Universidad Autónoma de Querétaro
Facultad de Ingeniería

INGENIERÍA EN AUTOMATIZACIÓN

Implementación de un Sistema de Visión en Robot Humanoide para la Detección de Objetos

---

TESIS

Que como parte de los requisitos para obtener el título de Ingeniero en Automatización con Línea Terminal en Mecatrónica.

Presenta: **Pilar Sarahi Sánchez Ortiz**
Director: **Dr. Gerardo Israel Pérez Soto**
Co-Directora: **Dra. Karla Anhel Camarillo Gómez**

Dr. Gerardo Israel Pérez Soto _____
Presidente

Dra. Karla Anhel Camarillo Gómez _____
Secretario

Dr. Jesús Carlos Pedraza Ortega _____
Vocal

Dr. Juan Manuel Ramos Arreguín _____
Suplente

Dr. J. Jesús de Santiago Pérez _____
Suplente

1 de mayo de 2025

Dedicatoria

"A mi familia, por ser mi apoyo y mi fuerza. Gracias por su amor, su paciencia y por no soltarme en los momentos más difíciles.

A mi tita, quien ya no está físicamente conmigo, pero siempre está en mi corazón y en cada uno de mis logros.

A mi pareja, por impulsarme a ser siempre la mejor versión de mí misma, por entender mis silencios y por seguir creyendo en mí incluso cuando yo no podía hacerlo.

A mi familia de cuatro patas y alas, cuya presencia ilumina y llena de alegría mi vida. Gracias por ser mi refugio y mi fuente constante de motivación, recordándome que, incluso en los momentos más difíciles, nunca estoy sola en este camino."

Agradecimientos

Agradezco al Dr. Gerardo Israel Pérez Soto, mi director de tesis, por brindarme la oportunidad de realizar este trabajo, por su acompañamiento académico, sus observaciones y orientación.

Al Dr. Jesús Carlos Pedraza, por su apoyo incondicional, su experiencia y su guía a lo largo de todo este proceso y su enorme calidad humana. Gracias por impulsarme a crecer tanto académicamente como personalmente.

Al Dr. Juan Manuel Ramos Arreguín, por todas las atenciones brindadas durante el desarrollo de este trabajo y por los valiosos consejos académicos que contribuyeron significativamente.

Al Tecnológico Nacional de México Campus Celaya y a la Dra. Karla Anhel Camarillo Gómez, por brindarme amablemente el acceso al robot que fue pieza clave para el desarrollo experimental de esta tesis.

Al Dr.J. Jesús de Santiago Pérez por su amabilidad y tiempo otorgado.

Mi gratitud a la Dra. María del Carmen Cabrera Hernández, a la Mtra. Rosa María Cabrera Hernández y a Tony, por su constante compañía, su apoyo incondicional y por escucharme en los momentos más importantes.

Índice

Índice de figuras

Índice de tablas

Abreviaciones

USB	Universal Serial Bus
FIRA	Federation of International Robot Sports Association
HSV	Hue, Saturation, Value
OpenCV	Open Source Computer Vision Library
ROS	Robot Operating System
HaViMo	Hamid's Vision Module
SIFT	Scale-Invariant Feature Transform
UART	Universal Asynchronous Receiver-Transmitter
IDLE	Integrated Development and Learning Environment
PCB	Printed Circuit Board
FPS	Frames Per Second
TTL	Transistor-Transistor Logic
RGB	Red, Green, Blue
CMYK	Cyan, Magenta, Yellow and Key
RS-485	Protocolo de comunicación en serie para controladores y receptores en aplicaciones industriales.
CAN	Controller Area Network
ARM	Advanced RISC Machine
GPIO	General Purpose Input/Output
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
SRAM	Static Random Access Memory
ADC	Analog-to-Digital Converter
PWM	Pulse Width Modulation
RNC	Convolutional Neural Network
HAT	Hardware Attached on Top
DDR3L	Double Data Rate 3 Low Voltage
RAM	Random Access Memory

Resumen

En este trabajo se presenta el diseño e implementación de un sistema de visión basado en operaciones morfológicas y algoritmos de detección de formas geométricas para la detección de objetos. El sistema utiliza una tarjeta *Raspberry Pi 5* para el procesamiento de imágenes obtenidas mediante una cámara *USB*. Así como la interfaz para el ajuste de los parámetros correspondientes al sistema de visión. También cuenta con un microcontrolador *Raspberry Pi Pico* para el control de movimiento de los motores *Dynamixel*. Se implementa una arquitectura abierta, es decir, flexibilidad en software mediante el uso de componentes de código abierto basado en *OpenCV*. Los resultados sugieren que el sistema propuesto es capaz de realizar una detección de objetos con distintas formas geométricas.

Palabras clave: Sistema de visión, arquitectura abierta, *OpenCV*, operaciones morfológicas.

Abstract

This paper presents the design and implementation of a vision system based on morphological operations and geometric shape detection algorithms for object detection. The system uses a *Raspberry Pi 5* for the processing of images obtained by a camera *USB*. As well as the interface for the adjustment of the parameters corresponding to the vision system. It also has a *Raspberry Pi Pico* microcontroller for motor motion control *Dynamixel*. An open architecture is implemented, i.e., software flexibility through the use of open source components based on *OpenCV*. The results suggest that the proposed system is capable of performing detection of objects with different geometric shapes.

Keywords: Vision system, open architecture, *OpenCV*, morphological operations.

1 Antecedentes

En esta sección se ofrece una introducción a la problemática abordada, junto con una descripción del estado del arte.

1.1 Introducción

El avance tecnológico ha transformado el mundo de la automatización, llevando a la invención de robots humanoides idóneos para realizar tareas complejas de gran precisión. La capacidad de estos robots para comprender e interactuar con su entorno depende principalmente de su sistema de visión, que debe integrar tecnologías avanzadas para ofrecer una percepción efectiva y en tiempo real. Con la evolución de la ingeniería en sus distintas disciplinas como la mecánica, electrónica e informática, múltiples modelos de robots han sido desarrollados con tecnologías mas sofisticadas.

En la actualidad, la importancia de estos sistemas de visión son evidentes en diversas aplicaciones y sectores. En caso del sector industrial, los robots humanoides con sistemas de visión son empleados para la evaluación y el soporte de equipos, así como para la ejecución de tareas detalladas. La capacidad de estos robots para realizar inspecciones visuales precisas y en tiempo real permite detectar anomalías para realizar ajustes y operar con un alto nivel de precisión, lo que contribuye a la eficiencia y seguridad en las fábricas.

Otra aplicación es en el uso de *robots humanoides* pertenecientes a la empresa *ROBOTIS*, la cuál es líder en el mercado de robótica ofreciendo diferentes modelos de robots formados por su marca estelar *Dynamixel*. Los cuales suelen ser utilizados en competencias o con fines de investigación.

Por otro lado, la robótica humanoide descrita por algunos autores como un mecanismo desarrollado con el fin de imitar la morfología corporal y movimientos de un ser humano, típicamente formados por un torso, dos extremidades superiores y dos extremidades inferiores [1], también está avanzando en el ámbito de la investigación y el entretenimiento. Siendo las competencias de robots humanoides creadas por *La Federación y Asociaciones Deportivas Internacionales (FIRA)*, unas de las más importantes, creada en Corea en el año de 1996 por el Prof. Jong-Hwan Kim, KAIST. Con la finalidad de recurrir a los deportes como casos de estudio para la investigación avanzada en robótica y otras disciplinas afines. Hoy en día, cuentan con diversas categorías de las cuales destacan la competición FIRA Air orientada a robots voladores autónomos, FIRA Challenge para la investigación robótica con gran impacto social, como robots de búsqueda y rescate urbanos, y FIRA Youth destinada para formar nuevas generaciones de científicos [2]. Dentro de estas categorías destaca la importancia de un sistema de visión debido a la identificación y localización de objetos, como lo es, los balones de fútbol, basquetbol, etc. Así como, la identificación de jugadores para la toma de decisiones.

En el presente trabajo, se implementa un sistema de visión en un robot humanoide para la detección de objetos. Utilizando una tarjeta *Raspberry Pi 5* y un microcontrolador *Raspberry Pi Pico* para el control de movimientos, ambos de código libre y arquitectura abierta, adoptando una técnica flexible del sistema.

1.2 Estado del arte

En esta sección se presenta la Tabla 1 y Tabla 2 , correspondiente a una búsqueda en artículos y tesis respectivamente. La Tabla 2 muestra los resultados de una búsqueda limitada a tesis de la Universidad Autónoma de Querétaro. Ambas tablas incluyen información sobre la cita, el año, el título del documento y la contribución del autor en relación con el trabajo de esta tesis.

Abordando los temas descritos en la Tabla 1, se muestran trabajos anteriores desde el año 2018 y 2019 sobre sistemas de visión para robots humanoides; el primero de estos desarrolló un sistema que integra una cámara *Logitech* a una *Raspberry Pi 3*, mediante el uso del modelo de color *HSV* y operaciones morfológicas como la dilatación y erosión, se realizó un algoritmo para definir el color, eliminar el ruido y detectar correctamente el objeto de interés para la activación del robot [3]. Por su parte, el segundo trabajo implementa el uso de redes neuronales convoluciones entrenadas previamente para la detección de personas con el objetivo de controlar un robot *KUKA* [4]. Dicho sistema se desarrolló con la combinación la librería de visión por computadora de código abierto *OpenCV*, *Python* y *ROS*.

Se observa que en el año 2023 existe una creciente a la investigación de estos robots bajo diversas técnicas. En [5] se muestra la implementación de un sistema de visión simple orientado en un vehículo de 4 ruedas, el cual consiste en el uso de la librería de código abierto de *OpenCV* para seleccionar el color a detectar utilizando el modelo de color *HSV*. Aunado a *OpenCV* [6] implementa técnicas que permiten la identificación y rastreo en tiempo real del objeto. Por su parte [7] propone un diseño del sistema de control mediante *OpenPose*.

Finalmente, se muestra en 2024 la implementación de técnicas como *soft computing* para sistemas de visión por computadora para la detección de objetos.

Cita	Año	Título	Contribución
[3]	*2018*	*Development and implementation of a vision system for decision making in the movements control of humanoid robots*	Sistema de visión y control con Raspberry Pi 3 en el robot humanoide, superando las limitaciones del del controlador original. Con *OpenCV*, se implementó un algoritmo que simplifica las formas y reduce el ruido.
[4]	*2019*	*Human rescue based on autonomous robot KUKA youbot with deep learning approach*	Navegación autónoma del robot KUKA YouBot utilizando una combinación de *OpenCV*, *Python* y el *Robot Operating System (ROS)*.
[5]	*2023*	*Development of a Raspberry PI-Controlled VEX Robot for a Robotics Technology Cours*	Desarrollo de un robot *VEX* controlado por una *Raspberry Pi*, utilizando el hardware *VEX* y aprovechando el entorno Linux de la *Raspberry Pi* para programar y ejecutar algoritmos avanzados.
[6]	*2023*	*Object Tracing with Colour range filtering in HSV Colour space and Object Following with Ros2*	Método de seguimiento de objetos mediante *OpenCV* y *ROS2* en una Raspberry Pi. Mediante el filtrado de la gama cromática en el espacio HSV, se identifican y rastrean objetos en tiempo real.
[7]	*2023*	*Humanoid Robot control system based on human pose detection*	Diseño de un sistema de control para robots humanoides basado en la detección de posturas humanas mediante el método *OpenPose*.
[8]	*2024*	*Using Soft Computing and Computer Vision to Create and Control an Integrated Autonomous Robotic Manipulator Process*	Implementación de soft computing, a través de algoritmos evolutivos, utilizando tecnologías como *Arduino*, *Raspberry Pi*, y procesamiento de imágenes para identificar objetos y realizar manipulaciones de forma eficiente.
[9]	*2024*	*Experiments with cooperative robots that can detect object's shape, color and size to perform tasks in industrial workplaces*	Sistema robótico que utiliza la visión por computadora para la detección con gran precisión de 99,8 en formas y 100 en colores y tamaños para clasificar objetos, guiando un brazo robótico para manipularlos.

Tabla 1: Revisión del estado del arte y comparación entre artículos relacionados.

Al tratar los temas de tesis descritos en la Tabla 2, se observa un amplio campo para la implementación de este trabajo, dado que en la institución existen algunos trabajos de tesis que combinen robótica humanoide y sistemas de visión. Como se puede apreciar en el año 2008 por [10], se desarrolló un sistema de visión aplicando operaciones morfológicas y filtrado de color. Por su parte [11] en 2021 integro una cámara de vídeo para el procesamiento de imágenes mediante *Raspberry Pi 3*. Continuando con esta misma linea de investigación en [12] realizó diversos métodos para la manipulación de imágenes con la finalidad de orientar la asistencia del conductor.
Por otra parte, en [13] se observa la implementación de sistema de visión utilizando RNC para la detección de emociones mediante las expresiones faciales y el control de un robot humanoide.

Finalmente, se aprecia en [14] la implementación de un controlador de motores *Dynamixel*, siendo este de arquitectura abierta para el movimiento de un robot humanoide con una *Raspberry Pi Pico*, este trabajo se utiliza de referencia para el diseño del controlador de este proyecto.

Cita	Año	Título	Contribución
[10]	*2008*	*Procesamiento morfológico, espacios color y quaternions, con biometría de la imagen y otras aplicaciones*	Caracterización de espacios de color, segmentación y filtrado en el procesamiento de imágenes.
[11]	*2021*	*Sistema de seguridad basado en Raspberry Pi con cámara para día y noche con enlace IoT y procesamiento de imágenes*	Implementación de una cámara para el procesamiento de imágenes utilizando una *Raspberry Pi 3* con *Python*
[12]	*2023*	*Detección automática de líneas de carril basada en Inteligencia Artificial orientada a la asistencia del conductor*	Detección de geometrías y cambios en espacio de color en imágenes.
[14]	*2023*	*Desarrollo de un controlador de movimiento para un robot humanoide*	Diseño de un controlador de arquitectura abierta compatible con la marca *Robotis*.
[13]	*2023*	*Desarrollo e implementación de un sistema de visión usando RNC para el reconocimiento de expresiones faciales*	Implementación de sistema de visión en *Raspberry Pi 3* usando tecnicas de RNC aplicado en robot humanoide.

Tabla 2: Revisión del estado del arte y comparación entre tesis relacionadas.

1.3 Descripción del problema

Actualmente, los robots humanoides son utilizados en gran escala para la detección e identificación de objetos de interés, por lo cual es de vital importancia contar un sistema de visión por computadora robusto, capaz de enfrentar obstáculos. Dentro de los cuales destaca el módulo"*Hamid's Vision Module (HaViMo)*", fabricado por la empresa *ROBOTIS*. Sin embargo, este cuenta con limitaciones para su escalabilidad y flexibilidad debido al hardware y software proporcionado por la compañía.

Por ello, es necesario contar con el conocimiento previo del sistema, como lo es en los dispositivos de arquitectura abierta, lo que permite al diseñador o usuario adecuar las características del modelo a las necesidades especificas de la región de interés. Es fundamental para un desarrollador contar con mayor control sobre el procesamiento de imágenes o vídeo dentro del sistema, lo que garantiza un mejor manejo de la información, que finalmente, resulta en una correcta manipulación del movimiento de un robot humanoide.

1.4 Justificación

Una de las aplicaciones de los robots humanoides es la detección de objetos. Esto se realiza con ayuda de un sistema de visión por computadora y con el control de movimiento de los grados de libertad, correspondientes a la cabeza móvil del mecanismo. Por lo qué, la supervisión de ambas cosas permite la detección y la ejecución de la función establecida.

En la actualidad, existen sistemas de visión por computadora capaces de realizar esta actividad. De acuerdo con la bibliografía se conoce el sistema "*HaViMo (Hamid's Vision Module)*", diseñado para conectarse y comunicarse con el controlador CM-530 de ROBOTIS. Este controlador forma parte de la serie *Dynamixel*. En la Tabla 3 se presentan las características del sistema *HaViMo* descritas por [15].

HaViMo v.1.5 y v.2.0	**Sistema en *Raspberry Pi 5***
▪ **Región de color:** Hasta 15 regiones (blobs) por cuadro. ▪ **Detección de región(blob):** Separadas con el mismo color o diferente. ▪ **Procesamiento y resolución de imágenes:** 8 FPS a 160x120 px en formato YCrCb. ▪ **Filtro de ruido incorporado y ajustable:** Permite la detección de objetos grandes y pequeños según el color, incluso en imágenes con mucho ruido. ▪ **Umbral:** Ajustable remueve automáticamente las regiones pequeñas indeseadas. ▪ **Baud rate:**1Mbps ▪ **Conexión TTL:**Utiliza la misma conexión TTL de BIOLOID, facilitando la conexión al bus de comunicación. ▪ **Ajustes de cámara:** Exposición automática/balance de blancos o configuración manual. ▪ **Depuración o calibración de imagen:** De la lookup table a través del programa de calibración provisto. ▪ **Conectividad:** RoboBuilder[31] y otras plataformas via full-duplex. ▪ **Soporte:** Aplicaciones de RoboPlus. ▪ **Algoritmo de visión:** Griding.	▪ **Región de color:** Delimitadas por los parámetros establecidos dentro del código. ▪ **Detección de región(blob):** Separadas con el mismo color o diferente. ▪ **Procesamiento y resolución de imágenes:** Puede manejar resoluciones de vídeo hasta 4K a 60 FPS, dependiendo de la configuración y uso. ▪ **Baud rate:** 921,600 bps ▪ **Conexión:** USB, Ethernet, Wi-Fi y Bluetooth, dependiendo del protocolo empleado. ▪ **Filtros:** Puede implementar filtros y procesamiento de imágenes personalizado utilizando software como *OpenCV* y otras bibliotecas de código abierto. ▪ **Soporte:** Aplicaciones de arquitectura abierta. ▪ **Algoritmo de visión:** Puede ejecutar una variedad extensa de algoritmos de visión y aprendizaje automático debido a su procesador Arm Cortex-A64 de cuatro núcleos de 76 bits y el soporte para bibliotecas como OpenCV y TensorFlow.

Tabla 3: Características del sistema *HaViMo vs* un sistema en *Raspberry Pi 5*.

Analizando la Tabla 3 una de las principales desventajas que se puede observar es que al ser un módulo especializado, *HaViMo* puede tener limitaciones en términos de flexibilidad, debido a su arquitectura cerrada, especialmente sí es necesario implementar algoritmos de visión más complejos como lo es el método *SIFT* (Scale-Invariant Feature Transform) o inclusive operaciones morfológicas. A diferencia de sistemas con arquitectura abierta, como lo es la *Raspberry Pi 5*, donde las limitaciones del sistema corresponden en su mayoría a las características del dispositivo. Otra desventaja, son los costos, debido a que en comparación con otras alternativas suele ser más elevado. Además, de ofrecer un rendimiento y resolución estática en el procesamiento de imágenes, y brindar poco soporte al usuario.

Un ejemplo de un sistema genérico que ofrece estas ventajas es la tarjeta *Raspberry Pi 5*, como se muestra en la Tabla 3, el cual al ser un sistema de arquitectura abierta, permite flexibilidad de adecuar todas estas características como es la eliminación del ruido, segmentación y filtrado de color, etc. Además de brindar un amplió soporte en *hardware y software*. Debido a estas características y a otras ventajas adicionales, se utiliza una *Raspberry Pi 5* para el desarrollo de esta investigación de tesis.

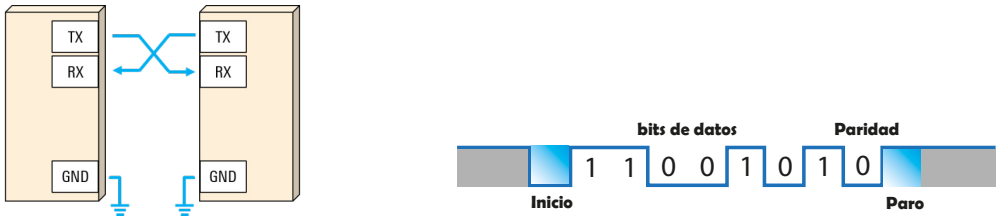
En el presente trabajo, se implementa un sistema de visión en robot humanoide para la detección de objetos utilizando una tarjeta *Raspberry Pi 5* y un microcontrolador *Raspberry Pi Pico* para el control de movimientos, abriendo la posibilidad de un sistema de arquitectura abierta y a su vez satisfacer las necesidades especificas del procesamiento de imágenes.

2 Fundamentos teóricos

En esta sección se presentan los conceptos fundamentales requeridos para el desarrollo del presente trabajo de tesis.

2.1 Protocolo UART

El protocolo UART (Universal Asynchronous Receiver-Transmitter) permite intercambiar datos en serie entre dos dispositivos con un cable para transmisión y otro para recepción, así como una conexión a tierra, representado en la Figura 2.1a. Dicho protocolo es asíncrono, lo que significa que el transmisor y receptor no comparten una señal de reloj, por lo que ambos deben operar a la misma velocidad preestablecida (4800, 9600, 19.2K, 57.6K, o 115.2K baudios) y usar los mismos parámetros de trama para mantener la sincronización de bits. [16].



