UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE VISIÓN ARTIFICIAL MEDIANTE UNA CÁMARA RGB-D PARA LA INTERPRETACION DE SEÑALIZACION INDUSTRIAL EN UN ROBOT BÍPEDO

POR

Nicolás Arturo Vergara Pasten

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero(a) Civil Electrónico(a)

Profesor Guía
Mario Rubén Medina Carrasco

Abril 2026
Concepción (Chile)

A todos quienes hicieron posible que hoy cumpliera esta meta les estoy profundamente agradecido con mis padres Francisca y Luis quienes estuvieron constantemente dándome su apoyo y amor. Además, debo destacar a cada uno de mis compañeros y amigos por recordarme lo importante que es en medio de todo este proceso disfrutar de instancias y momentos para el aprendizaje.

Agradecimientos

Me gustaría dar a conocer mi más sincero agradecimiento a todas las personas que hicieron posible la elaboración de esta memoria de título.

En primer lugar, reconozco profundamente a mis padres, por su apoyo en cada etapa de mi formación académica y personal, brindándome su confianza, paciencia y amor.

Extiendo un singular agradecimiento a la Universidad de Concepción, al Decano de la Facultad de Ingeniería y a los profesores Mario Medina Carrasco y José Espinoza Castro, por su valiosa colaboración en la financiación de este proyecto.

Finalmente deseo agradecer a mis amigos y compañeros, por su vital importancia en este proceso, brindándome apoyo emocional y ayudándome a poder mantener un equilibrio entre el estudio y las instancias de recreación.

Sumario

Esta memoria de título abarca el desarrollo del diseño e implementación de un sistema de visión artificial mediante una cámara integrada en un robot bípedo, con la finalidad de que pueda identificar e interpretar señaléticas industriales.

El principal propósito fue proporcionar al robot de capacidades visuales básicas que le permitan poder percibir su entorno y por medio de ello lograr responder a estímulos relevantes, que pueden ser, la identificación objetos por medio de su forma o colores específicos.

El sistema fue diseñado empleando una cámara Intel RealSense D435i, la cual fue integrada a una plataforma de procesamiento basada en Raspberry Pi 5, mediante el uso de la biblioteca OpenCV, donde se utilizaron técnicas de procesamiento de imágenes, logrando la capacidad de detectar características visuales como formas y colores que se relacionan con la señalización de seguridad en entornos industriales.

Además, la cámara proporciona información sobre la profundidad, lo que permite al sistema calcular la distancia entre el robot y los objetos detectados, lo que a su vez facilita generar acciones de navegación en relación a la cercanía de la señal detectada, tales como reducir la velocidad, detenerse o cambiar de dirección.

Se realizaron pruebas experimentales enfocadas a evaluar el funcionamiento del sistema ante diferentes escenarios de detección de señalización industrial. Los resultados adquiridos mostraron que el sistema puede identificar correctamente varias señales de seguridad y tomar decisiones dentro del rango operativo del sensor de la cámara.

Summary

This thesis covers the design and implementation of a machine vision system using a camera integrated into a bipedal robot, with the aim of identifying and interpreting industrial signage.

The main objective was to provide the robot with basic visual capabilities that allow it to perceive its environment and, through this, respond to relevant stimuli, such as identifying objects by their shape or specific colors.

The system was designed using an Intel RealSense D435i camera, which was integrated into a Raspberry Pi 5 based processing platform using the OpenCV library. Image processing techniques were employed, achieving the ability to detect visual characteristics such as shapes and colors related to safety signage in industrial environments.

Furthermore, the camera provides depth information, allowing the system to calculate the distance between the robot and the detected objects. This, in turn, facilitates navigation actions based on the proximity of the detected signal, such as reducing speed, stopping, or changing direction.

Experimental tests were conducted to evaluate the system`s performance under different industrial signal detection scenarios. The results showed that the system can correctly identify various safety signals and make decisions within the camera sensor`s operating range.

