sensor Infrarrojo	8	Sensores de detección de objetos alrededor del robot.
Sensor Ultrasonido	5	Sensores de detección de objetos alrededor del robot.
Cámara	1	Cámara de color situada al frente del robot.
Sensores de luz	4	Sensores usados para la detección de colores.
Contactos para estación de acoplamiento	1	Cargar o comunicarse con dispositivos externos.
Rueda	2	Ruedas de comportamiento diferencial.
Pegatina	1	Se ubica el número de serie del robot.
Tuerca M3 inferior	4	Para acoplar una posible extensión.
Conector Extensión KB-250	1	Conexión de módulos de extensión al robot.
Tuerca M3 superior	4	Para acoplar una posible extensión.
Imanes	3	Sujetar al accesorio pinza o cualquier otro modulo.
LED RGB	3	Identificación de robot.

Las dos ruedas locas permiten una buena estabilidad con y sin carga, además de módulos de extensión como una garra, sin embargo, es imposible para él realizar recorridos de caminos con sobresaltos.

El interruptor de encendido y apagado permite el estado del robot, ya que está conectado a los controladores, permite cargar el dispositivo sin tener que encenderlo.

El conector Mini-USB B connector permite abrir un enlace de comunicación entre el robot y una computadora, además con él se realiza la primera configuración para establecer la comunicación con una red WiFi.

Los 8 sensores infrarrojos permiten la lectura de objetos cercanos desde 2[mm] a 20[cm], con una lectura dentro del sistema desde 0 a 1023, donde es inversamente proporcional a la lectura real (1023 corresponde a 0[mm]). Además, estos se encuentran ubicados 45 grados respecto uno del otro formando una circunferencia.

En el robot existen 5 sensores ultrasónicos, estos están ubicados en la parte frontal del robot a 45 grados separados uno respecto del otro, a su vez estos detectan distancias entre 25 [cm] a 2,5[m] aproximadamente, por lo que es importante estar en contacto con ellos al momento de detectar objetos entre esas distancias, de lo contrario tendrán más relevancias los sensores infrarrojos. Estos son energizados por un voltaje de 85[V_{p-p}], con una frecuencia de 40[kHz] a una tolerancia de $\pm$1[kHz].

La carga de la batería dura aproximadamente 5 [$horas$] con los motores trabajando al 100 [%] y 7 [Hr] con los motores apagados con una tensión nominal de 7.4 [V] y una corriente de 3400 [mA], a su vez existen tres maneras de cargar la batería, la cual es desde el Jack, desde los contactos situados bajo el robot y desde la extensión de los conectores KB-250. El tiempo de carga es relativamente entre 4 a 5 [Hr].

Su cámara situada en la posición frontal del robot y puede ser usada para tomar imágenes y videos para ser procesadas. Su resolución máxima es de 752x480 pixeles con un ángulo de visión horizontal de 131 grados.

Sus dos micrófonos, situados uno a la entrada del MIC MAIN y el otro a la entrada del MIC SUB, poseen una ganancia de 20 [dB] a una tensión de alimentación de 2.5 [V]. Un parlante de 0.7 [W] como potencia nominal con resistencia de 8 [Ω]. Este tiene una distorsión máxima de un 5[%] y trabaja en un rango de frecuencias de entre 400 a 20000[Hz].

El robot posee tres LED RGB en el TOP de la placa principal, cada uno de estos tiene una guía de luz. Los LED son impulsados por un controlador LED dedicado que proporciona una resolución de 6 bits para cada color. Estos LED pueden usarse para ubicar el robot debajo de una cámara (en el techo) y diferenciar cada robot (en aplicaciones de enjambre). Como estos LED forman un triángulo isósceles, la dirección del robot también puede ser detectada.

Este robot fue diseñado para insertar una placa del procesador Gumstix Overo, el módulo del computador viene montado por defecto en el robot y es el Gumstix Overo FireSTROM COM. Este procesador viene con arquitectura ARM Cortex-A8 de Texas Instruments modelo OMAP3730 de 800[MHz] y WiFi 802.11 [b/g]. Todas las características del robot son mencionadas en su manual citado en [41].

3 Preparación y desarrollo del sistema

Para las experimentaciones, con el fin de cumplir el objetivo principal, es necesario acomodar el sistema donde las condiciones sean óptimas, además de preparar todos los elementos y materiales. En esta sección se detalla todo el desarrollo del trabajo realizado, desde la preparación del entono de trabajo, construcción de señales para la detección, construcción de cascada de clasificadores de todas las señales (desde la obtención de imágenes negativas) y por último una propuesta de un sistema para el robot Khepera IV en la detección de objetos a través de su cámara.

3.1 Entorno de trabajo

Para obtener una experiencia satisfactoria en el desarrollo de los experimentos que se llevan a cabo, se debe tener claro el entorno en donde se ejecutaran las pruebas. Para la robótica móvil siempre es recomendable realizar experimentaciones tanto de manera real como simulada, con el fin de eliminar posibles problemas el cual los costos podrían ser muy elevados.

3.1.1 Entorno simulado

Dentro de los programas en simulación de robots móviles, en [42] se detalla una comparación de las características más importantes en los simuladores más usados en la actualidad.

Tabla 3-1: Comparación de simuladores para la robótica móvil [42].

		ARGoS	Webots	RFCSIM	V-REP
Usabilidad		✗	✓	✓	✓
Escalabilidad		✓	✓	✓	✓
Visual	2D	✓	✓	✓	✓
	3D	✓	✓	✗	✓
Lenguaje	C	✗	✓	-	✓
	C++	✓	✓	-	✓
	Java	✗	✓	✓	✓
Licencia libre		✓	✗	✓	✓
Interactivo		✗	✓	✓	✓

Siempre al momento de realizar una experimentación nueva, el cual no se hayan obtenido mucha información, se debe realizar un modelado del sistema, el cual permitirá predecir los riesgos que puedan tener, ya sean por un costo elevado para el sistema o un peligro para las personas. Por sus múltiples características mostrados en la Tabla 3-1, se prefirió utilizar el entorno de simulación V-REP, para ello se utilizó el modelo del robot Khepera IV.

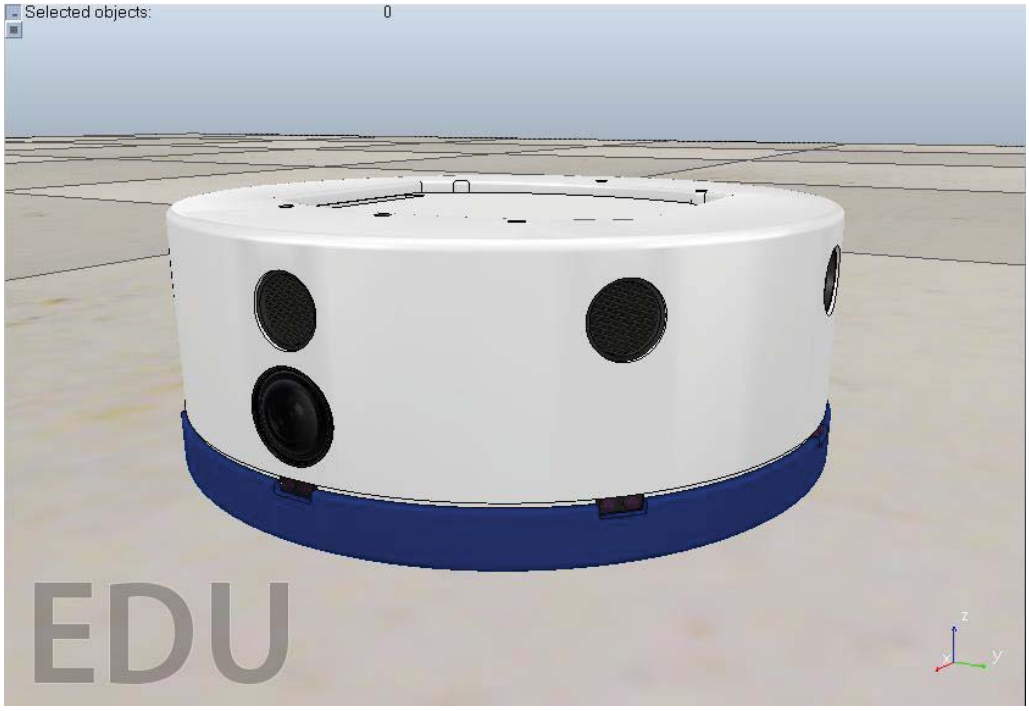

Figura 3-1: Robot Khepera IV modelado en la plataforma V-REP.

El robot Khepera IV (Figura 3-1) fue modelado en el entorno V-REP, realizado por el exestudiante Emmanuel Peralta. Aunque se destaca su capacidad de programación en un script de V-REP, en donde el modelo consta de 8 sensores infrarrojos con una distancia máxima de 20 [cm], 5 sensores ultrasónicos con una distancia mínima soportada de 20 [cm] y una distancia máxima de 2 [m], incluye el modelo de la cámara, la cual se puede acceder al igual que en el robot real, y ser tratadas en diferentes entornos de programación (por ejemplo Python), además de incluir los dos actuadores correspondiente a sus ruedas paralelas [21]. Este modelo fue de gran ayuda ya que además de que el robot real podría estar comprometido, este es de muy alto costo, por lo que el modelo genera una gran confianza a la realización de pruebas reales.

3.1.2 Entorno real

Las experimentaciones reales siempre son recomendables debido a que en entorno simulado no es posible simular todos los aspectos que un entono real si las tiene, por ejemplo, en la robótica móvil sería el estado de las ruedas, deslizamiento de estas en una plataforma no uniforme, problemas de iluminación, entre otros. La Universidad cuenta con varios robots móviles que permitirán la realización de pruebas con el objetivo de compararlas con las experimentaciones virtuales.

Plataforma de experimentación

Antes de todo, para la realización de todas las pruebas con el robot, fue necesario la utilización de la plataforma que se encuentra en el laboratorio de robótica (Figura 3-2). Esta es una plataforma lisa de 2x1.5 $[m^2]$ el cual posee una cámara PlayStation eye PS3 en la parte superior con el fin de visualizar todos los puntos de la plataforma, además, está conectada a un computador que además de realizar el procesamiento de imagen, permite conectarse con el robot móvil.

Figura 3-2: Plataforma de experimentación.

Una técnica que permitirá saber la posición del robot, es por medio de un código que se ubica en su parte superior visualizada por la cámara PS3 y la utilización de un programa llamado SwisTrack. Esta es una herramienta para el rastreo de robots móviles, el cual utiliza la librería de OpenCV para el procesamiento de imágenes. Luego de una configuración del entorno, cada componente que este en vista de la cámara reflejara una posición y orientación, convirtiéndola en un detector de objetos en tiempo real [43].

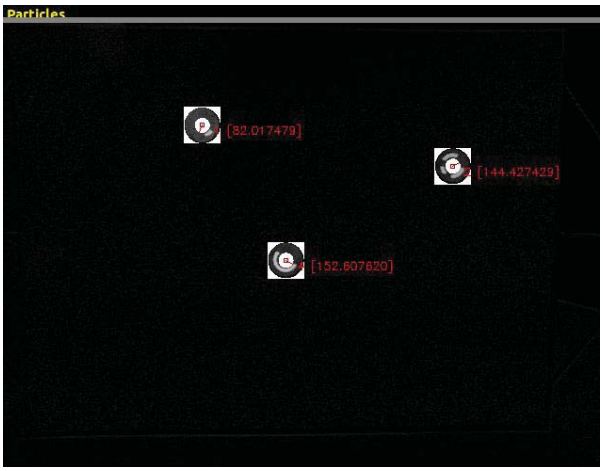

Figura 3-3: Detección de tres robots con aplicación SwisTrack.

En la Figura 3-3, se muestra un ejemplo de la aplicación SwisTrack, en ella se observan 3 robots móviles que están distribuidos por toda la plataforma, rastreando su posición y orientación. Con la ayuda del programa Easy Java Simulations (EJS), SwisTrack le enviará las posiciones y orientación para que estas sean enviadas al robot, en la Figura 3-4 se observa un diagrama de los programas dichos anteriormente para la comunicación de la posición y orientación del robot.

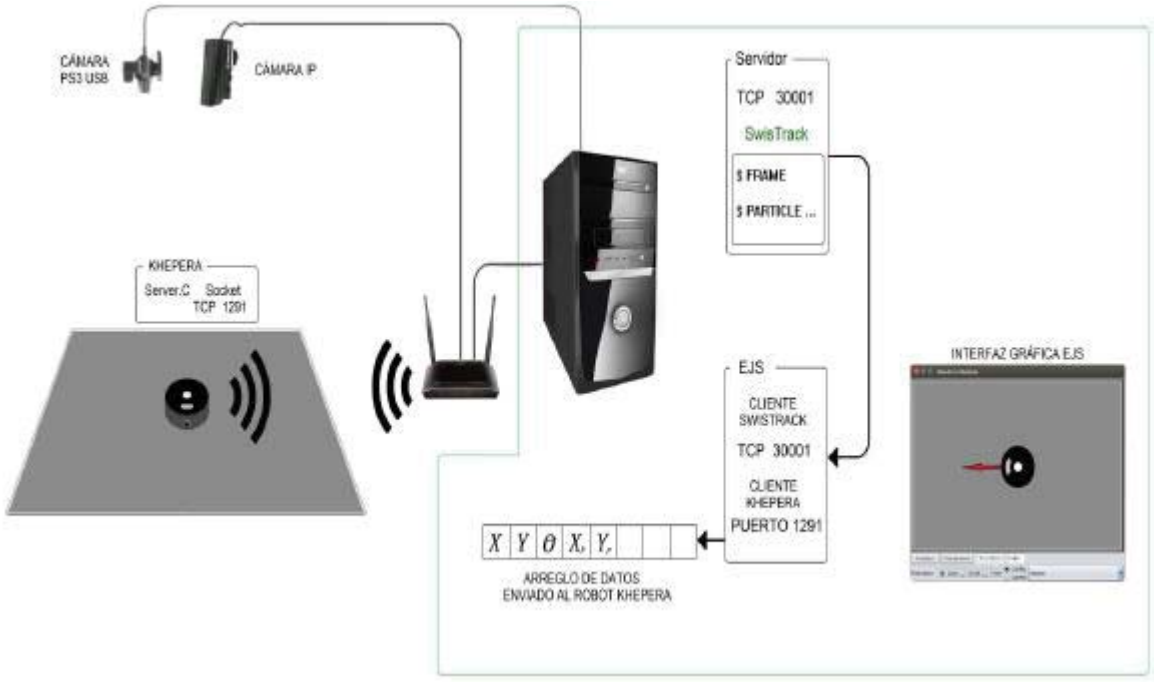

Figura 3-4: Diagrama de hardware y software de la plataforma experimental [21].

3.2 Construcción de señales de tránsito

Para que el robot realice un movimiento autónomo dependiendo de lo que está visualizando su cámara, es necesario realizar un control de los elementos que debería estar analizando, es por ello que en este apartado se muestra la construcción de las señales de tránsito que serán utilizadas para su detección. Este punto es importante debido a que es necesario utilizar algún soporte para que la cámara del robot aprecie las señales de buena manera, para ello se optó por diseñarlas en la aplicación Autodesk Inventor para luego ser imprimidas.

El software Autodesk Inventor proporciona una gran cantidad de herramientas para que los diseñadores puedan diseñar mecanismos en 3D de manera profesional, a su vez este proporciona simulación, visualización y documentación [44]. Como se puede observar en la Figura 3-5, se aprecia el entorno de diseño del programa Inventor el cual posee múltiples funciones que permiten la facilidad de crear cualquier pieza que uno desee. El programa inicia primero con la creación del boceto en dos dimensiones, para luego crear su profundidad.

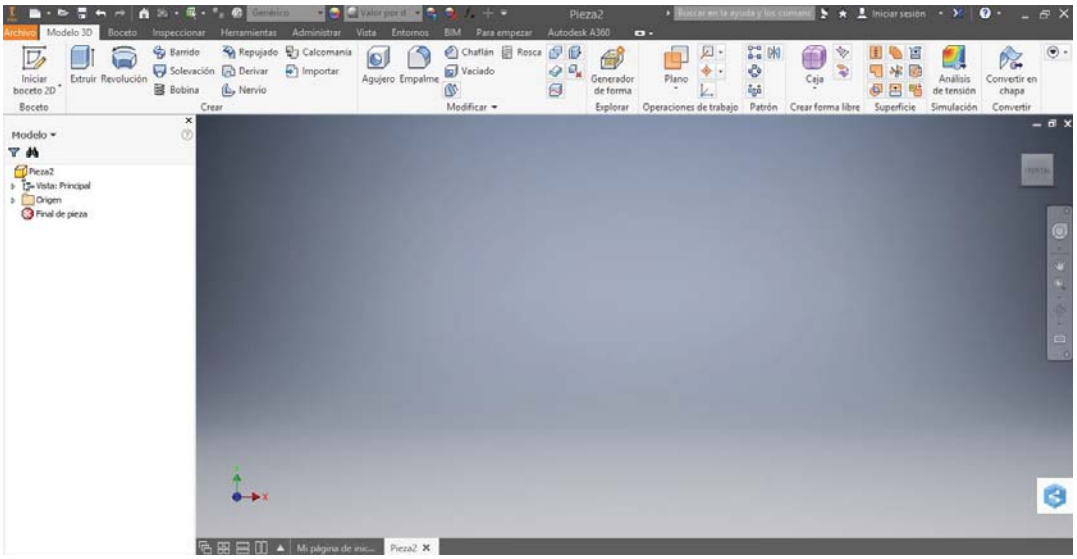

Figura 3-5: Entorno de diseño Inventor.