(a) Diagrama de conexión [16]. (b) Formato de trama de datos[16].

Figura 2.1: Protocolo *UART*.

Su funcionamiento, el cual se ve representado en la Figura 2.1b parte del bit de inicio, el cual marca el comienzo de los datos, pasando de un estado alto a bajo, seguido por los bits de datos. Estos pueden variar entre cinco y nueve, se transmiten comenzando por el bit menos significativo. Al final, un bit de parada, que regresa al estado alto, señala el fin de los datos [16].

2.2 Protocolo de comunicación de motores Dynamixel

Para realizar la comunicación con los motores *Dynamixel*, se emplea el protocolo UART (del inglés Universal Asynchronous Receiver-Transmitter) semidúplex, el cual constituye un protocolo de comunicación serie donde no es posible utilizar las líneas de transmisión (TxD) y recepción (RxD) simultáneamente. Este método se adopta habitualmente cuando es necesario conectar varios dispositivos a un único bus de comunicación. Dado que más de un dispositivo se encuentra conectado al mismo bus, todos los dispositivos adicionales deben operar en modo de entrada mientras uno de ellos está transmitiendo información [17].

El controlador principal, encargado de dirigir los actuadores *Dynamixel*, establece la dirección de comunicación en modo de entrada. Solo cuando se está transmitiendo un Paquete de Instrucciones, el Controlador Principal cambia la dirección del bus a modo de salida, permitiendo así la transmisión de datos de manera efectiva [17].

Para la comunicación, se requiere enviar una trama de datos o paquete de instrucciones como se muestra en la Tabla 4.

En 1	En 2	ID Motor	Longitud	Instrucción	Par 1	...	Par N	Suma comprobación
0xFF	0xFF	ID Motor	Longitud	Instrucción	Par1	...	Par N	CHKSUM

Tabla 4: Paquete de instrucciones

Donde:

- **En1, En2:** Son valores predeterminados en el paquete de datos, ambos con valor en hexadecimal de 0xFF.
- **ID Motor:** Es el ID o dirección del motor, los cuales pueden ir de 1 a 252.
- **Longitud:** Este valor se define como la cantidad de bytes a enviar en el paquete de datos, mas dos unidades.
- **Instrucción:** Este valor define el tipo de instrucción a realizar (Escritura o lectura). La Lista de instrucciones se puede consultar en el manual oficial del protocolo de los motores [17].
- **Par1, ..., Par N:** Son los parámetros o registros a los que se desea acceder (Posición, velocidad, torque, voltaje, etc.). La lista de registros o parámetros se pueden consultar en el manual oficial de cada uno de los modelos de los motores [18].
- **Suma de comprobación:** Este valor se define como el inverso de bits o complemento A1 de la suma de toda la trama de datos definida anteriormente. el cual nos sirve para comprobar que el paquete a enviar a sido escrito correctamente.

2.3 Modelo de color HSV

Un modelo de color es un sistema utilizado para describir la manera en la cual los colores pueden ser representados como combinaciones de valores. Suelen dividirse en dos categorías los adictivos y sustractivos, haciendo referencia a la formación del matiz negro. Sirven para la captura, visualización y manipulación de procesamiento de imágenes, existen diversos modelos, los mas comunes son: RGB, HSV y CMYK.

El modelo de color *HSV* (Hue, Saturation, Value) es una representación de los colores de forma mas intuitiva y preceptivamente a como lo hace el ojo humano. El modelo *HSV* se centra en cómo los colores se perciben y manipulan en términos de sus características visuales considerándose las siguientes:

- **Hue (Tono)**: Representa el color en su estado más puro, se mide en grados y va de 0 a 180°.
- **Saturación**: Indica la intensidad del color, que va desde el gris hasta el color más puro.
- **Valor**: Representa el brillo del color, que va del negro al brillo máximo.

En la Figura 2.2 se muestra la representación del modelo de color *HSV* en forma cónica, donde el punto más bajo representa el valor mínimo del brillo, mientras que la parte más alta representa el valor máximo. Por otro lado la saturación se ilustra mediante el radio, el cual, si aumenta, el color será mas puro, a menor distancia tenderá a tonos grises. Finalmente, el tono se describe a partir del color rojo con un valor en grado 0 al tono cían con un valor de grado 180.

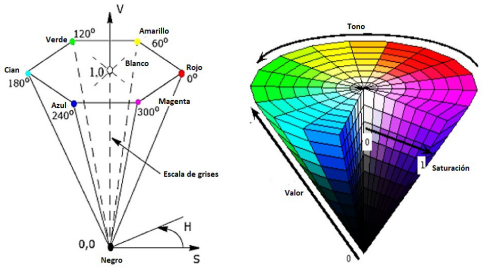


Figura 2.2: Representación del espacio de color *HSV* [19].

2.4 Filtro de suavizado

Los filtros de suavizado permiten reducir la cantidad de ruido en las imágenes. Su principio de funcionamiento es promediar los valores de los píxeles en una vecindad, en donde los píxeles más cercanos al centro tienen mayor peso que los lejanos con el objetivo de reducir las diferencias entre ellos, generando así, una imagen más homogénea. La ecuación que lo modela es la siguiente:

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{1}{2}(\frac{x^2}{\sigma_x}+\frac{y^2}{\sigma_y})} \tag{2.1}$$

Donde:

- σ_x y σ_y: Son las desviaciones estándar en las direcciones x y y respectivamente.
- $2\pi\sigma_x\sigma_y$: Es el factor de normalización para que la integral de dicha función sea de 1.

En la Figura 2.3 se muestran dos kernel o matrices de convolución con distribución gaussiana, en las que se observa como los píxeles centrales tiene un número mayor que los de los extremos. También se observa una gráfica tridimensional de la distribución gaussiana [20].

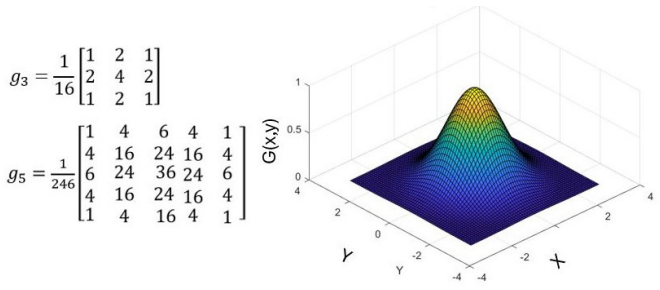


Figura 2.3: Representación matemática y gráfica de un Filtro *Gaussiano*.

2.5 Operaciones morfológicas

La morfología matemática es ampliamente empleada en el procesamiento de imágenes. Las operaciones morfológicas ayudan a optimizar los datos de una imagen, conservar los aspectos clave y eliminar las estructuras irrelevantes o ruido.

Las aplicaciones de la morfología matemática más utilizadas descritas por [21]:

- **Reprocesamiento de imágenes:** Eliminación de ruido, optimización de formas.
- **Extracción de rasgos:** Obtención del esqueleto, marcado de objetos, ampliación y reducción de la imagen.
- **Descripción cuantitativa de rasgos:** Área, perímetro, diámetro, volumen, etc.

La morfología matemática sustenta su conjunto de operaciones en el uso de dos operadores: la erosión y la dilatación correlacionadas con un elemento estructurante (ES). Este elemento sirve como modelo de referencia formado por una cantidad de píxeles que presentan una estructura geométrica sencilla, como un rectángulo, círculo, cruz, etc. Relacionado con la esquema o composición sobresaliente en los elementos de la imagen [22].

Dilatación en imágenes binarias

Consiste en la absorción de píxeles de fondo en cuyo vecindario exista al menos un píxel perteneciente al objeto, se convierten en objeto. Su salida resulta en aumentar el tamaño de los objetos o cerrar aquellos huecos diminutos. Se denota mediante la expresión I $\bigoplus$ H , donde I representa la imagen y H un vecindario dado.

En la Figura 2.4 se muestra en el inciso a) la imagen original, posteriormente en el inciso b) se muestran los píxeles correspondientes a la adhesión de píxeles del fondo a la figura original. Finalmente, el inciso c) muestra el resultado de aplicar la dilatación a la imagen original.

Erosión en imágenes binarias

Se basa en la eliminación de los píxeles del objeto en cuyo vecindario exista al menos un píxel perteneciente al fondo. Su resultado es la reducción del tamaño de los objetos o la eliminación de aquellos que son muy pequeños.

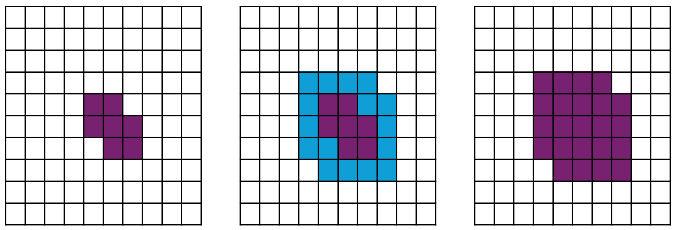


Figura 2.4: Dilatación en imagen binaria. a) Figura original, b) Proceso de dilatación, c) Imagen con dilatación.

Se denota por la expresión I Θ H, donde I representa la imagen y H un vecindario dado.

En la Figura 2.5 se muestra en el inciso a) la imagen original, posteriormente en el inciso b) se muestra la supresión de los píxeles correspondientes de la figura original. Finalmente, el inciso c) muestra el resultado de aplicar una erosión a la imagen original.

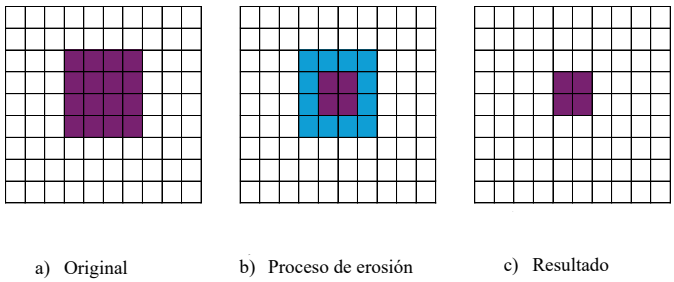


Figura 2.5: Erosión en imagen binaria. a) Figura original, b) Proceso de erosión, c) Imagen erosionada.

Apertura y Cierre

La operación de apertura consiste en la implementación de la erosión seguida de la operación de dilatación. El resultado de esta operación consiste en un suavizado de los contornos de un objeto y la eliminación de ruido de píxeles diminutos.

Por otro lado, la operación de cierre se basa en el uso de la dilatación seguida por la operación de erosión sobre una imagen. El efecto de este proceso es un suavizado en los contornos de la imagen, además de eliminar los huecos diminutos y cerrar espacios en el contorno.En cualquier caso, el resultado de la aplicación sucesiva de erosión y dilatación remueve aspectos triviales sin distorsionar la vista general del objeto [21].

2.6 Enmascaramiento con operadores bitwise

El enmascaramiento de imágenes se utiliza para permitir y negar el paso de píxeles de una imagen con una mascara binaria. Esto quiere decir que, en los píxeles de la máscara que sean blancos permitirán el paso del píxel

de la imagen original.

Para lograr el enmascaramiento se requiere del uso de las operaciones con operadores lógicos adaptados en arreglos matriciales conocidos como *bitwise* [20].

- **bitwise_and:** Operación de Y a nivel de bits entre dos valores.
- **bitwise_or:** Operación de OR a nivel de bits entre dos valores.
- **bitwise_not:** Operación de NOT a nivel de bits sobre un valor.

A	B	A AND B	A OR B	NOT A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

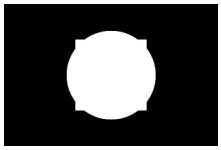
Tabla 5: Tabla de verdad para operadores AND, OR y NOT

La Tabla 5 muestra las tablas de verdad de los tres operadores lógicos mencionados anteriormente. En caso de los operadores de *AND* y *OR* requieren dos entradas, mientras que *NOT* solo requiere una.

Para mayor entendimiento de los operadores, se muestra en la Figura 2.6a un ejemplo de las imágenes de entrada en la que se hacen las operaciones con un cuadrado y un círculo. La Figura 2.6b muestra el resultado de la operación tipo AND, en la cual se observa que el cuadro fue filtrado casi en su totalidad a excepción de las esquinas, esto se debe porque son las únicas zonas que no comparten en común ambas figuras de entrada. Por otro lado, la Figura 2.6c muestra el resultado de la operación OR, la cual muestra a la figura del círculo con las esquinas del cuadrado, esto ocurre porque ambas figuras se suman. Finalmente, la Figura 2.6d muestra la operación NOT de la figura del cuadrado, dando como resultado una inversión de bits, donde lo negro ahora es blanco y viceversa.

(a) Imágenes de entrada.

(b) Resultado de operador AND.



(c) Resultado de operador OR.

(d) Resultado de operador NOT.

Figura 2.6: Enmascaramiento con operadores AND, OR y NOT.

3 Hipótesis

El robot humanoide es capaz de identificar objetos mediante un sistema de visión implementado en una *Raspberry Pi 5*.

4 Objetivos

4.1 Objetivo general

Implementar un sistema de visión de código abierto en un robot humanoide para la detección de un objeto empleando una tarjeta *Raspberry Pi 5*.

4.2 Objetivos específicos

- Establecer números de identificación a los motores mediante *DinamixelWizard* para el control de movimiento.
- Desarrollar un controlador empleando el microcontrolador *Raspberry Pi Pico* que permita la comunicación y control de los motores *Dynamixel*.
- Implementar rutinas de movimiento por medio del control de posición de cada motor para el desplazamiento del robot.
- Comunicar la tarjeta electrónica utilizando protocolo *UART* con la *Raspberry Pi 5*.
- Desarrollar un sistema de visión utilizando la librería de *OpenCV* en *Python*, que permita la identificación de color y geometría del objeto de estudio.
- Implementar un algoritmo para la localización y el seguimiento del objeto de estudio utilizando coordenadas rectangulares y áreas del objeto.

5 Descripción de *Software* y *Hardware*

En este trabajo se emplea una computadora *Lenovo Notebook IdeaPad 500-14ISK* con un procesador Intel Core i5-6200U de 2 núcleos a 2.8 GHz, memoria *RAM* de 12 GB *DDR3L* a 1600 MHz. Así mismo, se utiliza una tarjeta *Raspberry Pi 5* con un sistema operativo *Debian* V.12 *Bookworm*, con un *software* de *Python* V.3.11.2 con las librerías de *OpenCV* V.4.9.0, *numpy* V.1.24.2 y *serial* V.3.5. También se usa una tarjeta *Raspberry Pi Pico* con un lenguaje de *Python* V.3.4.0 y un intérprete *Micropython* V.1.22.2.

5.1 Robot Tonatiuh

El robot utilizado en el presente trabajo es nombrado"*Tonatiuh*"[1] con un diseño personalizado, adoptando la configuración en 18 grados de libertad similar al robot humanoide *Premium tipo A de la marca ROBOTIS* como se observa de forma ilustrativa en la Figura 5.1.

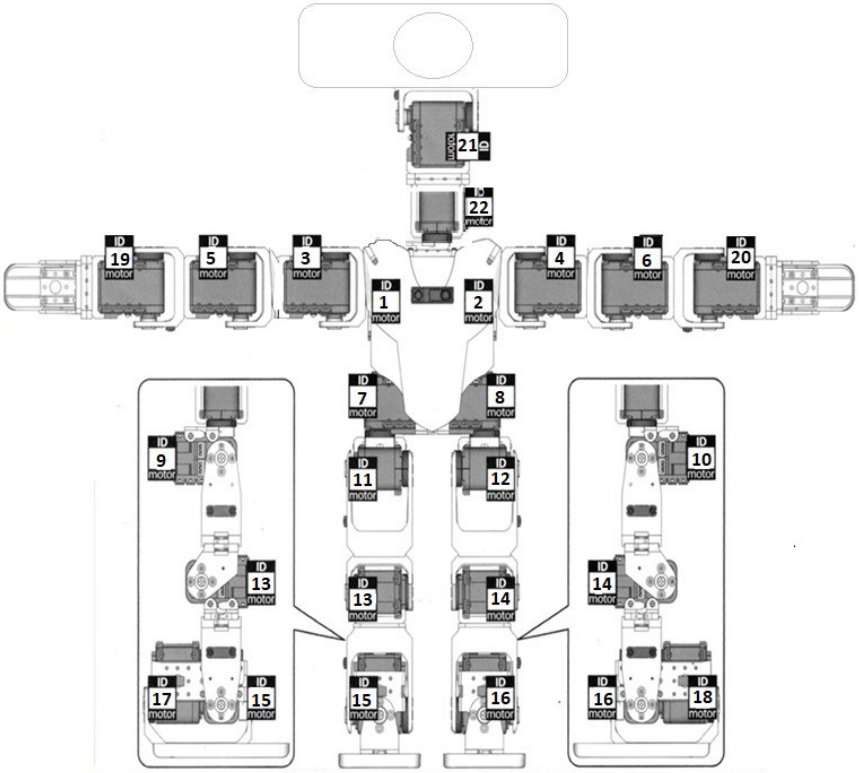


Figura 5.1: Morfología del robot Tonatiuh.

Dicho robot cuenta con 22 grados de libertad, distribuidos de la siguiente manera:

- 2 *GDL*: Son empleados para el movimiento rotatorio de la cámara.

[1]El robot *Tonatiuh* fue desarrollado como parte de una colaboración entre el Instituto Tecnológico de Celaya (ITC) y la Universidad Autónoma de Querétaro (UAQ).

- 12 *GDL*: Son distribuidos en las extremidades inferiores del mecanismo (6 por cada extremidad).
- 8 *GDL*: Son distribuidos en las extremidades superiores (4 por cada extremidad).

Empleando motores de la marca *Dynamixel*, conformado por distintos modelos, para la parte superior se construyó con motores *AX-12A*, y la inferior con *MX-28*. En la Figura 5.2 se observa al robot *Tonatiuh*, en el cual se puede apreciar el mecanizado, observando el diseño particular.

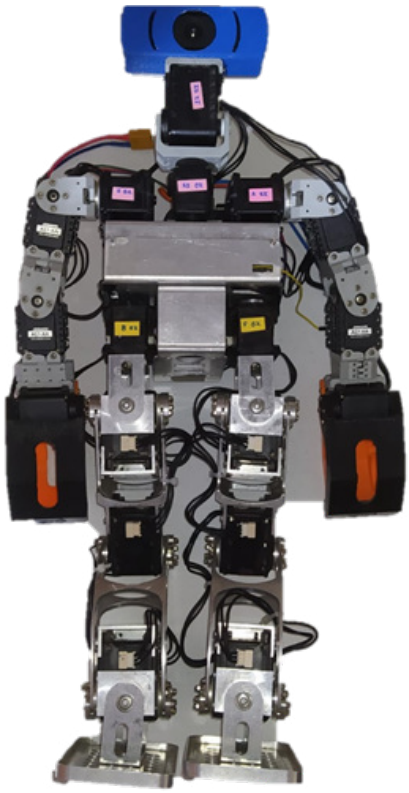


Figura 5.2: Morfología del robot Tonatiuh.

5.2 Motores Dynamixel

Los motores *Dynamixel* son una serie de actuadores inteligentes ampliamente utilizados en aplicaciones robóticas debido a su precisión, versatilidad y capacidad de control avanzada. Desarrollados por la empresa coreana *ROBOTIS*, estos motores son reconocidos por su diseño modular y su capacidad para comunicarse entre sí y con otros dispositivos electrónicos a través de buses de datos como *UART, RS-485 o CAN*.

Una de las características distintivas de los motores *Dynamixel* es su capacidad para realizar control de posición, velocidad y torque con alta precisión, lo que los hace ideales para una amplia gama de aplicaciones en robótica, desde manipuladores industriales hasta robots educativos y de investigación. Además, ofrecen una amplia variedad

de opciones de configuración y control, lo que permite adaptar su funcionamiento a las necesidades específicas de cada aplicación.

(a) Modelo AX-12A [18]. (b) Modelo MX-28 [23].

Figura 5.3: Motores *Dynamixel*.

Para comprender a fondo las características específicas de los modelos utilizados en el presente trabajo, se hace referencia a los manuales oficiales de los motores *Dynamixel*. Específicamente, se consulta el manual del modelo *AX-12A* [18] como se muestra en la Figura 5.3a y del modelo *MX-28* [23] correspondiente a la Figura 5.3b. A continuación, en la Tabla 6 se detallan las características más relevantes de ambos modelos de motores *Dynamixel*. Los aspectos comparados incluyen parámetros clave como los pulsos de encoder, rango de giro, la velocidad de transmisión (baudios), la resolución, la relación de transmisión y el voltaje requerido. Estos factores fueron considerados durante el diseño del robot, ya que influyen directamente en su rendimiento. El Modelo *MX-28* se caracteriza por una mayor resolución de pulsos de encoder y un rango mayor de giro, lo que lo hace ideal para tareas que requieren mayor precisión. Por otro lado, el modelo *AX-12A* ofrece una menor resolución de pulsos de encoder y un rango menor de giro.

