CENTRO DE INGENIERÍA Y DESARROLLO INDUSTRIAL

# Implementación de algoritmo de detección de objetos en entornos abiertos en hardware embebido

# REPORTE DE PROYECTO FINAL

En obtención al título de:

**ESPECIALIDAD EN TECNÓLOGO EN MECATRÓNICA**

PRESENTA:

Ranyel Morales Suero

ASESORES:

M.C. Alberto Vázquez Cervantes

Dr. Hugo Jiménez Hernández

Dr. Jorge Alberto Soto Cajiga

SANTIAGO DE QUERÉTARO, QUERÉTARO,
23 de agosto de 2017

# Agradecimientos

Agradezco a CIDESI por permitirme formar parte de su comunidad en la realización de este proyecto y así poder culminar una etapa más en mi desarrollo profesional.
De igual manera agradezco a CONACYT por haberme otorgado su apoyo financiero para mi desarrollo profesional.
A mis asesores el M.C y T Alberto Vázquez Cervantes, al Dr Hugo Jiménez Hernández y al Dr Jorge Alberto Soto Cajiga por haberme dado la oportunidad de recurrir a sus capacidades de orientación y conocimiento científico para guiarme durante el desarrollo de este proyecto. Gracias por su apoyo y confianza en mí.
Agradezco a mis grandes amigos Fatima Daniela González Mateo, Leisis López Rodríguez, Noé Morales Velasco, Jaquelin de los Ángeles Viveros y a Erik Emmanuel Pérez Martínez, por brindarnos su grata amistad, apoyo y aliento a lo largo de este año. Ellos son mi familia mexicana, los quiero muchísimo a todos.
Por último pero no menos importante agradezco a Mario Dagoberto Díaz Orgaz, Julio Díaz e Isis Martínez Díaz, por acogenos como parte de su familia también y brindarme apoyo y consejo en mi desarrollo humano y profesional.
Agradezco infinita y especialmente a mis padres Nelson Paulino González Morales y Clarisbel Suero Ramírez por su apoyo y eterna confianza en mi, ellos son mi mayor tesoro.
A todas estas personas muchas gracias por todo.

CONACYT

CIDESI

**CENTRO DE INGENIERÍA Y DESARROLLO INDUSTRIAL**

**CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN TECNOLÓGICA**

# AUTORIZACIÓN
# PUBLICACIÓN EN FORMATO ELECTRÓNICO DE TESIS

**Fecha:** 18/08/2017

El que suscribe Alumno (a) RANYEL MORALES SUERO

CURP MOSR891223HNERRN03 CVU 789563

ORCID 0000-0001-5653-9986

Correo electrónico (opcional) ranyel.morales@gmail.com

Egresado (a) de ESPECIALIDAD DE TECNÓLOGO EN MECATRÓNICA

Autor de la Tesis titulada IMPLEMENTACIÓN DE ALGORITMO DE DETECCIÓN DE OBJETOS EN ENTORNOS ABIERTOS EN HARDWARE EMBEBIDO

Por medio del presente documento autorizo[1] en forma gratuita y permanente a que la Tesis arriba citada sea divulgada y reproducida para publicarla mediante almacenamiento electrónico que permita el acceso al público a leerla y conocerla visualmente, así como a comunicarla públicamente en Página Web.
La única contraprestación que condiciona la presente autorización es la del reconocimiento del nombre del autor en la publicación que se haga de la misma.

Atentamente

RANYEL MORALES SUERO

Nombre y firma del tesista

[1] Ley Federal de Derechos de Autor
Para obtener tu ORCID regístrate en: **https://orcid.org/register**

Av. Playa Pie de la Cuesta No. 702 Desarrollo San Pablo. C. P. 76125
Santiago de Querétaro, Qro., México México.
Tel. 01 (442) 211 98 40
E-mail: jvite@cidesi.edu.mx

# Índice general

# Índice de figuras

# Índice de tablas

# Resumen

En el presente documento se desarrolla la implementación de un algoritmo en hardware embebido para detección de objetos en entornos abiertos en un vehículo autónomo, utilizando la técnica de visión por computadora de la Segmentación y dentro de esta el Método del valor umbral. Para la detección localización y seguimiento del objeto por color se emplean las bondades que ofrece la biblioteca de Visión por Computadora de Código Abierto (OpenCV) en el lenguaje de programación Python. En este se muestra el poder de procesamiento de los nuevos sistemas embebidos como son la Raspberry Pi-3 para controlar un robot móvil por medio de un algoritmo de visión. En este caso particular se busca desde un robot móvil analizar un objeto con base a su color, tal que el dispositivo pueda moverse de manera autónoma hacia el objeto. Además, este sistema continúa su seguimiento aún ante dificultades que se presentan en el ambiente con el que interactúa. Algunas de estas dificultades son los cambios lumínicos en el ambiente, así como los cambios de perspectiva y escala del objeto producto a su movimiento.

# Introducción.

Hoy en día la Robótica se ha convertido en un tema de gran interés, con grandes adelantos debido a una gran cantidad de proyectos que se han desarrollado en todo el mundo. Como resultado de esto, se han logrado conseguir robots con amplia interacción con el medio, lo cual ha abierto una inmensa gama de aplicaciones como la toma muestras, análisis del medio, detección de gases, fugas, envió de señales de audio y video y viceversa, y más, todo ello de forma remota para evitar el riesgo humano [7], [8].

El uso de los sensores ayuda al problema de navegación en un ambiente determinado al robot. Actualmente en muchos proyectos de robótica se hace uso de la visión por computadora utilizando como elemento sensor las cámaras. El uso de la visión por computadora no es exclusivamente para el uso en la robótica, sino que también se utiliza en aplicaciones tales como guiado automático de máquinas, procesamiento de imágenes como radiografías, resonancias magnéticas, identificación de imperfecciones en objetos, reconocimiento y clasificación de objetos y personas, inspección y control de la calidad y muchas otras aplicaciones.

Las funciones de la visión computacional en la robótica es dar seguridad, verificando el estado del sistema en general. Otro objetivo de la visión computacional en la robótica industrial es dar mayor flexibilidad, es decir, permitir la adaptación del sistema cuando haya cambios y aumentar la tolerancia a errores de posicionamiento, pero sin perder la eficiencia de trabajo. Un objetivo que se busca en la industria es abaratar costos, con esto se sustituye la mano de obra, pero a la vez también se disminuyen los riesgos para las personas [9].

Los robots móviles tienen una amplia aplicación dentro de la industria, entre las que se encuentran el transporte de materiales peligrosos, robots desactivadores de explosivos, exploración de áreas que el hombre no puede explorar entre los cuales destacan algunos robots desarrollados por la NASA como lo son el Spirit y Oportunity [10].

Esta tendencia es la causa de la gran potenciación de la invetigación de nuevas técnicas aplicables a la robótica, y buen ejemplo de ello son la visión por computadora, la inteligencia artificial o el

aprendizaje autónomo.

La visión por computadora es campo de la Inteligencia Artificial enfocado a que las computadoras puedan extraer información a partir de imágenes, ofreciendo soluciones a problemas del mundo real. Actualmente en el área de inspección, vigilancia, búsqueda y rescate existen problemáticas en aplicaciones específicas en cuanto a la detección, identificación y fundamentalmente en el seguimiento de objetos debido al alto grado de complejidad y costo computacional que presentan los algoritmos de visión por computadora empleados para este fin.

Algunos de los algoritmos de visión por computadora utilizados para realizar la detección y seguimiento de objetos son:

1. La comparación de plantilla (Template Matching).
2. Sustracción de fondo (Background Subtraction).
3. Clasificador en casca (Haar Cascade).
4. Algoritmo SURF (Speeded-Up Robust Features).

Una de las aplicaciones más destacadas de la visión por computadora se presenta en el área de la robótica móvil. Dentro de la robótica móvil se encuentran los vehículos autónomos; los cuales se han convertido en la plataforma para el desarrollo de investigaciones y proyectos [11], [12]. Esto se debe a la flexibilidad y efectividad que los mismos ofrecen. Se han utilizado en aplicaciones donde existen riesgos para el ser humano, tales como búsqueda y rescate [13], [14], [15], inspección [16] y vigilancia [17]. Dentro de la robótica móvil; los Seguidores de objetos (Object tracking) es una importante área en el campo de visión por computadora, esta consta de tres fases principales:

1. Detectar el objeto de interés.
2. Seguir el objeto dentro de los cuadros de videos.
3. Analizar el comportamiento de la ruta del objeto.

Actualmente se emplean los sistemas embebido (S.E) como la microcomputadora Raspberry Pi en los sistemas de detección, identificación y seguimiento de objetos o personas. Estos se encuentran instalados en robots y vehículos autónomos, debido a su bajo peso y gran potencia de procesamiento, esto evidentemente permite una rápida fluidez de la información, logrando que los sistemas de visión artificial tengan gran rapidez en la detección, identificación y seguimiento de objeto, además de permitir que sean muy flexibles a los cambios del medio y proporcionarle más autonomía al vehículo.

# Planteamiento del Problema

En la actualidad los sistemas tecnológicos están enfocados a resolver problemas concretos, lo que aumenta su efectividad. Por ejemplo, existen dispositivos que deben interactuar en situaciones en donde es de vital importancia detectar objetos, identificarlos y darle seguimiento.

La visión por computadora provee un mecanismo natural de detección y seguimiento de características en los UAV (Air Unmanned Vehicles) y MAUV (Micro Air Unmanned Vehicles), por ser las cámaras livianas y eficientes en consumo de energía.

Algunas de las técnicas de visión por computadora para realizar la detección y seguimiento de objetos, pero que computacionalmente las hacen inviables, debido a su alto grado de complejidad y costo computacional son Scale-invariant feature transform (SIFT) y Speeded-Up Robust Features (SURF).

Los vehículos autónomos de monitoreo de objetos requieren de un procesador robusto para aplicar las técnicas de localización de objetos. Normalmente el vehículo se encuentra comunicado con una computadora, la cual se encuentra en una central y esta procesa la información. Pero es evidente que colocar una computadora tan robusta en un vehículo autónomo lo haría de gran tamaño, peso y costoso. Por esta razón se plantea como problema fundamental del trabajo es diseñar un vehículo autónomo localizador de objeto que presente el menor costo computacionalmente posible y logre el seguimiento en línea del objeto.

# Justificación

En los vehículos autónomos localizadores de objetos se presentan inconvenientes como son el procesamiento computacional o el movimiento constante. Esto evidentemente impide que estos vehículos tengan una completa autonomía, ya que al aumentar la cantidad de datos a procesar se incrementa considerablemente la complejidad del algoritmo y requiere más potencia de computo. Por tal motivo existe la necesidad de desarrollar un algoritmo en el sistema embebido que presente una eficiencia computacional aceptable y sea robusto para mantener el desempeño.

Todo lo anterior ha motivado el desarrollo de este proyecto, basado en la visión por computadora para el control de un vehículo terrestre autónomo(UGV del ingles: Unmanned Ground Vehicle) localizador de objetos.

# Objetivos

## Objetivo general.

Crear un vehículo que utilizando técnicas de visión por computadora que logre hacer un seguimiento de un objeto en un sistema embebido (Opciones: seguidor de color, seguimiento de formas geométricas).

## Objetivos específicos.

1. Seleccionar el método de seguimiento.
2. Construcción del vehículo.
3. Realizar la instrumentación del vehículo (Cámara (visión), motores (para el movimiento), sistema embebido (Raspberry Pi 3)).
4. Diseñar el algoritmo de control empleando la biblioteca de Visión por Computador de Código Abierto (OpenCV) y utilizando el lenguaje de programación Python; que garantice el funcionamiento del sistema embebido de visión para el vehículo.
5. Implementar el algoritmo en el vehículo autónomo.
6. Validar experimentalmente el diseño.
7. Redactar el Reporte de Proyecto final.