Para explicar lo mencionado anteriormente, en la Figura 3-6 se muestra un ejemplo de la creación del logo de la Pontificia Universidad Católica de Valparaíso, en la parte de la izquierda se realiza el diseño en 2D y en la imagen de la derecha la profundidad.

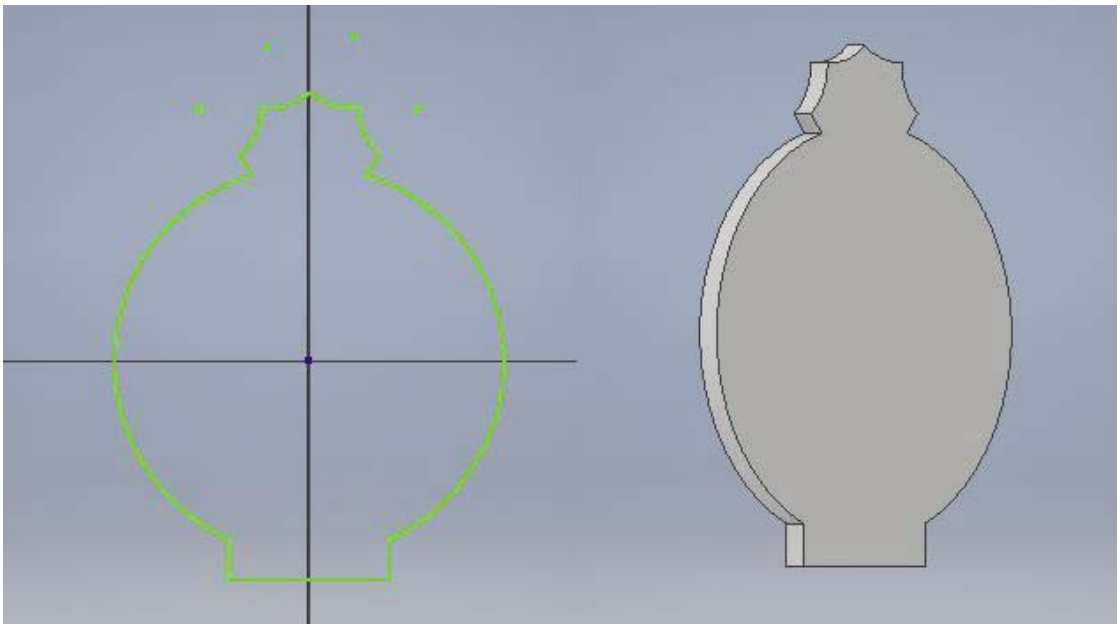

Figura 3-6: Creación logo PUCV en Inventor.

Ahora teniendo la pieza diseñada, es momento de crear el archivo que permitirá imprimirla en una impresora 3D. En el laboratorio de Robótica de la Escuela de Ingeniería Eléctrica existe una impresora llamada Replicator 2 de la empresa MakerBot, esta impresora de cuarta generación posee una resolución de 100 micrones, lo que quiere decir un poco más que el diámetro de un

cabello humano [45]. Esta impresora es una de las más rápidas y fácil de utilizar al realizar modelos profesionales, en la Figura 3-7 se puede observar la estructura de la señal diseñada.

Figura 3-7: Pieza logo PUCV creada en impresora 3D.

La impresión de la pieza del logo PUCV tanto la parte frontal y trasera quedo en buen estado, además se diseñó un soporte para que la pieza quedara de manera vertical. Para la detección de más señales se realizó el diseño de todas ellas para su seguida detección, estas señales fueron la señal Stop, giro izquierdo, ceda el paso, Aeropuerto, límite de velocidad 60, límite de velocidad 120, giro derecha y Mc Donald's. Para que las pruebas sean aún más llamativas, se realizó la impresión de un total de 14 piezas; en la Figura 3-8 se ilustran algunas de las señales mencionadas.

Figura 3-8: Señales de tránsito creadas en impresora 3D.

3.2.1 Construcción de semáforo

Para la navegación en robótica móvil con la detección de señales de tránsito, existe una gran cantidad de señales que poseen distintos significados, el cual permiten tener un orden y seguridad para las personas tanto peatonales como conductores. Un elemento muy importante es la señal semáforo, ya que dependiendo de sus colores (verde, amarillo y rojo) tendrán diferentes significados. Para la detección de este elemento es necesario diseñarlo tanto para una experimentación simulada como una real.

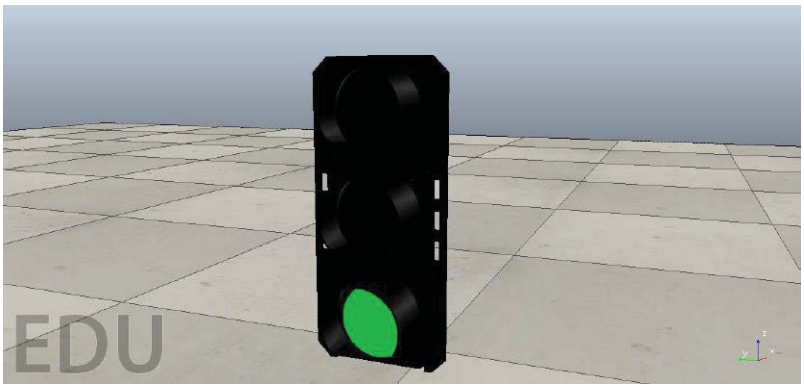

Figura 3-9: Modelo Semáforo en el entorno V-REP.

El diseño del semáforo fue extraído por el sitio web Grabcad, el cual es una comunidad en línea de diseñadores profesionales, ingenieros, fabricantes y estudiantes [46]. En la Figura 3-9 se puede observar el modelo del semáforo dentro de la plataforma V-REP, la programación de este fue de igual manera a uno real, empezando por el encendido de la luz verde, después la luz amarilla y por último la luz roja; repitiendo estos pasos hasta cuando el usuario termine la simulación del robot.

Para el proceso en el entorno real, se utilizó una placa de desarrollo que permite programar el semáforo de tal manera que su funcionamiento sea realizado en una estructura muy pequeña. El digispark attiny85 es una placa de pequeño tamaño y bajo costo, compatible con el entorno de arduino (IDE). Esta alimentado por un procesador de Atmel, un procesador de 8 bits AVR con arquitectura RISC [47]. Este procesador incorpora las siguientes características:

- 8KB de memoria flash.
- 512B de memoria EEPROM.
- 512B de memoria SRAM.
- 6 líneas de I/O.
- 32 registros.
- 2 Timers.
- 1 conversor ADC de 4 canales con 10Bits.
- Watchdog interno.
- Rango de operación de 1.8 a 5.5 [volts].

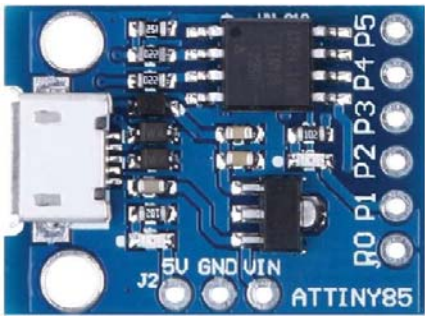

Figura 3-10: Digispark Attiny 85 (fuente: https://gotgadgets.com.au).

En la Figura 3-10 se puede observar el aspecto físico del componente, el cual fue ideal su utilización debido a que además de accesible y muy fácil de utilizar, su tamaño es realmente pequeño. La conexión del circuito está representada en la Figura 3-11 por la plataforma fritzing:

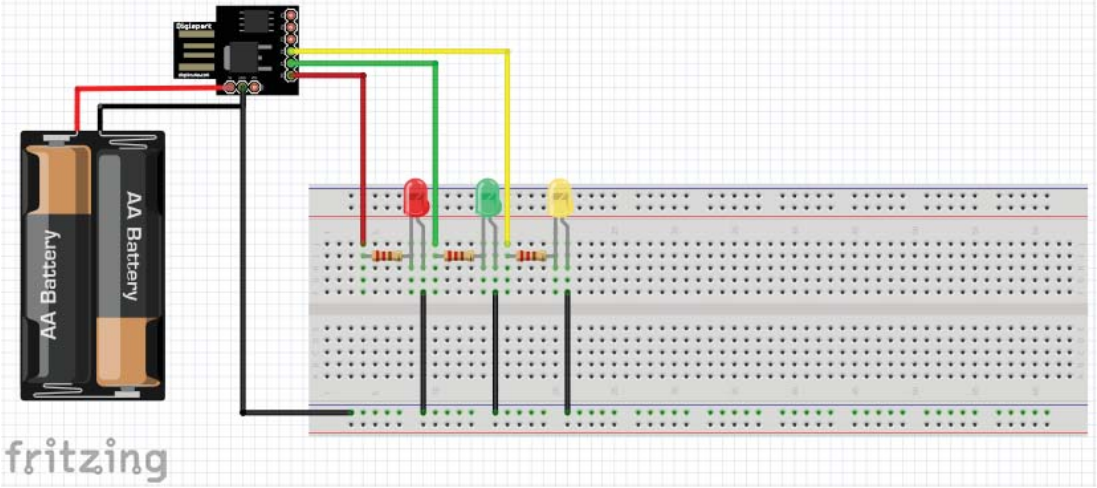

Figura 3-11: Circuito esquemático semáforo.

Fritzing es una aplicación OpenSource (código abierto) el cual permite que los usuarios puedan diseñar, documentar, compartir y enseñar sus proyectos de manera segura, elegante y rápida. Este software posee muchos componentes electrónicos produciendo que el usuario pueda diseñar una gran cantidad de circuitos, además este puede diseñarse hasta construir circuitos impresos (placas PCBs) [48]. En la Figura 3-11 se puede observar el circuito diseñado para el semáforo, en ella existe una fuente de energía que permite la alimentación del dispositivo, tres diodos led que permiten la emulación de los colores del semáforo, tres resistencias de 220 [Ω] cada una, el cual acotan la corriente que pasan por los diodos, cables de conexión y un protoboard que permiten el prototipo del circuito. De la misma manera que las señales de tránsito, se diseñó un soporte que permite el ensamblado del semáforo.

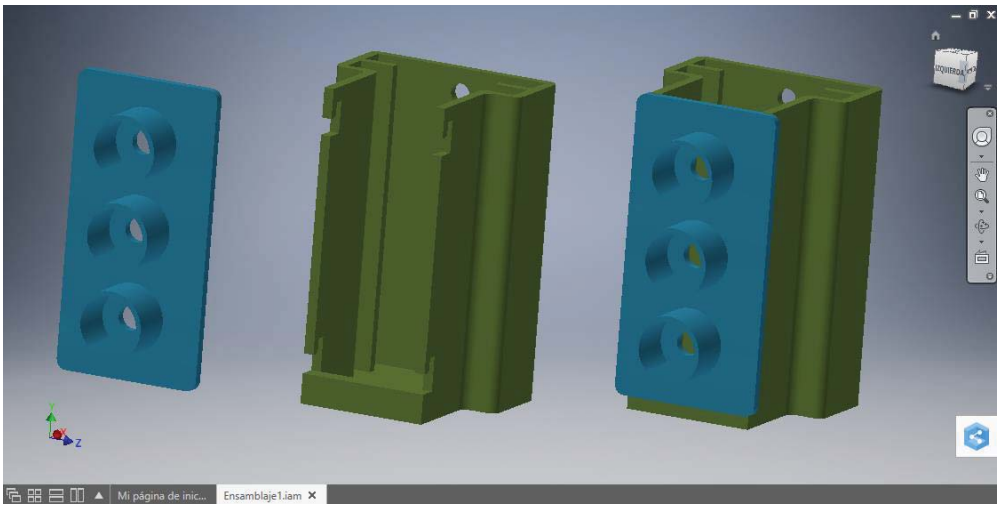

Figura 3-12: Semáforo diseñado en la plataforma Inventor.

En la Figura 3-12 se puede observar el diseño del semáforo, este cuenta con dos soportes donde el primero tiene la función de soportar los diodos led, y el segundo la batería con la placa de desarrollo. Este modelo se realizó por separado, debido a que al ingresar los componentes sea una manera más cómoda. Las dimensiones de este diseño fueron de 52x63x40 [mm^3] lo que lo hace un dispositivo muy compacto, en la Figura 3-13 se puede observar la pieza impresa y ensamblada.

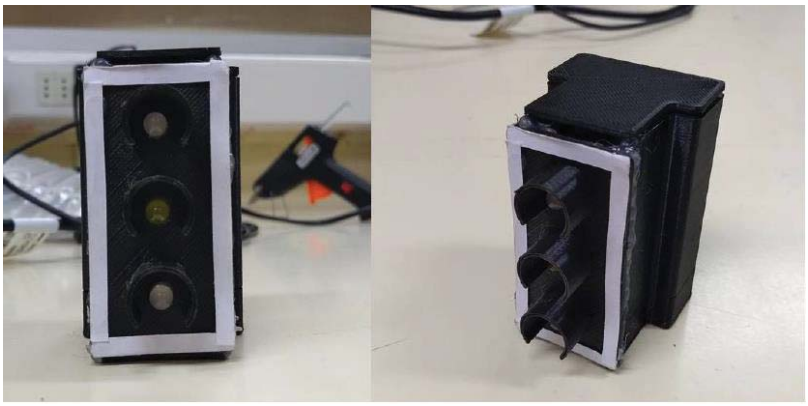

Figura 3-13: Semáforo impreso y ensamblado.

3.3 Construcción de clasificadores en cascada

Gracias a las herramientas que tiene la librería OpenCV, esta posee lo necesario para construir una cascada de clasificadores del tipo HAAR, LBP, entre otros. Cuando se necesita construir una cascada es necesario tener imágenes negativas y positivas. Las imágenes negativas pueden ser extraídas por cualquier base de datos en la que no estén involucradas las imágenes de interés. En las imágenes positivas hay que especificar la región de interés o mejor dicho el lugar donde se encuentra el elemento a detectar, con estos aspectos se construye un archivo vectorial que es básicamente la representación de la posición y el nombre de la imagen positiva.

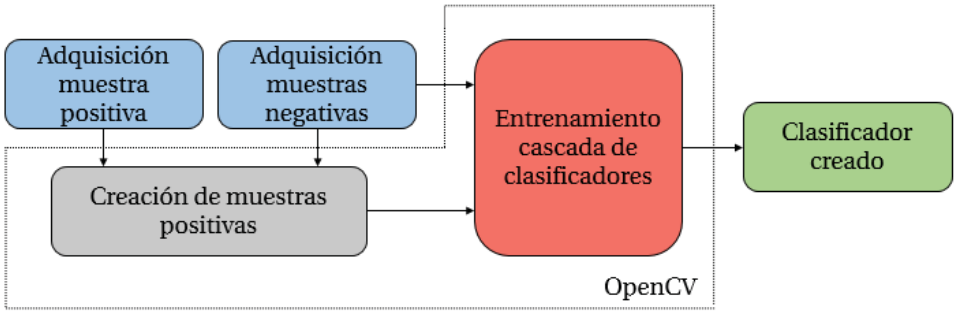

Figura 3-14: Diagrama de bloques Creación cascada de clasificadores.

En la Figura 3-14 se detalla en un diagrama de bloques la creación de una cascada de clasificadores. En primer lugar, se realiza la adquisición de solamente una muestra positiva, en este caso sería una señal de tránsito y junto con las muestras negativas se realiza una unión entre ellas, produciendo nuevas imágenes positivas. Este sistema es muy favorable al momento de no tener una base de datos donde estén las imágenes positivas, ya que por lo general es muy difícil

de conseguir. Por último, estas imágenes son llevadas a una etapa de entrenamiento que permite la realización de la cascada.

3.3.1 Creación de muestras positivas

Existen varias bases de datos que contienen imágenes relacionadas a la señal que uno quiere clasificar, con el fin de poder realizar un buen entrenamiento, pero existen señales que nunca varían, por ejemplo, la señal de transito Stop, el cual es la misma para todos los lugares en el planeta, por lo que con un mismo modelo bastaría para poder discriminarlas unas de otras. Cuando se quiere clasificar una imagen en específico (por ejemplo, una señal ceda el paso) no existe o es muy difícil de conseguir una base de datos que posea esta imagen, por lo que se deben buscar de forma manual, este paso puede ser un poco tedioso y a su vez imposible, por lo que es necesario buscar una herramienta que permita crear estas imágenes. El comando *opencv_createsamples* de la librería OpenCV, crea muestras positivas utilizando las imágenes negativas, esta imagen se superpondrá en las muestras negativas creando nuevas muestras, por lo que este comando necesita como entrada las imágenes negativas y la imagen positiva a detectar.

Figura 3-15: Señal Aeropuerto.