AX-12A	**MX-28**
▪ **Velocidad de transmisión:** 7.834 bps - 1 Mbps	▪ **Velocidad de transmisión:** 7.834 bps - 3 Mbps
▪ **Resolución:** 0.29 °	▪ **Resolución:** 0.088 °
▪ **Rango de giro:** 0 - 300 °	▪ **Rango de giro:** 0 - 360 °
▪ **Pulsos de encoder:** 1023	▪ **Pulsos de encoder:** 4095
▪ **Relación de transmisión:** 254:1	▪ **Relación de transmisión:** 193:1
▪ **Voltaje de entrada:** 9 - 12 V	▪ **Voltaje de entrada:** 9 - 12 V
▪ **Torque:** 1.5 Nm a 12 V	▪ **Torque:** 2.5 Nm a 12 V

Tabla 6: Características de los motores *Dynamixel*.

5.3 PCB para el Control del Robot Tonatiuh

La placa de circuito impreso que actúa como el controlador principal del robot humanoide *Tonatiuh* requiere de diferentes componentes electrónicos para su funcionamiento. En la Tabla 7 se detallan los materiales así como

la cantidad indispensable para su diseño e implementación.

Componente	Cantidad
Resistencia $10k\Omega$	3
Resistencia 330Ω	2
LED	2
Circuito integrado 74LS241	1
Convertidor lógico bidireccional 3.3-5v	1
Regulador Buck	1
Conector Molex 3P	6
Conector Molex 2P	5
Interruptor	1
Fusible europeo a 5A	1
Raspberry Pi Pico	1

Tabla 7: Lista de Materiales de la tarjeta electrónica del controlador

5.4 Logitech, Inc. HD Pro Webcam C920

Para la captura de vídeo del sistema de visión se emplea una cámara web Logitech modelo C920 como se muestra en la Figura 5.4. Este modelo cuenta una óptica de lente Carl Zeiss Tessar que permite mejorar la nitidez y minimizar la distorsión de la imagen, posee una corrección automática de la iluminación. Además, tiene un campo visual de 78° con una resolución HD 1080p, es decir, 920 × 1080 píxeles. Por otro parte, tiene una demanda de energía de 500 mA a 5 V.[24].

Figura 5.4: WebCam C920 [24].

5.5 Raspberry Pi Pico

El control de los motores *Dynamixel* se realiza mediante un microcontrolador *Raspberry Pi Pico* mostrado en la Figura 5.5. Idóneo por sus principales características de acuerdo con [25] son:

- Procesador ARM Cortex M0+ de doble núcleo, reloj flexible que funciona hasta 133 MHz.
- 264kB de SRAM, y 2MB de memoria Flash a bordo.
- USB 1.1 con soporte para anfitrión o dispositivo
- 26 pines GPIO multifunción.

- 2×SPI, 2×I2C, 2×UART, 3×12-bit ADC, 16×canales PWM controlables.
- Reloj y temporizador preciso en el chip.
- 8 Maquinas de estado con IO programables (PIO) con soporte para uso personalizado de periféricos.

Figura 5.5: Microcontrolador *Raspberry Pi Pico* [26].

5.6 Raspberry Pi 5

El sistema de visión se realiza con una tarjeta *Raspberry Pi 5* como se muestra en la Figura 5.6, donde se puede apreciar su distribución electrónica, como los puertos y entradas. A continuación se describen algunas de sus características [25].

- Broadcom BCM2712 CPU Arm Cortex-A76 de 64 bits y cuatro núcleos a 2.4GHz, con extensiones de criptografía, cachés L2 de 512KB por núcleo y una caché L3 compartida de 2MB.
- GPU VideoCore VII, compatible con OpenGL ES 3.1, Vulkan 1.2.
- Salida de pantalla dual HDMI® 4Kp60 con soporte HDR.
- Wi-Fi® de doble banda 802.11ac.
- Ranura para tarjeta microSD, con soporte para modo de alta velocidad SDR104.
- 2 puertos USB 3.0, compatibles con operación simultánea de 5Gbps.
- 2 puertos USB 2.0.
- 2 transceptores MIPI de 4 carriles para cámara/pantalla
- Interfaz PCIe 2.0 x1 para periféricos rápidos (requiere un HAT M.2 separado u otro adaptador)
- Alimentación de 5V/5A DC a través de USB-C, con soporte para Power Delivery
- Encabezado estándar de 40 pines de Raspberry Pi
- Reloj en tiempo real (RTC), alimentado por una batería externa
- Botón de encendido

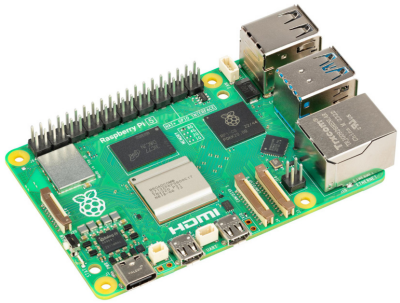


Figura 5.6: Tarjeta *Raspberry Pi 5* [27].

5.7 Batería Li-Po

En este trabajo se emplea una batería Lipo *Gens ace*, la cual suministra un voltaje de 11.1 V con una capacidad de 5000 mAh conformada por tres celdas de carga como se muestra en la Figura 5.9. Este componente es el encargado de suministrar la fuente de alimentación para los motores *Dynamixel*.

Figura 5.7: Batería Lipo 11.1 V a 5000 mAh

5.8 Placa de Expansión de Alimentación para Raspberry Pi 5

Para la alimentación de la tarjeta *Raspberry Pi 5* se utiliza un modulo de alimentación expansivo como se muestra en la Figura 5.8. Se selecciona esta placa porque brinda un voltaje de salida de 5 V con una corriente de 5 A, lo cual satisface la compatibilidad de alimentación con la *Raspberry Pi 5*.

Para satisfacer las necesidades energéticas descritas por el fabricante de la tarjeta *Yahboom* se selecciono un banco de baterías de litio recargable de 6000 mAh a 12 V, como se muestra en la Figura 5.9. No obstante, es posible seleccionar algún otro modelo que se adecué al tipo de proyecto cuidando las especificaciones de voltaje de entrada como se describe en su ficha técnica.

5.9 Librería OpenCV

OpenCV es una biblioteca utilizada para el procesamiento de imágenes con la ventaja de ser de código abierto, creada en 1999 por la empresa *Intel Corporation* a través de Gary Bradski. *OpenCV* fue diseñado con el objetivo

Figura 5.8: Tarjeta de expansión de alimentación para *Raspberry Pi 5*.

Figura 5.9: Banco de baterías de litio recargables.

de aportar a la visión por computadora y la inteligencia artificial. La biblioteca está disponible en *C, C++, Python, Java, MATLAB* y otros lenguajes, incluyendo la adaptación de la biblioteca para *Android y iOS* para aplicaciones móviles. Además es posible ejecutarlo en *Linux, Windows y Mac OS X*.

El objetivo principal de *OpenCV* es proporcionar una base robusta de visión por computadora practica y amigable con el usuario, brindado alternativas del sistema sofisticadas, debido a esto, esta biblioteca contiene más de 500 funciones que abarcan diversas áreas de la visión [20].

5.10 Entorno de desarrollo integrado Thonny

Thonny es un entorno de desarrollo integrado *(IDE)* de software libre especifico para lenguaje de *Python*. Es una herramienta que permite programar, depurar y corregir errores, entre otras funciones. Una de sus principales ventajas es la compatibilidad entre sistemas operativos como *Linux* y *Windows*. Además, de contar con soporte para aplicaciones de *Micropython* utilizadas para la tarjeta *Raspberry Pi Pico*, este *IDLE* se encuentra preinstalado en el sistema operativo de *Raspberry Pi* por defecto, lo que permite trabajar sin necesidad de instalar software adicional.

5.11 Implementación del sistema de visión para la detección de objetos

La implementación del sistema de visión para la detección de objetos se lleva a cabo en base a la metodología en espiral mostrada en la Figura 5.10, donde el proceso esta dividido en subprocesos en el que en cada uno de ellos existe la planificación, análisis, desarrollo y evaluación.

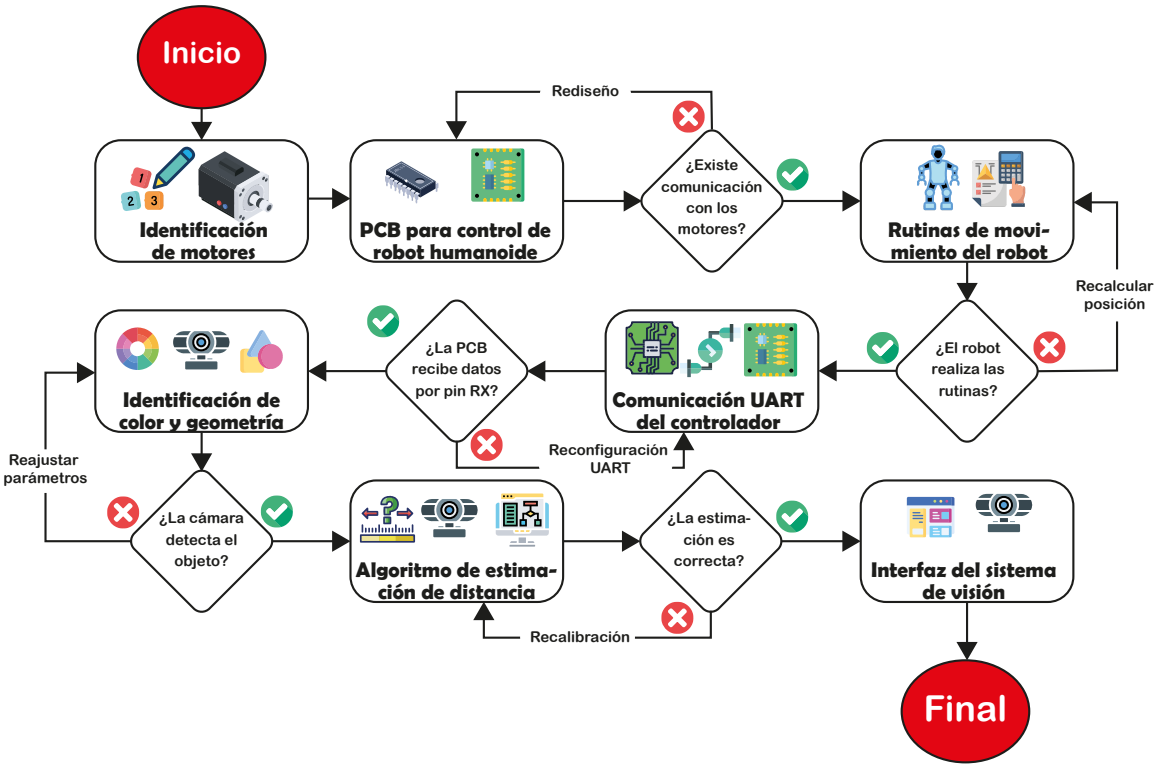


Figura 5.10: Diagrama de la metodología

La Figura 5.10 presenta en forma de diagrama de flujo los pasos necesarios para la implementación, así como las pruebas y realimentaciones necesarias para la corrección de posibles fallas. Se parte de la identificación y modificación de los números de identidad de los motores *Dynamixel*, seguido por el diseño de la tarjeta electrónica controladora a través del microcontrolador *Raspberry Pi Pico*.

La primera prueba a realizar consiste en la comunicación del controlador con los motores con el fin de manipular su valor de posición deseado. Por otro lado, para el desarrollo de la rutina de caminado, por medio de la programación del controlador, se define la posición de los 22 motores y se verifica que el robot pueda avanzar por cuenta propia. Hasta este punto se asegura el funcionamiento del robot y el controlador, por lo que se procede a vincular con la tarjeta *Raspberry Pi 5* para la transmisión de información del sistema de visión con el robot. Una vez vinculadas ambas tarjetas, se desarrollan los algoritmos del sistema de visión, correspondientes a la detección y seguimiento del objeto de estudio.

A continuación, se presenta con mayor detalle cada una de las etapas de la metodología que comprenden este trabajo.

5.12 Identificación de motores

En la primer etapa de la metodología, se requiere la identificación y nomenclatura de los 22 motores *Dynamixel* por diversas razones. La primera, dado que en una línea de comunicación por protocolo *UART*, no debe existir 2 o más dispositivos con el mismo número de identificación. De lo contrario, generaría problemas en la recepción de los motores.

Otra razón es que, al conocer y redefinir los números de identificación se simplifica la programación de la secuencia de movimiento, la cual se explicará posteriormente en otra subsección.

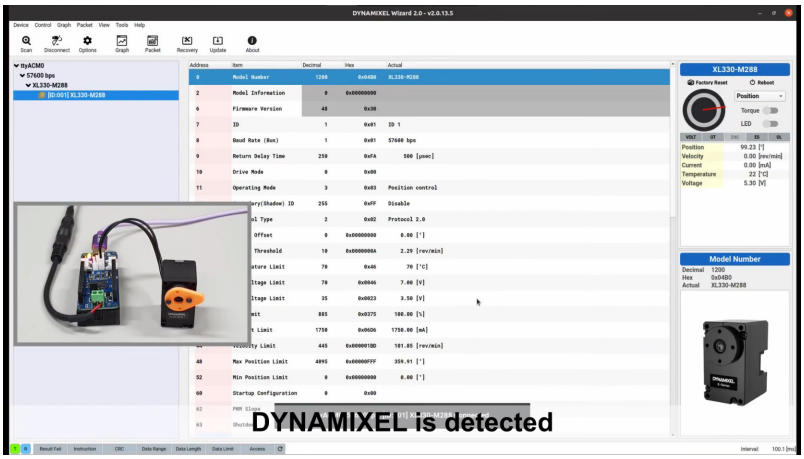


Figura 5.11: Comunicación con motor *Dynamixel* mediante *DynamixelWizad*

Utilizando el software de *DynamixelWizard* mostrado en la Figura 5.11, mediante el adaptador *USB* de la marca *Dynamixel*, permite escanear a todos los motores que se encuentren en la línea de comunicación. El software permite realizar la lectura de los valores de los registros, así como la escritura en aquellos que lo permiten. De acuerdo a [18] y [23], ambos modelos de motores permiten modificar el valor del tercer registro, también llamado dirección, que permite editar su número de identificación.

5.13 PCB para Control de Robot Humanoide

De acuerdo a la información consultada por el fabricante [18] y [23], el circuito mínimo para la etapa electrónica de la comunicación de los motores *Dynamixel* se muestra en la Figura 5.12.

La Figura 5.12 muestra que para la transmisión de datos, se requiere un sistema bidireccional, ya que en los motores, el cable de datos funciona como receptor y transmisor. Para resolver esta conexión, se requiere un circuito

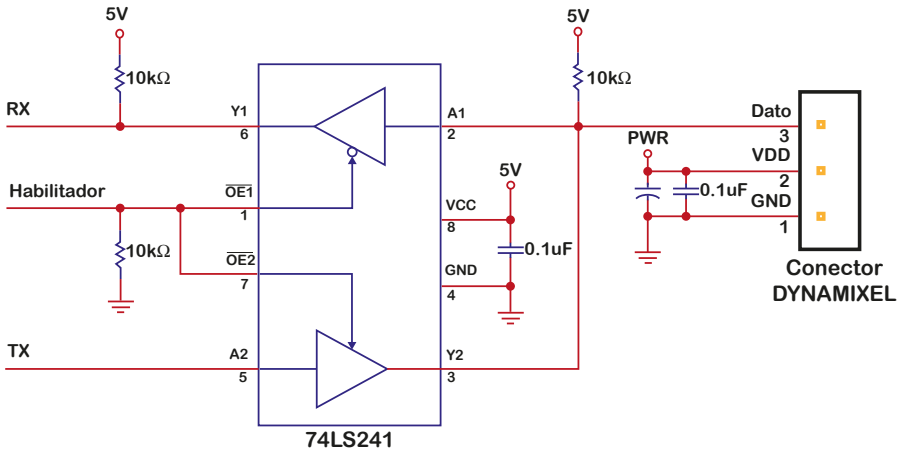


Figura 5.12: Diagrama eléctrico para comunicación TTL [18].

capaz de alternar la dirección de comunicación mediante alta impedancia. El CI 74LS241 es una opción, dado que permite cambiar la dirección con un solo pin habilitador para asegurar que no se activen ambas direcciones.

Para adaptar el circuito anterior con la tarjeta *Raspberry Pi Pico*, se requiere convertir los niveles lógicos de las señales, debido que el microcontrolador mencionado trabaja con lógica de *3.3 V*, mientras que el 74LS241 opera con *5 V*.

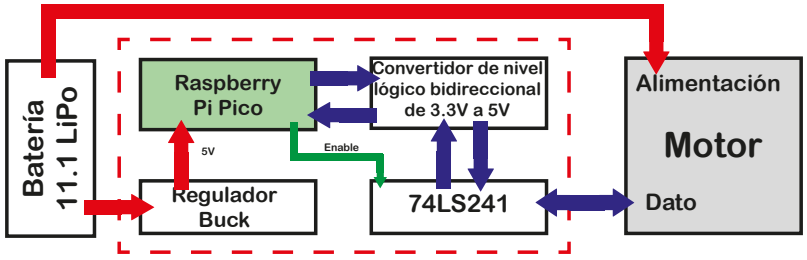


Figura 5.13: Diagrama del actuador de control.

La Figura 5.13 muestra de forma general la interconexión entre los componentes más esenciales para el desarrollo de la PCB que actúa como controlador. La batería *LiPo* suministra el voltaje a los 22 motores del robot, así como al regulador *Buck*, el cual está ajustado a una salida de voltaje constante de *5 V* para alimentar a toda la etapa digital del sistema. La tarjeta *Raspberry Pi Pico* utiliza el protocolo *UART* para enviar y recibir las tramas de datos mediante su transmisor y receptor, respectivamente. Dichas señales pasan a través del convertidor lógico bidireccional para elevar su valor de *3.3 V* a *5 V*. Las conexiones continúan por el circuito integrado 74LS241, el cual es controlado a través de su habilitador con una salida digital del microcontrolador con el fin de evitar que los datos enviados y recibidos se crucen. La salida de la compuerta, que actúa como una conexión bidireccional, conecta finalmente con el pin de datos de los motores.

Para la elaboración de la tarjeta electrónica o PCB, se utiliza el software de *Proteus*. La Figura 5.14 muestra

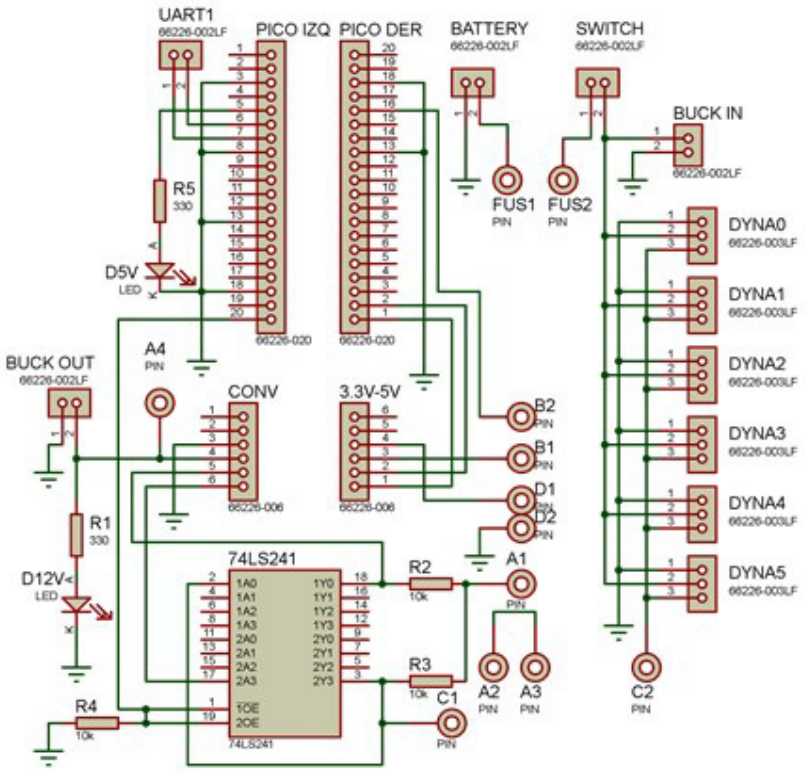


Figura 5.14: Diagrama esquemático de la PCB.

el diagrama esquemático empleado, en el que se asigna un conector a cada extremidad del robot. Por otro lado, se consideran conectores de 2 pines para el interruptor, la batería *LiPo*, la entrada y salida del regulador *Buck* y el puerto *UART* para la conexión con la tarjeta *Raspberry Pi 5*.