# Antecedentes

(Paredes, Héctor López, 2011). Desarrolló una investigación sobre la mejor forma de monitorear actividades en escenarios donde es necesario buscar y reconocer un objeto de interés.
(Naidoo, Yogianandh y otros, 2011). Propusieron como solución al monitoreo de actividades en escenarios donde es necesario buscar y reconocer objetos, el uso de un UAV; en este trabajo se emplean algoritmos tales como Template Matching, Background Subtraction y Haar Cascade; pero estos algoritmos presentan baja tolerancia a los cambios lumínicos, rotaciones del objeto y cambios de escala.
(Wang, Wenying y otros, 2015). Utilizaron mejores técnicas como es el algoritmo SURF, este algoritmo presenta una gran robustez y velocidad de cálculo respecto a los anteriores. Presenta alta complejidad y un coste computacional elevado.
(González, Carlos Pérez, 2016). Realizó la detección y el seguimiento de objetos en un S.E Raspberry Pi1. En este se muestran los resultados obtenidos empleando las herramientas del OpenCV a través del lenguaje de programación Python.
(Mateo, Raquel Muñoz, 2016). Desarrolló de un vehículo robótico de exploración para entornos terrestres en interiores, mostrando las posibilidades de control que posee la Raspberry Pi. Sin embargo, el seguimiento de objetos presenta inconvenientes al producirse oclusiones en el campo visual de la cámara o cambios de escala o geométricos del objeto, lo que provoca que el movimiento del robot sea lento.
En los trabajos mencionados se muestran diversas técnicas para el análisis del entorno de desplazamiento a partir de las imágenes, sin embargo, los algoritmos empleados a pesar de lograr realizar la detección y seguimiento del objeto, presentan fundamentalmente un alto costo computacional, lo cual los hace no viable para implementarlo de manera directa en un sistema embebido como lo es la Raspberry Pi.

# Hipótesis

Con la implementación del algoritmo en el hardware embebido para la detección de objetos en un vehículo autónomo, será posible realizar la detección de objetos tomando en cuenta las dificultades que se presentan en el movimiento, logrando así una disminución en la complejidad computacional haciendo viable el seguimiento en videos de vehículos autónomos.

# Alcances

## Alcances del proyecto

1. Implementar el algoritmo en el hardware embebido (Raspberry Pi).
2. Lograr la detección y seguimiento del objeto por color en línea.
3. Esta limitado al seguimiento de objetos por color. Seguimiento a solo un objeto.
4. El control del vehículo será básico pero que logre mantener el desempeño.
5. El sistema a desarrollar será colocado en un vehículo de dimensiones reducida.

## Cronograma de actividades

| Actividad | Meses (semanas) | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Mayo | | | | Junio | | | | Julio | | | | Agosto | | | |
| | S1 | S2 | S3 | S4 | S1 | S2 | S3 | S4 | S1 | S2 | S3 | S4 | S1 | S2 | S3 | S4 |
| Investigación (Marco teórico Estado del Arte) | ■ | ■ | ■ | | | | | | | | | | | | | |
| Diseño de prototipo de vehículo autónomo. | | | | ■ | | | | | | | | | | | | |
| Diseño e Implementación de algoritmo de control garantice el funcionamiento del sistema embebido de visión para el vehículo. | | | | | ■ | ■ | ■ | ■ | ■ | ■ | | | | | | |
| Validar experimentalmente el diseño | | | | | | | | | | | ■ | ■ | ■ | | | |
| Correcciones y/o mejoras. | | | | | | | | | | | | | | ■ | ■ | ■ |

Figura 1: Cronograma de actividades.

Capítulo 1

# FUNDAMENTO TEÓRICO

Los robots autónomos o inteligentes son máquinas capaces de percibir, modelar el entorno, planificar y actuar para alcanzar los objetivos sin la intervención o con una intervención muy pequeña de supervisores humanos. Estos son desde el punto de vista del procesamiento de información los más evolucionados.

Generalmente el control de estos robots es complejo y aumenta con el tipo de ambiente en donde se desenvuelven. Sin embargo, una de sus principales ventajas es que se emplean en aplicaciones donde existen riesgos para el ser humano. ejemplo de esto son los vehiculos de exploración Sojouner y Spirit (MER-A). Estos dos vehículos comparten varias característica en común pero una de las más relevantes es que emplean la visión por computadora para moverse en el ambiente, es decir, a partir de las imágenes que obtienen a través de la cámara toman decisiones, siendo capaces de evadir ostáculos e identificar objetos.

## 1.1. Estado del arte

### 1.1.1. Sistemas de video vigilancia

Los Sistemas de video vigilancia consisten en cámaras fijas o móviles que tienen como objetivo grabar el comportamiento de objetos o personas que se produce en la zona sobre la que tienen campo de visión.

Un caso particular de sistemas de video vigilancia muy importante debido a la gran cantidad de posibilidades que ofrece son los sistemas de video vigilancia inteligente. El objetivo de estos sistemas consiste en la monitorización en tiempo real de objetos en un escenario determinado y su correspondiente interpretación automática para la interpretación de los escenarios y la comprensión y predicción de acciones e interacciones en los objetos observados a partir de la información tomada por los sensores.

En la actualidad la Video-vigilancia IP es una tecnología de vigilancia visual que combina los beneficios analógicos de los tradicionales CCTV (Circuito Cerrado de Televisión) con las ventajas

digitales de las redes de comunicación IP (Internet Protocol), permitiendo la supervisión local y/o remota de imágenes así como el tratamiento digital de las imágenes, para aplicaciones como el reconocimiento de matrículas o reconocimiento facial, entre otras [18], [19].

### 1.1.2. Aplicaciones de los Sistemas de video vigilancia

Las aplicaciones a las que se pueden destinar los sistemas de video vigilancia son muy diversas, pero todas ellas podrían clasificarse en tres grandes grupos: aplicaciones basadas en píxel, aplicaciones basadas en objeto y aplicaciones especializadas.

**Aplicaciones basadas en píxel:**

1. Detección de movimiento.
2. Detección de manipulación de cámaras y mejora de imagen.

**Aplicaciones basadas en objeto:**

1. Detección de intrusos.
2. Contabilización de objetos o personas.
3. Seguimiento de objetos.
4. Detección de objetos abandonos.

Las aplicaciones especializadas utilizan una combinación de píxel y objeto para tomar sus decisiones. Algunos ejemplos son:

1. Reconocimiento de matrículas.
2. Reconocimiento facial.
3. Detección de fuego y humo.

Ejemplos de sistemas de vigilancia móviles son los Vehículo Aéreo No Tripulado (VANT), UAV (Unmanned Aerial Vehicle) o comúnmente llamado dron, también se encuentran los Robots Terrestres No Tripulados (Unmanned Ground Vehicle).

**Ventajas y desventajas**

Las principales ventajas que muestran estos sistemas de vigilancia móvil, es que no presentan la necesidad de depender de una persona (vigilante, supervisor), que se encuentre revisando los

monitores y detectando cualquier anomalía. Los sistemas móviles permiten llegar a zonas y lugares de difícil acceso lo cual los provee de cierta autonomía, además de que los mismos se utilizan en aplicaciones donde existen riesgos para el ser humano, como son la detección de incendios, inspección de estructuras, rescates (búsquedas de persona), etc.

### 1.1.3. Robots móviles

Un robot móvil es una máquina que tiene la habilidad de moverse dentro de un ambiente, puede realizar ciertas operaciones para completar una tarea específica [20].

#### 1.1.3.1. Clasificación de los robots

De acuerdo a su grado de autonomía, los robots pueden clasificarse en teleoperados, de funcionamiento repetitivo y autónomos o inteligentes [7].

1. Robots teleoperados: En los robots teleoperados las tareas de percepción del entorno, planificación y manipulación compleja son realizadas por humanos.

2. Robots de funcionamiento repetitivo: Se utilizan en las cadenas de producción industrial. Trabajan normalmente en tareas predecibles, e invariantes, con una limitada percepción del entorno.

3. Robots autónomos o inteligentes: Son máquinas capaces de percibir, modelar el entorno, planificar y actuar para alcanzar los objetivos sin la intervención o con una pequeña intervención de supervisores humanos.

### 1.1.4. Seguimiento visual de objetos

La visión por computador genera información que puede ser empleada para realizar el posicionamiento de un robot relativo a un objetivo, lo que suele denominarse control servo visual.
Un sistema de control visual en función de la información obtenida de la imagen, puede estar basado en posición o imagen, 3D o 2D, respectivamente. Sin embargo, si la estructura de control visual genera referencias al controlador interno del robot o actúa directamente sobre los actuadores, puede decirse es un control visual indirecto o control visual directo, respectivamente.

#### 1.1.4.1. Sistemas de control visual, en función de la estructura de control

**Control visual indirecto:** Si el sistema de control tiene la estructura de dos etapas, es decir, el sistema posee un bucle interno rápido o controlador interno del robot, y otro externo visual y más lento que genera las referencias en todo momento a ese controlador interno.
Si los bucles operan en forma secuencial, se denomina control visual estático. Si la información está presente en todo momento, aun cuando el robot está en movimiento, se denomina control visual dinámico.
**Control visual directo:** Las consignas de error provenientes del análisis visual de las imágenes se introducen directamente en el control a los actuadores del robot. A este control comúnmente se suele llamar control servo visual [1].

#### 1.1.4.2. Sistemas de control visual, en función de la información visual

**Control visual basado en posición:** Este tipo de control se basa en la información tridimensional de la escena que el análisis de las imágenes proporciona. En la figura 1.1 se observa el diagrama en bloques correspondiente.

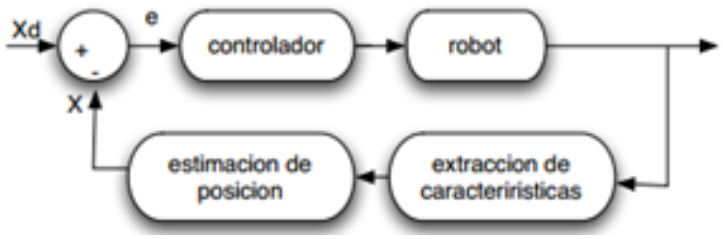


Figura 1.1: Esquema de control visual basado en posición [1].

**Control visual basado en imagen:** En este control la señal de error que se genera en la imagen se usa directamente como señal de control. Ver figura 1.2.

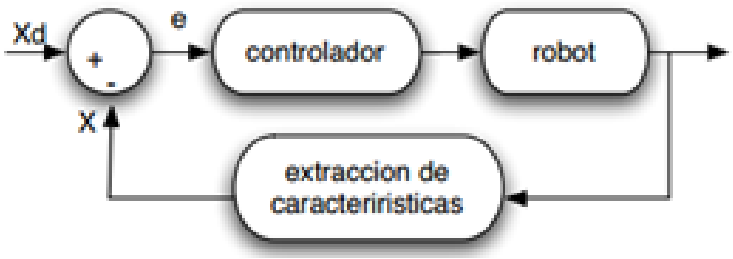


Figura 1.2: Esquema de control visual basado en imagen [1].

Existe también la combinación de los esquemas anteriores donde la tarea de control se realiza en el plano de la imagen y en el espacio de trabajo del robot, simultáneamente [1].

### 1.1.5. Visión por computadora y detección de objetos

La Visión por Computadora (VC) es un campo de la informática que enseña a las computadoras a ver. Usualmente la imagen es procesada primeramente en un nivel bajo para mejorar su calidad, removiendo ruido, etc. Luego, la imagen se vuelve a procesar, pero esta vez para detectar patrones y formas [1].

Si bien es cierto que no podemos esperar que la VC reproduzca exactamente la función del ojo humano, si hay aspectos en que se aprecia ventaja. Por ejemplo, la visión humana puede distinguir distancias relativas entre los objetos bastante bien, pero es pobre cuando de distancias absolutas se trata, lo contrario a VC. La VC es buena estimando diferencias absolutas, pero con una pobre resolución para diferencias relativas. La gran ventaja de nuestro cerebro es que puede encontrar más fácilmente relaciones entre conceptos de objetos que no son exactamente iguales, sin embargo, esto es toda una pesadilla para la VC puesto que abstraerse es mucho más complicado para la lógica.

Otro término comúnmente referido es la detección de objetos como la función responsable de descubrir e identificar la existencia de objetos o clases de estos. Puede ser considerado como método del procesamiento de imágenes para identificar objetos en las imágenes digitales. Una manera de hacer esto es simplemente clasificar los objetos por su color. Pero claro, esta aproximación de código de color no es 100 % confiable. Los experimentos muestran que las condiciones lumínicas son extremadamente determinantes.

#### 1.1.5.1. Algoritmos para la detección de objetos

Los sistemas de reconocimiento de objetos buscan objetos del mundo real partiendo de imágenes de dicho mundo, usando modelos de objetos que son conocidos a priori. La implementación de algoritmos para esta tarea en las máquinas es sorprendentemente difícil. El problema del reconocimiento puede ser definido como el problema de etiquetado basado en modelo de objetos ya conocidos. Formalmente, dado una imagen que contiene uno o más objetos de interés (y su fondo o background) y un conjunto de etiquetas correspondientes al conjunto de modelos de objetos conocidos al sistema, se debe asignar correctamente las etiquetas a las regiones o conjunto de estas en la imagen.