Tabla de Contenido

Lista de tablas

Lista de figuras

1. Introducción

1.1. Contexto

Debido a los avances tecnológicos que tenemos hoy en día, es posible el desarrollo de soluciones robóticas cada vez más sofisticadas que logran ambientarse de mejor manera a entornos reales, ya sea para la automatización, visión artificial, movilidad, entre otros. En particular, los robots bípedos son un campo complejo y desafiante dentro de la industria robótica móvil, debido a que tienen como finalidad simular el movimiento humano, para acceder a espacios o terrenos diseñados para personas. Es por ello que esta capacidad, es increíblemente útil y valiosa para ambientes industriales, dado que la infraestructura, generalmente no está optimizada para robots móviles que usan ruedas, orugas o múltiples patas [12].

Por otro lado, la incorporación de sistemas de visión artificial posibilita a los robots adquirir información importante de su entorno visual, ajustarse a cambios en tiempo real y tomar decisiones independientes. Las cámaras RGB-D, como el caso de las Intel RealSense, se utilizan debido a su posibilidad de combinar la imagen de color con percepción de profundidad, lo cual incrementa fundamentalmente la comprensión espacial.

La unión de estas tecnologías, tanto la movilización bípeda como la percepción visual artificial, ayuda a adquirir una mayor versatilidad, autonomía y precisión en la creación de plataformas móviles, por lo cual resulta ser beneficioso para el caso de tareas de monitoreo, asistencia, navegación o inspección en ambientes industriales estructurados.

1.2. Fundamentación

El uso de aplicaciones industriales cada vez más modernas necesita soluciones más flexibles, autónomas y con la capacidad de adaptarse en entornos dinámicos. Las tecnologías de captación visual en tiempo real en conjunto con la robótica bípedo, contribuyen notablemente a esta necesidad.

Este trabajo surge debido al impulso de examinar el potencial uso de la cámara Intel RealSense D435i en un robot bípedo, mediante el uso de una Raspebrry Pi 5 como plataforma de procesamiento, para lograr las habilidades visuales elementales en función de la comprensión del entorno. Se busca corroborar, si es factible llevar a cabo funciones simples de visión que permitan en escenarios industriales complejos la capacidad de poder adaptarse, mediante el uso de herramientas de software como OpenCV y SDK de RealSense.

1.3. Objetivos

1.3.1. Objetivo general

Diseñar e implementar un sistema de visión artificial sobre la base de una cámara RGB-D, que interprete la señalización industrial relevante y genere acciones de navegación, para optimizar el desempeño del robot bípedo, mejorando su capacidad de percepción, análisis y toma de decisiones en entornos industriales variables y complejos.

1.3.2. Objetivos específicos

- Implementar un sistema de captura de imágenes RGB y profundidad mediante la selección e incorporación física de la cámara RealSense en conexión a la Raspberry Pi 5 en la estructura del robot bípedo.
- Configurar mediante el sistema operativo Raspberry Pi OS el entorno de desarrollo, considerando las bibliotecas requeridas.
- Elaborar algoritmos para el uso de segmentación de colores y la detección de líneas, formas y distancias gracias al uso de OpenCV y el SDK de RealSense.

1.4. Alcances y limitaciones

Este trabajo se enfoca únicamente en añadir e incorporar un sistema de visión en un robot bípedo ya fabricado, sin cambiar el esquema de movimiento ni su gestión dinámica. Además, no se considera el uso de inteligencia artificial compleja o la identificación de objetos mediante redes neuronales, pero existe la posibilidad de añadir estas capacidades en trabajos futuros.

El sistema es sometido a pruebas en escenarios supervisados que simulan entornos industriales, pero no en instalaciones reales. Igualmente, el análisis de imágenes se restringe a funciones básicas, es decir, la detección de líneas, formas, segmentación de colores y estimación de distancias en entornos controlados, con la finalidad de evaluar su aplicabilidad a contextos industriales. Además, cabe destacar que el robot está actualmente en fase de ensamblaje y pruebas estructurales, por lo que no es posible abarcar la creación de modelos 3D ni la conducción autónoma. Entonces, como consecuencia las validaciones del sistema de visión se llevaron a cabo con el robot bípedo instalado en un mástil fijo, simulando la caminata bípeda del robot y su comportamiento frente a estas señaléticas.