La Figura 3-15 pertenece a la de un Aeropuerto el cual será nuestra muestra a clasificar, por lo general esta imagen debe ser pequeña, en este caso es de 44x44 pixeles. Ahora es necesario cientos de imágenes negativas, cuantas más imágenes mejor será el clasificador, pero siempre con el cuidado de no sobre entrenar el sistema. Para un buen entrenamiento, las imágenes negativas fueron extraídas de la cámara del robot en su entorno de trabajo.

Figura 3-16: Imágenes negativas.

La Figura 3-16 muestra las imágenes negativas adquiridas, siempre es importante que estas imágenes no tengan a la imagen positiva dentro, ya que producirá confusión y mal entrenamiento. Estas imágenes fueron alrededor de 1400, lo cual es un valor aceptable, con más imágenes negativas, menos se confundirá el clasificador. Para la creación de muestras positivas se debe crear una carpeta en el que se almacenen todos los archivos a utilizar, en esta deben estar los siguientes elementos:

- Carpeta de imágenes negativas (“negative”)
- Carpeta vacía para almacenar las imágenes positivas (“info”)
- Imagen positiva (“Aeropuerto.jpg”)
- Archivo txt donde estén la dirección y el nombre de las imágenes negativa (“bg.txt”). Por Ejemplo: “negative/1.bmp, negative/2.bmp, negative/3.bmp, etc.”)

Teniendo los elementos en ese orden, se realizarán la adquisición de muestras positivas. En la ventana de comandos y en la dirección de la carpeta, se ingresa el siguiente código:

```
opencv_createsamples -img Aeropuerto.jpg -bg bg.txt -info info/info.lst -
pngoutput info -maxxagle 0.5 -maxyangle 0.5 -maxzangle 0.5 -num 1242
```

Este comando se le ingresa la imagen positiva, la dirección de las imágenes negativas, el número de imágenes negativas y la desviación de intensidad máxima de píxeles en muestras del plano x, y, z. Este último es importante debido a que es recomendable tener la imagen en distintas posiciones y orientaciones de tal manera de obtener una mejor clasificación. Ahora al ejecutar el comando, en la carpeta “info” se crearán las imágenes positivas producto de las imágenes negativas y una única imagen positiva.

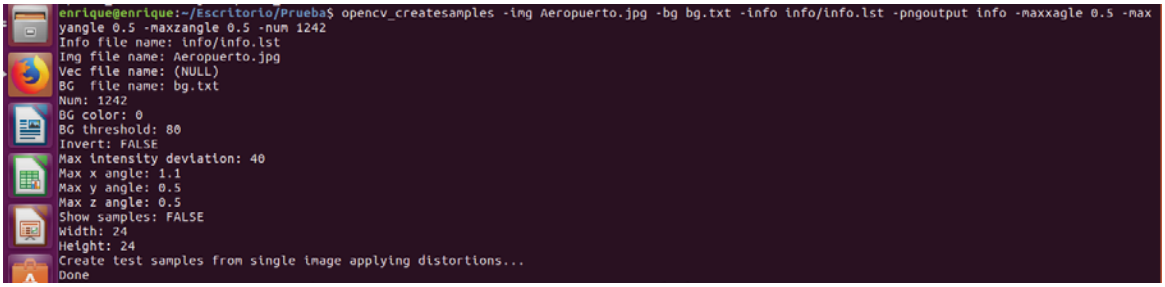

Figura 3-17: Creación de imágenes positivas en ventana de comandos.

En la Figura 3-17 se puede observar una representación de la ejecución del código, en ella se aprecia todos los elementos dichos anteriormente. A continuación, en la Figura 3-18 se presenta las imágenes positivas en producto de las imágenes negativas, en ellas se puede ver claramente la señal Aeropuerto por lo que ya no hay que preocuparse de la búsqueda de las imágenes positivas.

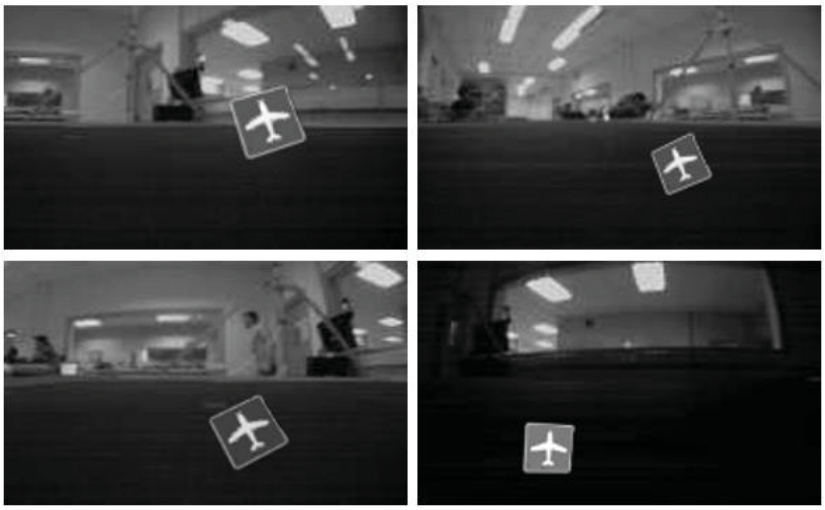

Figura 3-18: Imágenes positivas.

Además de la creación de muestras positivas, el comando `opencv_createsamples` crea un archivo de texto donde está la posición de la figura del Aeropuerto:

Tabla 3-2: Posición de la señal Aeropuerto.

Imagen	Cantidad de elementos	Posición			
1	1	109	30	43	43
2	1	33	21	70	70
3	1	92	6	71	71
4	1	37	28	37	37
5	1	72	49	47	47
6	1	24	56	43	43
7	1	77	32	37	37
8	1	110	31	69	69
9	1	58	26	73	73
10	1	92	31	57	57
11	1	91	14	66	66
12	1	88	35	65	65
13	1	106	30	39	39
14	1	72	54	46	46
15	1	74	18	70	70
⋮	⋮	⋮	⋮	⋮	⋮

Como se puede observar en la Tabla 3-2, este archivo además de entregar el nombre de la figura, esta entrega la posición en donde está el elemento a analizar (Aeropuerto) y la cantidad de ellos, lo cual es muy importante al momento de realizar el entrenamiento.

3.3.2 Entrenamiento

Teniendo las imágenes negativas y positivas es posible realizar el entrenamiento, por comodidades de OpenCV, es necesario crear un archivo vectorial, que es básicamente donde se unen todas las imágenes positivas en un mismo elemento, para realizarlo se ejecutó el siguiente comando:

```
opencv_createsamples -info info/info.lst -num 1242 -w 48 -h 48 -vec
positives.vec
```

Al tener el archivo vectorial ahora solo es necesario entrenar la cascada.

```
opencv_traincascade -data data -vec positives.vec -bg bg.txt -numPos 1000
-numNeg 1242 -numStages 5 -w 48 -h 48
```

En este caso se utilizó una muestra de 1000 imágenes positivas, 1242 imágenes negativas, cascada tipo HAAR y 5 etapas, cuantas más etapas mejor, pero el tiempo de entrenamiento crece exponencialmente, por lo que en esta parte fue necesario utilizar un servidor, el cual fue facilitado por la Escuela de Ingeniería Eléctrica.

Figura 3-19: Entrenamiento de la cascada.

En la Figura 3-19 se puede observar el comienzo del entrenamiento, en el que consta del estado cero hasta el estado cinco. Este proceso dura horas y horas el cual realiza todo el procesamiento de datos, al terminar esta entrega como salida un archivo XML que representa el clasificador en cascada. Todos los pasos anteriores fueron apoyados por el manual en la referencia [49].

3.4 Sistema de detección de objetos

Tanto en un entorno simulado como real, para la detección de objetos es necesario realizar un sistema de comunicación para poder realizar un procesamiento de la imagen desde la cámara del robot. Por lo tanto, este tipo de comunicación varía dependiendo del entorno de trabajo.

3.4.1 Entorno simulado

Para el procesamiento de imágenes, se utilizó la librería de visión artificial OpenCV, por defecto, el simulador V-REP trabaja con lenguaje LUA y para poder interactuar con Python se debe interaccionar desde una API remota. En otras palabras, realizando este tipo de comunicación, cualquier script externo podrá conectarse al puerto donde se ejecuta V-REP.

Listado 3-1: Script de comunicación en V-REP.

```
40 rightMotor=simGetObjectHandle('K4_Right_Motor')
41 leftMotor=simGetObjectHandle('K4_Left_Motor')
42 simSetJointTargetVelocity(leftMotor,0)
43 simSetJointTargetVelocity(rightMotor,0)
44 -- Put some initialization code here:
45 simSetThreadSwitchTiming(2)
46 simExtRemoteApiStart(19999)
47 -- Here we execute the regular thread code:
48 res,err=xpcall(threadFunction,function(err) return
   debug.traceback(err) end)
49 if not res then
50         simAddStatusbarMessage('Lua runtime error: '..err)
51 end
```

En el Listado 3-1 se muestra el script dentro de la plataforma de V-REP, donde en la línea 46 es cargada la comunicación para que el programa externo sea conectado. En el comando *simExtRemoteApiStart* se debe ingresar el puerto, el cual el programa externo será conectado, en este caso fue elegido el 19999. Por el otro lado, el lenguaje de programación elegido para el procesamiento de imagen es Python, ya que, en la literatura, se informa que muchos programas de visión artificial están ligados a él, además que este lenguaje es cómodo y popular. El lenguaje Python se ejecutó en el programa Spyder, en el que se obtuvo por medio de Anaconda, esta es una distribución libre y abierta de los lenguajes Python y R, utilizada en ciencia de datos y aprendizaje automático [50].

Listado 3-2: Script de comunicación en Python.

```
57 salir=0
58 v=0
59 vrep.simxFinish(-1) #Terminar todas las conexiones
60 clientID=vrep.simxStart('127.0.0.1',19999,True,True,5000,5) #Iniciar
   una nueva conexion en el puerto 19999 (direccion por defecto)
61 if clientID!=-1:
62     print ('Conexion establecida')
63 else:
64     sys.exit("Error: no se puede conectar") #Terminar este script
65 _, robot=vrep.simxGetObjectHandle(clientID, 'Khepera_IV',
   vrep.simx_opmode_oneshot_wait)
66 #Guardar la referencia de los motores
67 _, left_motor_handle=vrep.simxGetObjectHandle(clientID,
   'K4_Left_Motor', vrep.simx_opmode_oneshot_wait)
68 _, right_motor_handle=vrep.simxGetObjectHandle(clientID,
   'K4_Right_Motor', vrep.simx_opmode_oneshot_wait)
```

Dentro de ese programa, se debe ejecutar el Script que permite realizar la comunicación entre V-REP y Python. Tal como se muestra en el Listado 3-2, se observa que en la línea 60 se realiza la conexión, ya que se refleja que esta es conectada al mismo puerto (19999). En resumen, el script

permitirá el acceso directo de todas las funcionalidades del V-REP, lo que generará un completo control del robot y sus accesorios.

Lo más importante en la comunicación, es el envío de la información de la cámara del robot, la cual, al llegar a Python, con la ayuda de OpenCV se realizará todo el procesamiento de la cámara. Teniendo en cuenta que el objetivo es que el robot realice un proceso inteligente, como respuesta se debe tener las velocidades de los motores derecho e izquierdo, por lo que de la misma manera estás deben ser transmitidas al robot en V-REP.

3.4.2 Entorno real

Utilizando el robot real para el procesamiento de información, a diferencia del entorno simulado se debieron tomar ciertas medidas para que el sistema funcione en tiempo real.

Adquisición de imagen robot

Uno de los principales problemas en el robot, fue la no posibilidad del procesamiento de imagen dentro de él, ya que por muchas maneras que se intentó, no fue posible la instalación de una librería de visión artificial. Por lo tanto, la única solución encontrada, fue el envío de la imagen de la cámara.

Figura 3-20: Cámara Robot Khepera IV.

En la Figura 3-20 se muestra la vista frontal del robot, el driver que utiliza esta cámara es el *mt9v032*, el cual es un sensor de imagen digital CMOS de formato activo VGA de 1/3 pulgadas de ancho con operación de obturador global y alto rango dinámico (HDR). Este sensor VGA presenta una revolucionaria tecnología de imagen en ON semiconductor, que logra calidad de imagen CCD (basada en la relación señal-ruido y baja sensibilidad a la luz) mientras mantiene las ventajas inherentes de tamaño, costo e integración de CMOS. La matriz de pixeles de imagen activas es de 752x480 (Ancho por Alto). Incorpora sofisticadas funciones para el mejoramiento de la sensibilidad cuando se trabaja con resoluciones más pequeñas, además puede operar en modo determinado o programarse para tamaño de cuadro. Este sensor también presenta una salida serial de la señalización de bajo voltaje (LVDS), a su vez puede operar en una cámara estereoscópica, lo que puede fusionar lo datos de sí mismo [51].

Transmisión de imagen

Por defecto, el robot tiene una aplicación llamada MJPG-streamer, el cual permite adquirir y transmitir la información que captura la cámara. MJPG-streamer es una aplicación de línea de comandos que copia fragmentos de JPG de un único plugin de entrada a múltiples plugin de salida. Se puede usar para transmitir archivos JPEG a través de una red IP desde la cámara web a un visor como Chrome, Firefox, Camboloza, Videolancliente o incluso a un dispositivo Windows Mobile que ejecute el TCPMP-Player. Fue escrito para dispositivos integrados con recursos muy limitados en términos de RAM y CPU. Su origen (uvc_streamer) fue escrito por que las cámaras compatibles con Linux-UCV producen directamente datos JPEG, lo que permiten transmisiones M-JPEG rápidas y perfomistas incluso desde un dispositivo integrado que ejecuta OpenWRT. El módulo de entrada “input_uvc.so” captura tales cuadros JPG de una cámara web conectada [52].

Para ejecutar la transmisión de la cámara del robot se debe ejecutar el siguiente comando:

```
mjpg_streamer -i "input_uvc.so –yuv –f 15" –o "output_http.so -w
/usr/local/mjpg-streamer/www"
```

La aplicación MJPG-streamer permite modificar algunos aspectos de la información de la cámara a transmitir, entre ellas están:

- -d <device>: El dispositivo de video a usar (-d, /dev/video0).
- -r <resolution>: La resolución (-r 640x480, para 640x480 pixeles).
- -f <framerate>: La velocidad de fotogramas en frames por segundo (-f 10, por 10 fps).
- -q <quality>: La calidad de la compresión JPG a usar (-q 85, para compresión hasta 85%).
- -y: Usar si la cámara no admite MJPG, las imágenes se capturarán en YUVY.

A su vez, la aplicación MJPG-streamer posee un menú web donde pueden configurar varios parámetros.

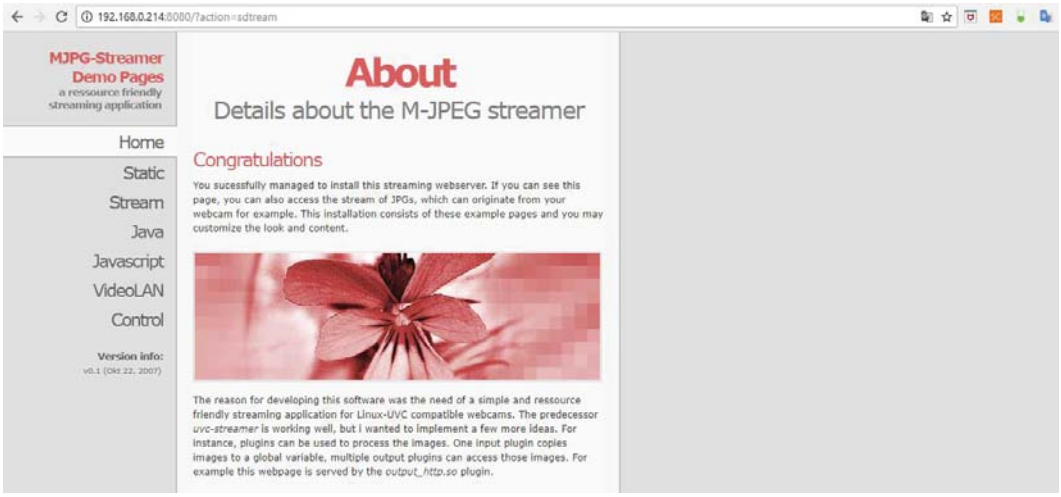

Figura 3-21: Menú MJPG-streamer en un navegador web.

En la Figura 3-21 se visualiza el menú web de la aplicación mencionada, el cual posee distintas características.

- Static: Muestra la imagen del robot estática.
- Stream: Muestra la imagen del robot en secuencia.
- Java: Muestra la transmisión mediante el uso de un applet de Java.
- Javascript: Muestra la secuencia de la imagen usando javascript.
- VideoLAN: Muestra la imagen utilizando una dirección URL definido.
- Control: Permite definir parámetros tales como: brillo, contraste, efectos de color y calidad de la imagen JPEG para la transmisión. En la Figura 3-22 se puede observar el menú de control.