A continuación, se tratan algunos de los algoritmos más usados en los sistemas de detección de objetos.

**Comparación por plantillas o Template Matching**

Una plantilla (template) es una imagen pequeña o sub-imagen.


Figura 1.3: La imagen muestra una plantilla usada para buscar ocurrencias en una imagen [2].

El objetivo con este algoritmo es encontrar ocurrencias de esta plantilla en una imagen más grande. A grandes rasgos eso es todo, simplemente encontrar coincidencias de la plantilla en la imagen. Por ejemplo:

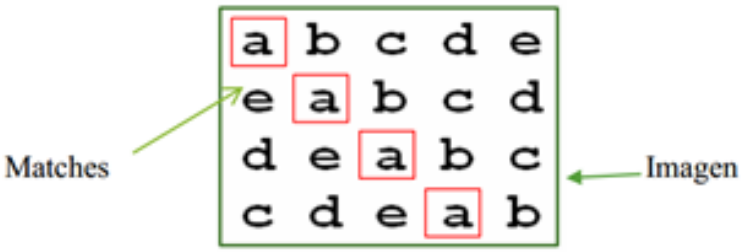


Figura 1.4: La imagen muestra la representación del funcionamiento del algoritmo de Comparación por Plantilla o Template Matching [2].

La comparación realizada se basa en la cercana proximidad entre los detalles de la plantilla y los de la imagen en cada una de sus coordenadas (i, j).

Limitaciones:

1. Las plantillas no tienen en cuenta variaciones en su escala o rotación.
2. Ligeros cambios de tamaño u orientación pueden causar problemas.

Para que sea eficaz se suelen usar varias plantillas para representar un mismo objeto, variando su tamaño y rotando la misma plantilla.

Pero por supuesto esto hace que el algoritmo sea una operación costosa computacionalmente. Especialmente si se busca en la imagen entera o se usan varias plantillas [2].

**Sustracción de fondo o Background Subtraction**

Para varias aplicaciones VC, Background Subtraction (BS) es una "rápida" manera de localizar objetos en movimiento en un video capturado por una cámara estática [3].

Este algoritmo es usualmente requerido para ser tan rápido y simple como sea posible. Consecuentemente, la mayoría de los métodos de etiquetado BS "en movimiento" se fija en las diferencias entre cada pixel, en un mismo tiempo t, y el fondo de la imagen. Esta solución ha sido exitosamente probada siempre y cuando la cámara utilizada permanezca rigurosamente estática y su fondo esté libre de ruido (ej: olas del mar, copas de los arboles movidas por el viento, etc.). De hecho, muchos videos con baja calidad, o ambiente ruidoso son causa de numerosos falsos positivos. Los falsos positivos pueden ser incluso inducidos por cambios de iluminación o temblor de la cámara, solo por nombrar algunos.

Aunque con algunas diferencias, la mayoría de las técnicas BS comparten un criterio en común, ellas asumen que la frecuencia de video observada está compuesta por un fondo estático delante del cual, objetos en movimiento son observados. Asumiendo también que cada objeto en movimiento está compuesto por colores diferentes del observado en el fondo, de lo contrario, se estaría en presencia del efecto camuflaje.

La siguiente figura ilustra acerca del resultado obtenido aplicando el algoritmo BS. Gracias al cual se puede detectar fácil y rápidamente los autos que transitan por una carretera, pudiendo a su vez contabilizarlos, medir su velocidad, el nivel de congestionamiento etc.

Figura 1.5: La imagen muestra el ejemplo de aplicar el método BS para detectar los autos que transitan por una carretera [3].

**Clasificador en cascada o Haar Cascade**

Un rasgo Haar-like considera regiones rectangulares vecinas como locaciones especificas en una ventana, suma las intensidades por pixel en cada región y calcula la diferencia entre esas sumas. Dicha resta es entonces usada para categorizar sub-secciones de la imagen [4].

Figura 1.6: La imagen muestra un ejemplo de reconocimiento realizado por el algoritmo Haar-Cascade [4].

El ejemplo más conocido puede ser el de detección de rostros humanos. Donde comúnmente, las áreas alrededor de los ojos son más oscuras que las áreas de las mejillas. Un rasgo Haar-like sería la colocación de dos áreas rectangulares vecinas entre las regiones de las ojeras y las mejillas.
Haar-Classification es una técnica de árbol donde en la fase de entrenamiento, una cascada basada en descartes por estadística es creada. Que sea por estadística significa que un clasificador potente es creado a partir de otros más débiles (en cascada). Y un clasificador débil, a su vez, es uno que toma la clasificación acertada en al menos el 50 % de los casos. Este avance hacia mejores clasificadores desde otros muchos menos potentes es posible incrementando el peso (penalización) en los ejemplos de errores de clasificación, significando que en la siguiente ronda o iteración de entrenamiento las hipótesis más acertadas serán escogidas.

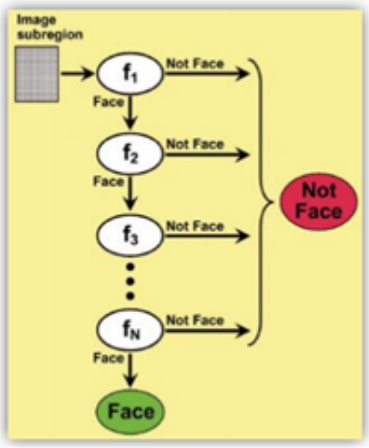


Figura 1.7: La imagen muestra la ejemplificación del trabajo en cascada del algoritmo. [4].

Que Haar-Classification use una cascada de descarte significa que el clasificador final en una gran cascada de muchos clasificadores más simples o sencillos, embebidos, y una región de interés debe entonces pasar por todos los estados o pasos de esa cascada para poder ser reconocido como tal. El orden de esos nodos es a menudo colocado según su dificultad de procesamiento, entonces muchos rasgos candidatos son reducidos y eliminados tempranamente, mejorando sustancialmente el tiempo de computación.

Existen cuatro métodos que usan el Haar-classification y están disponibles en la herramienta OpenCV: Real Adaboost, Discrete Adaboost, Logitboost y Gentle Adaboost.

La siguiente Tabla comparativa presenta algunas de las características más importantes entre los algoritmos mencionados, al ser implementados en un sistema embedido:

Tabla 1.1: La tabla muestra la comparación entre cuatro algoritmos al realizar la detección de objetos.

| Algoritmos | Invarianza ante cambios lumínicos | Invarianza ante cambios de escala | Invarianza ante de rotacion en el objeto | Coste computacional |
|---|---|---|---|---|
| Comparación por plantillas (template matching) | Baja | Baja | Baja | Alto |
| Substracción de fondo (background subtraction) | Baja | Baja | Baja | Bajo |
| Clasificador en cascada (Haar Cascade) | Alta | Alta | Alta | Alto |
| SURF (Speeded-Up Robust Features) | Alta | Alta | Alta | Alto |

#### 1.1.5.2. Herramienta OpenCV

Actualmente una de las herramientas más novedosas empleadas para realizar el procesamiento de imágenes y video es la herramienta de Visión por Computador de Código Abierto (OpenCV), esta

Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estérea y visión robótica.

La misma se encuentra insertada en sistemas de seguridad con detección de movimiento, sistemas de vídeo-vigilancia, aplicaciones de control de procesos donde se requiere reconocimiento de objetos y sistemas de seguimiento de objetos en vehículos autónomos, entre otros.

OpenCV ha sido diseñada para ser eficiente en cuanto a gasto de recursos computacionales y con un enfoque hacia las aplicaciones de tiempo real, por esta razón se hace una excelente solución para ser implementada en los sistemas embebidos para realizar la detección y seguimiento de objetos en vehículos autónomos [1], [2].

La librería OpenCV se encuentra presente en los lenguajes C y C++. Cuenta con un desarrollo activo en interfaces para Python, Ruby, Matlab y otros lenguajes y es compatible con Linux, Windows y Mac OS X.

OpenCV tiene una estructura modular, lo que quiere decir que que el paquete completo incluye varias bibliotecas compartidas. los módulos que incluyen son:

**Funcionalidad básica (Core Functionality)**: En este módulo se definen estructuras básicas, como son la matriz multidimencional 'Mat' y funciones básicas utilizadas por otros módulos.

**Procesamiento de imagen (Image Processing)**: En este módulo se incluyen filtros de imagen lineales y no lineales, transformaciones geométricas de imagen, conversiones de espacios de colores, histogramas, etc.

**Video**: Un módulo para el análisis de video que incluye estimación de movimiento, sustracción de fondo y algoritmos de seguimiento de objetos.

**Calibración en 3D (Calib3d)**: Algoritmos básicos de geometrías de multiples vista, calibración simple y estéreo, estimación de la posicion de objetos, algoritmos de correspondencia estéreo y elementos de reconstrucción 3D.

**Características 2D (Features2d)**: detectores de características salientes y descriptores de coincidencia.

**Detección de Objetos (Objdetect)**: Algoritmos de detección de objetos e instancias de clases predefinidas (ej: ojos, caras, personas, coches, etc).

**Highgui**: Interfaz de uso fácil para simplificar la UI(Interfaz de usuario, del inglés: User Interface).

**Videoio**: Interfaz de uso fácil para capturar de video y códec (codificador-decodificador) de video.

**Gpu**: Algoritmo de diferentes módulos de OpenCV con aceleración de GPU (Unidad de

Procesamiento de Gráficos, del inglés: Graphics Processing Unit).

#### 1.1.5.3. Lenguaje de programación Python

Python es un lenguaje de programación desarrollado como proyecto de código abierto de alto nivel del tipo scripting, está diseñado para ser fácil de leer y simple de implementar. Es administrado por la empresa Python software Foundation. Este lenguaje permite varios estilos de programación: programación orientada a objetos, programación imperativa y programación funcional.
Esta considerado un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad. Su popularidad se debe principalmente a que es gratuito, además de la cantidad de librerías que contiene, datos y funciones incorporadas en el propio lenguaje que ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero.
Fue creado por Guido van Rossum y su nombre se debe a la afición de su creador a los humoristas británicos Monty Python [21], [18].

### 1.1.6. Sistema embebido Raspberry Pi

Raspberry Pi, es una placa computadora (SBC) de bajo coste, es un ordenador de tamaño reducido, desarrollado en el Reino Unido por la Fundación Raspberry Pi (Universidad de Cambridge) en 2011, con el objetivo de estimular la enseñanza de la informática en las escuelas, aunque no empezó su comercialización hasta el año 2012.
Este sistema embebido se caracteriza por su bajo precio, presenta un bajo consumo de energía. Son usados para realizar tareas especificas con ejecución concurrente y presentan la capacidad de anexar hardware (cámara, sensores infrarojos, teclados, pantallas táctiles). Todo esto sumado le proporciona a este dispositivo una gran adaptabilidad a practicamente cualquier aplicación [5].

#### 1.1.6.1. Sistema operativo (S.O)

El sistema operativo utilizado en la Raspberry Pi, es un S.O basado en Debian, con distribución de Linux, optimizado para el hardware de la Raspberry Pi. Este S.O inckuye mas de 35000 paquetes que permiten el obtener un mayor rendimiento posible. En concreto se se utilizará el sistema operativo Raspbian versión Jessi de agosto de 2107, última versión estable de este S.O.

Capítulo 2

# DESARROLLO DEL PROYECTO

El desarrollo del sistema se encuentra integrado por las siguientes etapas:

1. Realizar la instrumentación del vehículo (Cámara (visión), motores (para el movimiento), sistema embebido (Raspberry Pi 3)).
2. Diseño e implementación de una metodología de detección e identificación del objeto de interés utilizando el Procesamiento digital de imágenes con OpenCV.
3. Implementar el algoritmo en el vehículo instrumentado.
4. Validar experimentalmente el diseño de algoritmo de monitoreo en tiempo real.

## 2.1. Instrumentación del vehículo

El vehículo terrestre desarrollado es de diseño sencillo. Este se ha instrumentado con los bloques básicos para que pueda moverse por el entorno. Estos bloques son:
- Sensor (cámara).
- Sistema de movimiento.
- Sistema computador.
- Alimentación.
- Chasis del vehículo autónomo.