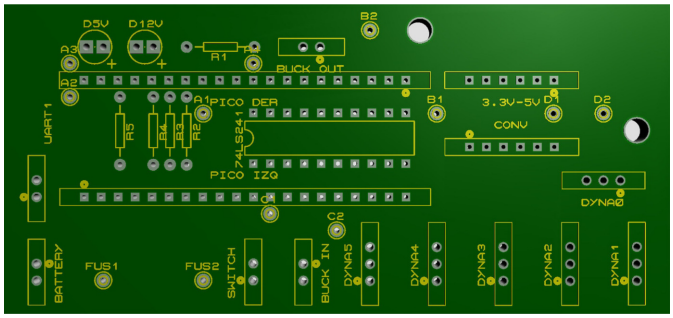


Figura 5.15: Vista de la *PCB* en *Proteus*.

En la Figura 5.15 se muestra el diseño implementado del controlador, donde todos los conectores de los motores se encuentran ordenados y alineados en el mismo sentido para evitar confusión al momento de su conexión.

5.14 Rutinas de movimiento del robot

Para la implementación de la rutina de movimiento del robot *Tonatiuh* se requiere que todos los motores se encuentren conectados al controlador. Como se mencionó anteriormente, ambos modelos de los motores *Dnamixel* permiten la conexión de múltiples dispositivos dentro de una línea de una comunicación para su control.

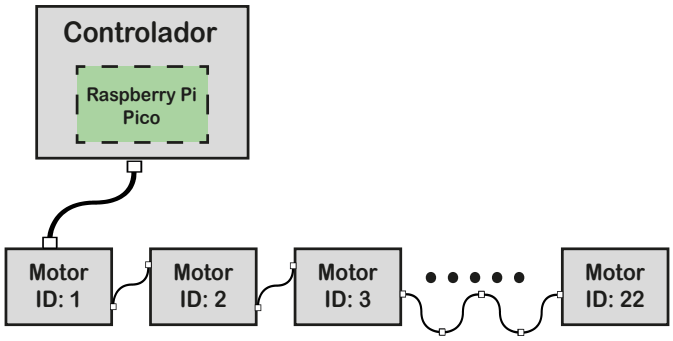


Figura 5.16: Representación de conexión de motores *Dynamixel* con el controlador.

La Figura 5.16 expone de forma general la conexión de todos los dispositivos. Ambos modelos de motores cuentan con dos conectores, esto con la finalidad de conectarse uno con otro. Como se mostró anteriormente en el diseño del controlador, este dispone de 5 conectores para los motores *Dynamixel* y para comunicar los 22 motores del robot.

Configuración del puerto *UART0*

Para la comunicación entre la tarjeta *raspberry Pi Pico* y los motores *Dynamixel* se configuro el puerto *UART0* mapeado a los pines 16 y 17 correspondientes a *TX* y *RX* respectivamente. Los demás parámetros de comunicación se establecieron como se muestra en la Tabla 8.

No.UART	Baudrate	Pin Tx	Pin Rx
0	1000000	16	17

Tabla 8: Configuración protocolo *UART* para motores *Dynamixel*.

Donde:

- **No. UART**: Corresponde al identificador de *UART*, únicamente puede ser un valor de 0 correspondiente al *UART0* o un valor de 1 correspondiente al *UART1*.
- **Baudrate:** Es la velocidad de comunicación de bits de transferencia por segundo (baudios).
- **Pin Tx:** Es el pin configurado para la transferencia de datos.
- **Pin Rx:** Es el pin configurado para la recepción de datos.

Después de la configuración del puerto, se programa el controlador en lenguaje de *Python*, en el cual se implementa un algoritmo para la generación de la trama de para los motores. Para la manipulación de la posición de

todos los motores, se utiliza el movimiento asíncrono, dado que este permite que al termino de la trama de datos, todos los motores se accionen simultáneamente. De acuerdo a [17], este comando recibe el nombre de *Sync Write* designado por el protocolo con un valor hexadecimal de *0x83*.

Es importante conocer que esta instrucción únicamente se puede emplear cuando la longitud de los datos es la misma y funciona solamente en una sola dirección [17]. Para la ejecución de dicha instrucción el paquete de parámetros que se debe enviar es el siguiente:

Item	Descripción
Instrucción	0x83
Parámetro 1	Dirección inicial
Parámetro 2	Longitud de datos para escribir
Parámetro 3	[1^{er} dispositivo] ID
Parámetro 4	[1^{er} dispositivo] 1^{er} byte
Parámetro 5	[1^{er} dispositivo] 2^{do} byte
...	...
Parámetro L+3	[1^{er} dispositivo] L-enésimo byte
Parámetro L+4	[2^{do} dispositivo] ID
Parámetro L+5	[1^{er} dispositivo] 1^{er} byte
Parámetro L+6	[1^{er} dispositivo] 2^{do} byte
...	...
Parámetro 2L+4	[1^{er} dispositivo] L-enésimo byte

Tabla 9: *Sync Write* paquete de instrucciones.

Haciendo énfasis en el parámetro 2 de la Tabla 9 la longitud de datos se calcula con la siguiente fórmula:

$$l = ((L + 1) * N) + 4 \tag{5.1}$$

Donde:

- **L**: Es la longitud de datos, en este caso en particular es igual a *0x04*.
- **N**: Número de motores *Dynamixel*.

Otro punto importante a considerar es que el *ID* del paquete se debe configurar como broadcast y se establece con el valor en hexadecimal de *0xFE*.

Diagrama de rutinas de movimiento

La Figura 5.17 presenta un diagrama de flujo que explica de forma general el funcionamiento del robot. Cuando se inicia el programa, el robot toma la pose inicial para sostenerse. Después queda a la espera de recibir un dato en el *buffer* por medio del protocolo de comunicación, en caso de no recibir nada, permanecerá estático, es decir,

en la pose inicial. Posteriormente, si se reciben datos, estos serán almacenados en sus respectivas variables. Lo que permite al algoritmo de control de movimiento conocer la localización de la esfera y accionar la rutina de movimiento correspondiente.

Sí el objeto no se encuentra en el rango de detección de la cámara, se activa la rutina de búsqueda para modificar la posición de los motores, los cuales controlan la orientación de la cámara, hasta encontrar el objeto. Cuando el objeto sea detectado, se activa la rutina de seguimiento, la cual consiste en modificar la posición de los motores que controlan la orientación de la cámara, con el fin de que esta quede centrada con respecto al centro de la circunferencia del objeto. De tal forma que, si se mueve el objeto, se mueve la cámara. En caso de que objeto se encuentre centrado con la cámara, y esta a su vez se encuentre alineada con el torso del robot, se calcula la distancia de la esfera.

Sí el objeto no se encuentra cerca del robot, para este caso, cerca se refiere a una distancia menor a 13 cm, entonces se activa la rutina de caminado para tratar de llegar al objeto. Cuando este sea detectado cerca, el robot activara la rutina de agarre para levantar sus extremidades superiores simulando el agarre del objeto. En caso de que sus extremidades ya se encuentren levantadas, no hay necesidad de repetir la rutina. En el Anexo 7.3, se muestra el algoritmo de programación aplicado.

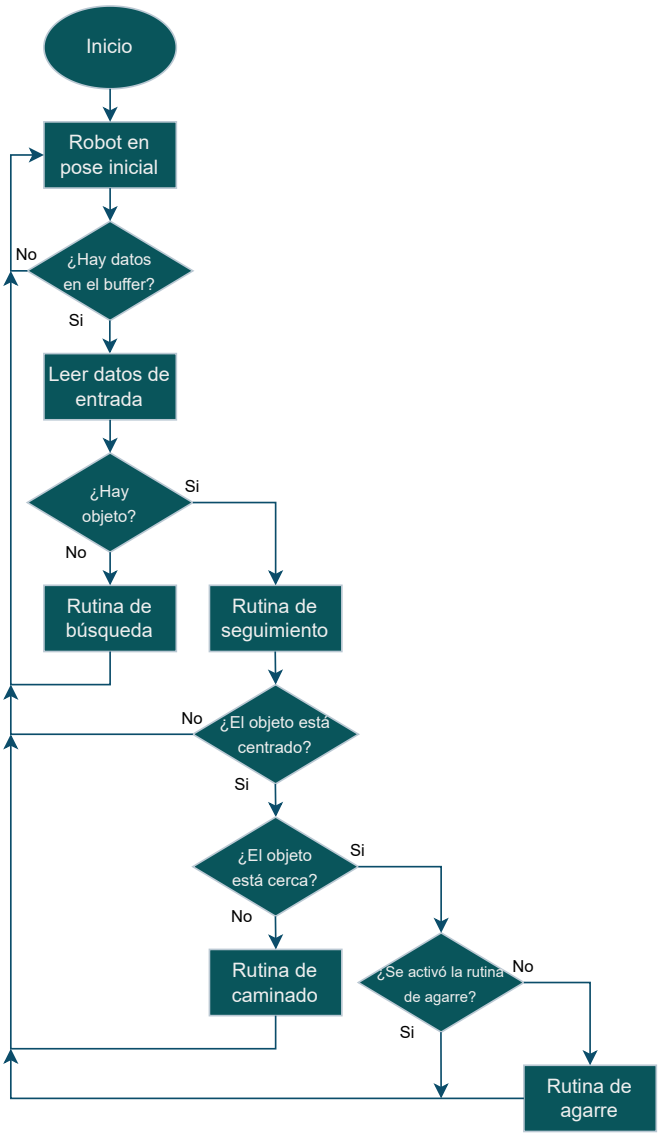


Figura 5.17: Diagrama de rutinas de movimiento.

5.15 Comunicación UART del controlador

Para la comunicación entre las tarjetas *Raspberry Pi 5* y *Raspberry Pi Pico*, se realiza mediante el protocolo *UART* con los *GPIO* de ambas tarjetas. Un dato importante a considerar es que el microcontrolador tenga al menos dos puertos *UART*. Puesto que, un puerto será configurado para la comunicación entre ambas tarjetas electrónicas. Mientras que el otro será configurado para la comunicación con los motores *Dynamixel*.

Configuración del puerto *UART1*

Para la comunicación entre ambas tarjetas se utilizo el puerto *UART1* mapeado a los pines 6 y 7 correspondientes a Tx y Rx. Los demás parámetros fueron configurados como se muestra en la Tabla 10.

No.UART	Baudrate	Pin Tx	Pin Rx
1	96000	6	7

Tabla 10: Configuración protocolo *UART1* para la tarjeta *Raspberry Pi 5*.

Donde:

- **No. UART**: Corresponde al identificador de *UART*, únicamente puede ser un valor de 0 correspondiente al *UART0* o un valor de 1 correspondiente al *UART1*.
- **Baudrate:** Es el valor correspondiente a la velocidad de comunicación de bits de transferencia por segundo (baudios).
- **Pin Tx:** Es el pin configurado para la transferencia de datos.
- **Pin Rx:** Es el pin configurado para la recepción de datos.

5.16 Identificación de color y geometría

El objeto de estudio a probar en el sistema de visión trata de una esfera de espuma de poliestireno de 8 cm de diámetro pintada de color azul. El presente trabajo fue limitado solo a dicho objeto, pero tiene la posibilidad de adecuarse al reconocimiento de figuras geométricas de 3 y 4 lados, así como a los diferentes colores espectro visual.

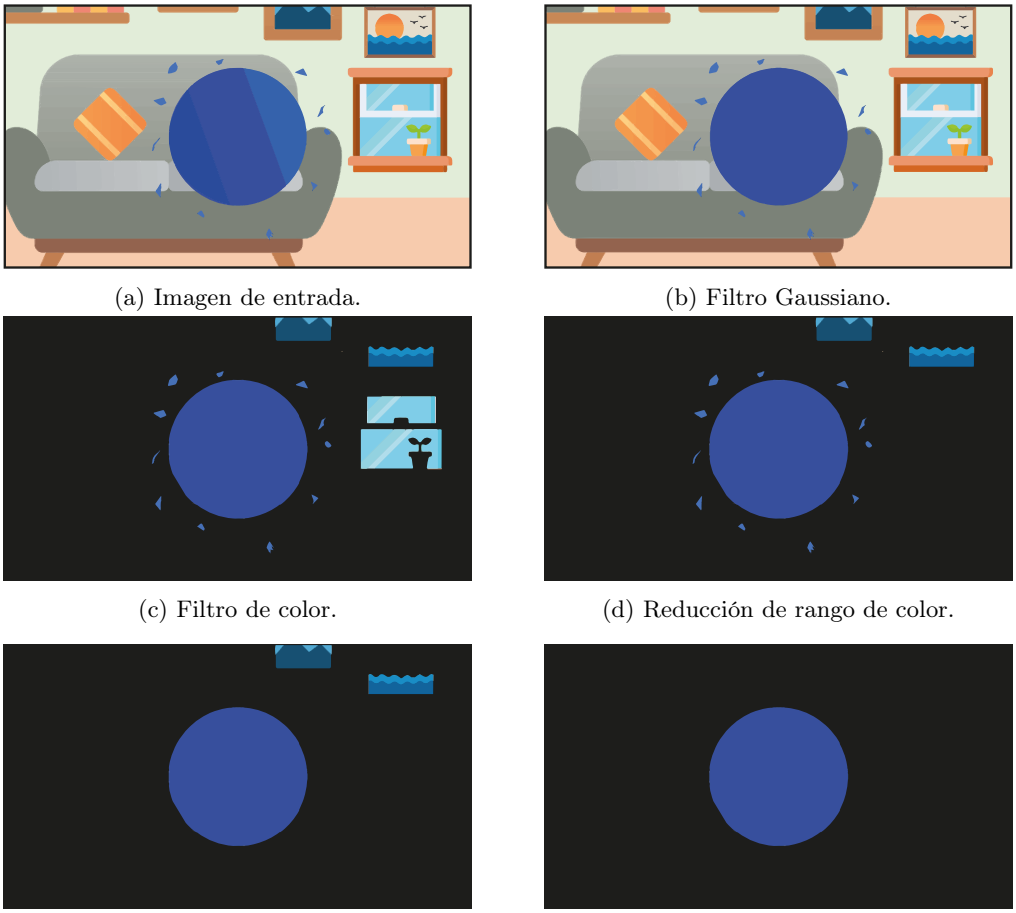
La Figura 5.18 muestra la secuencia pasos que representa de forma general el funcionamiento del algoritmo del sistema de visión para la identificación del objeto mediante su color y forma geométrica.

Para la detección del objeto a través del sistema de visión se emplea el lenguaje de programación de *Python*, con apoyo de la librería de *OpenCV*, la cuál se importa y utiliza bajo el nombre de *cv2*. En el Código 1, se muestran las librerías que se emplean. Entre estas se encuentran *math*, que permite realizar operaciones matemáticas, *serial* establece comunicación por puerto serial con los dispositivos, *time* que posibilita agregar retardos de tiempo en el código, *struct* que permite el empaquetado de datos y *numpy* crea arreglos de matriciales.

```
1 import cv2, math, serial, time, struct
2 import numpy as np
```

Código 1: Librerías de Python

En la Figura 5.18a, se observa una simulación de la imagen de entrada capturada por el sistema, en este caso una esfera de color azul, la cual muestra diferentes tonalidades generadas por el reflejo de la luz sobre sí misma. También se encuentra rodeada de pequeñas manchas azules las cuales simulan ruido dentro de la imagen. Este proceso de captura del sistema de visión se ejemplifica con el Código 2, donde a partir del objeto de captura *cap*



(a) Imagen de entrada.

(b) Filtro Gaussiano.

(c) Filtro de color.

(d) Reducción de rango de color.

(e) Dilatado y erosionado.

(f) Detección de circunferencias.

Figura 5.18: Funcionamiento de detección del objeto.

descrito en la línea 2 el sistema puede obtener los frames en tiempo real por medio de la cámara. Por su parte la función *cap.read* descrita en la línea 10 permite almacenar el frame en la variable *frame_original*.

```
1 # Objeto de captura de video desde la cámara
2 cap = cv2.VideoCapture(0)
3
4 # Condición de apertura de cámara
5 if not cap.isOpened():
6     print("Error: No se puede abrir la cámara")
7     exit()
8
9 # Captura de un frame
10 ret, frame_original = cap.read()
```

Código 2: Captura del sistema de visión

La Figura 5.18b ejemplifica el resultado de aplicar un filtro gaussiano, donde se aprecia que ahora la esfera no tiene tantas tonalidades. El objetivo de usar este filtro, consiste en obtener una superficie más homogénea de color en

la esfera. La homogeneidad dependerá del tamaño de la matriz de convolución, a mayor tamaño, mayor homogeneidad tendrá. Este algoritmo se describe en el Código 3 específicamente en la línea 2 con la función *cv2.GaussianBlur*.

Posteriormente se aplica un cambio de modelo de color de *BGR* a *HSV* que se muestra en la línea 3 con la función *cv2.COLOR_BGR2HSV*, con el objetivo de definir mediante el valor del tono, el rango de color que se desea filtrar. La Figura 5.18c ilustra el resultado de aplicar la operación de *InRange* con un enmascaramiento del tipo *bitwise AND* a la imagen original de entrada. El comando *cv2.inRange*, presentado en la linea 10 del código 3, permite delimitar el rango de color que sea desea conservar, definiendo un valor superior e inferior para cada uno de los canales del espacio de color. Dicha función devuelve una imagen binaria de salida que proviene de los pixeles que se encuentran dentro del rango de los canales, saturandolos al valor máximo de 255, mientras que los pixeles que se encuentran fuera del rango, son ajustados al valor mínimo de 0.

La Figura 5.18d representa el resultado de reducir el rango del tono de color en la imagen. Comparando con la Figura 5.18c, se aprecia la disminución del ruido proveniente de objetos distintos a la esfera, esto se debe a que su tonalidad y saturación se encuentran fuera de los límites permitidos.

Se observa aún ruido por lo que es necesario aplicar operaciones morfológicas de erosionado y dilatado como se muestra en la Figura 5.18e, donde se aprecia que el ruido alrededor de la esfera ha desaparecido. Esto ocurre al aplicar el erosionado, el tamaño del ruido era menor que al tamaño del elemento estructurante, por lo que al aplicar el dilatado para reconstruir la imagen, estas pequeñas zonas se pierden permanentemente. Sin embargo, sigue estando presente algunos objetos filtrados que generan ruido. Ese proceso se muestra en el Código 3 en la línea 13, donde se crea el elemento estructurante por medio de la función *cv2.getStructuringElemet*. Dicho elemento se propone con una forma circular definida por *cv2.MORPH_ELLIPSE*, y con un tamaño de kernel variable dependiendo el valor establecido en las trackbar. Por otra parte, en las líneas 16 y 17 se encuentra las operaciones morfológicas establecidas por las funciones *cv2.erode y cv2.dilate*, mientras que en la línea 18 se presenta la operación textitcv2.bitwise_and que realiza un enmascaramiento de la imagen dilatada con la imagen original.

```
1    # Suavizado y  Cambio de espacio de color
2    frame_suavizado = cv2.GaussianBlur(frame_original, (kernel_size, kernel_size),
     0)
3    frame_hsv = cv2.cvtColor(frame_suavizado, cv2.COLOR_BGR2HSV)
4
5    # Rango de color azul en HSV basado en los valores de los trackbars
6    lower_blue = np.array([lower_h, lower_s, lower_v])
7    upper_blue = np.array([upper_h, upper_s, upper_v])
8
9    # Obtención de mascara binaria del filtrado de color
10   frame_mascara = cv2.inRange(frame_hsv, lower_blue, upper_blue) # Mascara color
11
```

```
12      # Creación del elemento estructurante
13      element = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2 * size_element + 1, 2
         * size_element + 1))
14
15      # Erosionado, Dilatado y Enmascaramiento AND
16      frame_erosionado = cv2.erode(frame_mascara, element, iterations=1)
17      frame_dilatado = cv2.dilate(frame_erosionado, element, iterations=1)
18      frame_color_morf  = cv2.bitwise_and(frame_original, frame_original, mask =
        frame_dilatado)
```

Código 3: Aplicación de filtros

Después de realizar las operaciones morfológicas a la imagen por medio del algoritmo descrito en el código 3, se implementa una función para la detección de círculos. Dicha función tiene como valor de entrada el contorno existente en la imagen. En la línea 5 del código 4 se obtiene el perímetro del contorno detectado mediante la función *cv2.arcLength*. El objetivo de conocer el perímetro es para realizar una aproximación de la figura por medio de la función *cv2.approxPolyDP*. Esto se debe por la resolución de los pixeles, ya que al momento de contabilizar los lados de dicha figura, se podrían obtener valores muy altos, por lo que afectaría principalmente cuando se quisiera detectar figuras de pocos lados como triángulos y cuadriláteros. Si no se realiza la aproximación, los contornos serán reconocidos como figuras con demasiados lados.

En la línea 11 se encuentra una condicional en donde se compara la cantidad de lados de la figura aproximada para comprobar que no sean figuras menores a 4 lados. Cabe mencionar que se opta por 4 lados porque es poco común encontrar figuras de 5 lados o más, por lo que algo mayor a un cuadrilátero, será considerado círculo.