1.5. Planteamiento del problema y propuesta de solución

Hoy en día, la robótica en ambientes industriales se ha centrado en gran medida en robots estáticos o con ruedas, los que tienen dificultades para moverse en sitios diseñados para el desplazamiento de personas, como es el caso de escaleras, desniveles o rincones estrechos. Además, numerosas alternativas autónomas existentes no son capaces de visualizar directamente su entorno, lo cual reduce su habilidad para adaptarse y desplazarse de forma inteligente.

Por lo tanto, considerando este panorama, es crucial equipar a los robots bípedos de sistemas de visión artificial que les posibiliten obtener información y datos del ambiente por sí solos, de forma asequible y en tiempo real, simplificando de esta manera su operatividad en áreas industriales organizados y cambiantes. No obstante, cabe considerar que incorporar estos sistemas en plataformas de bajo presupuesto, con limitaciones de tamaño y procesamiento, representa un reto considerable.

Para abordar esta dificultad, se propone un sistema de visión artificial basado en la cámara Intel RealSense D435i, la cual se conectará a una Raspberry Pi 5, la cual a su vez se integrará físicamente al robot bípedo. La finalidad es obtener datos de profundidad para detectar estímulos visuales sencillos, como líneas o áreas destacadas por color, y así confirmar su utilidad en tareas de navegación o revisión básica en ambientes industriales controlados. En este caso específico, se busca la identificación de señales de seguridad industrial.

2. Marco teórico

2.1. Robótica bípeda

Se centra en diseñar, controlar y fabricar sistemas robóticos que emulen la forma o método de locomoción humana, es decir, utilizando dos extremidades inferiores. Esto permite que estos robots puedan incorporar múltiples articulaciones, actuadores y sistemas de control, los cuales le ayuden a poder caminar, detenerse y cambiar de dirección logrando mantener el equilibrio [12].