### 2.1.1. Sensor (cámara)

**Cámara Raspberry Pi**

La cámara oficial de Raspberry Pi Rev 2.1 es la plataforma utilizada para la adquisición de las imágenes. La cámara es de 8 Mpíxeles, esta se conecta directamente en el conector CSI de la Raspberry Pi. El conector CSI (del inglés: Camera Serial Interface, o Interfaz de Cámara Serie) es un estándar definido por la MIPI (Mobile Industy Processor Interface) que permite la conexión serie directa de la cámara con el procesador. Esto permite conectar cámaras con muy alta

resolución, siempre que el procesador permita el tratamiento de los datos [22], [23]. La siguiente imagen muestra el aspecto de la cámara de Raspberry Pi.

Figura 2.1: La imagen muestra el aspecto exterior de la cámara Raspberry Pi.

Las ventajas de emplear la cámara oficial es la compatibilidad y facilidad de instalación que presenta esta cámara con respecto a otras cámaras USB. Gracias al interfaz de tipo CSI, las velocidades de adquisición y procesamiento de imágenes son mucho mas rápidas que otras tipos de cámaras. La desventaja fundamental es el precio el cual ronda los 950 pesos, que resulta más caro que otras cámaras.

### 2.1.2. Sistema de movimiento

El movimiento del vehículo es llevado a cabo a través de dos servomotores. Estos son los encargados de proporsionarle movilidad al vehículo, de manera tal que este sea capaz de seguir al objeto identificado en su movimiento y posicionarse frente al mismo.
Las características técnicas de los servomotores empleados así como una vista externa de los mismos se presentan a continuación:

Tabla 2.1: La tabla muestra las características técnicas de del servomotor MG996R utilizado para el movimiento del vehículo.

| | |
|---|---|
| Tensión de operación (V) | 4.8 - 7.2 |
| Tipo de engranaje | metálicos |
| Cable conector (mm) | 300 |
| Torque a 4.8V | 9 kg-cm (317.5 oz-in) |
| Torque a 6V | 12 kg-cm (423.3 oz-in) |
| Peso (gramos) | 55 |
| Dimensiones (mm) | 40x19x43 |
| Velocidad de operación | 0.17seg / 60 grados (4.8V,sin carga) |
| Velocidad de operación | 0.13seg / 60 grados (6.0V sin carga) |
| Rango de temperatura (grados Celsius) | 0 a 55 |


Figura 2.2: La imagen muestra el aspecto exterior del servomotor MG996R utilizado en el vehículo.

### 2.1.3. Sistema computador

#### 2.1.3.1. Características de la Raspberry Pi-3

El modelo utilizado para el desarrollo del proyecto es una Raspberry Pi-3, ya que este modelo presenta excelentes características con respecto a los modelos Raspberry Pi-1 y Raspberry Pi-2; entre las que podemos mencionar la velocidad de reloj del microprocesador (CPU) y el tamaño de memoria de acceso aleatorio o memoria RAM. Esta dos características son vitales en la adquisición y procesamiento de imagen, ya que estos reaquieren de gran potencia de computo para garantizar la fuidez en el video [5].

Tabla 2.2: Tabla comparativa de las características para los modelos básicos de Raspberry Pi en el mercados.

| Características | Raspberry Pi-1 | Raspberry Pi-2 | Raspberry Pi-3 |
|---|---|---|---|
| Precio (pesos) | 720 | 720 | 990 |
| Tamaño (cm) | 8.6x5.4x1.7 | 8.6x5.4x1.7 | 8.6x5.4x1.7 |
| Peso (gramos) | 45 | 45 | 45 |
| Memoria (MB) | 512 | 512 | 1000 |
| Velocidad de reloj (MHz) | 700 | 900 | 1200 |
| CPU | ARM11 ARMV6 700 MHZ | ARM11 ARMv7 ARM Cortex-A7 | ARM Cortex-A53 |
| Núcleos del CPU | 1 | 4 | 4 |
| SoC | BROADCOM BCM2835 | BROADCOM BCM2836 | Broadcom BCM2387 |
| Flash | tarjeta SD (2-16) | tarjeta microSD (2-32) | tarjeta microSD (2-32) |
| S.O. | Distribuciones Linux | Distribuciones Linux | Distribuciones Linux |

A continuación se muestran las caracteríticas técnicas de la Raspberry Pi-3 en forma más detallada:

1. SoC: Broadcom BCM2837 (CPU + GPU + DSP + SDRAM + PUERTO USB)
   - CPU: Chipset Broadcom BCM2387, 1.2GHz, 64 bits, 4 núcleos ARM Cortex-A53.
   - GPU: Dual Core VideoCore IV Multimedia Co-procesador. Proporciona Open GL ES 2.0, OpenVG acelerado por hardware, y 1080p30 H.264 de alto perfil de decodificación.
   - Memoria RAM: 1GB LPDDR2.
   - Puertos USB: 4 USB 2.0

2. Conectividad: Ethernet socket Ethernet 10/100 BaseT, 802.11 b / g / n LAN inalámbrica y Bluetooth 4.1 (Classic Bluetooth y LE).

3. Entrada de video: Permite instalar un módulo de cámara desarrollado por la Raspberry Pi Fundation, conocido como la Picámara. Conector de la cámara de 15 pines cámara MIPI interfaz en serie (CSI-2).

4. Salidas de video: RCA compuesto (PAL y NTSC), HDMI rev 1.3 y 1.4 y pantalla de visualización Conector de la interfaz de serie (DSI), con conector de 15 vías plana flex cable con dos carriles de datos y un carril de reloj

5. Salida de audio: jack de 3,5 mm de salida de audio, HDMI.

6. Almacenamiento integrado: microSD

7. Conector GPIO: 40-clavijas de 2,54 mm (100 milésimas de pulgada) de expansión de 2x20 tira, proporciona 27 pines GPIO, así como 3,3 V, +5 V y GND líneas de suministro.

8. Consumo energético: 800mA (4W).

9. Fuente de alimentación: 5V vía Micro USB o GPIO Header.

10. Sistemas operativos soportados: Debian (Raspbian), fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux, Windows 10 optimizado.

Figura 2.3: La imagen muestra el aspecto exterior del sistema embebido Raspberry Pi-3.

### 2.1.4. Alimentación

Para que tanto los motores como el sistema embebido puedan funcionar correctamente y el vehículo se mueva libremente, se necesita de una fuente de alimentación. Debido a que el vehículo será autónomo

y se movera en espacios sin obstáculos, se utiliza como fuente de energía una batería de marca GearPower modelo GMP8k. Las características técnicas de la misma se presentan a continuación, así como su aspecto exterior.

Tabla 2.3: La tabla muestra las características técnicas de la batería utilizada para la alimentación eléctrica del vehículo.

| | |
|---|---|
| Marca | GearPower |
| Modelo | GMP8k |
| Tipo de batería | Ion de Litio |
| Conector de alimentación | USB/microUSB |
| Capacidad Amperes/horas | 8 A/h |
| Tensión de alimentación (V) | DC 5V / 2A |
| Tensiones de salidas (V) | DC 5V/1A y 5V/2.4A |
| Peso (gramos) | 320 g |
| Dimensiones (cm) | 2.1x10.4x7.6 |

La figura 2.4 muestra el aspecto exterior de la batería empleada:

Figura 2.4: La imagen muestra el aspecto de la batería GearPower modelo GMP8k empleada en el vehículo.

### 2.1.5. Chasis de vehículo autónomo

El chasis del robot desarrollado se corresponde con uno ya en existencia, el mismo presenta el siguiente aspecto:

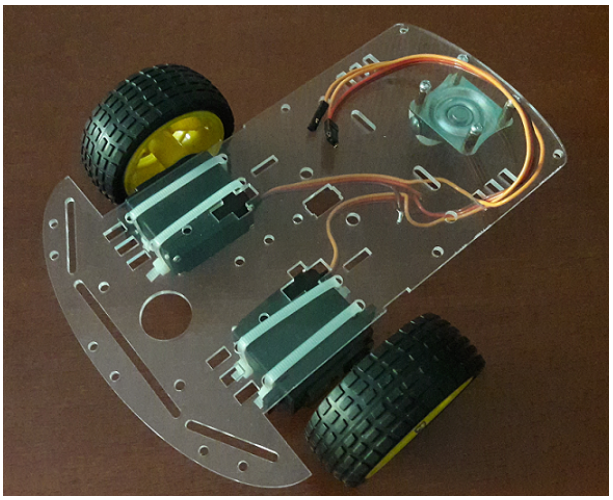

Figura 2.5: La imagen muestra el aspecto del chasis del vehículo.

El chasis del vehículo está soportado sobre dos ruedas motrices y una rueda libre. las ruedas motrices son accionadas por un servomotor cada una.

## 2.2. Diseño e implemetación del algoritmo para realizar la detección del objeto

Existen varias técnicas de visión por computadora para realizar la detección y el seguimiento de objetos según las funcionalidades que se requieren. Para lograr el funcionamiento adecuado y eficaz del algoritmo se hace necesario evaluar las características requeridas por el proyecto. Las características básicas que debe cumplir el proyecto son:

1. El algoritmo sea lo suficientemente eficiente para ser implementado en el sistema embebido Raspberry Pi-3 y capaz de ser ejecutado con los recursos computacionales disponibles.

2. Realizar la detección del centro de masa del objeto a seguir con el objetivo de que el vehículo se mueva, posicionándose siempre al frente del objeto.

3. La detección del objeto se realice en tiempo real.

4. El algoritmo debe ser capaz de ejecutarse en entornos dinámicos.

El algoritmo contruido se desarrolla en dos etapas fundamentales, una la detección y seguimiento del objeto y la segunda el movimiento del vehículo instrumentado. esto se observa a traves de este

diagrama de flujo en el cual se muestra de manera general el algoritmo desarrollado.

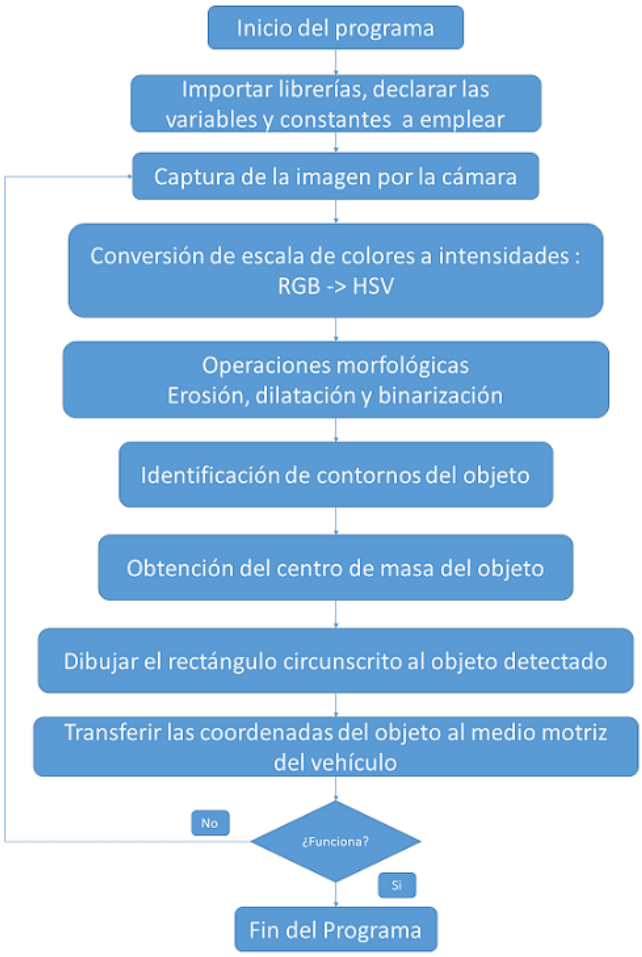


Figura 2.6: La imagen muestra el flujo del algoritmo general.

### 2.2.1. Detección del objeto

Exiten técnicas muy diversas para realizar la detección de objetos. Generalmente las líneas que siguen estos algoritmos es desarrollarse para un fin específico, por ejemplo en la detección de objetos por un color determinado, detección de caras y detección de personas. Esto permite que se simplifique considerablemente el algoritmo y se desminuya el tiempo de procesamiento y recursos que emplea el algoritmo.

Algunas de las técnicas de visión por comutadora son:

1. Detectores de regiones: Se utiliza para generar rasgos similares que sean de interés. Para esto los algoritmos verifican cada punto de la imagen con el fin de identificar estas regiones

similares. estos algoritmos son costosos computacionalmente, ya que recorren cada píxel y emplean mucho tiempo en ello, pero ofrece excelentes resultados.

2. Substracción de fondo: Se basa en eliminar el fondo de la imagen, destacando los objetos que no permanecen fijos de un cuadro al siguiente. Es mu eficiente en sistemas de visión que se encuentren fijos.