La línea 14 obtiene el área del contorno por medio de la función *cv2.ContourArea*, mientras que la linea 15 permite conocer el radio con la función *cv2.minEnclosingCircle*, la cual se encarga de calcular el círculo más pequeño posible que encierre completamente un contorno. Una vez conocido el radio del círculo estimado de la línea anterior, se calcula el área para este. Por lo que al conocer el área real del contorno y el área teórica del circulo estimado, se comparan en la línea 21 considerando una tolerancia que se ajusta a través del trackbar. Sí el área del circulo estimado se encuentra dentro de la tolerancia del área real, se regresa un *True*, en caso contrario, un *False*.

```
1 # Función de detección de circulos por contorno y area
2 def es_circulo(contorno):
3
4 # Calculo del perímetro del contorno
5       perimetro = cv2.arcLength(contorno, True)
6
7       # Aproximación del contorno
8       aproximacion = cv2.approxPolyDP(contorno, 0.04 * perimetro, True)
9
10      # Condición de cantidad de lados
```

```
11      if len(aproximacion) > 4:
12
13          # Calculo de área, radio y coordenadas del círculo aproximado
14          area = cv2.contourArea(contorno)
15          (cx, cy), radio = cv2.minEnclosingCircle(contorno)
16
17          # Calculo de area a partir del radio
18          area_circulo = np.pi * (radio ** 2)
19
20          # Comprobar si el área del contorno es similar al área del círculo
21          if min_tol * area_circulo/100 <= area <= max_tol * area_circulo/100:
22              return True
23
24      return False
```

Código 4: Detección de circunferencia.

En caso de que en la imagen se filtre más de un objeto dentro del rango de color y que también sea considerado círculo de acuerdo a la tolerancia definida, se implementa un un algoritmo para considerar únicamente al objeto de mayor área captada por la cámara.

```
1   # Inicializar variables para el objeto más grande
2   max_area = 0
3   max_contour = None
4
5   # Encontrar el contorno más grande
6   for contour in contornos_circulos:
7       area = cv2.contourArea(contour)
8       if area > max_area:
9           max_area = area
10          max_contour = contour
11
12  # si detectó la figura más grande
13  if max_contour is not None:
14
15      # Detección de circulo mayor y coordendas radio
16      (cx, cy), radio = cv2.minEnclosingCircle(max_contour)
```

Código 5: Detección Círculo Mayor.

En el código 5 se observa dos variables en las líneas 2 y 3 que se limpian antes de utilizarlas. En la línea 6 se implementa un ciclo *For* el cual se repite para cada contorno existente en la imagen de los contornos. Se calcula el área del contorno actual y se compara con el valor del área máxima en la linea 9. Para la primera iteración, el valor máximo es 0, por lo que tomará inmediatamente el valor del primer contorno. En caso de encontrar un contorno mayor se actualizan las variables hasta terminar el ciclo. La línea 10 permite identificar sí existe por lo menos un

contorno para calcular su radio a partir del círculo estimado utilizando la función de *cv2.minEnclosingCircle* en la línea 16.

Finalmente, la Figura 5.18f muestra el resultado de aplicar este algoritmo de detección de contornos descrito en el Código 4 y 5 por lo que se obtiene unicamente el objeto deseado filtrado. El Anexo 7.2 contiene el código completo de la programación realizada.

5.17 Algoritmo de estimación de distancia

Una vez detectado el objeto, se busca un algoritmo que permita calcular sus coordenadas del centro, así como la distancia a la que se encuentra con respecto al lente de la cámara. Para abordar este problema, se implementa la solución que se muestra en la Figura 5.19, la cual consiste en colocar la esfera azul a 9 cm de la cámara para que el algoritmo descrito en la subsección anterior detecte el objeto.

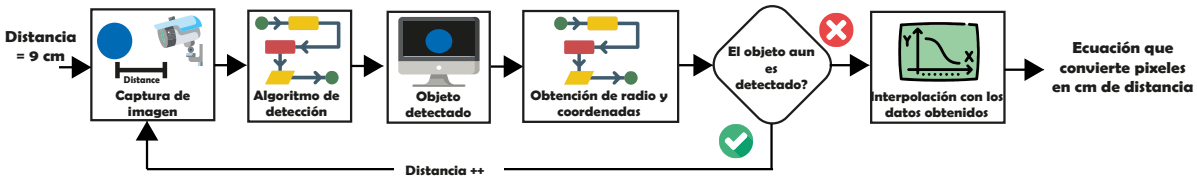


Figura 5.19: Diagrama de funcionamiento del seguimiento del objeto.

Por medio del cálculo de mínimo círculo que envuelve el contorno de la función *cve.minEnclosingCircle* se obtiene las coordenadas en pixeles del centro del círculo detectado así como el radio en pixeles. Si el sistema sigue captando la esfera, se aleja la esfera un centímetro y se repite el mismo proceso hasta que la esfera ya no sea reconocida por la cámara. Con los datos obtenidos se realiza una interpolación del radio en pixeles con la distancia de la esfera para obtener una función que como entrada reciba la detección de la esfera y de como salida la distancia de la esfera.

La Figura 5.20 presenta el resultado de la muestras realizadas al colocar la esfera frente a la cámara a una distancia conocida. Partiendo de una distancia de 9 cm se realiza la captura de la imagen para posteriormente calcular el valor del radio en pixeles. Se repite este proceso incrementando 1 cm en cada prueba, hasta alcanzar la distancia de 30 cm. Finalmente, se obtiene la gráfica mostrada donde su comportamiento esta descrito por una ecuación exponencial y un coeficiente de determinación R^2 de 0.9997. Esto indica una alta fiabilidad en los resultados, facilitando la predicción de valores a partir del radio en píxeles obtenido de la cámara.

```
1         # Detección de circulo y coordendas radio
2         (cx, cy), radio = cv2.minEnclosingCircle(max_contour)
3
4         # Calculo de distancia a partir de radio pixeles
```

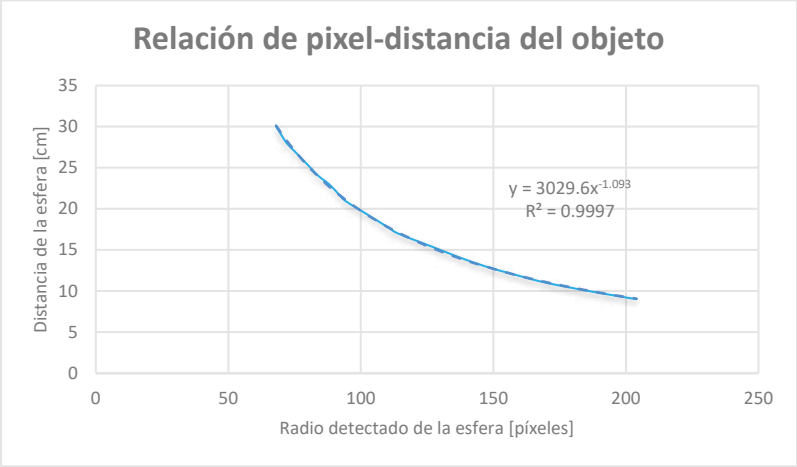


Figura 5.20: Gráfica de interpolación.

```
5        dist = 3029.6*radio**(-1.093)
6        dist = round(dist, 2)
```

Código 6: Cálculo de distancia del objeto

Como se observa en el código 6, se obtiene el radio del círculo en la línea 2, y en la línea 5 se utiliza la ecuación obtenida de la interpolación para conocer el valor de la distancia en centímetros del objeto. se aplica un redondeo a dos decimales en la línea 6. En el Anexo 7.2, se presenta el código completo del programa implementado.

5.18 Interfaz del sistema de visión

Para el ajuste de captura del sistema de visión, se desarrolla una interfaz visual utilizando *OpenCV*. Esta interfaz permite visualizar en tiempo real la imagen capturada por la cámara del robot, así como ajustar dinámicamente los parámetros de los filtros implementados mediante una ventana de trackbars. De esta manera, el usuario puede calibrar el sistema de visión de forma intuitiva y adaptarlo a diferentes condiciones del entorno.

La figura 5.21 presenta la ventana de controles que permite ajustar los parámetros de cada unos de los filtros, así como la tolerancia permitida para la detección del círculo. El primer *trackbar* permite cambiar entre las diferentes vistas aplicando los correspondientes filtros, similar a como se presenta en 5.18. El *Trackbar Kernel* permite modificar el tamaño de la matriz que forma al kernel para el filtro de suavizado. Las siguientes cuatro , las *Trackbars: Lower and Upper H and S* permiten modificar los valores superiores e inferiores de los canales del tono y la saturación del modelo de color *HSV* para la función *cv2.InRange*, la cual se utiliza para el filtro de color. La *Trackbar Size Morph* permite modificar el tamaño del elemento estructurante de tipo circular para las operaciones morfológicas de erosión y dilatado. Por último las *Trackbars Min y Max Tol* ajustan la tolerancia permitida para la detección de los círculos.

Todos los trackbars se encuentran declarados en el código 7. Estos se muestran en la ventana llamada *win2* y sin ninguna función ya que el todo el código se encuentra en un ciclo infinito, por lo que siempre se está leyendo el

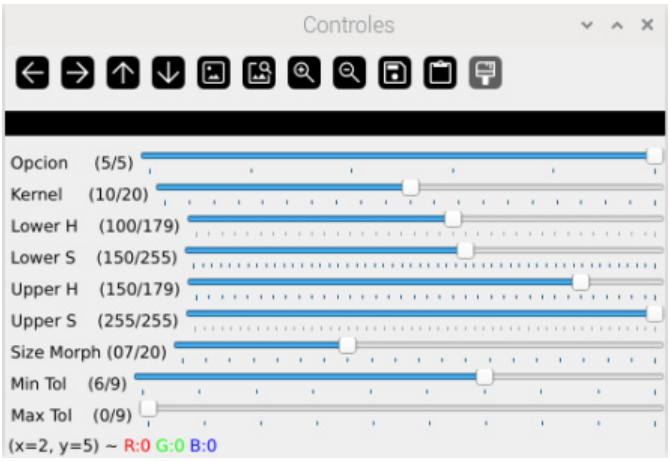


Figura 5.21: Ventana de Trackbars.

valor de cada *Trackbar*.

```
# Trackbar para cambiar de opciones en la ventana de controles
cv2.createTrackbar('Opcion', win2, 5, 5, nothing)

# Trackbar para ajustar el tamaño del kernel del filtro de suavizado
cv2.createTrackbar('Kernel', win2, 10, 20, nothing)

# Trackbars para ajustar el rango de color azul en HSV
cv2.createTrackbar('Lower H', win2, 100, 179, nothing)
cv2.createTrackbar('Lower S', win2, 150, 255, nothing)
cv2.createTrackbar('Upper H', win2, 150, 179, nothing)
cv2.createTrackbar('Upper S', win2, 255, 255, nothing)

# Trackbar para ajustar el tamaño del kernel del elemento estructurante
cv2.createTrackbar('Size Morph', win2, 7, 20, nothing)

# Trackbar para ajustar la tolerancia de detección del circulo
cv2.createTrackbar('Min Tol', win2, 6, 9, nothing)
cv2.createTrackbar('Max Tol', win2, 0, 9, nothing)
```

Código 7: Trackbars

Para ejecutar el código del sistema de visión, previamente se requiere tener un entorno virtual con las librerías mostradas en el Código 1. La Figura 5.22 muestra el procedimiento para ejecutar el código del sistema de visión desde el entorno virtual mediante la terminal del sistema de la tarjeta *Raspberry Pi 5*. En la primer línea se escribe el comando *Source* seguido de la ruta en donde se encuentra el entorno y el comando *activate* para activar el entorno virtual. En la segunda línea se observa que aparece el nombre del entorno, por lo que ahora se puede acceder a la ruta donde se encuentra el código del sistema de visión. Finalmente, en la tercera línea se muestra el nombre del archivo que se va a ejecutar.

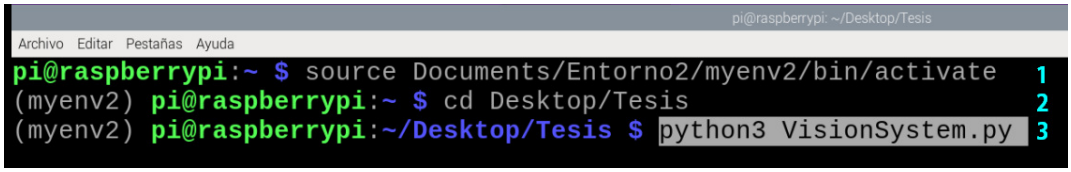


Figura 5.22: Ventana de Terminal de la tarjeta *Raspberry Pi 5*.

Para activar el funcionamiento de los motores del robot se requiere conectar la tarjeta *Raspberry Pi Pico*, la cual se encuentra montada sobre la *PCB* descrita en la sección 5.13, a un puerto *USB* de la tarjeta *Raspberry Pi 5*. Posteriormente, se ejecuta la aplicación de *Thonny*.

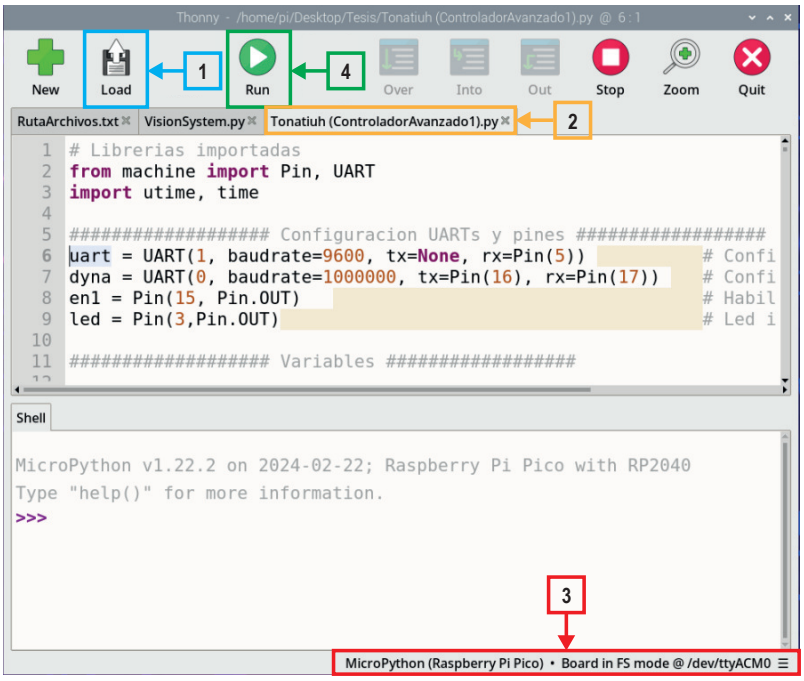


Figura 5.23: Ventana de *Thonny* en tarjeta *Raspberry Pi 5*.

La figura 5.23 muestra una captura de pantalla del *IDLE de Thonny* desde el sistema de la tarjeta *Raspberry Pi 5* y señala el orden de pasos para la activación del robot.

El primer paso es seleccionar la opción de cargar el archivo, donde se abrirá el buscador de archivos. El segundo paso corresponde a la selección del documento del controlador del robot. El tercer paso implica seleccionar el puerto serial correspondiente de la conexión de la tarjeta *Raspberry Pi Pico*. Como ultimo paso se ejecuta el archivo. El código fuente completo utilizado para el desarrollo del sistema se encuentra en el Anexo 7.2.

6 Resultados y Discusión

En esta sección se presentan los resultados obtenidos durante el desarrollo del presente trabajo, donde se abarcan diversas etapas clave que permiten demostrar su funcionalidad. En primer lugar, se detallan los logros alcanzados en el diseño y la implementación de la *PCB* para control de robot humanoide . Seguido, se describen los resultados obtenidos del sistema de visión, como de la comunicación entre las tarjetas *Raspberry Pi 5* y *Raspberry Pi Pico*. Por último, se presentan los resultados de las rutinas de movimiento del robot humanoide.

6.1 Estructura y función de la PCB para control de Robot Humanoide

La Figura 6.1 muestra el resultado del diseño de la *PCB* descrita en el apartado de metodología sección 5.13, esta funciona como controlador para los motores del robot humanoide. Su estructura electrónica esta compuesta por un microcontrolador *Raspberrry Pi Pico*, así como un regulador tipo buck el cual se alimenta directamente de una batería Lipo para suministrar una salida de *5 V* constante. También contiene un módulo convertidor lógico bidireccional ubicado debajo del regulador, con el fin de optimizar el espacio en la *PCB*. Finalmente, se presentan varios conectores tipo Molex: cinco de ellos están destinados para conectar las líneas de los motores, mientras que los demás se utilizan para la alimentación de la batería, el interruptor y el conector *UART*.

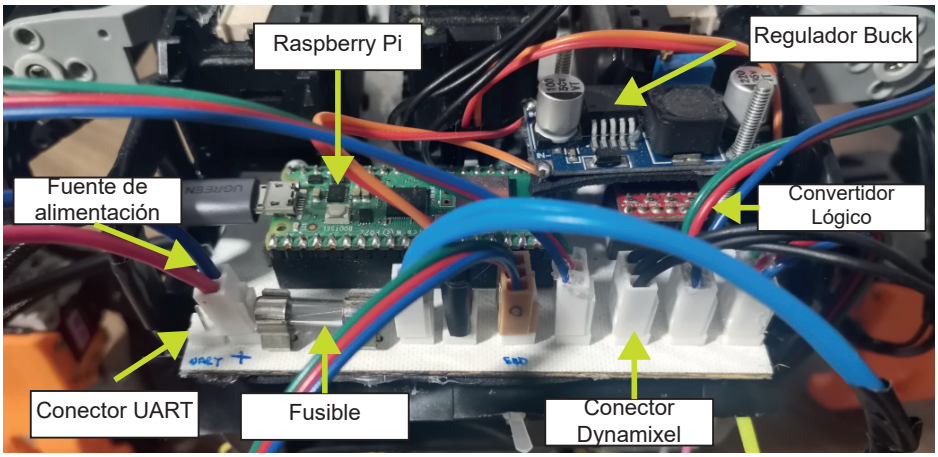


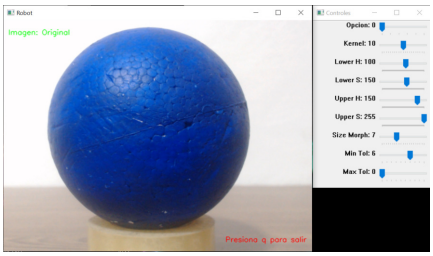
Figura 6.1: *PCB* para el Control de un Robot Humanoide.

6.2 Detección del objeto

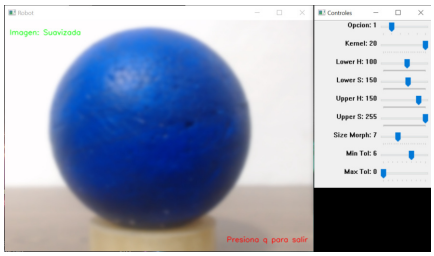
La Figura 6.2 presenta los resultados obtenidos en las distintas etapas del sistema de visión. En la Figura 6.2a se muestra la imagen de entrada, donde se puede observar el objeto de interés: una esfera azul ubicada frente a la cámara. Por otro lado, la Figura 6.2b ilustra el efecto de aplicar un filtro gaussiano de suavizado, que genera una imagen borrosa debido a la homogenización de los valores de los píxeles. Para este proceso, se utilizó un filtro con un tamaño de 21x21.

En la Figura 6.2c se presenta el enmascaramiento de la imagen original utilizando la máscara binaria generada con el filtro de color aplicado mediante la función *InRange*. Los rangos de valores utilizados fueron: Hue entre 60 y 150, y la saturación entre 103 y 255. Posteriormente, se realizaron operaciones morfológicas de erosión y dilatación para refinar la máscara. Por su parte, la Figura 6.2d muestra el resultado del filtro correspondiente a la detección del círculo de mayor tamaño, destacando tanto el contorno de la circunferencia calculada como su área y centroide, obtenidos a partir del objeto filtrado.

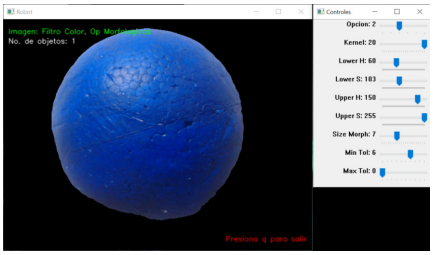
Por otro lado, la Figura 6.2e muestra a la imagen original, en la que se superpone el contorno de la circunferencia detectada del paso anterior, así como las coordenadas del centroide para localizar a la esfera azul. Finalmente, la Figura 6.2f muestra el resultado obtenido de la figura anterior, indicando el cuadrante en el que se encuentra el centro de la esfera y el valor del radio de la esfera expresado en píxeles superpuesto en la imagen.



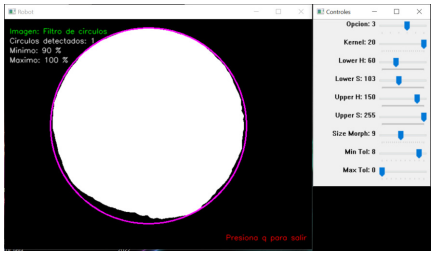
(a) Imagen de entrada.