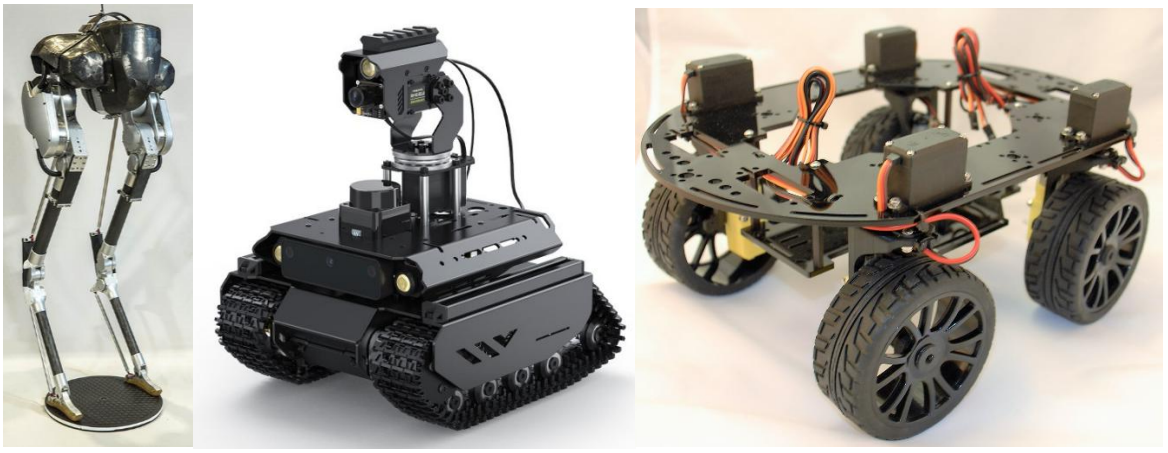


Figura 2.1: Tipos de robots: Robot Bípedo, Robot Oruga y Robot con Ruedas

Otros robots móviles, como son aquellos que utilizan ruedas u orugas, los cuales poseen una cinemática más simple, debido a que cuentan con un punto de apoyo continuo y estable sobre el suelo. Los robots bípedos en cambio, deben lidiar con la dificultad de poder mantenerse en equilibrio, encontrar un control preciso del centro de masa y de las fuerzas que actúan sobre el robot en cada instante y coordinar movimientos en terrenos irregulares. La locomoción bípeda presenta desafíos técnicos significativos, donde el robot debe lidiar con la alternancia constante entre fases de apoyo simple y doble, lo que implica que va presentar instancias donde tendrá solo un pie en contacto con el suelo, por lo que se debe asegurar que el centro de masa permanezca dentro de una región estable para evitar caídas. Por lo tanto, es necesario tener una estabilidad dinámica mediante la integración de sensores, motores, algoritmos de control y modelos dinámicos complejos para la correcta coordinación de movimientos.

El entorno por el cual se desplazará el robot influye directamente en su desempeño y estabilidad, debido a que obstáculos, diferentes tipos de suelo, terrenos irregulares, pendientes, escaleras, pueden alterar y modificar las condiciones dinámicas del movimiento. Por esta razón es imprescindible que

sea capaz de interpretar y percibir su alrededor, logrando anticipar posibles obstáculos o instancias de riesgo antes de ejecutar una acción, tales como poder ajustar los parámetros de la longitud del paso, la altura del pie, la velocidad de marcha o trayectoria de desplazamiento, para así tener una locomoción más eficiente y segura [13].

Dado este contexto, la visión artificial resulta ser una herramienta importante para dotar a estos tipos de robots de la información visual relevante en relación a su entorno. Mediante el procesamiento de imágenes y datos visuales, el sistema consigue identificar obstáculos, estimar distancias, obtener referencias espaciales que favorecen la toma de decisiones durante el desplazamiento, y reconocer superficies transitables. Esta mejora significativa en la autonomía y robustez de los robots bípedos, permitiéndoles ser la base para el desarrollo de estrategias de navegación y control avanzadas.

2.2. Visión artificial

Es una rama de la inteligencia artificial que facilita a los sistemas computacionales adquirir, procesar, interpretar y analizar la información visual obtenida mediante imágenes o secuencias de video por cámaras, con la finalidad de extraer conocimiento útil del entorno [1]. En consecuencia, actúa como un medio por el cual el sistema puede percibir el sentido de la vista de las personas.

Figura 2.2: Ilustración Referencial de la Visión artificial

El procesamiento digital de imágenes está estrechamente vinculado a esta área, porque proporciona las herramientas matemáticas y algorítmicas necesarias para poder transformar, mejorar y analizar las imágenes. El PDI se enfoca principalmente en la manipulación de la imagen en sí. En cambio, la visión artificial busca más allá, tratando de interpretar el contenido visual y utilizar la información para apoyar la toma de decisiones del sistema robótico.

Este PDI se genera por medio de un flujo estructurado que permite transformar datos visuales en información útil para el control del robot [2]. Se divide en las siguientes etapas:

- Captura de imágenes: se basa en la adquisición de información visual mediante sensores como cámaras, las cuales producen imágenes en tiempo real del entorno.
- Preprocesamiento: se aplican técnicas enfocadas a mejorar la calidad de la imagen o facilitar su análisis, como son corrección de la iluminación, filtrado del ruido, normalización o conversión de espacios coloridos.
- Extracción de características: una vez preprocesadas las imágenes, identifican elementos relevantes como contornos, bordes, regiones de interés, formas geométricas o colores dominantes, para reducir la cantidad de información útil para el análisis posterior.
- Toma de decisiones: la información extraída es empleada por el sistema de control para efectuar acciones, tales como evitar obstáculos, ajustar parámetros de locomoción o modificar la trayectoria del robot.

La visión artificial abarca una gran variedad de técnicas, las cuales para este proyecto se concentraron en la identificación de elementos visuales del entorno, teniendo como prioridad realizar soluciones eficientes y compatibles con sistemas embebidos.

La segmentación por color nos permite separar regiones de la imagen en función de características cromáticas, ayudando a facilitar la identificación de objetos o zonas específicas dentro de una escena. Esta técnica resulta especialmente útil en entornos controlados o industriales, ya que los colores pueden ser utilizados como referencias visuales.