3. Apendizaje automático (Machine learning): Este permite a las computadoras aprender sin haber sido explícitamente programada. Esto se realiza mostrando de una serie de ejemplos al algoritmo y éste posteriormente es capaz de identificar objetos similares a esos ejemplos. Presenta el inconveniente de la necesidad de aportar ejemplos del objeto que se desea identificar y la potencia de procesamiento requerida para analizar los datos de los ejemplos y compararlos con la captura de la cámara.

4. Segmentación: Es el proceso de particionar una imagen digital en multiples regiones, que no son mas que un conjunto de píxeles tambien llamados superpíxeles. El objetivo de esto es simplificar la imagen para facilitar el análisis. Como resultado de la aplicación de la función de segmentación a una imagen se obtienen regiones con características similares. Las regiones adyacentes son significativamente diferentes en cuanto a sus propiedades características, como pueden ser el color, la intensidad, la textura, etc. Existen varios algoritmos de segmentación, proporcionando cada uno resultados diferentes. Esto sse debe a que no existe una manera general para segmentar una imagen y en función de los resultados buscados pueden usarse diferentes técnicas o incluso conbinar varias de ellas. Algunas de estas técnicas o algoritmos son: Métodos de agrupamiento (Clustering), Métodos basados en histogramas, Detección de bordes, Método del valor umbral (Threshold), Segmentación basada en modelos, etc.

El método a empleado en nuestro proyecto es el del valor umbral (Threshold), ya que el coste computacional es menor que el de los métodos anteriores y permite ser implementado en entornos dinámicos.

En este caso concreto se va a utilizar como característica de identificación el color de los objetos. Para ello es necesario que el objeto a identificar tenga un color diferente al del fondo, con el fin de separar el objeto de éste y visualizar únicamente el objeto. A continuación se muestran las diferentes funciones en las que se divide el programa:

### 2.2.2. Conversión de escala de colores

Una vez que la imagen ha sido capturada por la cámara es necesario comenzar con el preprocesado de ésta. Cabe destacar que la captura de video por defecto de OpenCV se realiza mediante la escala de color BGR. Esto es, cada imagen se representa usando 3 matrices, una para cada uno de los 3 colores de la escala (Blue, Green, Red – Azul, Verde, Rojo). Los elementos de la matriz representan un píxel y sus valores van desde 0 hasta 255, siendo 0 el color negro y 255 el color al que represente esa matriz. La siguiente figura presenta un ejemplo de una imagen a color representada mediante las 3 matrices de BGR.

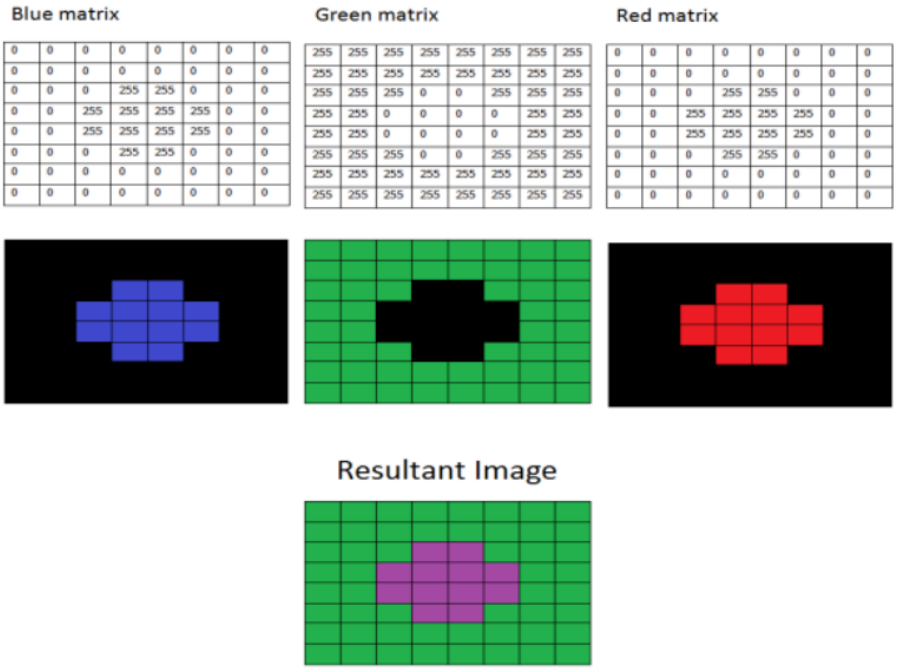


Figura 2.7: La figura muestra la representación de una imagen mediante matrices RGB [5].

En el código se ha realizado la conversión de los datos obtenidos en BGR al modelo de color HSV (del inglés: Hue, Saturation, Value – Matiz, Saturación, Valor) [6]. Este modelo de color nos permite una segmentación de la imagen mucho más eficiente, ya que permite tener en cuenta las condiciones de luminosidad de la imagen y la intensidad del color, que con la escala BGR no se podría realizar. Esto se consigue separando la componente "chroma" (que aporta la información del color, es la única componente que se utiliza en BGR) de la componente "luma" (aporta la intensidad del color). Resulta muy útil en visión artificial por varias razones, tales como conseguir un sistema más robusto frente a los cambios de luz o eliminar sombras.

Cabe destacar que existen más escalas de color que también separan el color de la intensidad tales

como YcbCr, Lab, etc. Pero en este Trabajo se ha utilizado HSV porque la conversión BGR-HSV está muy extendida y es fácilmente implementada, ya que se encuentra disponible en la biblioteca de OpenCV.

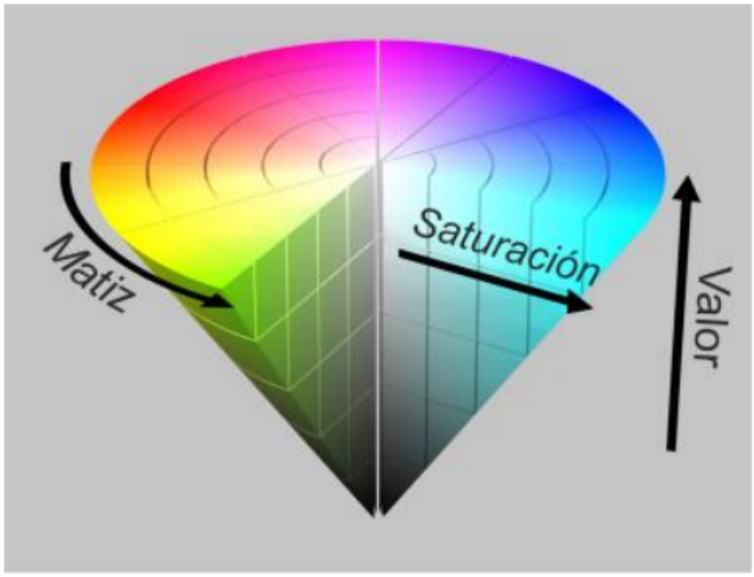


Figura 2.8: La imagen muestra la representación gráfica del modelo de color HSV [6].

El modelo de color HSV se representa usualmente en forma de cono invertido utilizando coordenadas cilíndricas y consta de las siguientes componentes:

1. Hue (Matiz): Representado como un ángulo entre 0 y 360º. Cada uno de los valores corresponde a un color. OpenCV realiza la conversión de este valor a la escala de valores enteros: 0-180, ya que se trata de una imagen de 8 bits y se aplica la conversión siguiente: H - H/2 (360 - 180).

2. Saturation (Saturación): Se define como la distancia al eje de blanco-negro. Los valores se encuentran entre 0 y 100 %. OpenCV realiza la conversión a la escala de valores enteros: 0-255.

3. Value (Valor): Definido como la altura en el eje blanco-negro. Los valores se encuentran entre 0 y 100 %, en el que 0 siempre es negro (vértice del cono). OpenCV realiza la conversión a la escala de valores enteros: 0-255.

### 2.2.3. Operaciones morfológicas

La segunda funcionalidad que incluye este proyecto es la función de operaciones morfológicas. Para este caso se va a realizar la segmentación de la imagen capturada mediante la conversión de ésta a una imagen binaria (blanco y negro).

Para realizar esta transformación se utilizará una operación de procesado de imagen que permite convertir una imagen en color a una binaria en función de los parámetros que se le asignen. Si el valor del píxel analizado es mayor que cierto umbral o "Threshold", se le asignará el color blanco al píxel; en caso contrario, se le asignará el color negro. De esta forma se obtiene en color blanco el color que hayamos seleccionado y el resto en color negro, lo que facilita la separación del objeto del fondo [24], [25]. Esta operación, en la cual se basa el programa aquí realizado, es la función de OpenCV: inRange. En este caso se realiza la operación con 3 canales de entrada y la función matemática que aplica el comando inRange es la mostrada a continuación:

$$imgBin = minH \leq imgHSV_0 \leq maxH \& minS \leq imgHSV_1 \leq maxS \& minV \leq imgHSV_2 \leq maxV \tag{2.1}$$

Para realizar esta operación y permitir seleccionar el color deseado por el usuario se ha creado una interfaz que, mediante la utilización de "Trackbars". Esta interfaz permite modificar los parámetros de la función inRange, o sea, los rangos de valores de color HSV deseado. La interfaz mencionada se muestra en la figura:

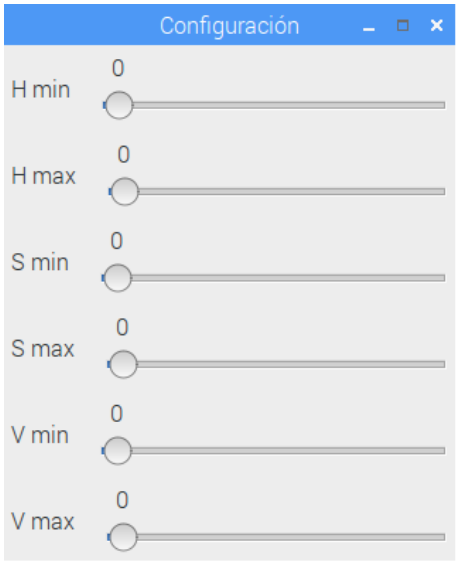


Figura 2.9: La imagen muestra el interfaz gráfica deseñada para el control de color HSV.

Los valores de los parámetros HSV se definen en la ventana de control en función de las condiciones de iluminación para que la detección sea más precisa. En este caso tienen valores de: H: 110 - 130; S: 50 - 255; V: 50 - 255. Los objetos rojo, amarillo, han han sido obviados, ya que no cumplen los requisitos de los parámetros de Threshold seleccionados. En este caso el color seleccionado para las pruebas es el color azul.

Además, se utilizan dos funciones muy útiles en visión por computadora que permiten reducir en gran medida el ruido de la imagen. El ruido es la variación aleatoria (no presente en la imagen original) de la información del brillo o color en imágenes digitales, producido por el dispositivo de captura de la imagen y que no se corresponde con la realidad. Es necesario reducir en la medida de lo posible este fenómeno para obtener una imagen fiel a la realidad. Para ello se utilizarán las funciones erode (erosión) y dilate (dilatación).

La función "erode" del OpenCV actúa de la siguiente forma:

$$imgBin(x,y) = {}_{(x',y')}min_{(x',y')\neq 0} imgBin(x+x', \quad y+y') \tag{2.2}$$

Para aplicar tanto la función de erosión como de dilatación es necesario definir previamente un elemento estructurante que se encargará de realizar esta operación.

El elemento estructurante es un conjunto de forma conocida que permite realizar la extracción de estructuras geométricas en los conjuntos sobre los que se opera. Usualmente se emplean círculos o cuadrados. En este trabajo se utiliza un cuadrado de 3x3 píxeles para realizar la erosión y un cuadrado de 8x8 píxeles para realizar la dilatación. Es importante realizar la erosión con un elemento estructurante menor para evitar reducir mucho el tamaño del objeto y la dilatación con un elemento estructurante mayor para homogeneizar los objetos. Con esta combinación de valores es con la que se han obtenido los mejores resultados.

Una vez creado el elemento estructurante se realiza la erosión de la imagen binaria, lo cual supone tomar el valor mínimo de la imagen en el entorno vecino definido por este elemento. La erosión en este caso se utiliza para eliminar pequeños objetos que no sean de interés en la identificación del objeto y puedan llevar a error. La ecuación que describe la función "dilate" es:

$$imgBin(x,y) = {}_{(x',y')}max_{(x',y')\neq 0} imgBin(x+x', \quad y+y') \tag{2.3}$$

Esta es la función opuesta a la erosión, ya que toma el valor máximo de la imagen en el entorno vecino definido por el elemento estructurante. La dilatación se utiliza aquí para agrandar los objetos, cerrando agujeros y grietas.