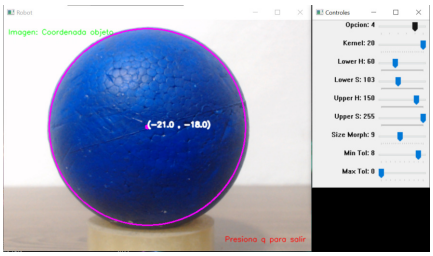
(b) Imagen con filtro de suavizado.



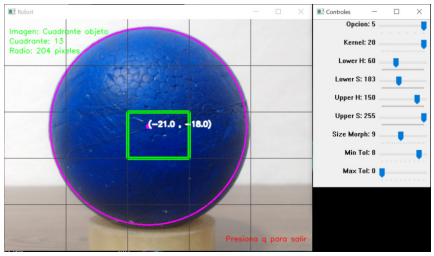
(c) Imagen con filtro de color y morfología.



(d) Imagen detección de círculos.



(e) Imagen con coordenadas del centroide.



(f) Imagen con objeto localizado en cuadrante.

Figura 6.2: Resultados del sistema de visión.

6.3 *Transmisión de datos entre Raspberry Pi 5 y Raspberry Pico*

La Figura 6.3 muestra una captura de pantalla tomada desde la *Raspberry Pi 5*, en la cual se aprecia la ventana de Python donde se captura la imagen de la cámara por medio de la interfaz desarrollada señalada en color verde. De manera similar, el cuadro azul muestra los datos enviados de la localización y distancia de la esfera detectada, por medio del protocolo *UART*. Por su parte, el cuadro rojo destaca la interfaz de la *Raspberry Pi Pico* con su entorno de *MicroPython* conectado al puerto USB *ttyACM0*. Finalmente, en color magenta se visualiza en consola en el *IDLE* de *Thonny*, los datos recibidos por el microcontrolador, correspondientes a la localización y distancia de la esfera detectada.

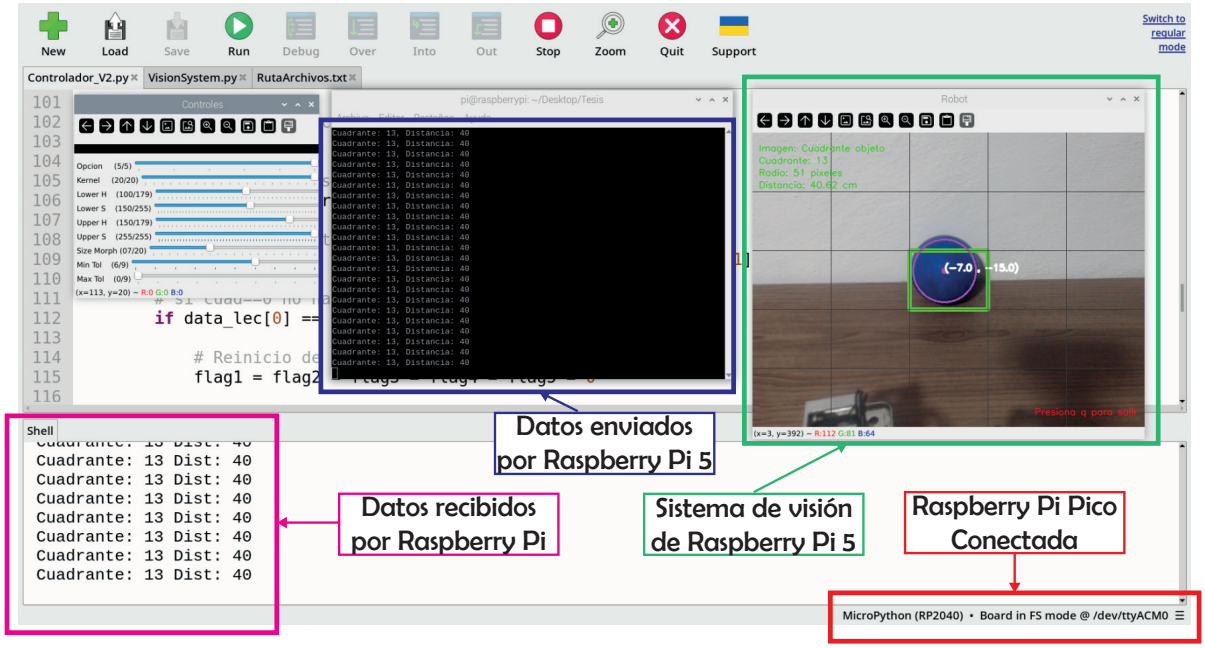
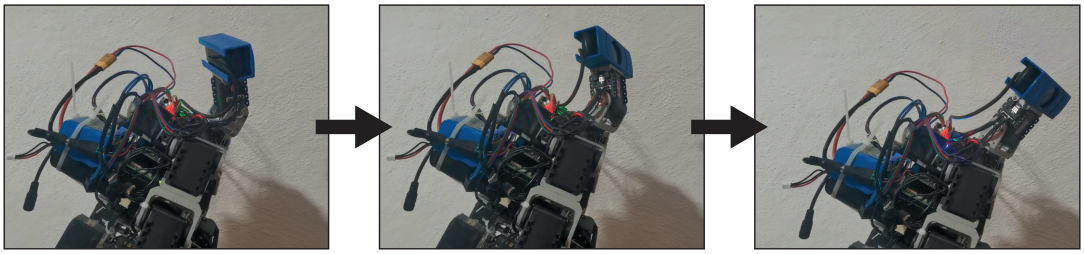


Figura 6.3: Fotografía de transmisión de datos.

6.4 *Rutina de búsqueda del objeto*

La Figura 6.4a muestra la rutina de búsqueda del objeto realizada por el robot, donde se observa como los motores correspondientes al movimiento de la cámara, en este caso los motores con *ID* 21 y 22, realizan un barrido de izquierda a derecha y posteriormente de arriba a abajo hasta detectar el objeto. Por otro lado, la Figura 6.4b muestra la vista capturada por el sistema de visión durante la ejecución de esta rutina, donde se aprecia que al no detectar el objeto continua de forma cíclica con la búsqueda.



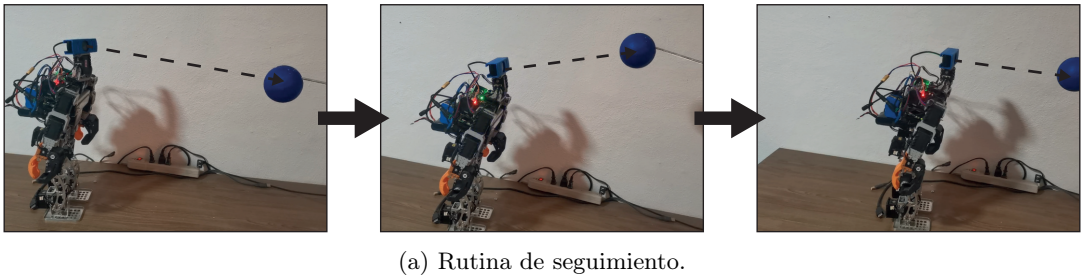
(a) Rutina de búsqueda.

(b) Vista adquirida por el sistema de visión.

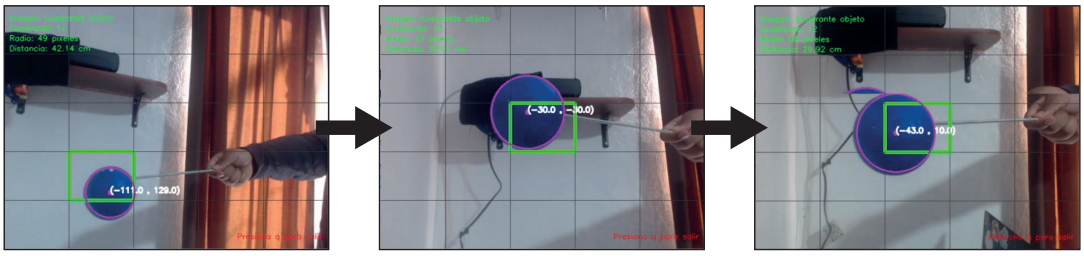
Figura 6.4: Búsqueda del objeto.

6.5 Rutina de seguimiento del objeto

La Figura 6.5a presenta una secuencia de imágenes que muestran el proceso de seguimiento del objeto que realiza el robot, en el cual se observan los movimientos de la cámara para mantener el objeto dentro de su rango de visión. Por otra parte, en la Figura 6.5b se aprecia la secuencia de imágenes correspondientes a la captura del sistema de visión, donde la esfera azul es desplazada para visualizar la rutina de seguimiento hasta que la cámara logra centrarla para dar paso a la rutina de movimiento.



(a) Rutina de seguimiento.



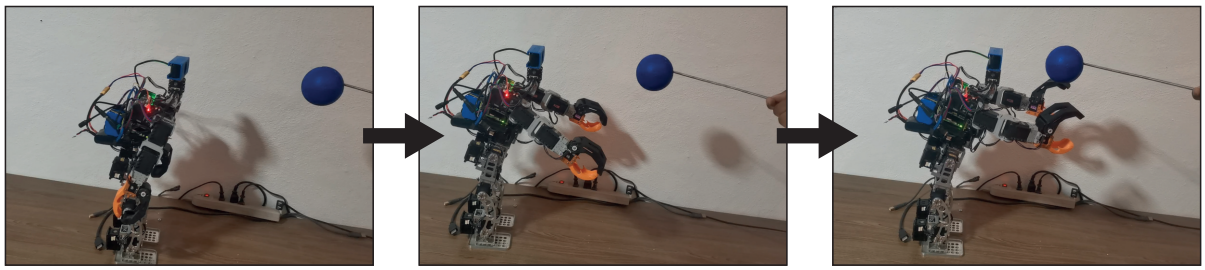
(b) Vista adquirida por el sistema de visión

Figura 6.5: Rutina de seguimiento.

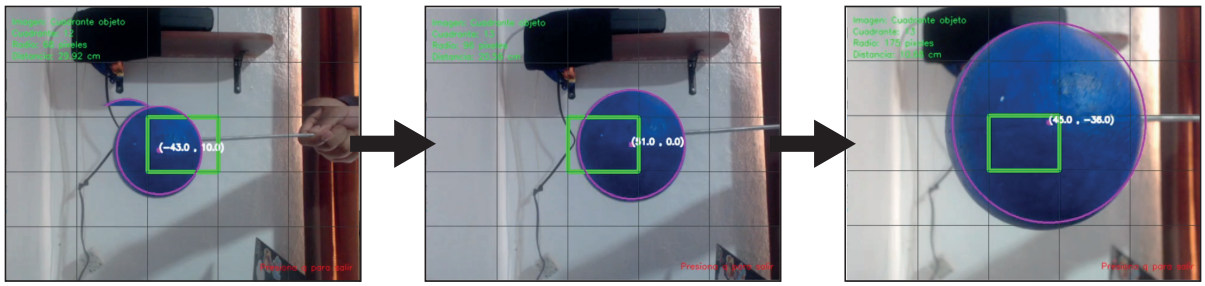
6.6 Rutina de agarre

La Figura 6.6a presenta una serie de imágenes que ilustra el funcionamiento de la rutina de movimiento del robot, la cual es accionada cuando la esfera se encuentra centrada y a una distancia menor de 13 *cm*. Mientras el robot permanece estático visualizando la esfera y esta no se encuentra centrada, únicamente ajusta la orientación de la cámara mediante los motores ID 21 y 22. Una vez que el sistema de visión detecta la esfera centrada, el robot mueve sus extremidades superiores como señal de que se dirige hacia ella. Finalmente, cuando el sistema detecta que la esfera se encuentra a una distancia menor de 13 *cm*, el robot ajusta la posición de sus extremidades y abre sus pinzas, simulando sujetarla.

Por otro lado, la Figura 6.6b muestra como es visualizada la esfera a través del sistema de visión, evidenciando cómo el tamaño del objeto aumenta progresivamente a medida que disminuye la distancia entre este y la cámara.



(a) Rutina de movimiento.



(b) Vista adquirida por el sistema de visión

Figura 6.6: Rutina de agarre.

6.7 Rutina de caminado

La Figura 6.7 muestra al robot Tonatiuh en distintas posiciones, donde se puede apreciar como las extremidades inferiores son desplazadas mientras se ejecuta la rutina de caminado. Durante esta rutina, el robot ejecuta una secuencia de movimiento alternado tipo zancada, en el que una extremidad se eleva ligeramente mientras la otra mantiene el equilibrio. De igual manera, se aprecia cómo el centro de masa del robot se desplaza lateralmente para el levantamiento de la extremidad contraria, lo que permite una caminata.

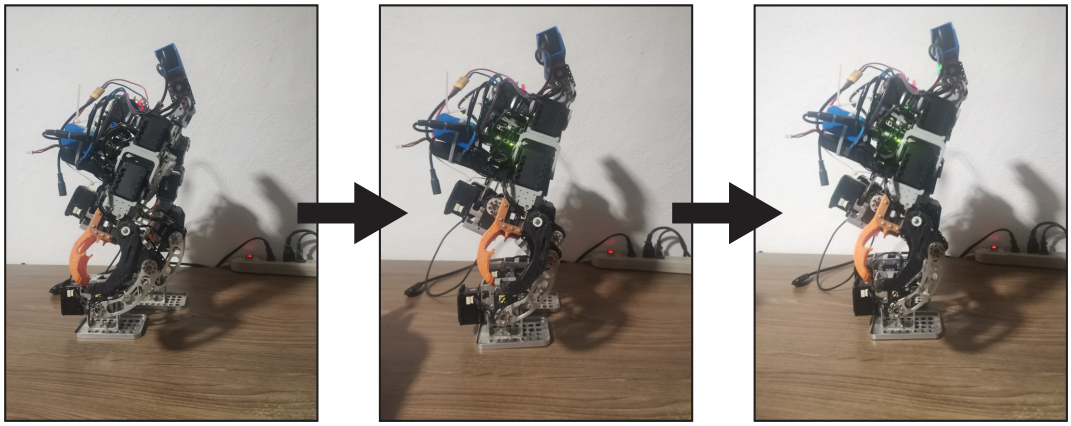


Figura 6.7: Rutina caminado.

6.8 Discusión de Resultados

El sistema de visión desarrollado cumple con los objetivos de detección y localización del objeto de interés, en este caso, una esfera azul. La implementación del algoritmo de filtrado redujo el ruido presente en las imágenes capturadas, presentando mejoras en la segmentación del color azul del objeto en diferentes condiciones de iluminación. La combinación de técnicas de suavizado de ruido y filtrado por color permitió aislar de manera efectiva los píxeles correspondientes a la esfera.

Así mismo, la detección de contornos permite identificar la circunferencia de la esfera, lo que habilita la obtención de parámetros geométricos como el centroide y el radio del objeto. Estos datos son utilizados posteriormente en la retroalimentación del sistema de control, facilitando la comunicación y toma de decisiones en el robot. Cabe destacar que el algoritmo de detección de contornos se mantiene estable bajo distintas condiciones, como lo es la iluminación, distancia y rango de tono.

La *PCB* implementada para el control de robot humanoide cumple con los requerimientos de funcionamiento propuestos. Esta tarjeta es capaz de recibir los datos procesados por la tarjeta *Raspberry Pi 5*, generando las tramas de datos necesarias para el control de los motores *Dynamixel* del robot. Por otro lado, la arquitectura de este muestra una buena capacidad de integración con el sistema de visión, manteniendo comunicación entre los componentes.

7 Conclusión

Este trabajo presenta la implementación de un sistema de visión orientado a la detección de un objeto específico, una esfera de color azul, cuya identificación permite el control del movimiento de un robot humanoide. A partir de los resultados obtenidos durante el desarrollo del sistema de visión para un robot humanoide, basado en *Raspberry Pi 5*, se puede concluir que la solución propuesta representa una alternativa ventajosa frente al modulo de visión por computadora *HaViMo* predeterminado por la empresa *Robotis*, tanto por su flexibilidad como por su potencial de personalización.

A continuación, se detallan algunas de las ventajas previamente mencionadas.

- **Capacidad de procesamiento**: La tarjeta *Raspberry Pi 5* tiene un *CPU ARM Cortex-A76* de hasta 2.4 GHz, con 8 *GB de RAM*, con posibilidad de usar procesamiento paralelo, aceleradores de *IA* como *HATs* o módulos externos. A diferencia del modulo *HaViMo* que esta diseñado con un procesador limitado a sistemas de visión simples.

  Por otra parte, la tarjeta *Raspberry Pi 5* permite un almacenamiento de altos volúmenes de imágenes que suelen utilizarse para el entrenamiento de redes neuronales. Así mismo, cuenta con una amplia compatibilidad con cámaras de alta resolución. De este modo se obtiene una mejora en la velocidad de procesamiento.

- **Flexibilidad del sistema para expandirse**: Tanto la tarjeta *Raspberry Pi 5*, como el microcontrolador *Raspberry Pi Pico* empleados en el sistema de visión y control de movimiento del robot, son dispositivos de arquitectura abierta, lo que permite a los fabricantes la creación de software y hardware compatible con ambas tarjetas. Esta característica facilita la integración de distintos componentes electrónicos, además de brindar una herramienta flexible que puede adaptarse a las necesidades específicas de cada proyecto. Así mismo, al ser sistemas ampliamente utilizados existe una amplia documentación, por lo que es posible implementar mejoras constantes y actualizaciones periódicas que añadan nuevas funcionalidades.

- **Accesibilidad de programación**: La tarjeta *Raspberry Pi 5*, al igual que el microcontrolador *Raspberry Pi Pico*, ofrece una amplia accesibilidad en cuanto a lenguajes de programación. En particular, la *Raspberry Pi 5* permite la instalación de múltiples intérpretes y entornos de desarrollo, lo que brinda la posibilidad de trabajar con diversos lenguajes como *Python, C/C++, Java*, entre otros. Esta versatilidad facilita al usuario la elección de la interfaz y el lenguaje que le resulte más sencillo o adecuado a su proyecto, siendo la única limitante el nivel de conocimiento en programación del desarrollador.

- **Actualización de software**: La tarjeta *Raspberry Pi 5*, cuenta con compatibilidad con diferentes sistemas operativos siendo el oficial el *Raspberry Pi OS*, esto permite realizar actualizaciones constantes tanto del sistema como de las bibliotecas instaladas. Esta capacidad de actualización facilita la incorporación de nuevas funcionalidades, correcciones de seguridad y mejoras en el rendimiento. Cabe resaltar que el sistema admite la instalación de bibliotecas de última generación mediante gestores de paquetes como `apt`, `pip` o `conda`, lo que permite integrar herramientas modernas para visión por computadora, inteligencia artificial, manejo

de *hardware*, entre otros. A diferencia del modulo *HaViMo* que al ser de arquitectura cerrada, no permite actualizaciones ni mejoras.

7.1 Trabajo futuro

A pesar de los resultados obtenidos existen varias mejoras que deberán abordarse en trabajo futuro.

- Implementación de algoritmos avanzados de visión, como el aprendizaje profundo, segmentación, detección de objetos, reconocimiento facial o redes neuronales convolucionales.
- Integración de inteligencia artificial para la toma de decisiones del robot humanoide.
- Integración de sensores visuales con capacidades de adaptación dinámica a condiciones de iluminación variables, así como la incorporación de cámaras con percepción estereoscópica o sensores de profundidad para la obtención de mapas tridimensionales del entorno.
- Diseño de una interfaz de usuario que permita la modificación dinámica de los parámetros de los filtros de procesamiento de imagen, sino también la selección interactiva del objeto a identificar. Así mismo, incorporar mecanismos que faciliten al usuario la configuración de las acciones a ejecutar por el robot en función del reconocimiento realizado.

Referencias

[1] G. X. Ortiz Morales, J. N. Garrido Vázquez, A. Hernández Cadena, J. A. Magaña, J. M. Gómez Zea y D. M. de la O, "Acercamiento a la robótica: Robot humanoide," *Innovación y desarrollo tecnológico revista digital*, vol. 12, n.º 4, 2020.

[2] Federation of International Robot-soccer Association, *FIRA World Cup*, Accedido el 9 de septiembre de 2024, 2024. dirección: https://firaworldcup.org/about/.

[3] K. Valladares-Yanez, A. Monroy-Meza, R. Suarez-Rivera, J. Rodriguez, G. Pérez-Soto y K. Camarillo-Gómez, "Development and implementation of a vision system for decision making in the movements control of humanoid robots," sep. de 2018, págs. 1-6. DOI: 10.1109/COMROB.2018.8689417.

[4] C. Gordon, H. Lema, D. Leon y P. Encalada, "Human rescue based on autonomous robot KUKA youbot with deep learning approach," Cited by: 4, 2019, págs. 318-323. DOI: 10.1109/ICEDEG.2019.8734311. dirección: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85068365767&doi=10.1109%2fICEDEG.2019.8734311&partnerID=40&md5=416a12036f2c0c0f7227911d6b9e7086.