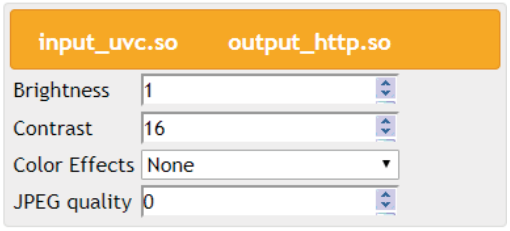

Figura 3-22: Menú de control MJPG-streamer en un navegador web.

Dentro del robot, al ejecutar el comando mencionado, se comienza a transmitir la información de la cámara, que implica que con cualquier computador conectado a la misma red, es posible visualizar la imagen.

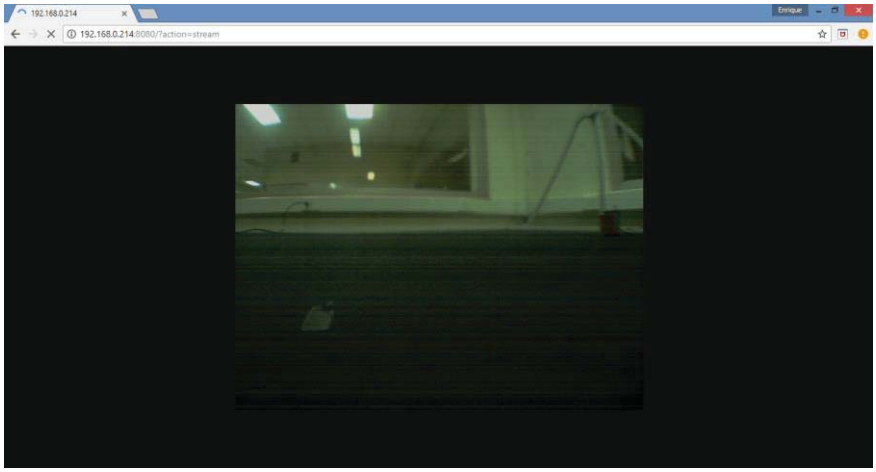

Figura 3-23: Imagen del robot capturada mediante un computador externo.

En la Figura 3-23 se puede observar la imagen recibida por un navegador web, que en este caso fue Google Chrome, además con el comando *ROBOT:8080/?action=stream* se puede ingresar a la imagen, donde *ROBOT* es la dirección IP del robot. De la misma manera que la manera simulada, se utiliza el lenguaje de programación Python para el procesamiento de imagen, aunque ahora este debe conectarse en la cámara real. En el Listado 3-3, se observa un segmento del código, el cual, la línea 31 presenta la dirección de la imagen de la cámara y desde la línea 180 a la 185, se realiza un pequeño procesamiento para que esta sea leída como frame. El motivo para que la información de la cámara sea leída como frame es porque al ser transmitida como stream, se percató de que existía un gran retraso, lo que producía que el sistema fuese ineficiente para trabajarlo en tiempo real.

Listado 3-3: Script de comunicación con robot real.

```
 31 url="http://192.168.0.106:8080/?action=snapshot"
 32 servidor ="192.168.0.106"
 33 puerto = 8888
 34 cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
 35 cliente.connect((servidor, puerto))
180 imgResp=urllib.request.urlopen(url)
181 imgNp=np.array(bytearray(imgResp.read()),dtype=np.uint8)
182 frame=cv2.imdecode(imgNp,-1)
183 resolution = frame.shape
184 scaledsize=resolution
185 frame = np.array(frame, dtype = np.uint8)
186 frame.resize([resolution[0], resolution[1], 3])
```

Por último, mediante una comunicación socket TCP/IP, se realiza él envío de las velocidades de los motores al robot, esta operación es similar a la realizada en el entorno de simulación, ya que al realizar el cálculo de las velocidades en el lenguaje Python, estas tienen que ser enviadas al robot por una comunicación Api Remota.

3.5 Sistema propuesto

Finalmente, debido a todas las pruebas realizadas, contando las técnicas utilizadas, se realizó un esquema que muestra desde la conexión del robot a un servidor externo, hasta el envío de velocidades por un proceso anterior de procesamiento de imagen.

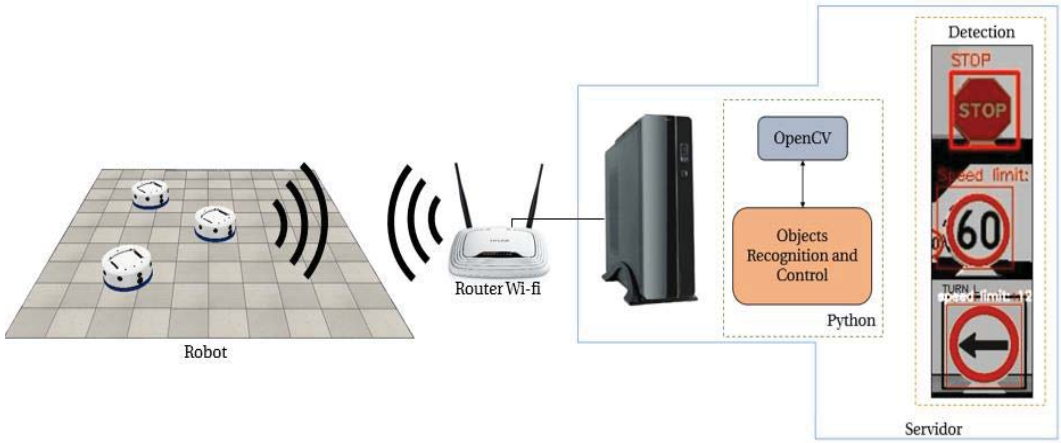

Figura 3-24: Esquema conexión Robot-Servidor.

En la Figura 3-24 se muestra el esquema dicho anteriormente, en primera instancia el robot envía la imagen que obtiene la cámara mediante una conexión TCP/IP, la cual es recibida por un servidor externo. Este servidor procesa toda la información, es decir, realiza todo el trabajo de procesamiento de la imagen, además de tomar todas las decisiones, queriendo decir que las velocidades de las ruedas se asignan en ese lugar. Por último, estas velocidades son enviadas de la misma manera pasando por el router hasta llegar al robot, para que estas sean ejecutadas. Por ejemplo, si el robot envía una imagen en donde haya una señal de Stop, el clasificador detectará esa señal produciendo que los motores de las ruedas se les asignara un valor de cero, y esas serán enviadas al robot; De igual manera que una señal de giro a la derecha, ya que se realizará todo el

procesamiento anterior y las velocidades que recibirá el robot será una mayor que la otra. Todo el procedimiento de este sistema puede ser visto por el siguiente diagrama de bloques:

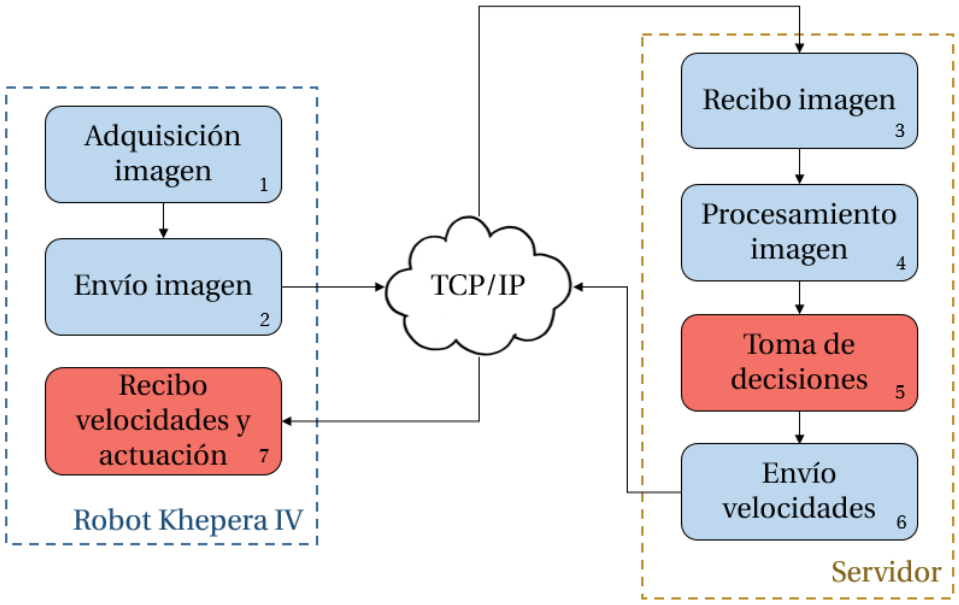

Figura 3-25: Diagrama de bloques conexión Robot-Servidor.

La Figura 3-25 muestra un diagrama de bloques representando al esquema de la Figura 3-24. En la Figura 3-26 se realizó un modelo que representa un análisis más profundo de cada bloque, el cual se explica a continuación:

- Adquisición y envío de imagen: Utilizando la aplicación MJPG-streamer, el robot recibirá la imagen de la cámara y la aplicación misma la subirá a la red para que cualquier servidor la tome y la procese.
- Recibo imagen: El servidor ingresa a la dirección donde se está transmitiendo la información de la cámara y la guarda.
- Procesamiento imagen: Lee la imagen, la convierte en escala de grises ya que solo con una matriz es posible obtener las mismas características que con una imagen en RGB, además de ejecutar la cascada de clasificadores para la detección de señales.
- Toma de decisiones: Dependiendo de la señal encontrada, las velocidades de los motores cambiaran, por ejemplo, si es una señal Stop, las velocidades disminuirán a cero hasta un cierto tiempo, si es un giro, la velocidad de un motor será mayor que la otra, etc. Si no se detecta ninguna señal, se ejecutará el seguidor de pista, utilizando la técnica de enmascaramiento.
- Envío datos al robot: Dependiendo de las velocidades que tomen los motores, estas serán enviadas al robot.
- Recibo velocidades del robot: El robot al recibir las velocidades entregadas por el servidor, las ingresara a su sistema para que sean actualizadas en sus motores.

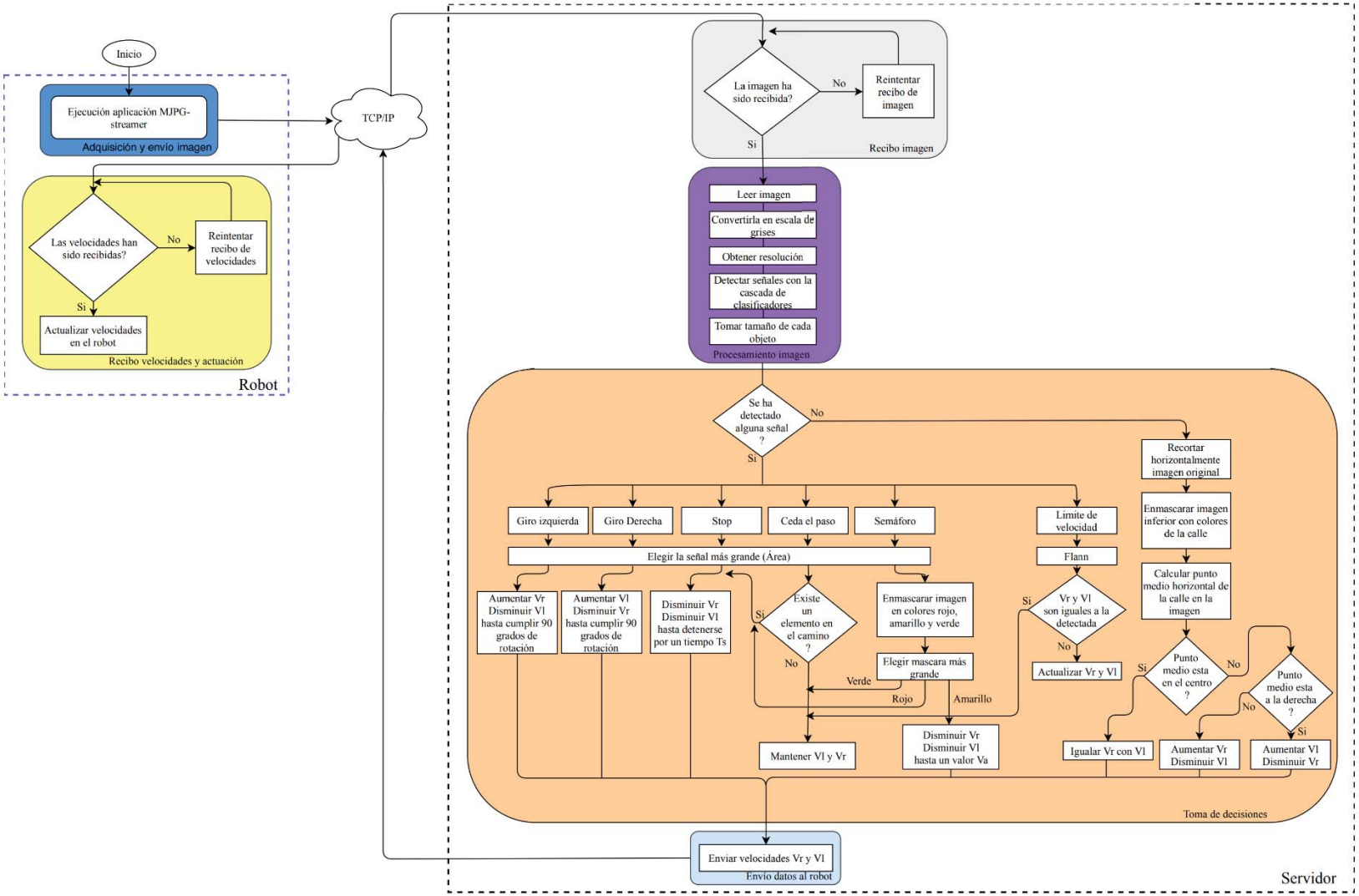

Figura 3-26: Diagrama de bloques profundo conexión Robot-Servidor.

4 Experimentación

En este capítulo, se presentan todas las pruebas que se realizaron para la detección de objetos robustos en robótica móvil por medio de visión artificial. Los resultados son representados desde la detección de señales básicas (detección de flechas de un mismo color) hasta señales más robustas, el cual están involucradas las pruebas de los clasificadores diseñados (presentados en el capítulo anterior), tanto como señales de tránsito y nuevos escenarios. Todo ello con el fin de tener resultados que reflejen la eficiencia de este nuevo algoritmo tanto simulado como real.

4.1 Detección de objetos simples

Para dar en marcha el algoritmo propuesto, se realizó una simulación de un escenario simple con la detección de señales de un mismo color.

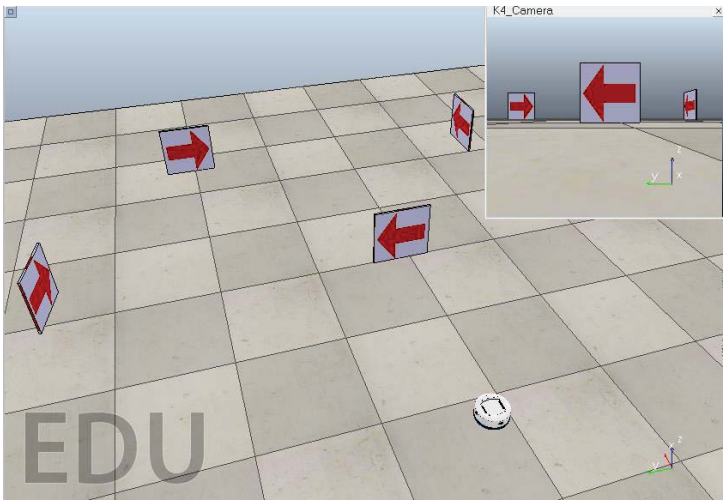

Figura 4-1: Escenario simulado detección de flechas.

En la Figura 4-1 se observa el robot posicionado detrás de un conjunto de flechas, como objetivo él debe realizar la lectura de estas flechas y determinar a qué dirección están apuntando para poder asignar una velocidad a los motores y poder girar, con el fin de realizar un movimiento inteligente. Como la señal es de un solo color, solamente utilizando la técnica de la umbralización es posible realizar la detección de la flecha. En el capítulo Marco teórico se presentaron diversos tipos de espacios de colores para procesar la imagen.

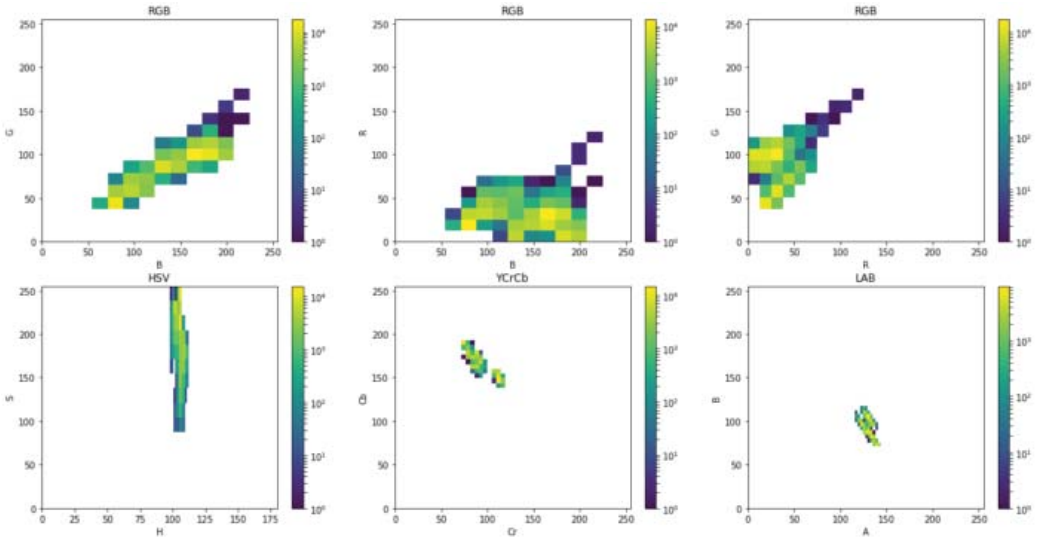

Figura 4-2: Diagrama de densidad de variación de iluminación en color azul (fuente: http://www.learnopencv.com).