Estas funciones, cuando actúan de forma conjunta, permiten realizar las operaciones de Apertura y Cierre Morfológicos. La apertura morfológica es la utilización en cascada de erosión y dilatación para subsanar el efecto de disminución de tamaño de los objetos que causa la erosión. Por otro lado, el Cierre Morfológico es la aplicación en cascada de una dilatación y una erosión, con la finalidad de reducir el efecto de ensanchamiento de los objetos causado por la dilatación.
La no aplicación de las operaciones morfológicas conduce al análisis de más píxeles de la imagen, que da a lugar a resultados incorrectos en la detección de objetos.

### 2.2.4. Identificación de contornos

Tras la aplicación de las operaciones anteriores se obtiene una imagen binaria, con el objeto que se desea seguir en color blanco y el resto de la imagen en color negro. A primera vista parece inmediato realizar una función que siga el objeto blanco en el fondo negro en esta imagen binaria, pero el problema aparece cuando el objeto se encuentra en movimiento, ya que las matrices con las que hay que tratar son de grandes dimensiones. Por tanto, es necesario aplicar una operación intermedia que reduzca la cantidad de información que hay que tener en cuenta para poder realizar un seguimiento del objeto en tiempo real y con la mayor simplicidad posible de cálculo.
OpenCV incorpora una funcionalidad que permite simplificar esta tarea mediante la reducción de una imagen original a otra nueva formada sólo por los contornos de la imagen original. Al tratarse de una imagen binaria proporciona unos resultados excelentes que reducen de forma cualitativa la cantidad de datos a procesar en el seguimiento de un objeto. La función de la biblioteca de OpenCV que se va a utilizar es la siguiente: *findContours*. Esta función está basada en un algoritmo desarrollado por Satoshi Suzuki, y obtiene los bordes de los objetos que se encuentran en una imagen binaria [26].
La función consta de los siguientes parámetros: cv2.findContours ( image (InputOutputArray), contours (OutputArrayOfArrays), hierarchy (OutputArray), mode, method )
-**Image**: Imagen de entrada (binaria) a la que se va a aplicar la detección de contornos.
-**Contours**: Es un vector de contornos en el que cada contorno es un vector de puntos.
-**Hierarchy**: Contiene información sobre la topología de la imagen, es decir, cómo se relacionan entre sí los contornos detectados. Contiene tantos elementos como el número de contornos detectados.
-**Mode**: Permite definir un modo de recuperación de contornos. Existen modos que permiten eliminar las relaciones de jerarquía, o establecer nuevas relaciones que identifiquen sólo los

contornos externos, etc. En el caso de seguimiento del objeto propiamente dicho se va a aplicar la operación en el modo "RETR-CCOMP" que permite obtener los contornos con sus relaciones jerárquicas sin modificar. Con ello se podrá llevar a cabo la identificación del centro de masas con la función moments que se explicará más adelante.

-**Method**: Método de aproximación de contornos. Dispone de varios modos de trabajo, como por ejemplo: no realizar aproximación y almacenar todos los puntos del contorno, realizar una aproximación simple comprimiendo horizontal y verticalmente para reducir los puntos del contorno, o incluso aplicar algoritmos de aproximación muy complejos. En este caso se aplicará el modo "CHAIN-APPROX-SIMPLE", que ofrece resultados muy buenos y con el mínimo de datos del contorno posible. De esta forma se consigue reducir en gran medida los cálculos que el procesador tiene que llevar a cabo, fomentando la fluidez del programa de seguimiento.

### 2.2.5. Seguimiento de Objetos

De forma sencilla, se puede definir el seguimiento de objetos como un problema de estimación de la trayectoria de un objeto en una imagen plana según se mueve por la escena. Esto es, un algoritmo cuyo fin sea seguir objetos debe mostrar cuál es la situación del objeto que está siguiendo en cada frame del video.

Para el desarrollo de este proyecto se decidio utilizar el centro de masas de los objetos como punto principal a seguir. Además, se ha definido una funcionalidad extra que permite calcular de forma simple el tamaño del objeto dentro de la escena. Por tanto, cada objeto que haya sido identificado tendrá asociado su centro de masas (en forma de coordenadas dentro de la imagen) y su tamaño (de forma encuadrada, ancho y alto).

La utilización de ambas funcionalidades a la vez permite evitar falsos positivos, que puedan ser debidos a objetos de muy pequeño tamaño o muy grandes, y realizar un seguimiento eficaz, gracias el seguimiento del centro geométrico de cada objeto y no a este por completo [27].

### 2.2.6. Obtención del centro de masa del objeto

Para obtener el centro de masas del objeto que se desea seguir se utiliza la función *moments* de OpenCV, la cual había sido mencionada previamente. Esta función devuelve los momentos de área de hasta tercer orden de un polígono en una estructura de tipo "Moments". Esta estructura almacena

todos los momentos calculados mediante la ecuación:

$$m_{ji} = \sum_{x,y}^{x_n,y_n} (array(x,y) * x^j * y^i \tag{2.4}$$

Los valores obtenidos a partir de la ecuación 2.4 se utilizan para calcular el centro de masas del objeto en función de sus momentos de área, utilizando las fórmulas siguientes:

$$\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}} \tag{2.5}$$

El momento espacial "$m_{00}$" es el área encerrada por el contorno, ya que es la suma de todos los contornos que componen el objeto. Y así es como queda definido el centro de masas por sus coordenadas x e y.

Cabe destacar que el origen de coordenadas por defecto en OpenCV se encuentra en la esquina superior izquierda, y, por tanto, las coordenadas x e y obtenidas del cálculo anterior parten con respecto a esta base.

### 2.2.7. Transferir coordenadas del objeto detectado al medio motriz del vehículo

A través del centro de masa se obtienen las coordenadas de ubicación del objeto detectado. El primer paso realizado es obtener un origen de coordenadas para la imagen, para posteriormente encontrar el error de la posición del objeto con respecto al origen de coordenadas de la imagen y posteriormente procesar esta información para enviar los pulsos necesarios a los servomotores. El origen de coordenadas se obtuvo dividiendo el ancho y el largo de la imagen capturada a la mitad, ya que al inicio del programa se estableció el tamaño de la imagen, con estos valores se ubican el centro de la imagen que funjira como el origen de coordenadas.

Después se realiza un ciclo en el programa de python, estableciendo una zona muerta, esta garantiza que si el objeto detectado sale de este rango el vhículo se movera en esa dirección hasta que la posición del objeto vuelva a situarse dentro de esta zona. Para mayor ilustración se muestra la figura 2.10 en la cual se muestran las líneas realizadas para obtener el centro de la imagen, asi el recuadro que comprende la zona muerta o zona de inmovilidad del vehículo.

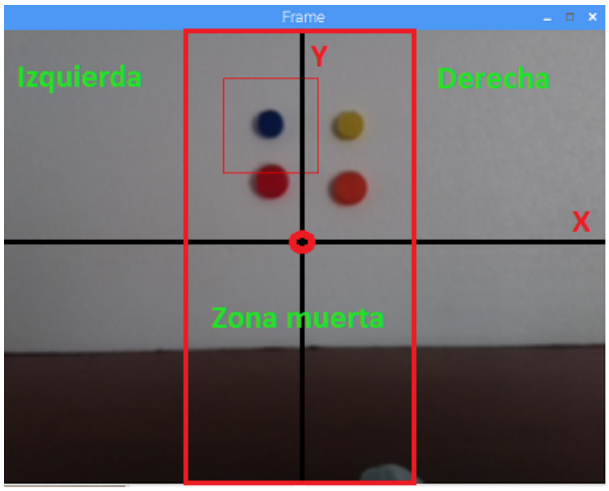


Figura 2.10: La imagen muestra como queda ubicado el origen de coordenadas y el área de la zona muerta.

Una vez obtenida el valor del error se compara con el valor umbral de la zona muerta este valor fue fijado en 60 puntos para el eje de las abscisas, si este superaba el valor el vehículo se movera hacia la derecha hasta que el objeto se ubique en la zona muerta o en caso de que la comparación del error con el valor umbral sea un valor negativo quiere decir que el objeto se encuentra moviéndose hacia la izquierda.

Para el movimiento del vehículo se emplean los pines 17 y 18 de la Raspberry Pi, a través de los cuales se envían los pulsos de control a los servomotores.

Desde el programa se envia un valor numérico en forma de cadena de caractéres a un segundo programa llamado "$pi - blaster$", este es el encargado de convertir esa cadena de caractéres a los pulsos que son enviados cada servomotor. En este caso la velocidad de cada motor sera invariable y constante.

El código desarrollado se muestra en el anexo 1, para mayor comprensión dirigirse a este y revizar esta parte del algoritmo.

# Resultados

### 2.2.8. Contrucción e intrumentación del vehículo

La instrumentación del vehículo esta compuesta por una Raspberry Pi-3, la cámara Raspberry Pi oficial rev 2.1 de 8 Megapíxeles, los servomotores MG996R y todo el sistema se encuentra alimentado por una batería de Ion de Litio con una capacidad de 8000 mA/h y tensiones de salida de 5V de corriente directa capaz de proporcionar corrientes de 1A y 2.4A respectivamente. Esta batería consta de dos conectores de tensiones de salida. El conector que proporciona 5V/1A es empleado en la alimentación del sistema embebido y el conector de 5V/2.4A es utilizado para alimentar a los servomotores, ya que estos presentan un mayor consumo de energía, el vehículo construido e instrumentado es el mostrado en la figura 2.11.

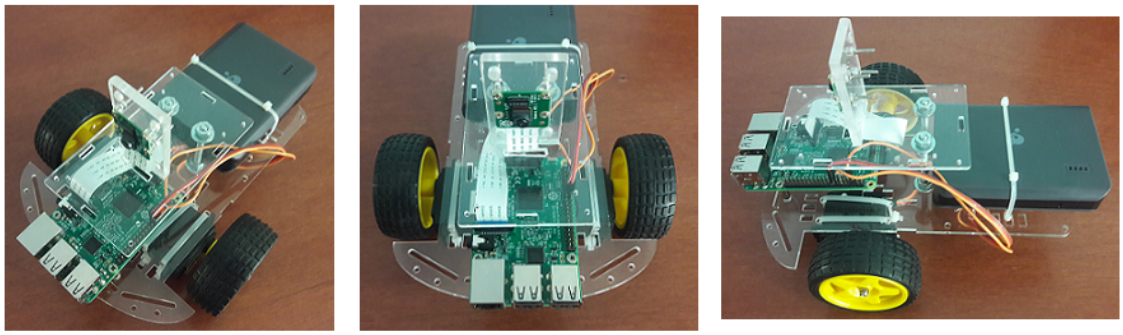

Figura 2.11: La imagen muestra el aspecto final del vehículo autónomo instrumentado.

### 2.2.9. Algoritmo diseñado

A través de la Segmentación, la cual fue la técnica de visión por computadora seleccionada y dentro de esta el método del Valor umbral, se logra realizar la detección y el seguimiento del objeto de color azul.

Este metodo presentó un consumo computacional bajo, ademas de presentar un buen desempeño en

la detección del objeto de color azul.

A continuación se presentan los resultados obtenidos al implementarse cada paso del algoritmo.

**Obtención de la imagen y conversión de RGB a HSV**

Después de obtener la imagen capturada por la cámara se realizó la conversión del modelo de color RGB a HSV. Esto permitió obtener un sistema de visión artificial más robusto ante cambios lumínicos y sombras, además de un ahorro considerable en cuanto al costo computacional, ya que el modelo HSV permite seleccionar un valor de matiz, saturación y brillo específicos, lo que se traduce en un menor análisis de píxeles y esto a su vez en un menor tiempo de procesamiento de la imagen. El resultado de realizar la conversión se presenta en la figura 2.12.