[5] L. Ma, J. Bartholomew, Y. Wang y X. Li, "Development of a Raspberry PI-Controlled VEX Robot for a Robotics Technology Course," Cited by: 0, 2023. dirección: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85172149912&partnerID=40&md5=50a67db6fd531301214facdd14752938.

[6] A. Vas y G. A. Dsouza, "Object Tracing with Colour range filtering in HSV Colour space and Object Following with Ros2," Cited by: 0, 2023. DOI: 10.1109/GCITC60406.2023.10426509. dirección: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85191723286&doi=10.1109%2fGCITC60406.2023.10426509&partnerID=40&md5=a7d5d4e2515a23400f218056875df293.

[7] J. Wen, X. Feng, Q. Zhang, H. Yuan y X. Peng, "Humanoid Robot control system based on human pose detection," Cited by: 0, vol. 12940, 2023. DOI: 10.1117/12.3010610. dirección: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85179548552&doi=10.1117%2f12.3010610&partnerID=40&md5=fae81a8fa375d4bdb7948bd66b6f7e6d.

[8] J. A. T. Rodrigues, S. D. Anjos, M. C. Silva, R. C. M. Câmara de Santos y R. A. R. Oliveira, "Using Soft Computing and Computer Vision to Create and Control an Integrated Autonomous Robotic Manipulator Process," Cited by: 0; All Open Access, Hybrid Gold Open Access, vol. 1, 2024, págs. 820-827. DOI: 10.5220/0012705600003690. dirección: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85193992014&doi=10.5220%2f0012705600003690&partnerID=40&md5=6ed849739d6d3024b8088c9708d95cd9.

[9] M. F. S. Titu, S. M. R. Haque, R. Islam, A. Hossain, M. A. Qayum y R. Khan, "Experiments with cooperative robots that can detect object's shape, color and size to perform tasks in industrial workplaces," *International Journal of Intelligent Robotics and Applications*, vol. 8, n.º 1, págs. 179-192, 2024, Cited by: 0. DOI: 10.1007/s41315-023-00305-y. dirección: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85178290323&doi=10.1007%2fs41315-023-00305-y&partnerID=40&md5=7fa2c14c31257832042d648dbc128b80.

[10] M. D. C. E. Gudiño, "Procesamiento morfológico, espacios color y quaternions, con biometría de la imagen y otras aplicaciones," Available: http://ri-ng.uaq.mx/handle/123456789/871, Ph.D. thesis, Facultad de Ingeniería, Universidad Autónoma de Querétaro, Querétaro, México, nov. de 2008.

[11] A. F. M. Vargas, "Sistema de seguridad basado en Raspberry Pi con cámara para día y noche con enlace IoT y procesamiento de imágenes," Available: http://ri-ng.uaq.mx/handle/123456789/3102, Bachelor's thesis, Facultad de Ingeniería, Universidad Autónoma de Querétaro, Querétaro, México, 2021.

[12] T. E. J. Vallejo, "Detección automática de líneas de carril basada en Inteligencia Artificial orientada a la asistencia del conductor," Available: http://ri-ng.uaq.mx/handle/123456789/4384, Master's thesis, Facultad de Ingeniería, Universidad Autónoma de Querétaro, Querétaro, México, ene. de 2023.

[13] A. L. Álvarez, "Desarrollo e implementación de un sistema de visión usando RNC para el reconocimiento de expresiones faciales," Universidad Autónoma de Querétaro, 2023.

[14] A. I. De la Fuente Sánchez, "Desarrollo de un controlador de movimiento para un robot humanoide," Universidad Autónoma de Querétaro, 2023. dirección: https://ri-ng.uaq.mx/handle/123456789/8239.

[15] G. Gismero, *Estado del Arte: Proyecto de Grado Visión de Robots*, https://www.fing.edu.uy/inco/grupos/mina/pGrado/vision2010/docs/pgvisrob_estado_arte.pdf, Tutores: Facundo Benavides, Gonzalo Tejera, Serrana Casella. InCo - FIng - UDELAR, Montevideo, Uruguay. Retrieved July 29, 2024, from http://www.fing.edu.uy/ pgvisrob - gegismero@gmail.com, sep. de 2012.

[16] Rohde & Schwarz. "Entendiendo el UART." Accedido: 2024-10-23. (n.d.), dirección: https://www.rohde-schwarz.com/lat/productos/prueba-y-medicion/essentials-test-equipment/digital-oscilloscopes/entendiendo-el-uart_254524.html.

[17] Y. ROBOTIS, *ROBOTIS e-Manual*, ROBOTIS e-Manual, s. f. dirección: https://emanual.robotis.com/docs/en/dxl/protocol1/.

[18] Y. ROBOTIS, *ROBOTIS e-Manual*, ROBOTIS e-Manual, s. f. dirección: https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/.

[19] N. A. Ibraheem, M. M. Hasan, R. Z. Khan y P. K. Mishra, "Understanding color models: a review," *ARPN Journal of science and technology*, vol. 2, n.º 3, págs. 265-275, 2012.

[20] A. Kaehler y G. Bradski, *Learning OpenCV 3*, First. Sebastopol, CA: O'Reilly Media, Inc., dic. de 2017, ISBN: 9781491937990.

[21] C. M. en P. Inteligencia e I. I. Volumen, "Una Perspectiva de la Inteligencia Artificial en su 50 Aniversario," en *Actas del Congreso de Inteligencia Artificial*, Uclm.es, 2006. dirección: https://www.dsi.uclm.es/personal/AntonioFdez/download/papers/conference/cmpi2006-volumeII.pdf#page=164.

[22] W. J. Torres, "Procesamiento de imágenes a color utilizando morfología matemática," *Iiisci.org*, Year. dirección: https://www.iiisci.org/journal/pdv/risci/pdfs/C382LR.pdf.

[23] Y. ROBOTIS, *ROBOTIS e-Manual*, ROBOTIS e-Manual, s. f. dirección: https://emanual.robotis.com/docs/en/dxl/mx/mx-28/.

[24] Logitech, *Cámara web HD PRO Logitech C920*, https://www.logitech.com/es-es/products/webcams/c920-pro-hd-webcam.960-001055.html, [Consultado: 06-mar-2025], 2025.

[25] Raspberrypi.com, *Raspberry Pi Pico Datasheet*, Accessed: 01-Sep-2024, 2024. dirección: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf?_gl=1*1mrc87g*_ga*MTQ5MzA4MzEyLjE3MjUyMzM5NzU.*_ga_22FD70LWDS*MTcyNTIzMzk3Ni4xLjEuMTcyNTIzNDA1NC4wLjAuMA..

[26] Raspberry Pi pico, *Raspberry Pi*, Recuperado de https://raspberrypi.cl/producto/raspberry-pi-pico/, 2024. dirección: https://raspberrypi.cl/producto/raspberry-pi-pico/ (visitado 02-08-2024).

[27] Raspberrypi.com, *Raspberry Pi 5 Product Brief*, Accessed: 01-Sep-2024, 2024. dirección: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf.

Anexos

7.2 Anexo A: Código de Python de Raspberry Pi 5

```python
# Librerias a usar
import cv2, math, serial, time, struct # OpenCV
import numpy as np              # Numpy abreviado

rep = 10
flag_enviar = 0

# Declaracion de puerto serial
ser = serial.Serial('/dev/ttyAMA0', 9600)

# Nombres de las ventanas
win1 = 'Robot'      # Ventana de imagen
win2 = 'Controles'  # Ventana de controles

# Función de trackbars
def nothing(x):
    pass

# Función de detección de circulos por contorno y area
def es_circulo(contorno):

    # Calculo del perímetro del contorno
    perimetro = cv2.arcLength(contorno, True)

    # Aproximación del contorno
    aproximacion = cv2.approxPolyDP(contorno, 0.04 * perimetro, True)

    # Condición de cantidad de lados
    if len(aproximacion) > 4:

        # Calculo de área y el radio del círculo aproximado
        area = cv2.contourArea(contorno)

        # Calculo de coordendas y radio del objeto
        (cx, cy), radio = cv2.minEnclosingCircle(contorno)

        # Calculo de area a partir del radio
        area_circulo = np.pi * (radio ** 2)

        # Destruir variable innecesaria
        del cx,cy

        # Comprobar si el área del contorno es similar al área del círculo
        if min_tol * area_circulo/100 <= area <= max_tol * area_circulo/100:
            return True

    return False

# Ventanas
cv2.namedWindow(win1)
cv2.namedWindow(win2,cv2.WINDOW_NORMAL)

# Trackbar para cambiar de opciones en la ventana de controles
cv2.createTrackbar('Opcion', win2, 5, 5, nothing)

# Trackbar para ajustar el tamaño del kernel del filtro de suavizado
cv2.createTrackbar('Kernel', win2, 10, 20, nothing)

# Trackbars para ajustar el rango de color azul en HSV
cv2.createTrackbar('Lower H', win2, 100, 179, nothing)
cv2.createTrackbar('Lower S', win2, 150, 255, nothing)
cv2.createTrackbar('Upper H', win2, 150, 179, nothing)
cv2.createTrackbar('Upper S', win2, 255, 255, nothing)

# Trackbar para ajustar el tamaño del kernel del elemento estructurante
```

```
cv2.createTrackbar('Size Morph', win2, 7, 20, nothing)

# Trackbar para ajustar la tolerancia de detección del circulo
cv2.createTrackbar('Min Tol', win2, 6, 9, nothing)
cv2.createTrackbar('Max Tol', win2, 0, 9, nothing)

# Variables para los putText
font = cv2.FONT_HERSHEY_SIMPLEX     # Fuente del texto
font_scale = 0.5                    # Escala del texto
color_blanco = (255, 255, 255)      # Color del texto en BGR
color_verde = (0, 255, 0)           # Color del texto en BGR
color_rojo = (0, 0, 255)           # Color del texto en BGR
color_gris = (60, 60, 60)           # Color del texto en BGR
thickness = 1                       # Grosor del texto
pos1 = (10, 30)                     # Posición del texto (x, y)
pos2 = (10, 50)                     # Posición del texto (x, y)
pos3 = (10, 70)                     # Posición del texto (x, y)
pos4 = (10, 90)                     # Posición del texto (x, y)
posf = (460, 460)                   # Posición del texto (x, y)

# Textos para las imagenes de los frames
t_frame_original   = f"Imagen: Original"
t_frame_suavizado  = f"Imagen: Suavizada"
t_frame_morfologia = f"Imagen: Filtro Color, Op Morfologicas"
t_frame_circulos   = f"Imagen: Filtro de circulos"
t_frame_circulomax = f"Imagen: Filtro Circulo Mayor"
t_frame_coordenada = f"Imagen: Coordenada objeto"
t_frame_cuadrante  = f"Imagen: Cuadrante objeto"

# Texto para recordar que hace cada frame
# frame_original        Frame original
# frame_suavizado       Frame suavizado
# frame_hsv             Frame cambiado a HSV
# frame_mascara         Frame de mascara binaria Inrange
# frame_erosionado      Frame de erosion
# frame_dilatado        Frame de dilatacion
# frame_color_morf      Frame de color y operacion morfologica
# frame_circulos_3ch    Frame de circulos filtrados 3 canales
# frame_circulos_1ch    Frame de circulos filtrados 1 canal
# frame_coordenada      Frame de coordenadas XY del circulo grande
# frame_cuadrante       Frame de cuadrante del objeto

# Objeto de captura de video desde la cámara
cap = cv2.VideoCapture(0)

# Condición de apertura de cámara
if not cap.isOpened():
    print("Error: No se puede abrir la cámara")
    exit()

# Captura de un frame
ret, frame_original = cap.read()

# Obtención de dimensiones del frame
height, width, _ = frame_original.shape
print(f"Altura del frame: {height}, Ancho del frame: {width}")

fondo = np.zeros((height//20, width, 3), dtype=np.uint8)

# Contador para enviar datos
z = 0

# Vector para guardar bytes a enviar
data = [0,0]

# Ciclo infinito
while True:

    # Captura de frame
    ret, frame_original = cap.read()
```

```
136
137     # Condición de captura frame
138     if not ret:
139         print("Error: No se puede recibir frame (stream end?). Saliendo ...")
140         break
141
142     # Captura el valor de la opción
143     opc = cv2.getTrackbarPos('Opcion', win2)
144
145     # Captura del valor del trackbar del kernel del suavizado
146     kernel_size = cv2.getTrackbarPos('Kernel', win2)
147     if kernel_size % 2 == 0:
148         kernel_size += 1
149     if kernel_size < 1:
150         kernel_size = 1
151
152     # Aplicación de suavizado al frame original
153     frame_suavizado = cv2.GaussianBlur(frame_original, (kernel_size, kernel_size), 0)
154
155     # Cambio de espacio de color al frame suavizado
156     frame_hsv = cv2.cvtColor(frame_suavizado, cv2.COLOR_BGR2HSV)
157
158     # Obtención de valores de trackbars del HSV
159     lower_h = cv2.getTrackbarPos('Lower H', win2)
160     lower_s = cv2.getTrackbarPos('Lower S', win2)
161     lower_v = 0
162     upper_h = cv2.getTrackbarPos('Upper H', win2)
163     upper_s = cv2.getTrackbarPos('Upper S', win2)
164     upper_v = 255
165
166     # Rango de color azul en HSV basado en los valores de los trackbars
167     lower_blue = np.array([lower_h, lower_s, lower_v])
168     upper_blue = np.array([upper_h, upper_s, upper_v])
169
170     # Obtención de mascara binaria del filtrado de color
171     frame_mascara = cv2.inRange(frame_hsv, lower_blue, upper_blue) # Mascara color
172
173     # Captura del valor del trackbar del tamano del elemento estructurante
174     size_element = cv2.getTrackbarPos('Size Morph', win2)
175
176     # Creación del elemento estructurante
177     element = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2 * size_element + 1, 2 *
        size_element + 1))
178
179     # Erosionado de la máscara binaria
180     frame_erosionado = cv2.erode(frame_mascara, element, iterations=1)
181
182     # Dilatado del erosionado
183     frame_dilatado = cv2.dilate(frame_erosionado, element, iterations=1)
184
185     # Enmascaramiento AND para color filtrado
186     frame_color_morf  = cv2.bitwise_and(frame_original, frame_original, mask = frame_dilatado)
187
188     # Detección de contornos en la máscara dilatada
189     contornos_dilatado, _ = cv2.findContours(frame_dilatado, cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
190
191     # Conteo de objetos blancos
192     num_objectos_dilatado = len(contornos_dilatado)
193
194     # Creación de imagen vacia
195     frame_circulos_3ch = np.zeros_like(frame_original)
196
197     # Contador de círculos
198     contador_circulos = 0
199
200     # Lectura de trackbar
201     min_tol = (cv2.getTrackbarPos('Min Tol', win2) + 1)*10
202     max_tol = (cv2.getTrackbarPos('Max Tol', win2) + 10)*10
203
```

```
    # Detección de circulos de los objetos detectados
    for contorno in contornos_dilatado:

        # Llama función que detecta el circulo en rango tolerancia
        if es_circulo(contorno):

            # Dibuja en frame los circulos filtrados
            cv2.drawContours(frame_circulos_3ch, [contorno], -1, (255,255,255),
    thickness=cv2.FILLED)
            contador_circulos += 1

    # Converción de la imagen de circulos filtrados de 3ch a 1ch
    frame_circulos_1ch = cv2.cvtColor(frame_circulos_3ch, cv2.COLOR_BGR2GRAY)

    # Deteccion de contornos en el frame de circulos filtrados
    contornos_circulos, _ = cv2.findContours(frame_circulos_1ch, cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)

    # Conteo de circulos
    num_circulos = len(contornos_circulos)

    # Inicializar variables para el objeto más grande
    max_area = 0
    max_contour = None

    # Copia a nuevo frame
    frame_coordenada = frame_original.copy()
    frame_cuadrante  = frame_original.copy()

    # Reiniciar variables a enviar
    cuad = 0
    dist = 255

    # Encontrar el contorno más grande
    for contour in contornos_circulos:
        area = cv2.contourArea(contour)
        if area > max_area:
            max_area = area
            max_contour = contour

    # si detectó la figura más grande
    if max_contour is not None:

        # Detección de circulo mayor y coordendas radio
        (cx, cy), radio = cv2.minEnclosingCircle(max_contour)

        # Calculo de nuevas coordenadas con oofset
        X = int(cx) - width/2
        Y = int(cy) - height/2

        # Calculo de distancia a partir de radio pixeles
        dist = 3029.6*radio**(-1.093)
        dist = round(dist, 2)

        if dist >= 255:
            dist = 255

        # Cuadrante del centroide
        if(cx > 0 and cx <= width*1//5 and cy > 0 and cy <= height*1//5):
            cv2.rectangle(frame_cuadrante, (0,0), (width*1//5,height*1//5), (0,255,0),5)
            cuad = 1
        elif(cx > width*1//5 and cx <= width*2//5 and cy > 0 and cy <= height*1//5):
            cv2.rectangle(frame_cuadrante, (width*1//5,0), (width*2//5,height*1//5), (0,255,0),5)
            cuad = 2
        elif(cx > width*2//5 and cx <= width*3//5 and cy > 0 and cy <= height*1//5):
            cv2.rectangle(frame_cuadrante, (width*2//5,0), (width*3//5,height*1//5), (0,255,0),5)
            cuad = 3
        elif(cx > width*3//5 and cx <= width*4//5 and cy > 0 and cy <= height*1//5):
            cv2.rectangle(frame_cuadrante, (width*3//5,0), (width*4//5,height*1//5), (0,255,0),5)
            cuad = 4
```

```
272         elif(cx > width*4//5 and cx <= width*5//5 and cy > 0 and cy <= height*1//5):
273             cv2.rectangle(frame_cuadrante, (width*4//5,0), (width*5//5,height*1//5), (0,255,0),5)
274             cuad = 5
275         elif(cx > 0 and cx <= width*1//5 and cy > height*1//5 and cy <= height*2//5):
276             cv2.rectangle(frame_cuadrante, (0,height*1//5), (width*1//5,height*2//5), (0,255,0),5)
277             cuad = 6
278         elif(cx > width*1//5 and cx <= width*2//5 and cy > height*1//5 and cy <= height*2//5):
279             cv2.rectangle(frame_cuadrante, (width*1//5,height*1//5), (width*2//5,height*2//5),
    (0,255,0),5)
280             cuad = 7
281         elif(cx > width*2//5 and cx <= width*3//5 and cy > height*1//5 and cy <= height*2//5):
282             cv2.rectangle(frame_cuadrante, (width*2//5,height*1//5), (width*3//5,height*2//5),
    (0,255,0),5)
283             cuad = 8
284         elif(cx > width*3//5 and cx <= width*4//5 and cy > height*1//5 and cy <= height*2//5):
285             cv2.rectangle(frame_cuadrante, (width*3//5,height*1//5), (width*4//5,height*2//5),
    (0,255,0),5)
286             cuad = 9
287         elif(cx > width*4//5 and cx <= width*5//5 and cy > height*1//5 and cy <= height*2//5):
288             cv2.rectangle(frame_cuadrante, (width*4//5,height*1//5), (width*5//5,height*2//5),
    (0,255,0),5)
289             cuad = 10
290         elif(cx > 0 and cx <= width*1//5 and cy > height*2//5 and cy <= height*3//5):
291             cv2.rectangle(frame_cuadrante, (0,height*2//5), (width*1//5,height*3//5), (0,255,0),5)
292             cuad = 11
293         elif(cx > width*1//5 and cx <= width*2//5 and cy > height*2//5 and cy <= height*3//5):
294             cv2.rectangle(frame_cuadrante, (width*1//5,height*2//5), (width*2//5,height*3//5),
    (0,255,0),5)
295             cuad = 12
296         elif(cx > width*2//5 and cx <= width*3//5 and cy > height*2//5 and cy <= height*3//5):
297             cv2.rectangle(frame_cuadrante, (width*2//5,height*2//5), (width*3//5,height*3//5),
    (0,255,0),5)
298             cuad = 13
299         elif(cx > width*3//5 and cx <= width*4//5 and cy > height*2//5 and cy <= height*3//5):
300             cv2.rectangle(frame_cuadrante, (width*3//5,height*2//5), (width*4//5,height*3//5),
    (0,255,0),5)
301             cuad = 14
302         elif(cx > width*4//5 and cx <= width*5//5 and cy > height*2//5 and cy <= height*3//5):
303             cv2.rectangle(frame_cuadrante, (width*4//5,height*2//5), (width*5//5,height*3//5),
    (0,255,0),5)
304             cuad = 15
305         elif(cx > 0 and cx <= width*1//5 and cy > height*3//5 and cy <= height*4//5):
306             cv2.rectangle(frame_cuadrante, (0,height*3//5), (width*1//5,height*4//5), (0,255,0),5)
307             cuad = 16
308         elif(cx > width*1//5 and cx <= width*2//5 and cy > height*3//5 and cy <= height*4//5):
309             cv2.rectangle(frame_cuadrante, (width*1//5,height*3//5), (width*2//5,height*4//5),
    (0,255,0),5)
310             cuad = 17
311         elif(cx > width*2//5 and cx <= width*3//5 and cy > height*3//5 and cy <= height*4//5):
312             cv2.rectangle(frame_cuadrante, (width*2//5,height*3//5), (width*3//5,height*4//5),
    (0,255,0),5)
313             cuad = 18
314         elif(cx > width*3//5 and cx <= width*4//5 and cy > height*3//5 and cy <= height*4//5):
315             cv2.rectangle(frame_cuadrante, (width*3//5,height*3//5), (width*4//5,height*4//5),
    (0,255,0),5)
316             cuad = 19
317         elif(cx > width*4//5 and cx <= width*5//5 and cy > height*3//5 and cy <= height*4//5):
318             cv2.rectangle(frame_cuadrante, (width*4//5,height*3//5), (width*5//5,height*4//5),
    (0,255,0),5)
319             cuad = 20
320         elif(cx > 0 and cx <= width*1//5 and cy > height*4//5 and cy <= height*5//5):
321             cv2.rectangle(frame_cuadrante, (0,height*4//5), (width*1//5,height*5//5), (0,255,0),5)
322             cuad = 21
323         elif(cx > width*1//5 and cx <= width*2//5 and cy > height*4//5 and cy <= height*5//5):
324             cv2.rectangle(frame_cuadrante, (width*1//5,height*4//5), (width*2//5,height*5//5),
    (0,255,0),5)
325             cuad = 22
326         elif(cx > width*2//5 and cx <= width*3//5 and cy > height*4//5 and cy <= height*5//5):
327             cv2.rectangle(frame_cuadrante, (width*2//5,height*4//5), (width*3//5,height*5//5),
    (0,255,0),5)
```

```
328             cuad = 23
329         elif(cx > width*3//5 and cx <= width*4//5 and cy > height*4//5 and cy <= height*5//5):
330             cv2.rectangle(frame_cuadrante, (width*3//5,height*4//5), (width*4//5,height*5//5),
    (0,255,0),5)
331             cuad = 24
332         elif(cx > width*4//5 and cx <= width*5//5 and cy > height*4//5 and cy <= height*5//5):
333             cv2.rectangle(frame_cuadrante, (width*4//5,height*4//5), (width*5//5,height*5//5),
    (0,255,0),5)
334             cuad = 25
335 
336         # Texts en frames
337         t_coord_objeto = "Objeto encontrado"            # Texto de que si se detectó objeto
338         t_coord_centro = f"({X} , {Y}) "                # Texto de que coordendas centro objeto
339         t_cuad_cuad    = f"Cuadrante: {cuad}"           # Texto de cuadrante centro
340         t_cuad_radio   = f"Radio: {int(radio)} pixeles" # Texto de radio en pixeles
341         t_cuad_dist    = f"Distancia: {dist} cm"        # Texto de distancia esfera
342 
343         # Impresión de circulo calculado
344         cv2.circle(frame_circulos_3ch, (int(cx), int(cy)), int(radio), (255, 0, 255), 2) #
    Circulo mayor de radio detectado contorno en frame circulos 3ch
345         cv2.circle(frame_coordenada, (int(cx), int(cy)), int(radio), (255, 0, 255), 2)   #
    Circulo mayor de radio detectado contorno en frame coordenada
346         cv2.circle(frame_coordenada, (int(cx), int(cy)), 5, (255, 0, 255), -1)           #
    Circulo punto centro en frame coordenada
347         cv2.circle(frame_cuadrante, (int(cx), int(cy)), int(radio), (255, 0, 255), 2)    #
    Circulo mayor de radio detectado contorno en frame cuadrante
348         cv2.circle(frame_cuadrante, (int(cx), int(cy)), 5, (255, 0, 255), -1)            #
    Circulo punto centro en frame coordenada
349 
350         # Texto de impresión en coordendas centro en frame coordendas y cuadrante
351         cv2.putText(frame_coordenada, t_coord_centro, (int(cx), int(cy)), font, font_scale, (255,
    255, 255), 2, cv2.LINE_AA)
352         cv2.putText(frame_cuadrante, t_coord_centro, (int(cx), int(cy)), font, font_scale, (255,
    255, 255), 2, cv2.LINE_AA)
353         cv2.putText(frame_cuadrante, t_cuad_cuad, pos2, font, font_scale, color_verde, thickness,
    cv2.LINE_AA)
354         cv2.putText(frame_cuadrante, t_cuad_radio, pos3, font, font_scale, color_verde,
    thickness, cv2.LINE_AA)
355         cv2.putText(frame_cuadrante, t_cuad_dist, pos4, font, font_scale, color_verde, thickness,
    cv2.LINE_AA)
356 
357     # Si no detecto objeto
358     else:
359         # Texts en frames
360         t_coord_objeto = "Objeto NO encontrado" # Texto de que NO se detectó objeto
361 
362     for i in range(1, 5):  # range(1, 5) incluye 1 pero excluye 5
363         cv2.line(frame_cuadrante, (width*i//5, 0) , (width*i//5, height), color_gris, 1)
364         cv2.line(frame_cuadrante, (0, height*i//5), (width, height*i//5), color_gris, 1)
365 
366 
367     # Selección de opción
368     if opc == 0:
369         # Texto de frame original
370         cv2.putText(frame_original, t_frame_original, pos1, font, font_scale, color_verde,
    thickness, cv2.LINE_AA)
371 
372         # Frame
373         frame_display = frame_original
374 
375     elif opc == 1:
376         # Texto de frame suavizado
377         cv2.putText(frame_suavizado, t_frame_suavizado, pos1, font, font_scale, color_verde,
    thickness, cv2.LINE_AA)
378 
379         # Frame
380         frame_display = frame_suavizado
381 
382     elif opc == 2:
383         # Texto de cantidad de objetos en filtro color y morfologia
```

```
        t_morf_no_obj = f"No. de objetos: {num_objectos_dilatado}"

        # Textos en frame morfologia
        cv2.putText(frame_color_morf  , t_frame_morfologia, pos1, font, font_scale, color_verde, thickness, cv2.LINE_AA)
        cv2.putText(frame_color_morf  , t_morf_no_obj, pos2, font, font_scale, color_blanco, thickness, cv2.LINE_AA)

        #Frame
        frame_display = frame_color_morf

    elif opc == 3:
        # Textos de tolerancia y numero circulos
        t_cir3ch_min_tol = f"Minimo: {min_tol} %"
        t_cir3ch_max_tol = f"Maximo: {max_tol} %"
        t_cir3ch_no_cir = f"Circulos detectados: {contador_circulos}"

        # Textos en frame circulos 3ch
        cv2.putText(frame_circulos_3ch, t_frame_circulos, pos1, font, font_scale, color_verde, thickness, cv2.LINE_AA)
        cv2.putText(frame_circulos_3ch, t_cir3ch_no_cir, pos2, font, font_scale, color_blanco, thickness, cv2.LINE_AA)
        cv2.putText(frame_circulos_3ch, t_cir3ch_min_tol, pos3, font, font_scale, color_blanco, thickness, cv2.LINE_AA)
        cv2.putText(frame_circulos_3ch, t_cir3ch_max_tol, pos4, font, font_scale, color_blanco, thickness, cv2.LINE_AA)

        # Frame
        frame_display = frame_circulos_3ch

    elif opc == 4:
        # Textos en frame coordenada
        cv2.putText(frame_coordenada, t_frame_coordenada, pos1, font, font_scale, color_verde, thickness, cv2.LINE_AA)
        cv2.putText(frame_coordenada, t_coord_objeto, pos2, font, font_scale, color_blanco, thickness, cv2.LINE_AA)

        # Frame
        frame_display = frame_coordenada

    elif opc == 5:
        # Text frame cuadrante
        cv2.putText(frame_cuadrante, t_frame_cuadrante, pos1, font, font_scale, color_verde, thickness, cv2.LINE_AA)


        # Frame
        frame_display = frame_cuadrante

    if cv2.waitKey(1) & 0xFF == ord('b'):
        flag_enviar = 1

    # Envio de datos
    z = z + 1
    if data[0] == 0:
        rep = 7
    else:
        rep = 13

    if z >= rep and flag_enviar == 1:
        data[0] = cuad
        data[1] = int(dist)
        ser.write(bytearray(data))
        print("Cuadrante: {}, Distancia: {}".format(data[0],data[1]))
        z = 0

    # Texto para salir
    t_cerrar = f"Presiona q para salir"
    cv2.putText(frame_display, t_cerrar, posf, font, font_scale, color_rojo, thickness, cv2.LINE_AA)
```

```
444
445
446     # Visualización de frame
447     cv2.imshow(win1, frame_display)
448     cv2.imshow(win2, fondo)
449
450     # Salir del loop cuando se presione la tecla 'q'
451     if cv2.waitKey(1) & 0xFF == ord('q'):
452         break
453
454 # Liberar el objeto de captura y cerrar todas las ventanas
455 cap.release()
456 cv2.destroyAllWindows()
```