Utilizando la misma imagen de los cubos rubiks y aplicando el diagrama de densidad de variación de iluminación en el color azul (Figura 4-2) se percató que el espacio de colores donde la respuesta es más acotado es el espacio HSV. Se puede analizar que el diagrama de densidad para el color RBG varía en demasía, ya que la variación de los canales es muy alta y la fijación de un umbral es un gran problema. La comparación entre YCrCb y LAB es parecida en el ámbito de la compacidad, pero de igual manera abarca un espacio en el plano donde se puede mezclar otro color. En HSV solo la componente H contiene el color absoluto para una imagen oscura y brillosa, por lo que solamente utilizando un parámetro se puede distinguir el color. Por conclusión, es recomendable utilizar el espacio de colores HSV, ya sea por su precisión e indiferencia a distintas iluminaciones.

Teniendo la imagen que detecta el robot en el programa, se realiza el enmascaramiento de color rojo, lo que permitirá solo tener las características de la flecha roja, tal como se muestra en la Figura 4-3.

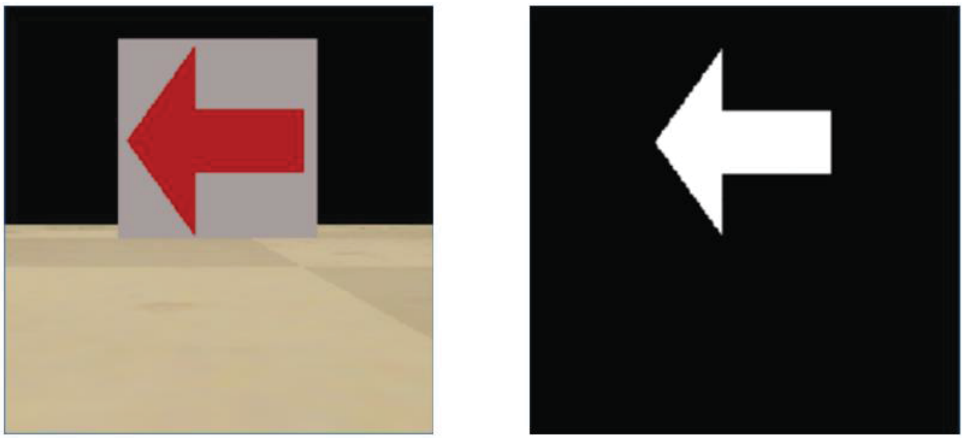

Figura 4-3: Señal capturada y enmascarada con el color rojo en experimentación simulada.

La imagen de la izquierda es la que visualiza la cámara del robot y la imagen de la derecha es la misma pero aplicada con un enmascaramiento de color rojo, es decir, se segmentó la imagen de tal manera que existiera un margen en donde solo el color rojo apareciera. En esa imagen puede haber varias flechas, por lo que se debe encontrar la flecha más grande, ya que se supone que esta será la más cercana al robot, por ende, tendrá más relevancia.

Figura 4-4: Ejemplo de técnicas de cálculo de orientación de un objeto (fuente: http://docs.opencv.org).

OpenCV brinda múltiples herramientas para conocer la orientación de un objeto, entre ellas están cv.minAreaRect, cv.fitEllipse y cv.fitLine (Figura 4-4), en este caso se utilizó la última mencionada. La función cv.fitLine entrega cuatro variables y con ellas es posible relacionarlas y conseguir dos variables llamadas lefty y righty. En el caso de la detección de flechas, si esta apunta a la izquierda el valor de lefty será positivo y righty será negativo, en cambio si la flecha apunta a la derecha, estos valores cambiaran su signo. Utilizando dicha función, se llevó a cabo la prueba simulada, obteniendo los siguientes resultados:

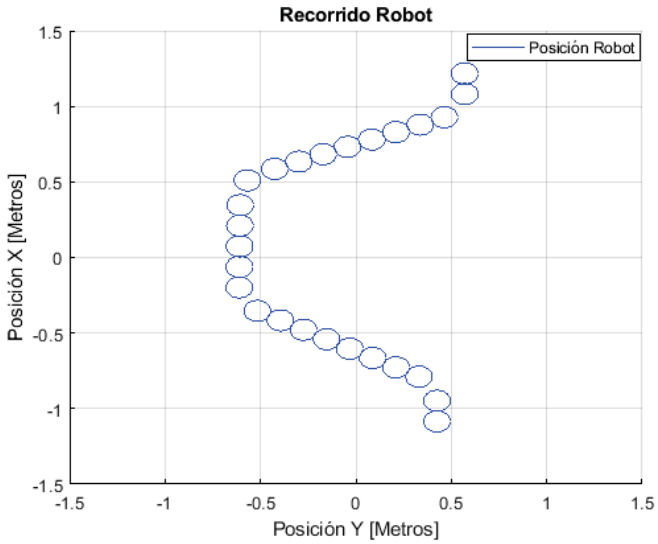

Figura 4-5: Recorrido del robot en detección de flechas en experimentación simulada.

En la Figura 4-5 se observa el recorrido del robot, en ella se visualiza que el robot está realizando un comportamiento autónomo, ya que se está detectando y evitando las señales de giro a la derecha e izquierda. Como se muestra en la Figura 4-6, se puede revelar los comportamientos de

los motores en la plataforma. En la primera etapa, la velocidad del motor derecho es mayor al del izquierdo, por lo que el robot realiza un giro hacia la izquierda, luego sus velocidades vuelven a ser las mismas. En la segunda etapa la velocidad del motor izquierdo es mayor, por lo que el robot gira a la derecha. La tercera etapa el robot vuelve a girar a la derecha y en la última etapa el robot gira hacia la izquierda. En conclusión, el robot detecta cuatro flechas y afirmativamente este las evita produciendo un algoritmo de detección y evitación de flechas.

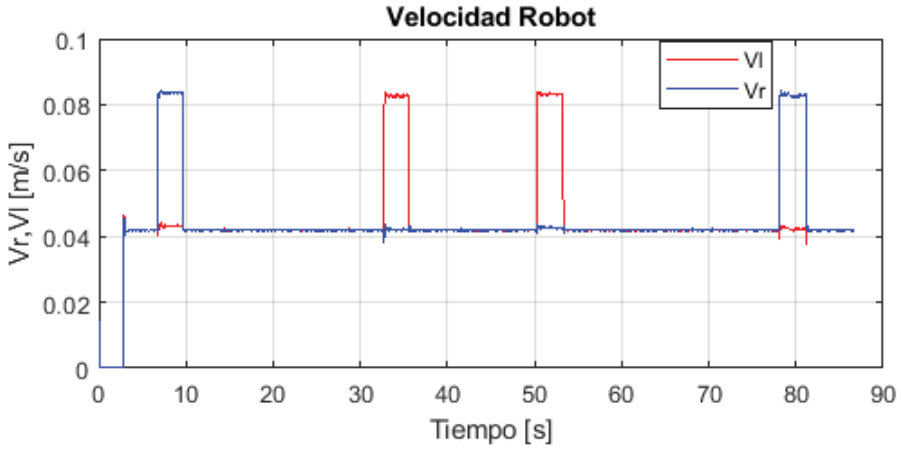

Figura 4-6: Velocidad del robot en detección de flechas en experimentación simulada.

Al igual que en la experiencia simulada, en la experiencia real también se realizó un experimento que involucra la detección de flechas.

Figura 4-7: Escenario real detección de flechas.

La Figura 4-7 muestra el robot junto a tres flechas de color rojo, las cuales están posicionadas a una distancia de 80 [cm] y con una orientación de 90 grados una respecto a las otras, como misión el robot debe ser capaz de detectar estas flechas y tomar una decisión al respecto. En la Figura 4-8 se muestra la visualización de la cámara del robot con la flecha de orientación a la derecha

(imagen izquierda) y su enmascaramiento del color rojo (imagen derecha). En la experiencia virtual, se utilizó la función cv.fitLine, pero en este caso la función al ser aplicada no entrego una buena respuesta, ya que esta no detecto con claridad la orientación de la flecha, por lo tanto se utilizó otra técnica según el siguiente criterio: Una flecha posee siete vértices, pero dos de ellos se sitúan en la parte superior e inferior de la imagen, por ende, dependiendo de la ubicación de esos puntos en el eje *x*, el robot sabrá la orientación de la flecha.

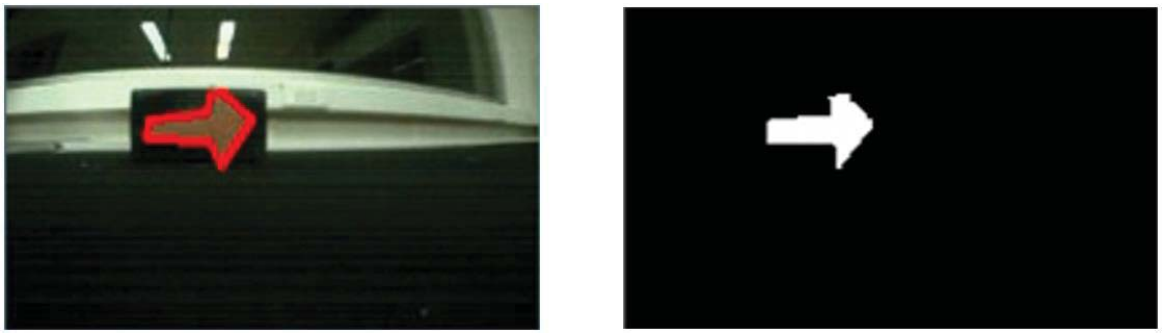

Figura 4-8: Señal capturada y enmascarada con el color rojo en experimentación real.

El robot al realizar la detección de flechas, se obtuvo un recorrido (mostrado en la Figura 4-9), en ella se puede apreciar las distintas posiciones que recorre mientras está detectando las flechas.

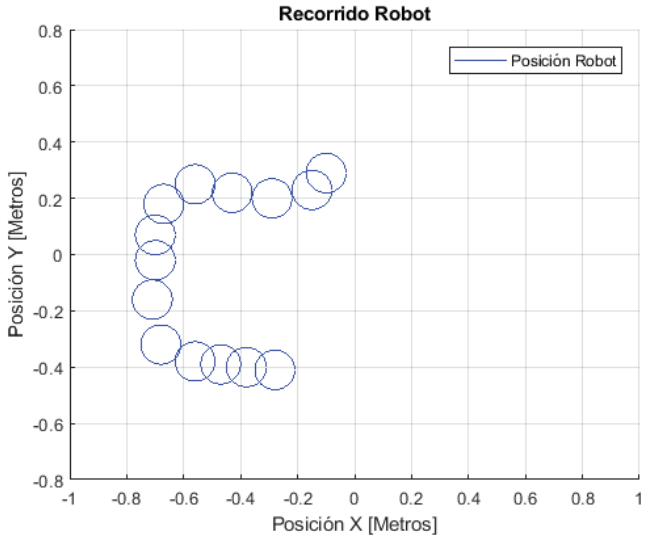

Figura 4-9: Recorrido del robot en detección de flechas en experimentación real.

En la Figura 4-10 se aprecia la orientación del robot con relación al tiempo que recorre, las pendientes más pronunciadas significan que el robot hizo un giro, en este caso fueron tres, por lo que afirmativamente el robot giró con respecto a las señaléticas encontradas.

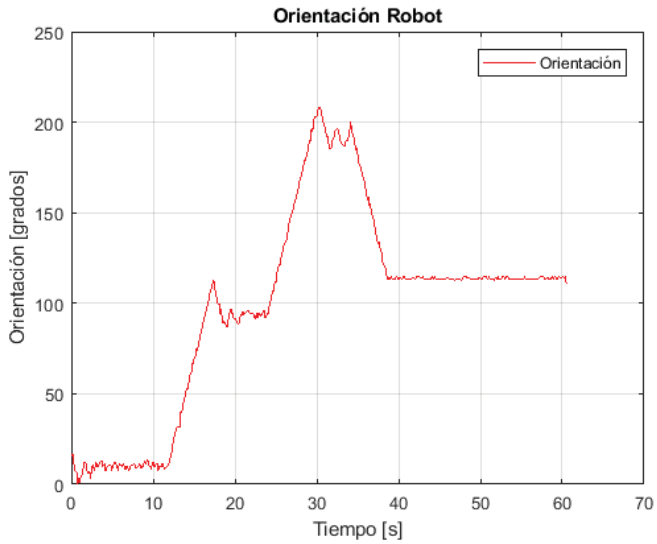

Figura 4-10: Orientación del robot en detección de flechas en experimentación real.

4.2 Prueba de clasificadores

Para la realización de detección de señales más robustas, como reconocimiento de señales de tránsito, se realizaron nueve cascadas, las cuales son Aeropuerto, flecha derecha, flecha izquierda, logo McDonald's, logo PUCV, límite de velocidad, stop, semáforo y ceda el paso.

Figura 4-11: Señales a clasificar.

En la Figura 4-11 solo se muestran algunas señales, cada una de estas fueron entrenadas para que fuesen reconocidas por el robot. Las cascadas fueron creadas de la misma manera que se realizó en el capítulo Preparación y desarrollo del sistema (Construcción de señales de tránsito), por lo que se utilizaron similares características, estas imágenes fueron especialmente elegidas debido a que son una de las más importantes en la vía pública. Ingresando estas cascadas en un código en Python y llamando a la figura anterior, se obtuvo la siguiente respuesta:

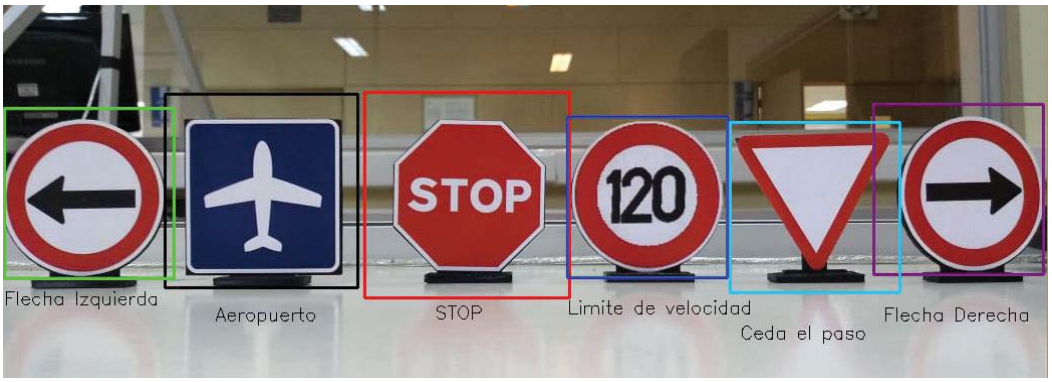

Figura 4-12: Señales detectadas.

Tal como se esperaría, en la Figura 4-12 se observa como las imágenes son detectadas, cabe destaca que el tamaño de los cuadros varía según la detección, ya que pertenece a toda la región donde el clasificador piensa que se encuentra la señal.

4.2.1 Detección semáforo

Para que el semáforo sea reconocido por el Robot, este tuvo que ser entrenado por OpenCV, lo cual se tomaron las siguientes imágenes:

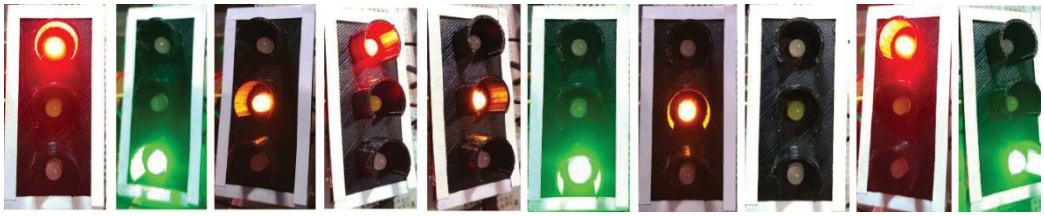

Figura 4-13: Posturas horizontales del semáforo.

La Figura 4-13 muestra todas las posturas horizontales posibles que se pueden obtener del semáforo, además estas se realizaron con sus luces encendidas, con el fin de que el semáforo sea detectado adecuadamente con el fin de mejorar el entrenamiento.

Figura 4-14: Detección semáforo.