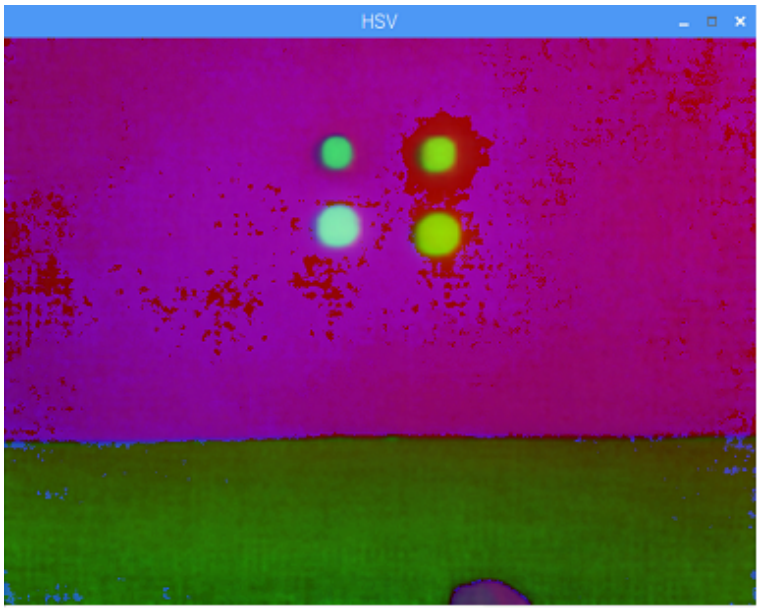


Figura 2.12: La imagen muestra el resultado obtenido al convertir del modelo de color RGB a HSV.

Los valores utilizados para realizar la detección y seguimiento del objeto de color azul se muestran en la figura 2.13, a través de este interfaz se establecieron los valores de Matiz, Saturación y Brillo.

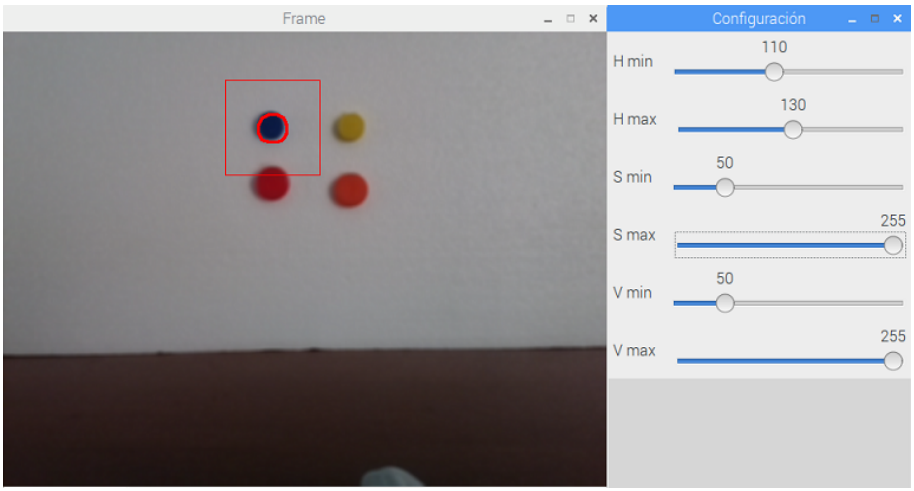


Figura 2.13: La imagen muestra selección del color utilizando un rango de valores máximos y mínimos de HSV.

### Binarización de la imagen

Con la binarización de la imagen se logró separar el objeto del fondo. Esto nos permitió darle un mejor tratamiento a la imagen, ya que esto reduce conciderablemente los píxeles a analizar. En la siguiente figura se muestra los resultados obtenidos con la binarización de la imagen.


Figura 2.14: La imagen muestra la binarización obtenida al aplicar técnicas morfológicas.

### Filtrado de la imagen

El filtrado de la imagen se realiza primero empleando un filtro de erosión y después otro de dilatación, estos permitieron eliminar los ruidos de la imagen, producto a fundamentalmente al dispositivo de captura.

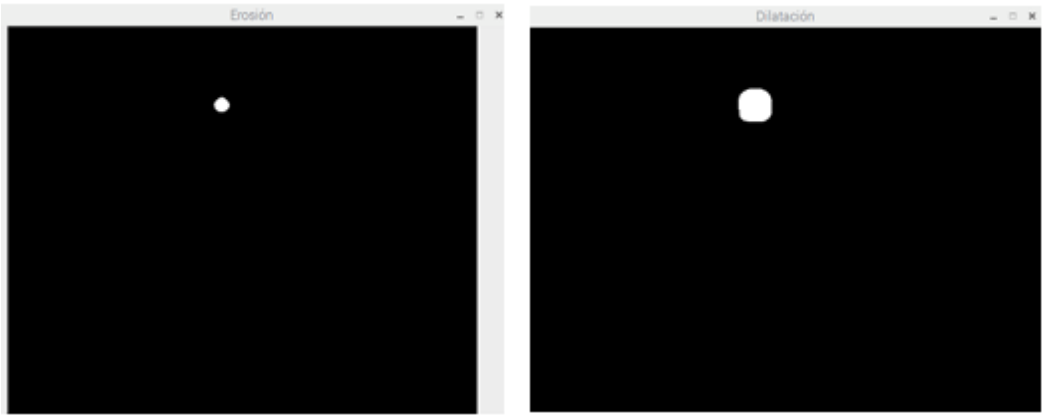


Figura 2.15: La imagen muestra el resultado obtenido al aplicar los filtros de erosión y dilatación de la imagen binarizada.

A continuación se aplica un filtro Gaussiano, deficiéndose mucho mejor los contornos del objeto a seguir, así como un contorno mucho más suave. Esto mejora considerablemente la detección y el seguimiento del objeto. Permite realizar la obtención del centro de masa del objeto de una manera mucho más exacta.

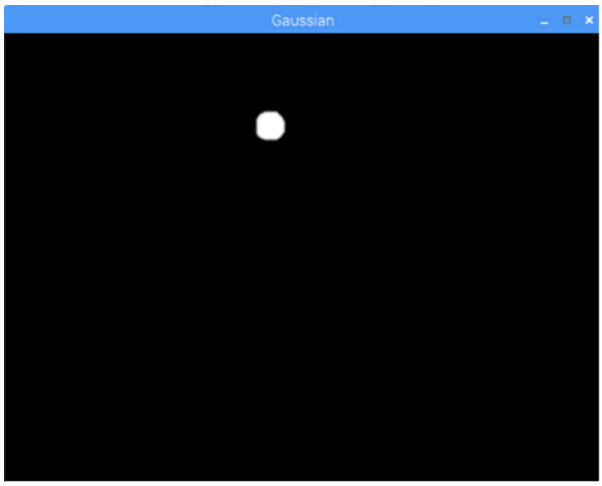


Figura 2.16: La imagen muestra el resultado obtenido al aplicar el filtro gaussiano a la imagen anterior.

**Identificación del contorno del objeto y la obtención de su centro de masa**

Una vez desarrollados los algoritmos que permiten la detección, localización y seguimiento del objeto por color es necesario realizar la identificación del contorno del objeto y la obtención de las coordenadas del centro del objeto con respecto al centro de la imagen. A continuación se muestran

los resultados obtenidos al determinar el contorno y las coordenadas del centro del objeto.

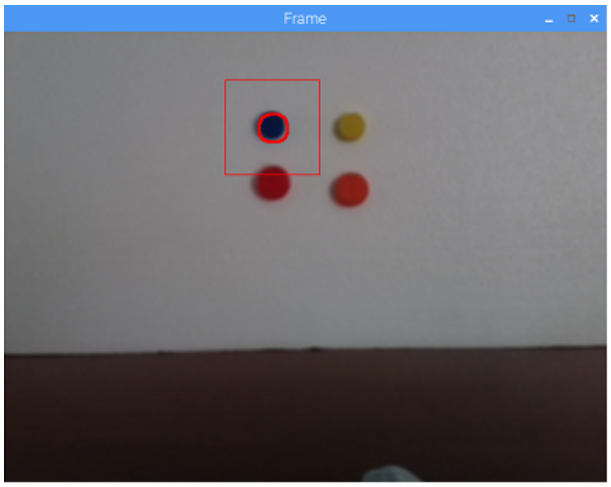


Figura 2.17: La imagen muestra el contorno del objeto al ser detectado el objeto de color azul.

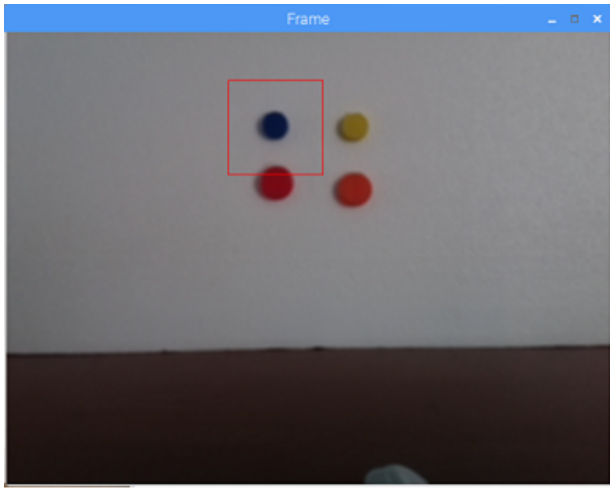


Figura 2.18: La imagen muestra el rectángulo inscrito sobre el objeto al determinar su centro de masa.

Una vez obtenido el centro de masa del objeto se calcula la distancia a la que se encuentra el objeto del origen de coordenadas, con estos valores se determina si el vehículo debe permanecer inmóvil o moverse hacia los lados, según sea el caso, esto se evidencia a través de la siguiente imagen, para mayor información consultar el código desarrollado. En las figuras se muestran de manera gráfica las partes en las cuales se dividió la imagen para obtener el centro de la misma. De tal manera, que

sí el objeto permanece en la zona muerta el robot queda inmóvil pero si el objeto se mueve y sale de esta zona o rango, entonces el vehículo se deberá mover en esa dirección.

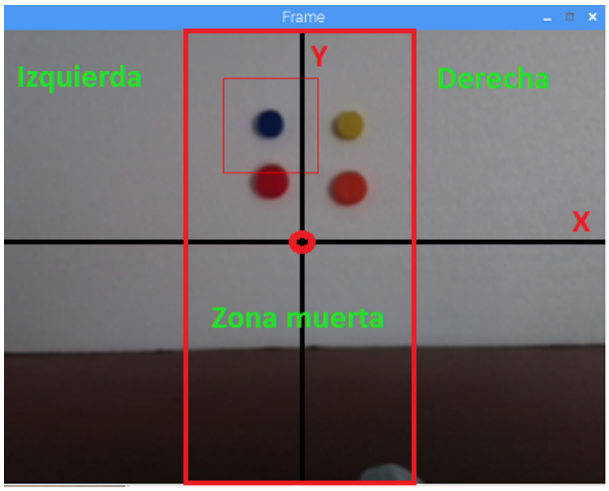


Figura 2.19: La imagen muestra la división de la pantalla para obtener el centro de origen de coordenadas.

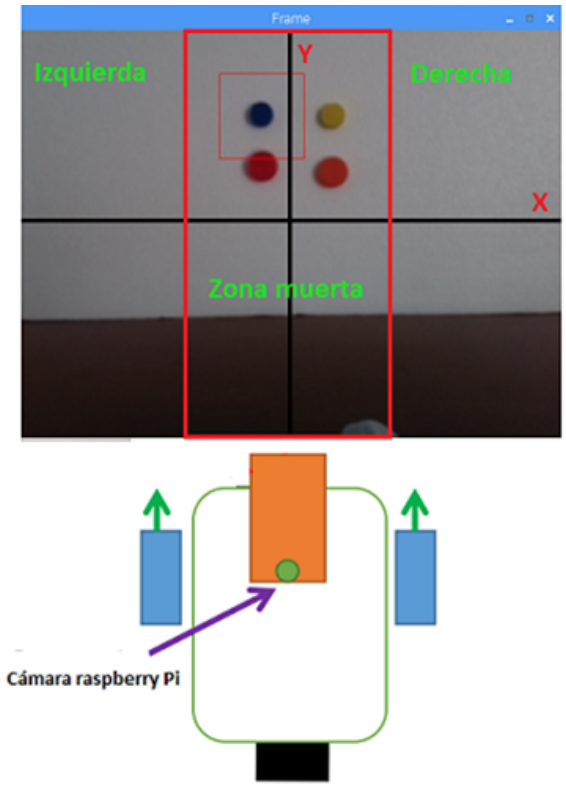


Figura 2.20: La imagen muestra la división de la pantalla para obtener el centro de origen de coordenadas y la posición que debe mantener el vehículo con respecto a la imagen.

# Conclusiones

1. Con el sistema embebido Raspberry Pi-3, es posible realizar la detección y seguimiento de un objeto por color, así como la conducción, operación y monitoreo del vehículo autónomo a través de la visión artificial.

2. El seguimiento del objeto se realiza en línea.

3. Al realizar la conversión de modelo RGB al HSV se logró una mejor segmentación de la imagen con un bajo costo computacional, ademas que se consigue un sistema de visión artificial más robusto ante cambios lumínicos y de sombras a las que esta sometido el sistema de captura de la imagen.