Código 8: Código de Python de *Raspberry Pi 5*

7.3 Anexo B: Código de Python de Raspberry Pi Pico

```
1  from machine import Pin, UART
2  import utime, time
3
4  uart = UART(1, baudrate=9600, tx=None, rx=Pin(5))
5  dyna = UART(0, baudrate=1000000, tx=Pin(16), rx=Pin(17))
6  en1 = Pin(15, Pin.OUT)
7  led = Pin(3,Pin.OUT)
8
9  RangoM21 = [480,570,660]
10 RangoM22 = [422,512,602]
11 MaxM21 = RangoM21[2]
12 MaxM22 = RangoM22[2]
13 CenM21 = RangoM21[1]
14 CenM22 = RangoM22[1]
15 MinM21 = RangoM21[0]
16 MinM22 = RangoM22[0]
17 inc = 5
18 tol = 20
19 ref = 13
20 registro = 0b00000000
21 ve = 25
22 final =  15
23 tiempo = 0.18
24 tiempo2 = 3
25 opcion = 's'
26 vec_ID = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]2
27 vec_vel = [ve, ve, ve, ve, ve, ve, ve, ve, ve, ve, ve, ve, ve, ve, ve, ve, ve, ve, ve, ve, ve, ve]
28 pose= [[2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1300, 3000, 2500, 0600, 2600,
       1700, 2050, 2080, 0490, 0530, 0680, 0512],#0
29         [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1280, 2980, 2470, 0600, 2590,
       1690, 2050, 2080, 0490, 0530, 0680, 0512],#1
30         [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1260, 2960, 2500, 0600, 2550,
       1680, 2050, 2100, 0490, 0530, 0680, 0512],#2
31         [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1240, 2940, 2500, 0570, 2570,
       1670, 2100, 2100, 0490, 0530, 0680, 0512],#3
32         [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1220, 2920, 2480, 0570, 2550,
       1660, 2100, 2100, 0490, 0530, 0680, 0512],#4
33         [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1220, 2920, 2510, 0570, 2550,
       1660, 2100, 2100, 0490, 0530, 0680, 0512],#5
34         [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1200, 2900, 2510, 0560, 2550,
       1650, 2050, 2100, 0490, 0530, 0680, 0512],#6
35         [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1180, 2880, 2510, 0550, 2525,
       1600, 2050, 2100, 0490, 0530, 0680, 0512],#7
36         [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1160, 2860, 2525, 0550, 2500,
       1550, 2050, 2080, 0490, 0530, 0680, 0512],#8
37         [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1140, 2840, 2525, 0590, 2500,
       1550, 2050, 2080, 0490, 0530, 0680, 0512],#9
38         [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1120, 2820, 2500, 0610, 2450,
       1530, 2050, 2080, 0490, 0530, 0680, 0512],#10
39         [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1100, 2800, 2490, 0650, 2400,
       1500, 2050, 2080, 0490, 0530, 0680, 0512],#11
```

```
        [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1120, 2820, 2490, 0650, 2400, 1500, 2035, 2070, 0490, 0530, 0680, 0512],#12
        [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1140, 2840, 2490, 0650, 2400, 1520, 2035, 2060, 0490, 0530, 0680, 0512],#13
        [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1160, 2860, 2490, 0650, 2430, 1550, 2035, 2040, 0490, 0530, 0680, 0512],#14
        [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1180, 2880, 2490, 0650, 2460, 1550, 2035, 2040, 0490, 0530, 0680, 0512],#15
        [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1200, 2900, 2490, 0650, 2490, 1550, 2035, 2040, 0490, 0530, 0680, 0512],#16
        [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1220, 2920, 2490, 0615, 2520, 1580, 2035, 2040, 0490, 0530, 0680, 0512],#17
        [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1240, 2940, 2500, 0590, 2530, 1600, 2035, 2050, 0490, 0530, 0680, 0512],#18
        [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1260, 2960, 2500, 0590, 2520, 1640, 2035, 2060, 0490, 0530, 0680, 0512],#19
        [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1280, 2980, 2500, 0580, 2560, 1690, 2050, 2070, 0490, 0530, 0680, 0512],#20
        [2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1290, 2990, 2500, 0590, 2580, 1690, 2050, 2080, 0490, 0530, 0680, 0512]]#21

pos2= [[2150, 1850, 0600, 0450, 0500, 0512, 2000, 2000, 2100, 2000, 1300, 3000, 2500, 0600, 2600, 1700, 2050, 2100, 0490, 0530, 0570, 0512],#0
        [2725, 1200, 0625, 0420, 0400, 0612, 2000, 2000, 2100, 2000, 1300, 3000, 2500, 0600, 2600, 1700, 2050, 2100, 0490, 0530, 0570, 0512],#1
        [3300, 0700, 0650, 0380, 0300, 0712, 2000, 2000, 2100, 2000, 1300, 3000, 2500, 0600, 2600, 1700, 2050, 2100, 0320, 0700, 0570, 0512]]#3

n = len(vec_ID)
l = 4
log = ( (l+1)*n ) + 4
bid = 0xFE
pasos = len(pose)
motores = len(pose[0])
pasos2 = len(pos2)
brazos = 0
led.value(0)

def secuencia():

    for i in range(0,final):
        data = [0xFF, 0xFF, bid         , log     , 0x83      , 0x1E          , l       ]
        print(f"Fila {i}")

        for j in range (0,motores):
            data.append(vec_ID[j])
            MSB = pose[i][j] >> 8
            LSB = pose[i][j] & 0xFF
            data.append(LSB)
            data.append(MSB)
            MSB = vec_vel[i] >> 8
            LSB = vec_vel[i] & 0xFF
            data.append(LSB)
            data.append(MSB)

        checksum = 0
        for k in range(2,len(data)):
            checksum = checksum + data[k]
        checksum = ~checksum
        checksum = checksum & 0x00FF
        data.append(checksum)

        en1.value(1)
        utime.sleep_us(10)
        dyna.write(bytearray(data))
        utime.sleep(0.003)
        en1.value(0)
        utime.sleep_us(10)
        time.sleep(tiempo)

```

```
def posicionar():
    data = [0xFF, 0xFF, bid         , log     , 0x83      , 0x1E         , l       ]

    for i in range(0,n):
        data.append(vec_ID[i])
        MSB = pulsos_q[i] >> 8
        LSB = pulsos_q[i] & 0xFF
        data.append(LSB)
        data.append(MSB)
        MSB = vec_vel[i] >> 8
        LSB = vec_vel[i] & 0xFF
        data.append(LSB)
        data.append(MSB)

    checksum = 0
    for j in range(2,len(data)):
        checksum = checksum + data[j]
    checksum = ~checksum
    checksum = checksum & 0x00FF
    data.append(checksum)

    en1.value(1)
    utime.sleep_us(10)
    dyna.write(bytearray(data))
    utime.sleep(0.003)
    en1.value(0)
    utime.sleep_us(10)

def busqueda(flags, PosM21, PosM22):
    flagM21 = 0
    flagM22 = 0
    print("Buscando")

    if flags == 0b00000000:
        if PosM22 >= MaxM22:
            flags = 0b00010000
        else:
            PosM22 = PosM22 + inc

    if flags == 0b00010000:
        if PosM21 >= MaxM21:
            flags = 0b00001000
        else:
           PosM21 = PosM21 + inc

    if flags == 0b00001000:
        if PosM22 <= MinM22:
            flags = 0b00000100
        else:
           PosM22 = PosM22 - inc

    if flags == 0b00000100:
        if PosM21 <= MinM21:
            flags = 0b00000010
        else:
           PosM21 = PosM21 - inc

    if flags == 0b00000010:
        if PosM22 >= MaxM22:
            flags = 0b00000001
        else:
           PosM22 = PosM22 + inc

    if flags == 0b00000001:
           flags = 0b00000000

    return [PosM21, PosM22, flags]

def seguimiento(PosM21,PosM22):
    flagM21 = 0
```

```
167     flagM22 = 0
168     flagCen = 0
169     texto = ""
170 
171     if cuad == 1 or cuad == 6 or cuad == 11  or cuad == 16 or cuad == 21 or cuad == 2 or cuad ==
        7 or cuad == 12 or cuad == 17 or cuad == 22:
172         PosM22 = PosM22 + inc
173         flagM22 = 0
174         if PosM22 >= MaxM22:
175             PosM22 = MaxM22
176 
177     if cuad == 4 or cuad == 9 or cuad == 14  or cuad == 19 or cuad == 24 or cuad == 5 or cuad ==
        10 or cuad == 15 or cuad == 20 or cuad == 25:
178         PosM22 = PosM22 - inc
179         flagM22 = 0
180         if PosM22 <= MinM22:
181             PosM22 = MinM22
182 
183     if cuad == 3 or cuad == 8 or cuad == 13  or cuad == 18 or cuad == 23:
184         if PosM22 >= CenM22 - tol and PosM22 <= CenM22 + tol:
185             flagM22 = 1
186 
187     if cuad == 1 or cuad == 2 or cuad == 3  or cuad == 4 or cuad == 5 or cuad == 6 or cuad == 7
        or cuad == 8 or cuad == 9 or cuad == 10:
188         PosM21 = PosM21 + inc
189         flagM21 = 0
190         if PosM21 >= MaxM21:
191             PosM21 = MaxM21
192 
193     if cuad == 16 or cuad == 17 or cuad == 18  or cuad == 19 or cuad == 20 or cuad == 21 or cuad
        == 22 or cuad == 23 or cuad == 24 or cuad == 25:
194         PosM21 = PosM21 - inc
195         flagM21 = 0
196         if PosM21 <= MinM21:
197             PosM21 = MinM21
198 
199     if cuad == 11 or cuad == 12 or cuad == 13  or cuad == 14 or cuad == 15:
200         if PosM21 >= CenM21 - tol and PosM21 <= CenM21 + tol:
201             flagM21 = 1
202 
203     if flagM21 == 1 and flagM22 == 1:
204         flagCen= 1
205 
206     texto = "Hor: " + str(CenM22-PosM22) + ", Ver: " + str(CenM21-PosM21)
207     print(texto)
208     return [flagCen, PosM21, PosM22]
209 
210 if opcion == 's':
211     final = final+1
212     secuencia()
213 
214 elif opcion == '2':
215     pulsos_q = pos2[1]
216     posicionar()
217     time.sleep(tiempo2)
218     pulsos_q = pos2[2]
219     posicionar()
220     time.sleep(tiempo2)
221 
222 elif opcion == 'p':
223     pulsos_q = pose[final]
224     posicionar()
225 
226 elif opcion == 'i':
227     pulsos_q = pose[0]
228     print("Pose inicial")
229     posicionar()
230 
231 elif opcion == 'c':
232     final = pasos
```

```
    print("Ciclo")
    while True:
        secuencia()

elif opcion == 'v':
    print("Estableciendo comunicación con Pi 5")
    pulsos_q = pose[0]
    final = pasos
    posicionar()

    while True:

        data_lec = bytes()

        if uart.any():
            led.value(1)
            data_lec += uart.read(2)
            cuad = data_lec[0]
            dist = data_lec[1]

            if cuad == 0:
                brazos = 0
                pulsos_q[-2], pulsos_q[-1], registro = busqueda(registro, RangoM21[1], RangoM22[1])
                RangoM21[1], RangoM22[1] = [pulsos_q[-2],pulsos_q[-1]]
                posicionar()

            else:
                centro, pulsos_q[-2], pulsos_q[-1] = seguimiento(RangoM21[1], RangoM22[1])
                RangoM21[1], RangoM22[1] = [pulsos_q[-2],pulsos_q[-1]]
                posicionar()

                if centro == 1:
                    if dist <= ref:
                        print("cerca")

                        if brazos == 0:
                            pulsos_q = pos2[1]
                            posicionar()
                            time.sleep(tiempo2)
                            pulsos_q = pos2[2]
                            posicionar()
                            time.sleep(tiempo2)
                            brazos = 1
                        else:
                            pulsos_q = pos2[2]
                            posicionar()


                    else:
                        print("lejos")
                        brazos = 0
                        pulsos_q = pose[0]
                        posicionar()
                        time.sleep(3)
                        secuencia()

                else:
                    brazos = 0
                    pulsos_q = pose[0]
                    posicionar()
```

Código 9: Código de Python de *Raspberry Pi 5*