Al ser entrenado de la misma manera que las señales de tránsito, la cascada se lleva al entorno de programación de Python para que el robot ahora pueda reconocer semáforos. En la Figura 4-14 se ilustra un ejemplo de la detección del semáforo, este es detectado con las luces verde, amarillo y rojo encendidos por lo que funciona adecuadamente. Al tener clasificado la señal es necesario saber qué color este encendido, por lo que se utiliza la misma técnica realizada en la prueba de Detección de objetos simples, es decir, utilizar la técnica de enmascaramiento. A diferencia de la

detección de flechas, en este caso se deben realizar tres mascaras que permitan distinguir los colores de las luces del semáforo, en la Tabla 4-1 se muestran los rangos de colores de cada luz:

Tabla 4-1: Valores parámetros HSV de los colores rojo, verde y amarillo.

	H	S	V
Rojos bajos 1	0	55	50
Rojos altos 1	35	255	255
Rojos bajos 2	150	60	50
Rojos altos 2	255	255	255
Verdes bajos	55	50	50
Verdes altos	100	255	255
Amarillos bajos	30	50	49
Amarillos altos	44	255	255

Teniendo estos parámetros en el sistema, se calcula el área de todas estas luces y dependiendo de cuál sea la más grande, esta tendrá la decisión para llevarla a sus motores. Si el área de la máscara de color rojo es mayor, las velocidades serán cero, si es verde las velocidades de los motores aumentaran de igual manera y si es amarillo las velocidades disminuirán.

4.3 Detección de señales

Poniendo en prueba la detección de señales de tránsito utilizando las cascadas de clasificadores, se realizaron pruebas tanto simulada como real.

4.3.1 Entorno simulado

La prueba simulada consta de un ejemplo clásico de un vehículo en una vía pública, en donde a un vehículo se le presentan distintas señales de tránsito.

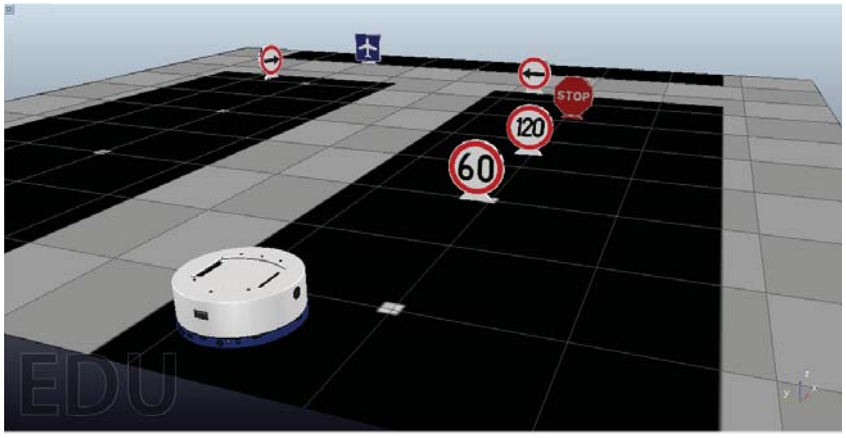

Figura 4-15: Detección de señales en experimentación simulada.

La Figura 4-15 muestra el escenario donde se ejecutará la simulación, la prueba consta que el robot debe detectar 6 señales de tránsito y estas deben ser obedecidas para que el robot llegue a su objetivo, que en este caso es la señal Aeropuerto.

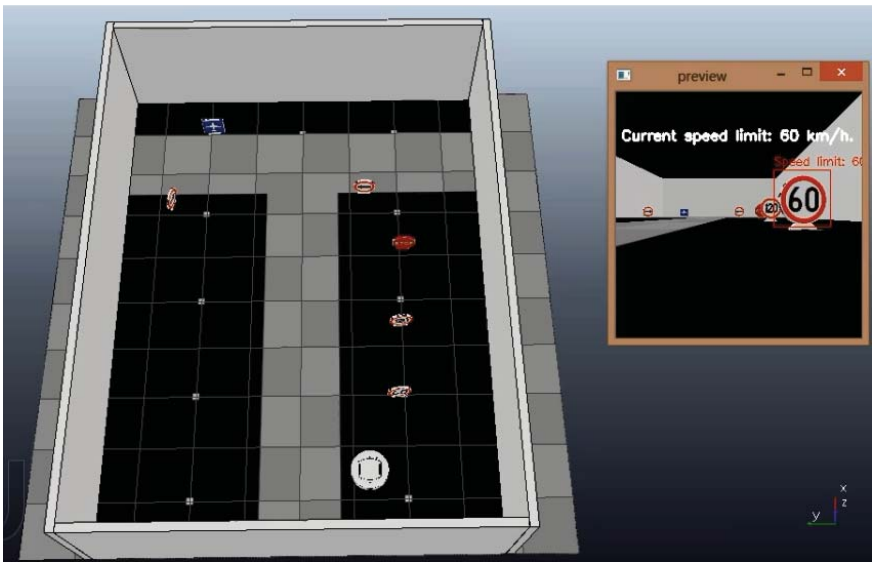

Figura 4-16: Inicio de prueba detección de señales en experimentación simulada.

Al ejecutar la simulación, el robot empezará a avanzar linealmente de tal manera que este recorra las señales, como se puede observar en la Figura 4-16, la imagen de la derecha muestra la vista de la cámara, la cual está detectando la señal límite de velocidad 60, por lo que su velocidad es actualizada, luego este debe detectar las señales que vienen más adelante produciendo el siguiente recorrido:

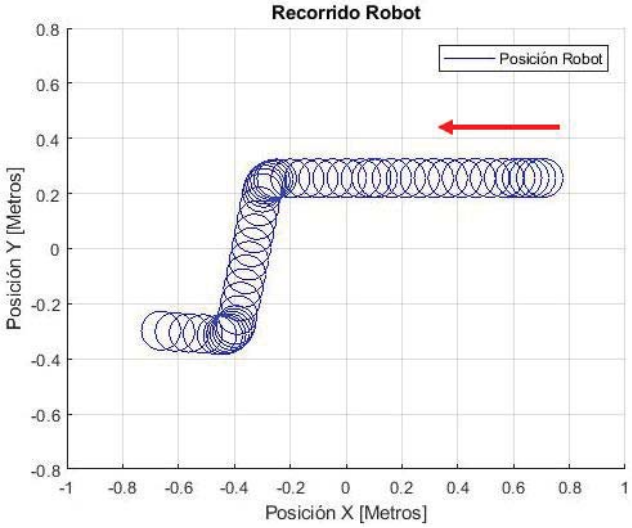

Figura 4-17: Recorrido robot en detección de señales en experimentación simulada.

En la Figura 4-17 se puede observar el recorrido del robot en sus coordenadas x e y, donde la flecha de color rojo representa el inicio y la dirección del robot. Este recorrido también puede ser representado por sus velocidades en el motor derecho e izquierdo.

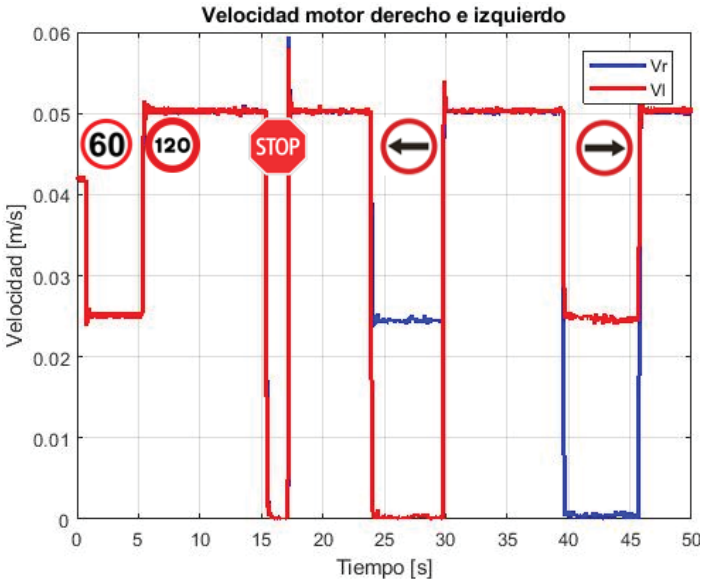

Figura 4-18: Velocidades del robot en la detección de señales en experimentación simulada.

Las velocidades de los motores del robot está apoyada por la Figura 4-18, en primer lugar el robot avanza de manera recta a una velocidad de 0,0418 $[m/s]$, luego al reconocer la señal 60, los motores bajan a una velocidad de 0,025 $[m/s]$ por 4,5 $[s]$, luego al reconocer la señal 120, su velocidad aumenta al doble, que es 0,05 $[m/s]$ por 9,7 $[s]$. Al seguir avanzando el robot se encuentra con la señal Stop, lo que sus velocidades disminuirán hasta que estas sean 0 $[m/s]$ con un tiempo de 2,15 $[s]$, al seguir avanzando, la señal de giro a la izquierda implicara que el robot gire a la izquierda produciendo que la rueda derecha tenga velocidad 0 $[m/s]$ y la rueda izquierda gire a una velocidad de 0,025 $[m/s]$ y todo en un tiempo de 5,85 $[s]$. Al pasar todo ese tiempo, el robot vuelve a su estado anterior y nuevamente se topa con otra señalética, que representa la señal de giro a la derecha en el que nuevamente el robot gira, llegando a la señal Aeropuerto y así completar el ciclo de la simulación.

4.3.2 Entorno real

En esta sección se presentarán algunos resultados acerca de la detección de señales tales como Logo PUCV, McDonald's y Aeropuerto; el cual habrá una serie de señales que le permitirán al robot poder crear una ruta y llegar a su objetivo.

Orientación Robot

La primera prueba que se realizó con el entorno real fue solucionar el problema de la orientación del robot, en simulación es más fácil tener un control sobre la posición y orientación del objeto, por lo que ahora se quiere realizar un nuevo control sobre ello. Para que este pueda tener control sobre su orientación, se propusieron las siguientes técnicas:

- Giroscopio: El robot tiene en su sistema un giroscopio que permite el control de su ángulo de orientación.

- SwisTrack: Este programa informa sobre la posición y ángulo del robot por medio de una cámara en la parte superior de la plataforma.
- Sensores de luz: Debajo del robot, trae a los costados sensores que permiten detectar el color de la pista, por lo que se puede realizar un seguidor de línea.
- Umbralización: Enmascarar la imagen el cual detecte la pista solamente utilizando procesamiento de imágenes desde la cámara del robot.

Tomando en cuenta todas esas técnicas, se prefirió utilizar la técnica de umbralización, aunque si se utilizaba el programa SwisTrack, podría ser una buena idea, pero el proyecto en sí es más atractivo con el uso de imágenes entregados por el robot, es decir, todo el procesamiento sea gracias a la cámara del robot.

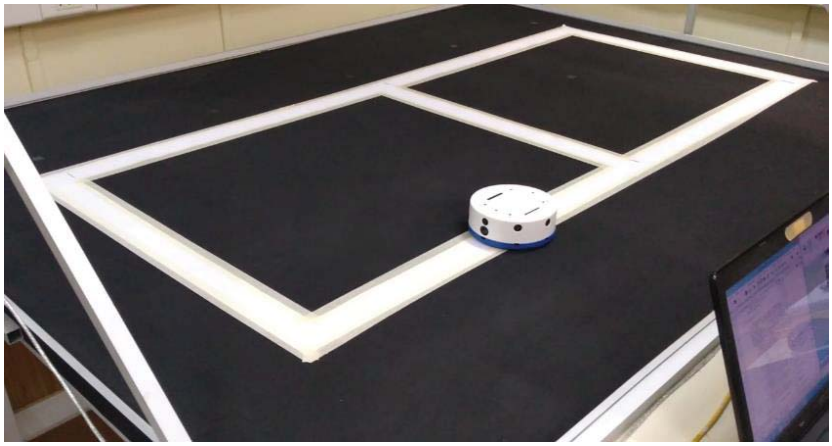

Figura 4-19: Plataforma del robot con pista blanca.

Para utilizar la técnica de la umbralización, en la plataforma se realizó un camino de color blanco con material cartulina y cinta de pegar, en la Figura 4-19 se aprecia un camino en la plataforma con forma del número 8. Como el color del camino es de color blanco, los valores de umbrales del color varían en [30, 0, 60] hasta [255, 255, 255] en código HSV. En la Figura 4-20 se muestra los pasos desde la toma de imagen hasta la toma de decisión de la orientación del robot.

Figura 4-20: Procedimiento detección de pista.

En (1) se observa la imagen que detecta la cámara del robot. En (2) la imagen es recortada ya que solo se necesita la información del camino, por lo que la parte superior de la imagen debe ser eliminada. En (3) actúa la técnica de la umbralización por lo que solo pasan los rangos de colores que se definen, en ella se calcula el promedio de ese color en el eje x, y en (4) se muestra con un punto rojo el centro del camino. A continuación, la Figura 4-21 representa todos los posibles casos para que el robot gire a la izquierda, gire a la derecha o posea una trayectoria recta.

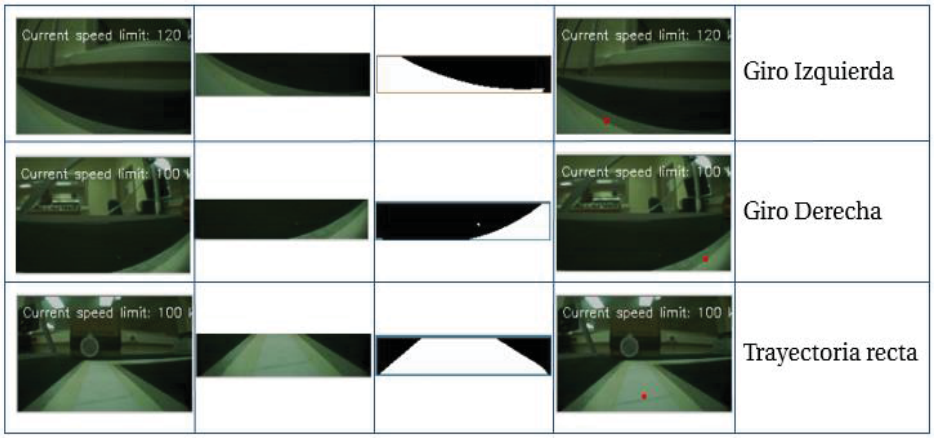

Figura 4-21: Casos posibles en la detección del camino.

Como se puede observar, la distinción del camino se realiza sin problema, por lo que la toma de decisiones será aceptable. Ahora teniendo un sistema de control de posición mediante la cámara, es posible realizar las pruebas reales.

Detección señal McDonald's

La primera prueba que se realizo fue la detección de la Señal McDonald's, la cual, junto con ella están involucradas 13 señales de tránsito. En la Figura 4-22 se aprecia el entorno donde se desplazará el robot, este deberá respetar todas las señales y seguir el camino (línea blanca) para que llegue a su objetivo. En primera instancia el robot deberá toparse con el orden de las siguientes señales: Stop, PUCV, giro derecha, giro izquierda, Stop, giro izquierda, giro derecha, giro derecha, limite velocidad 60, limite velocidad 120, giro izquierda y señal McDonald's.

Figura 4-22: Escenario detección señal McDonald's.

Si se sigue ese orden el robot llegará a su objetivo, en la Figura 4-23 se aprecia una gráfica que muestra el recorrido del robot.

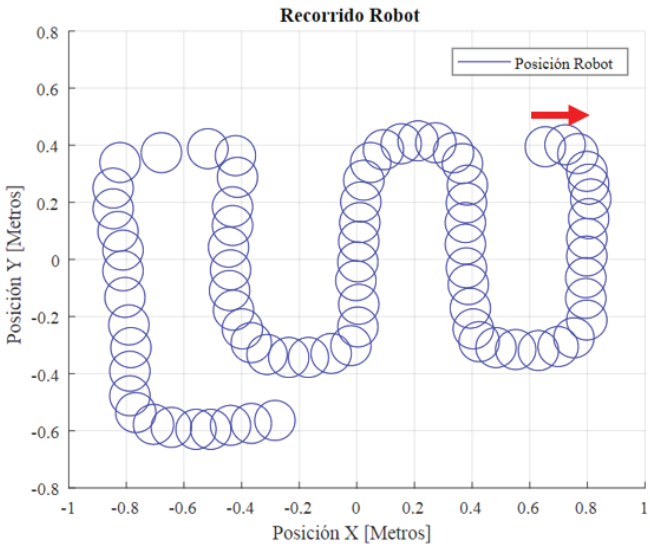

Figura 4-23: Recorrido del robot en detección señal McDonald's.

Desde la fecha roja, el robot comienza a avanzar, el cual el robot realiza una especie de zig-zag, lo que genera el recorrido pensado, por lo tanto, el algoritmo propuesto está funcionando adecuadamente, respetando todas las señaléticas de giro y detección de calle. Para saber si se respetan las señales de stop y límite de velocidad, se graficaron las velocidades de los dos motores.

La Figura 4-24 representa las velocidades de los motores en $[m/s]$, la línea azul representa la velocidad del motor derecho y la roja la velocidad del motor izquierdo. En los primeros 195 $[s]$ los motores toman velocidades entre 0 y 0,02 $[m/s]$ ya que su velocidad máxima es el equivalente de 100 $[km/hr]$, además dependiendo de las señales que se vayan encontrando y su posición en la ruta, las velocidades van subiendo o bajando. Luego se topa con la señal límite de velocidad 60, por lo que su velocidad máxima disminuye a 0,014 $[m/s]$, pero eso solo dura un tiempo aproximado de 10 $[s]$, para luego ser detectado por la señal límite de velocidad 120, por lo que su velocidad máxima aumenta a 0,027 $[m/s]$. El tiempo final de todo este trayecto fue de 300 $[s]$.