4. La combinación de los filtros de erosión y dilatación permitieron minimizar el ruido presente en la imagen y posteriormente la aplicar el filtro Gaussiano se logra un contorno del objeto más definido y suave en la imagen.

5. El tiempo de procesamiento requerido por el sistema embebido desde que capta la imagen hasta realizar el seguimiento del objeto es de aproximadamente de 50 ms a una resolución de 640x480 píxeles.

# Recomendaciones y trabajos futuros

1. Incorporar al algoritmo diseñado la capacidad de seguimiento de objetos por formas.
2. Optimizar el algoritmo para lograr invarianza antes cambios lumínicos bruscos en ambiente.
3. Implementar una programación a múltiples hilos, para reducir aun más la complejidad computacional.
4. Realizar la instalación de la cámara en una plataforma móvil, para obtener un mayor campo visual, proporcionándole mayor flexibilidad al sistema.

# Bibliografía


[1] L. Mejias Alvarez, *Control visual de un vehiculo aereo autonomo usando detección y seguimiento de caracterısticas en espacios exteriores.* PhD thesis, Industriales, 2006.

[2] P. M. Roth and M. Winter, "Survey of appearance-based methods for object recognition," *Inst. for Computer Graphics and Vision, Graz University of Technology, Austria, Technical Report ICGTR0108 (ICG-TR-01/08)*, 2008.

[3] Y. Benezeth, P.-M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger, "Comparative study of background subtraction algorithms," *Journal of Electronic Imaging*, vol. 19, no. 3, pp. 033003–033003, 2010.

[4] S. Soo, "Object detection using haar-cascade classifier," *Institute of Computer Science, University of Tartu*, 2014.

[5] C. Pérez González, "Detección y seguimiento de objetos por colores en una plataforma raspberry pi," 2016.

[6] A. Jepson, *Introduction to color theory.* http://infohost.nmt.edu/tcc/help/pubs/colortheory/web/hsv.html, 2017.

[7] A. O. Baturone, *Robótica: manipuladores y robots móviles.* Marcombo, 2005.

[8] F. Reyes, J. Cid, J. Méndez, G. Villegas, F. Porras, and A. Lara, "Diseño, modelado y construcción de un robot movil.," *Benemérita Universidad Autónoma de Puebla, México*, 2003.

[9] R. Rodríguez and J. Sossa, "Procesamiento y análisis digital de imágenes," *Ra-Ma Ed.: Madrid, España*, 2011.

[10] M. G. Villanueva, S. R. Zavala, and H. O. Reyes, "Robot móvil autónomo seguidor de línea con raspbery pi y sistema de visión artificial.,"

[11] T. Krajník, V. Vonásek, D. Fišer, and J. Faigl, "Ar-drone as a platform for robotic research and education," *Research and Education in Robotics-EUROBOT 2011*, pp. 172–186, 2011.

[12] T. Krajník, M. Nitsche, S. Pedre, L. Přeučil, and M. E. Mejail, "A simple visual navigation system for an uav," in *Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on*, pp. 1–6, IEEE, 2012.

[13] Y. Naidoo, R. Stopforth, and G. Bright, "Development of an uav for search & rescue applications," in *AFRICON, 2011*, pp. 1–6, IEEE, 2011.

[14] S. M. Adams and C. J. Friedland, "A survey of unmanned aerial vehicle (uav) usage for imagery collection in disaster research and management," in *9th International Workshop on Remote Sensing for Disaster Response*, p. 8, 2011.

[15] A. Chikwanha, S. Motepe, and R. Stopforth, "Survey and requirements for search and rescue ground and air vehicles for mining applications," in *Mechatronics and Machine Vision in Practice (M2VIP), 2012 19th International Conference*, pp. 105–109, IEEE, 2012.

[16] T. Luukkonen, "Modelling and control of quadcopter," *Independent research project in applied mathematics, Espoo*, 2011.

[17] R. L. Finn and D. Wright, "Unmanned aircraft systems: Surveillance, ethics and privacy in civil applications," *Computer Law & Security Review*, vol. 28, no. 2, pp. 184–194, 2012.

[18] H. L. Paredes, "Detección y seguimiento de objetos con cámaras en movimiento," *Proyecto fin de carrera, Universidad Autónoma de Madrid*, 2011.

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

[20] E. Aboudaya *et al.*, "Mobile teleoperation of a mobile robot," 2010.

[21] S. Salas Arriarán *et al.*, "Todo sobre sistemas embebidos [capítulo 1]," 2015.

[22] M. Alliance, *Camera Interface Specifications*. http://mipi.org/spesifications/camera-interface, 2017.

[23] R. P. Blog, *Raspberry Pi*. http://www.raspberrypi.org/downloads/, 2017.

[24] OpenCV, *Changing color spaces.* http://docs.opencv.org/master/df/d9d/tutorial-py-colorspaces.html, 2015.

[25] S. Fernando, *OpenCV Tutorial C++ Color detection and Object Tracking.* http://opencv-srf.blogspot.com.es/object-detection-using-color-seperation.html, 2010.

[26] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.

[27] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *Acm computing surveys (CSUR)*, vol. 38, no. 4, p. 13, 2006.

# Anexos

Código del algoritmo diseñado en Python:

```
"""
#------------------------"-Programa principal-"---------------------
#Titulo de programa:Template_Matching_20_07_2017.py
#Codigo escrito por: Ranyel Morales Suero
#Nombre del proyecto:En este programa se busca un objeto por color
# del objeto de interes a buscar.
#-------------------------------------------------------------------
"""

#Importando la biblioteca
import RPi.GPIO as GPIO
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2
import numpy as np
import sys
import os #invoca la terminal

#Declaro variable
umbral = 60  # Umbral de error (deadzone)
camera = PiCamera()
camera.resolution = (640, 480)
```

```
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(640, 480))
# Ancho y alto del video que se desea capturar
frame_w = 640
frame_h = 480
# Coordenadas del centro del video
fcx = int(frame_w/2)
fcy = int(frame_h/2)

# allow the camera to warmup
time.sleep(0.1)

def nothing(x):
pass
cv2.namedWindow('Configuracion')
#Creamos los controles para indicar el color que deseamos seguiremos.
cv2.createTrackbar ('H min', 'Configuracion', 0,255, nothing)
cv2.createTrackbar ('H max', 'Configuracion', 0,255, nothing)
cv2.createTrackbar ('S min', 'Configuracion', 0,255, nothing)
cv2.createTrackbar ('S max', 'Configuracion', 0,255, nothing)
cv2.createTrackbar ('V min', 'Configuracion', 0,255, nothing)
cv2.createTrackbar ('V max', 'Configuracion', 0,255, nothing)

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

blur = cv2.blur(image, (5,5)) ##desenfoque (filtro de caja normalizado)
## toma el promedio de todos los pixeles bajo el area del kernel y
```

```
## reemplaza al elemento central por este promedio.

#hsv compila la imagen de RGB a HSV
hsv = cv2.cvtColor(blur,cv2.COLOR_BGR2HSV)
#thresh = cv2.inRange(hsv,np.array((0, 200, 200)), np.array((20, 255, 255)))

#Declaramos las variables del rango de color que seguiremos.
Hmin = cv2.getTrackbarPos ('H min', 'Configuracion')
Hmax = cv2.getTrackbarPos ('H max', 'Configuracion')
Smin = cv2.getTrackbarPos ('S min', 'Configuracion')
Smax = cv2.getTrackbarPos ('S max', 'Configuracion')
Vmin = cv2.getTrackbarPos ('V min', 'Configuracion')
Vmax = cv2.getTrackbarPos ('V max', 'Configuracion')


 #Define el rango de color azul HSV
#lower = np.array([Hmin,Smin,Vmin],dtype="uint8")
lower = np.array([110,50,50],dtype="uint8")
#lower = np.array([Hmin,Smin,Vmin],dtype="uint8")
#lower = np.array([Hmax,Smax,Vmax],dtype="uint8")
upper = np.array([130,255,255],dtype="uint8")
#upper = np.array([Hmax,Smax,Vmax],dtype="uint8")

#Crear una mascara con solo los pixeles dentro del rango de azul
# Aplicar umbral a la imagen y extraer los pixeles en el rango de colores.
thresh = cv2.inRange(hsv, lower, upper)

#Crear un kernel de '1' de 3x3 (o matriz de convolucion)
kernel = np.ones((6,6),np.uint8)
#Empleo de las funciones erode y dilate
#aplicando la funcion de erosion
thresh_erosion = cv2.erode(thresh, kernel, iterations=2)
```

```
#aplicando la funcion de dilatasion
thresh_dilatacion = cv2.dilate(thresh_erosion , kernel, iterations=2)
#aplicacion de las transformaciones morfologicas
thresha = cv2.morphologyEx(thresh_dilatacion, cv2.MORPH_CLOSE, kernel)
threshb = cv2.morphologyEx(thresha, cv2.MORPH_OPEN, kernel)
thresh2 = threshb.copy()

#Difuminar la mascara para suavizar los contornos y aplicar filtro canny
blur1 = cv2.GaussianBlur(thresh2, (5, 5), 0)
edges = cv2.Canny(thresh2,1,2)

# find contours in the threshold image
contours,_ = cv2.findContours(edges,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

 areas = [cv2.contourArea(c) for c in contours]
i = 0
for extension in areas:
if extension > 600:
actual = contours[i]
approx = cv2.approxPolyDP(actual,0.05*cv2.arcLength(actual,True),True)
if len(approx)==4:
cv2.drawContours(blur,[actual],0,(0,0,255),2)
cv2.drawContours(thresh2,[actual],0,(0,0,255),2)
i = i+1
# finding contour with maximum area and store it as best_cnt
max_area = 0
best_cnt = 1
for cnt in contours:
area = cv2.contourArea(cnt)
if area > max_area:
max_area = area
best_cnt = cnt
```

```
#Buscando el centro x, y del objeto
M = cv2.moments(best_cnt)
cx,cy = int(M['m10']/M['m00']), int(M['m01']/M['m00'])
#if best_cnt>1:
#Dibujamos el centro con un rectangulo
cv2.rectangle(blur,(cx-50,cy-50), (cx+50, cy+50),(0,0,255), 0)
# Dibujar una linea del centro del frame al centroide
#cv2.line(blur, (cx, cy), (fcx, fcy), 255, 2)

#font = cv2.FONT_HERSHEY_SIMPLEX
#cv2.putText(blur,'OpenCV',(100,100), font, 4,(0,255,0),2)

# Calcular el error en X y Y
error_x = fcx- cx
error_y = fcy - cy
print('El objeto se encuentra en la abscisas: ',error_x)
print('El objeto se encuentra en la ordenadas: ',error_y)

if error_x < -umbral:
print('El objeto se mueve hacia la derecha')
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(blur,'Derecha',(480,50), font, 1,(0,255,0),2)
os.system("echo ""17=0.3""> /dev/pi-blaster")
else:
print('El objeto se encuentra en zona muerta')
os.system("echo ""17=0""> /dev/pi-blaster")
if error_x > umbral:
print('El objeto se mueve hacia la izquierda')
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(blur,'Izquierda',(20,50), font, 1,(0,255,0),2)
os.system("echo ""18=0.11""> /dev/pi-blaster")
```

```
else:
print('El objeto se encuentra en zona muerta')
os.system("echo ""18=0""> /dev/pi-blaster")

# Mostrando las imagenes
cv2.imshow("Frame", blur) #Muestra el segumiento del objeto por color
#Imagen convertida de RGB  a HSV
cv2.imshow("HSV", hsv)
#Imagen sin aplicarle trasformaciones morfologicas
cv2.imshow("Imagen", thresh)
#Imagen obtenida al aplicarle trasformaciones morfologicas apertura
cv2.imshow("Morphology apertura", thresha)
#Imagen obtenida al aplicarle trasformaciones morfologicas cierre
cv2.imshow("Morphology cierre", threshb)
cv2.imshow('thresh2',thresh2) #Muestra el contorno de la figura
cv2.imshow('thresh1',thresh) #Muestra solo los pixeles dentro del rango de azul
cv2.imshow('Erosion',thresh_erosion)
cv2.imshow('Dilatacion',thresh_dilatacion)
#Aplicando filtro gaussiano
cv2.imshow('Gaussian',blur1)

#cv2.imshow("Frame", image)
key = cv2.waitKey(1) & 0xFF

# clear the stream in preparation for the next frame
rawCapture.truncate(0)

# if the `q` key was pressed, break from the loop
if key == ord("q"):
break
```