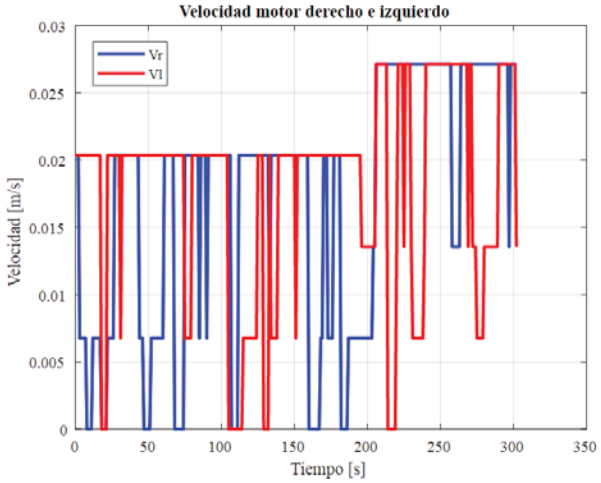

Figura 4-24: Velocidades de los motores del robot en detección señal McDonald's.

Detección señal Aeropuerto

La segunda prueba fue que el robot debería llegar a la señal Aeropuerto, en este caso existen 7 señales de tránsito que se ilustran en la Figura 4-25.

Figura 4-25: Escenario detección señal Aeropuerto.

En primera instancia, el robot deberá girar a la derecha, luego detectar el logo PUCV, después realizar un giro a la derecha, luego detenerse por dos señales Stop consecutivos y finalizando con la detección de la señal Aeropuerto.

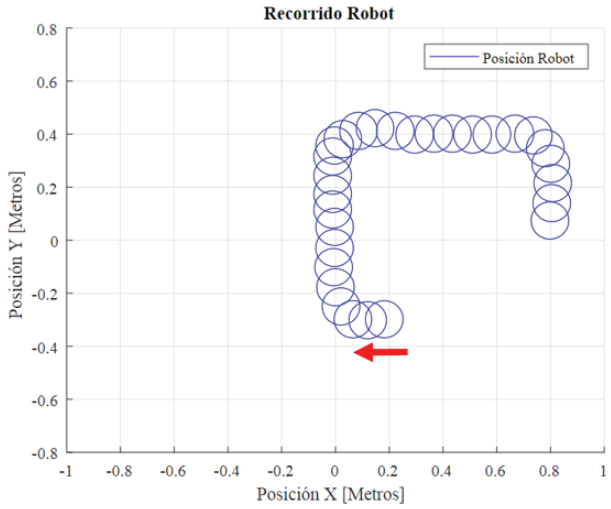

Figura 4-26: Recorrido del robot en detección señal Aeropuerto.

En la Figura 4-26, se muestra la gráfica del recorrido del robot, como se presenta, el robot realiza un movimiento autónomo lo que genera que el recorrido del robot sea lo esperado, ya que cumple con todas las detecciones de las señales, a pesar de que su trayecto no es totalmente fino, el robot es capaz de poder seguir la pista sin ningún inconveniente. A continuación, la Figura 4-27 muestra el recorrido del robot.

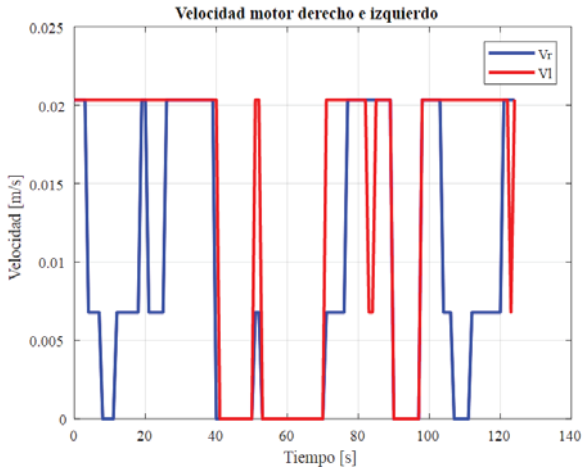

Figura 4-27: Velocidades de los motores del robot en detección señal Aeropuerto.

Como en este caso no existen límite de velocidades, la velocidad máxima será de 0,02 $[m/s]$. En los primeros 35 $[s]$, la velocidad del motor derecho es menor a la del izquierdo debido a que el robot está realizando un giro a la derecha, además de estabilizarse al centro de la pista. Luego se mantiene en línea recta hasta los 40 $[s]$, ya que empieza a realizar el otro giro a la derecha, sin embargo, ese proceso se ve interrumpido debido a que la señal Stop detiene los motores, eso dura aproximadamente 30 $[s]$ ya que reconoció dos veces la señal en ocasiones distintas. Entre los 70 y 75 $[s]$ el robot termina su giro, para luego seguir avanzando y toparse con otra señal Stop. Finalizando, el robot vuelve a hacer un giro, pero esta vez porque la pista se curva hacia la derecha, por lo que la velocidad del motor derecho es menor a la del izquierdo, hasta llegar a la señal Aeropuerto.

Detección señal PUCV

Finalizando, la última prueba se basó en la detección de la señal PUCV.

Figura 4-28: Escenario detección señal PUCV.

En este caso el robot debe detectar 8 señales, primero pasando por la señal Aeropuerto, luego por el límite de velocidad 60, Stop, límite de velocidad 120, giro a la derecha, señal McDonald's, giro

izquierda y por último señal Logo PUCV. En la Figura 4-28 se muestra el escenario para la detección del logo PUCV, el cual el robot debe transitar de extremo a extremo. El recorrido se ve reflejado en la Figura 4-29 empezando desde la flecha roja, lo cual toma como recorrido a 5 pistas, de la misma manera que las pruebas anteriores, el robot cumple con lo esperado.

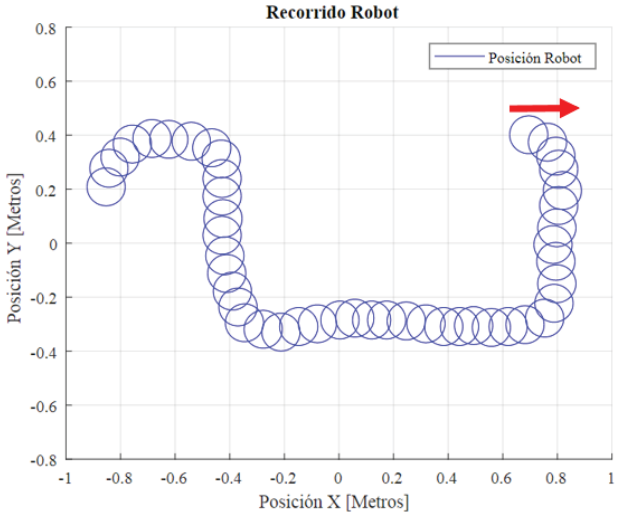

Figura 4-29: Recorrido del robot en detección señal PUCV.

También se realizó la gráfica de velocidades de los motores (Figura 4-30). En ella se puede observar que el tiempo de recorrido fue de 160 [s], cabe mencionar, que a diferencia de las otras pruebas, en esta se aprecia un pequeño problema en el instante de 75 [s] ya que ese momento es donde el robot se encuentra con la otra pista, produce que el robot intente girar hacia ese lugar, además como las velocidades son muy bajas, al robot le cuesta más estabilizarse.

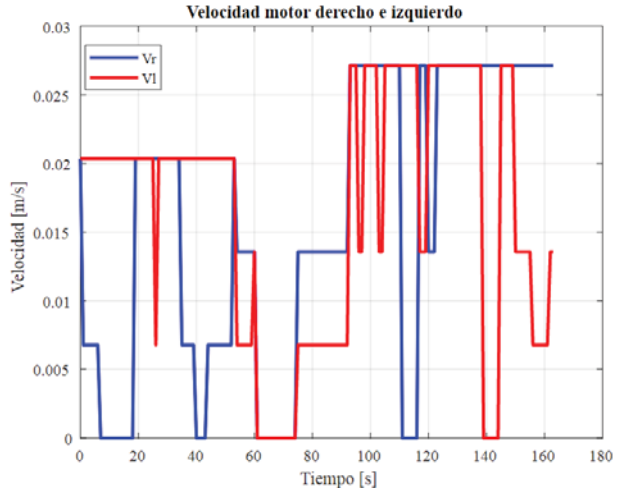

Figura 4-30: Velocidades de los motores del robot en detección señal PUCV.

Discusión y conclusiones

En este proyecto se realizó un estudio acerca de la robótica móvil de este último tiempo, por lo que brinda una gran ayuda a automatizar, realizar precisión y optimizar procesos en muchas áreas, como son los procesos industriales, medicinales, educación y navegación. A su vez, estas dos últimas atrae a los estudiantes de cualquier edad al mundo de las ciencias y la investigación. Además, la robótica se ha hecho tan popular que se encuentra accesible por todo tipo de personas, y gracias a su autonomía, las labores ya sean hogareñas o de trabajo, se realizan de una manera más eficiente y sencilla.

En la robótica móvil, este ha entregado un gran apoyo en temas de investigación y transporte, ya que el riesgo de que una persona sufra un daño por ingresar o actuar bajo circunstancias peligrosas ha disminuido. Por lo tanto, siempre se debe considerar que al momento de tomar una decisión en el que afecte significativamente nuestra salud, la robótica será una gran alternativa. Por otro lado, el estudio del funcionamiento de la operación de un robot móvil es importante ya que se tiene más claro los problemas que pueden suceder en la navegación de un robot, por ejemplo, problemas por el peso que genera el sistema de alimentación, o un mal sistema de control por la inadecuada instalación del sistema sensorial.

La navegación en robótica móvil no siempre brinda un buen funcionamiento, ya que también poseen problemas de localización, problemas de algoritmos, mala percepción del entorno e identificación. Pero se concluye que para todos esos problemas existen herramientas en la que el robot puede tomar y salir de ese paso engorroso. Estas herramientas han dado un amplio tema de estudio, por ejemplo: sistema multiagentes, inteligencia artificial y visión artificial. Esta última aporta significativamente al reconocimiento de objetos, por lo que el robot además de utilizar sus sensores tradicionales para detectarlos, la visión artificial percibe lo que realmente está detectando el robot.

La comunicación entre dos o más computadoras ha sido de gran relevancia para compartir y procesar información, ya que este mecanismo ayuda a que los procesos sean más rápidos. El problema ocurre cuando un computador no puede procesar una información que el otro si pueda, por ende, es necesario crear una comunicación entre ellos. Este problema se vio reflejado, debido a que el robot no es capaz de procesar la información de la cámara, por lo que es necesario que el robot envíe sus datos a un computador donde este si pueda, con el objetivo de facilitar su trabajo. Para que la comunicación sea fluida y eficiente, es necesario conocer en demasía el

comportamiento de las máquinas, por lo que hay que tener en cuenta las características del robot y de la máquina de procesado. A su vez, se debe tener en cuenta cómo será el traspaso de información, por lo que hay que conocer claramente el sistema de conexión de envió y recibimiento de datos. La cámara utilizada fue la integrada en el robot Khepera IV con el controlador *mt9v032.so*, lo que permitió configurar la imagen para su envío, en ella se puede modificar la calidad, tamaño, entre otros, por lo que es una herramienta útil para aumentar la velocidad de transmisión. Otra característica es que, al momento de procesar la imagen en un programa determinado, se tiene la opción de adquirir la información en forma de video o por imágenes, lo cual permite tener alternativas para elegir el modo de procesamiento.

Al procesar una imagen siempre hay que considerar las imperfecciones de esta, ya sea por el ruido, pérdida de definición o fidelidad de la imagen. A su vez, en imágenes existe un problema mayor con el tema de la iluminación, el cual, para poder segmentarla, es necesario definir un umbral de colores adecuado. Por lo tanto, es necesario saber qué espacio de colores es recomendable utilizar, en este caso fue utilizado el espacio de color HSV, ya que, con solo variar una componente, es posible discriminar un color del otro.

Otra técnica para la clasificación de una imagen, además de detección de bordes o colores, es la cascada de clasificadores que consiste en un conjunto de clasificadores que trabajan juntos para determinar un objeto en específico. Esta herramienta es bastante útil hoy en día, ya que permite discriminar cualquier objeto que uno desee, por lo tanto, en la detección de señales de tránsito es ideal. Este clasificador posee distintas etapas en la que uno debe seguir para construirlo, primero obtener imágenes positivas y negativas, por lo que fue necesario utilizar las herramientas de OpenCV para crear las muestras positivas ya que no es fácil conseguirlas y finalmente una etapa de entrenamiento, ya que, dependiendo de la cantidad de imágenes, tipo de extracción de características y etapas de entrenamiento, se creara un clasificador en cascada.

Se realizó un pequeño estudio acerca de los distintos escenarios de robótica móvil enlazado con la visión artificial, todos los sistemas tienen distintas técnicas para procesar la información de una señal, ya sea por la utilización de filtros, umbralización y clasificación; además se han utilizado técnicas muy relacionadas para otras funciones, como es la detección de caminos. Aunque en la literatura no se ha encontrado mucha documentación en donde se hayan realizado escenarios de pruebas con la navegación autónoma, en este estudio, se concluye que la detección de señales de tránsito utilizando cascadas de clasificadores en un entorno de laboratorio para robots móviles es algo novedoso, por lo que este trabajo es un aporte ya que da otro punto de vista a la problemática de la robótica móvil con la visión artificial.

Para la realización de experimentos en un robot, siempre es recomendable hacerlo primero en un modelo, ya que muchas veces el experimento real puede ser muy costoso o muy peligroso. El entorno de simulación V-REP proporciona una amplia gama de herramientas en el que se puede simular un robot, desde la simulación de escenarios donde se realicen experimentos, hasta la creación de modelos de robots con sus propias características.

Se realizó un esquema que relaciona un algoritmo propuesto para un sistema de visión artificial enlazado con la robótica móvil, este esquema fue diseñado especialmente para el robot Khepera

IV teniendo en cuenta sus capacidades y el entorno de trabajo establecido. Este esquema resume todos los procesos que deben pasar para que el robot realice un movimiento inteligente, lo cual facilita a entender el comportamiento del sistema propuesto.

Por último, se concluye que las pruebas se realizaron de forma exitosa, en primer lugar, se realizó una prueba simulada que conlleva al robot identificar si una flecha estaba con orientación a la izquierda o derecha para luego tomar una decisión; luego esta se realizó de forma real, determinando que no se pueden ocupar los mismos métodos que en la prueba virtual, ya que existen diferencias en la imagen obtenida (por ejemplo, la calidad). También se realizó una prueba simulada que involucra la detección y actuación de las señales de tránsito, se realizó un mapa el cual el robot debía llegar a un Aeropuerto, en el que poseía varias señales que lo ayudaban a llegar. Luego en el entorno real, se diseñó un sistema que detectara la orientación de robot, por lo que fue necesario crear un camino en la plataforma de tal que el sistema con solo procesamiento de imágenes pudiera detectarlo. A su vez, se realizó la detección de un semáforo creado, lo que involucro todo el proceso de creación de su cascada; esta tuvo buenos resultados por lo que se procedió a realizar la detección de los colores verde, amarillo y rojo con la técnica de la umbralización, por lo que se concluye que este método es la mejor opción para este tipo de características. Además, se realizaron varios escenarios en donde se vieron afectadas muchas señales, este es el caso de la detección de señal McDonald's, Aeropuerto y PUCV. El robot debía realizar la detección de objetos que involucraran que este girara, disminuyera o aumentara sus velocidades y se detuviera; además de todos estos procesos, tenía que seguir una ruta que permitiera su localización en el mapa, produciendo que el robot no se perdiera. A pesar de que el robot no siempre realizaba la detección de manera adecuada, este tuvo buenos resultados, ya que el sistema tuvo la capacidad de realizar movimientos inteligentes sin la intervención de la presencia humana.

Debido a que se están utilizando diversos tipos de clasificadores hoy en día, se propone como trabajos futuros la implementación de un sistema de detección de señales de tránsito mediante redes neuronales convolucionales, el cual se realizara el estudio de distintos algoritmos como redes CNN, SSD movilenet, YOLO, entre otros; además de realizar sus comparaciones junto con la cascada de clasificadores. En segundo lugar, se establecerá un nuevo sistema de control de posición, el cual posee distintas leyes que permitirán la realización de maniobras más eficientes, llevando al robot a su punto de destino en el menor tiempo posible. Por último, se propone relacionar la detección de señales mediante los sensores ultrasónicos e infrarrojos, lo cual se busca interpretar los datos adquiridos por la cámara a través de los sensores del robot.

Bibliografía

[1] R. Valero, Y. Ko, S. Chauhan, O. Schatloff, A. Sivaraman, R. Coelho, F. Ortega, K. Palmer, R. Sanchez-Salas, H. Davila, X. Cathelineau y V. Patel, «Cirugía robótica: Historia e impacto en la ensenanza,» *Actas Urológicas Españolas,* vol. 35, nº 9, pp. 540-545, 2011.

[2] J. F. Engelberger, Robotics in practice: management and applications of industrial robots, Springer Science & Business Media, 1980.

[3] M. L. Pinto Salamanca, N. Barrera Lombana y W. J. Pérez Holguín, «Uso de la Robótica educativa como herramienta en los procesos de enseñanza,» *Ingeniería Investigación y Desarrollo: I2+D,* vol. 10, nº 1, pp. 15-23, 2010.

[4] G. Andaluz, V. Andaluz y A. Rosales, «Modelación, Identificación y Control de Robots Móviles,» *Escuela Politecnica Nacional, Quito, Tesis de Ingeniería en Electrónica y Control,* 2011.

[5] G. Linda y C. Shapiro, Computer Vision, Seatle, Washington: Prentice Hall, Upper Saddle River, NJ, 2000.

[6] M. L. Guevara, J. D. Echeverry y W. A. Urueña, «Detección de rostros en imágenes digitales usando clasificadores en cascada,» *Scientia et technica,* vol. 1, nº 38, 2008.

[7] M. Zapata Ros, «Redes telemáticas: educación a distancia y educación cooperativa,» *Pixel-Bit. Revista de medios y educación,* vol. 8, pp. 57-79, 1997.

[8] H. Gonzáles y C. Mejia, «Estudio Comparativo De Tres Técnicas De Navegación Para Robots Móviles,» *Revista UIS Ingenierías,* vol. 6, nº 1, pp. 77-88, 2007.

[9] B. Jähne, Computer vision and applications: a guide for students and practitioners, Heidelberg, Germany: Elsevier, 2000.

[10] M. Fernández, D. Fernández y C. Valmaseda, «Planificación de trayectorias para un robot móvil,» Memoria Proyecto Sistemas Informáticos, Madrid, 2010.

[11] F. Jiménez, J. Moreno, R. Gonzales, F. Rodríguez y J. Sánchez, «Sistema de visión de apoyo a la navegación de un robot móvil en invernaderos,» *XXIX Jornadas de AutomÁtica,* pp. 3-5, 2010.

[12] C. Soria, R. Carelli, R. Kelly y J. Zannatha, «Control de robots Cooperativos por Medio de Visión Artificial,» *XVI Congreso de La Asociación Chilena de Control Automático,* vol. 223, 2004.

[13] M. Lakrouf, S. Larnier, M. Devy y N. Achour, «Moving Obstacles Detection and Camera Pointing for Mobile Robot Applications,» *roceedings of the 3rd International Conference on Mechatronics and Robotics Engineering,* pp. 57-62, 2017.

[14] A. Irawan, M. Yaacob, F. Azman, M. Daud, A. Razali y S. Ali, «Vision-based Alignment Control for Mini Forklift System in Confine Area Operation,» *2018 International Symposium on Agent, Multi-Agent Systems and Robotics,* pp. 1-6, 2018.

[15] F. Zaklouta y B. Stanciulescu, «Real-time traffic sign recognition in three stages,» *Robotics and autonomous systems,* vol. 62, nº 1, pp. 16-24, 2014.

[16] S. Hossain y Z. Hyder, «Traffic Road Sign Detection and Recognition for Automotive Vehicles,» *International Journal of Computer Applications,* vol. 120, nº 24, 2015.

[17] S. Archana, G. Thejas, S. Ramani y S. Iyengar, «Image Processing Approaches for Autonomous Navigation of Terrestrial Vehicles in Low Illumination,» *2017 2nd International Conference On Emerging Computation and Information Technologies,* pp. 1-6, 2017.

[18] A. De la Escalera, J. Armingol y M. Mata, «Traffic sign recognition and analysis for intelligent vehicles,» *Image and vision computing,* vol. 21, nº 3, pp. 247-258, 2003.

[19] H. Zhang, D. Hernandez, Z. Su y B. Su, «A Low Cost Vision-Based Road-Following System for Mobile Robots,» *Applied Sciences,* vol. 8, nº 9, p. 1635, 2018.

[20] H. Shah, M. Ab Rashid, Z. Kamis, M. Aras, N. Ali, F. Wasbari y T. Anuar, «Sign Detection Vision Based Mobile Robot Platform,» *Indonesian Journal of Electrical Engineering and Computer Science,* vol. 7, nº 2, pp. 524-532, 2017.

[21] E. Peralta, «Modelado, control y simulación en robótica móvil,» Tesis Ingeniero Eléctrico Pontificia Universidad Católica de Valparaíso, Valparaiso, 2016.

[22] X. Tan y B. Triggs, «Enhanced Local Texture Feature Sets for Face Recognition Under Difficult Lighting Conditions,» *IEEE transactions on image processing,* vol. 19, nº 6, pp. 1635-1650, 2010.

[23] Team github, «The VXL Homepage,» [En línea]. Available: https://vxl.github.io/.

[24] M. Cazorla, O. Colomina, P. Compañ, F. Escolano y J. Zamora, «JavaVis: Una librería para visión artificial en Java,» *Técnicas de Inteligencia Artificial,* 2001.

[25] Team LTI-lib, «LTI-lib,» 2010. [En línea]. Available: http://ltilib.sourceforge.net/doc/homepage/index.shtml.

[26] Team OpenCV Developers, «OpenCV,» 2018. [En línea]. Available: https://opencv.org/.

[27] L. OpenCV, «Color spaces in OpenCV (C++ / Python),» 2018. [En línea]. Available: https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/.

[28] B. Leo, «Bias, variance, and arcing classifiers.,» 1996.

[29] Z. H. Zhou, «Ensemble methods: foundations and algorithms,» *Chapman and Hall/CRC,* 2012.

[30] M. Kearns, «Thoughts on hypothesis boosting,» *Unpublished manuscript,* vol. 45, p. 105, 1988.

[31] M. Kearms y L. Valiant, «Cryptographic limitations on learning Boolean formulae and finite automata,» *Journal of the ACM (JACM),* vol. 41, nº 1, pp. 67-95, 1994.

[32] Coursera, «Cascada de clasificadores,» 2018. [En línea]. Available: https://es.coursera.org/lecture/deteccion-objetos/l5-5-cascada-de-clasificadores-pRnHu.

[33] A. Rosebrock, «Local Binary Patterns with Python & OpenCV,» Pyimagesearch, 7 Diciembre 2018. [En línea]. Available: https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/.

[34] Intel, «Histogram of Oriented Gradients (HOG) Descriptor,» software intel, 11 Septiembre 2018. [En línea]. Available: https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients-hog-descriptor.

[35] A. Van, «Histogram of Oriented Gradients (HOG) Boat Heading Classification,» The DownLinQ, 12 Agosto 2016. [En línea]. Available: https://medium.com/the-downlinq/histogram-of-oriented-gradients-hog-heading-classification-a92d1cf5b3cc.

[36] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. Gambardella y M. Dorigo, «ARGoS: a Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems,» *Swarm Intelligence,* vol. 6, nº 4, pp. 271-295, 2012.

[37] cyberbotics, «Webots Robot Simulator,» 2018. [En línea]. Available: https://cyberbotics.com/.

[38] E. Fabregas, G. Farias, S. Dormido-Canto y S. Dormido, «RFCSIM simulador interactivo de robótica móvil para control de formación con evitación de obstáculos,» de *XVI Congreso Latinoamericano de Control Automático*, Cancún, Quintana Roo, México, 2014.

[39] Team Coppeliarobotics, «V-REP,» [En línea]. Available: http://www.coppeliarobotics.com/.

[40] K-Team, «K Team Mobile Robotics,» 2015. [En línea]. Available: https://www.k-team.com/.

[41] K Team, User Manual Khepera IV, Switzerland: Revision 1.1, 2014.

[42] E. Peralta, E. Fabregas, G. Farias, H. Vargas y S. Dormido, «Development of a Khepera IV Library for the V-REP Simulator,» *IFAC-PapersOnLine,* vol. 49, nº 6, pp. 81-86, 2016.

[43] sourceforge, «SwisTrack,» [En línea]. Available: https://sourceforge.net/projects/swistrack/.

[44] Team Autodesk, «Inventor Professional,» 2018. [En línea]. Available: https://www.autodesk.com/.

[45] Team 3dnatives, «Impresora 3D : Replicator II,» 2018. [En línea]. Available: https://www.3dnatives.com/es/3D-compare/imprimante/replicator-2.

[46] Team grabcad, «Traffic Signals,» 2013. [En línea]. Available: https://grabcad.com/library/traffic-signals-1.

[47] «Datasheets Atmel 8-bit AVR Microcontroller with 2/4/8K Bytes In-System Programmable Flash ATtiny25/V / ATtiny45/V / ATtiny85/V,» Atmel, 2013. [En línea]. Available: http://www.farnell.com/datasheets/1698186.pdf.

[48] Team Fritzing, «Fritzing,» 2018. [En línea]. Available: http://fritzing.org/fritzing-creatorkit/.

[49] pythonprogramming, «Creating your own Haar Cascade OpenCV Python Tutorial,» [En línea]. Available: https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/.

[50] Anaconda Corporate Headquarters, «Anaconda,» [En línea]. Available: https://www.anaconda.com/.

[51] O. Semiconductor, «Datasheet MT9V032,» Semiconductor Components Industries, Mayo 2017. [En línea]. Available: https://www.onsemi.com/pub/Collateral/MT9V032-D.PDF.

[52] Team github, «mjpg-streamer-yu12,» 2018. [En línea]. Available: https://github.com/pranjalv123/mjpg-streamer-yu12/tree/master/mjpg-streamer-experimental.

A Comandos y códigos utilizados

A.1 Comandos para crear cascadas de clasificadores con OpenCV

Tabla A-1: Comandos opencv_createsamples.

Código	Descripción
`-vec <vec_file_name>`	Nombre del archivo de salida que contiene las muestras positivas para el entrenamiento.
`-img <image_file_name>`	Imagen del objeto fuente (p. Ej., Un logo tipo de la empresa)
`-bg <background_file_name>`	Archivo de descripción de fondo; contiene una lista de imágenes que se utilizan como fondo para versiones distorsionadas al azar del objeto.
`-num <number_of_samples>`	Número de muestras positivas para generar
`-bgcolor <background_color>`	Color de fondo (actualmente se presuponen imágenes en escala de grises); el color de fondo denota el color transparente. Dado que puede haber artefactos de compresión, la cantidad de tolerancia de color se puede especificar con -bgthresh. Todos los píxeles de entrada bgcolor, bgthreshy, bgcolor y bgthreshrango se interpretan como transparentes.
`-bgthresh <background_color_threshold>`	
`-inv`	Si se especifica, los colores se invertirán.
`-randinv`	Si se especifica, los colores se invertirán aleatoriamente.
`-maxidev <max_intensity_deviation>`	Desviación de intensidad máxima de píxeles en muestras de primer plano.
`-maxxangle <max_x_rotation_angle>`	

`-maxyangle <max_y_rotation_angle>`	
`-maxzangle <max_z_rotation_angle>`	Los ángulos de rotación máximos deben darse en radianes.
`-show`	Opción de depuración útil Si se especifica, se mostrará cada muestra. Al presionar continuara con el proceso de creación de muestras.
`-w <sample_width>`	Ancho (en píxeles) de las muestras de salida.
`-h <sample_height>`	Altura (en píxeles) de las muestras de salida.
`-pngoutput`	Con esta opción, la opencv_createsamples herramienta activada genera una colección de muestras PNG y una cantidad de archivos de anotación asociados, en lugar de un solo vec archivo.

Tabla A-2: Comandos opencv_traincascade.

Código	Descripción
`-data <cascade_dir_name>`	Donde se debe almacenar el clasificador entrenado.
`-vec <vec_file_name>`	Archivo vec con muestras positivas (creadas por opencv_createsamples).
`-bg <background_file_name>`	Archivo de descripción de fondo.
`-numPos <number_of_positive_samples>`	Número de muestras positivas utilizadas en el entrenamiento para cada etapa clasificadora.
`-numNeg <number_of_negative_samples>`	Número de muestras positivas / negativas utilizadas en el entrenamiento para cada etapa clasificadora.
`-numStages <number_of_stages>`	Número de etapas en cascada a entrenar.
`-precalcValBufSize <precalculated_vals_buffer_size_in_Mb>`	Tamaño del búfer para valores de características precalculados (en Mb).
`-precalcIdxBufSize <precalculated_idxs_buffer_size_in_Mb>`	Tamaño del búfer para índices de características precalculados (en Mb). Cuanta más memoria tengas, más rápido será el proceso de entrenamiento.
`-baseFormatSave`	Este argumento es real en el caso de características similares a Haar. Si se especifica, la cascada se guardará en el formato antiguo.
`-acceptanceRatioBreakValue`	Este argumento se utiliza para determinar la precisión con la que su modelo debe seguir aprendiendo y cuándo detenerse. Una buena guía es entrenar no más allá de 10e-5, para garantizar que el modelo no sobreentienda sus datos de entrenamiento. De forma predeterminada, este valor se establece en -1 para deshabilitar esta función.

A.2 Código Python

Listado A-1: Detección de señales de tránsito.

```
1  import cv2
2  import numpy as np
3
4  aeropuerto = cv2.CascadeClassifier('aeropuerto.xml')
5  flechaDerecha = cv2.CascadeClassifier('flechaDerecha.xml')
6  flechaIzquierda = cv2.CascadeClassifier('flechaIzquierda.xml')
7  Stop = cv2.CascadeClassifier('Stopsign.xml')
8  cedaElPaso = cv2.CascadeClassifier('yield.xml')
9  velocidad = cv2.CascadeClassifier('SpeedLimitHaarCascade_v2.xml')
10
11 img = cv2.imread('imagen1.jpg')
12
13 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
14 aeropuerto1 = aeropuerto.detectMultiScale(gray,2,10)
15 flechaDerecha1 = flechaDerecha.detectMultiScale(gray,1.5,5)
16 flechaIzquierda1 = flechaIzquierda.detectMultiScale(gray,1.5,5)
17 Stop1 = Stop.detectMultiScale(gray, 1.1, 5)
18 cedaElPaso1 = cedaElPaso.detectMultiScale(gray,1.3,5)
19 velocidad1 = velocidad.detectMultiScale(gray,1.9,5)
20
21 for (x1,y1,w1,h1) in aeropuerto1:
22     cv2.rectangle(img,(x1,y1),(x1+w1,y1+h1),(0,0,0),2)
23     cv2.putText(img,"Aeropuerto",(x1+50,y1-
   5+230),cv2.FONT_HERSHEY_SIMPLEX,0.6,(0,0,0),1,)
24     roi_gray = gray[y1:y1+h1, x1:x1+w1]
25     roi_color = img[y1:y1+h1, x1:x1+w1]
26 for (x2,y2,w2,h2) in flechaDerecha1:
27     cv2.rectangle(img,(x2,y2),(x2+w2,y2+h2),(128,0,128),2)
28     cv2.putText(img,"Flecha Derecha",(x2+10,y2-
   5+220),cv2.FONT_HERSHEY_SIMPLEX,0.6,(0,0,0),1,)
29     roi_gray = gray[y2:y2+h2, x2:x2+w2]
30     roi_color = img[y2:y2+h2, x2:x2+w2]
31 for (x3,y3,w3,h3) in flechaIzquierda1:
32     cv2.rectangle(img,(x3,y3),(x3+w3,y3+h3),(0,255,0),2)
33     cv2.putText(img,"Flecha Izquierda",(x3,y3-
   5+200),cv2.FONT_HERSHEY_SIMPLEX,0.6,(0,0,0),1,)
34     roi_gray = gray[y3:y3+h3, x3:x3+w3]
35     roi_color = img[y3:y3+h3, x3:x3+w3]
36 for (x4,y4,w4,h4) in Stop1:
37     cv2.rectangle(img,(x4,y4),(x4+w4,y4+h4),(0,0,255),2)
38     cv2.putText(img,"STOP",(x4+70,y4-
   5+230),cv2.FONT_HERSHEY_SIMPLEX,0.6,(0,0,0),1,)
39     roi_gray = gray[y4:y4+h4, x4:x4+w4]
40     roi_color = img[y4:y4+h4, x4:x4+w4]
41 for (x5,y5,w5,h5) in cedaElPaso1:
42     cv2.rectangle(img,(x5,y5),(x5+w5,y5+h5),(255,200,0),2)
43     cv2.putText(img,"Ceda el paso",(x5+10,y5-
   5+220),cv2.FONT_HERSHEY_SIMPLEX,0.6,(0,0,0),1,)
44     roi_gray = gray[y5:y5+h5, x5:x5+w5]
45     roi_color = img[y5:y5+h5, x5:x5+w5]
46 for (x6,y6,w6,h6) in velocidad1:
47     cv2.rectangle(img,(x6,y6),(x6+w6,y6+h6),(255,0,0),2)
48     cv2.putText(img,"Limite de velocidad",(x6,y6-
   5+200),cv2.FONT_HERSHEY_SIMPLEX,0.6,(0,0,0),1,)
49     roi_gray = gray[y6:y6+h6, x6:x6+w6]
50     roi_color = img[y6:y6+h6, x6:x6+w6]
51
52 cv2.imshow('img',img)
53 cv2.imwrite('save1.jpg', img)
54 cv2.waitKey(0)
```