Además, la detección de contornos es utilizada para la identificación de los límites de objetos presentes en la imagen, proporcionando información relacionada a su forma y posición. Adicionalmente, el análisis de formas facilita la caracterización geométrica de dichos objetos, lo que contribuye a su clasificación y a poder estimar su relación espacial en relación al robot.

A pesar de que es posible para la visión artificial abarcar aplicaciones avanzadas como el reconocimiento facial, la identificación de personas, el análisis de gestos o la identificación de objetos en 3D, este proyecto no se encuentra enfocado en dichas áreas. Su enfoque será la interpretación de elementos visuales del entorno industrial, como la señalización de seguridad, para apoyar la navegación del robot y la toma de decisiones, teniendo como prioridad la percepción de lo que lo rodea por sobre la identificación de individuos u objetos complejos.

2.3. Tecnologías de visión usadas en robots bípedos

Los robots bípedos necesitan sistemas que les permitan entender e interpretar su entorno y tomar decisiones a medida que se van desplazando [12]. Diferentes proyectos de robots bípedos han utilizado varios tipos de sensores de visión, entre los cuales resaltan las cámaras RGB convencionales, los sistemas de visión estéreo, cámaras LiDAR y las cámaras RGB-D.

Las cámaras RGB tradicionales son comúnmente usadas, porque tienen un bajo costo y son fáciles de integrar. Sin embargo, este tipo de sensores sólo proporciona información en dos dimensiones del entorno, lo cual limita la capacidad de estimar distancias y la percepción de la profundidad.

Los sistemas de visión estéreo usan dos cámaras para calcular la profundidad por medio de técnicas de triangulación, lo cual permite obtener información en tres dimensiones del entorno. Este esquema requiere un mayor procesamiento computacional, y de una calibración precisa entre ambas cámaras.

También existen los sensores LiDAR, los cuales se han vuelto muy populares últimamente debido a que entregan una alta precisión en la estimación de distancias, una efectiva detección de obstáculos y un funcionamiento independiente de la iluminación. Esto es gracias a que miden las distancias por medio de pulsos laser, los cuales generan impulsos que viajan a la velocidad de la luz, rebotan en los objetos circundantes y vuelven al sensor LiDAR, obteniendo los mapas 3D de su entorno. El uso de estos sensores implica un mayor consumo energético, una integración más compleja, y un costo mayor.

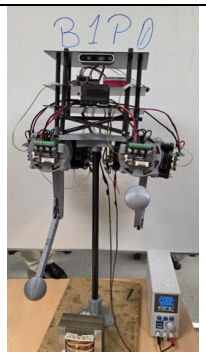
En este proyecto, se utilizó una cámara Intel RealSense D435i, la cual proporciona imágenes en color y mapas de profundidad al mismo tiempo, por lo que este tipo de sensores resultan muy útiles para aplicaciones robóticas, ya que permiten estimar distancias y detectar objetos con mayor facilidad. Aun así, este tipo de sensores también tiene algunas limitaciones, tales como que requieren más recursos computacionales que su alcance puede verse afectado por la iluminación.

2.3.1. Comparación de sistemas de visión de diferentes robots bípedos

Diversos robots bípedos fabricados en la actualidad incorporan diversas tecnologías de percepción visual para interactuar con su entorno [13]. Por ejemplo, robots humanoides como ASIMO usan sistemas de visión estéreo, para poder evaluar la profundidad del entorno, mientras que robots más avanzados como Atlas combinan múltiples sensores, además de cámaras y sistemas LiDAR, para poder adquirir una percepción visual más precisa del entorno. En cambio, robots enfocados a la interacción con los humanos, como NAO y Pepper, utilizan principalmente cámaras RGB junto con sensores para lograr el reconocimiento visual y el seguimiento de objetos. Asimismo, existen robots bípedos no humanoides, como el caso de Cassie y Digit, que utilizan sensores de visión y profundidad para mejorar la estabilidad y permitir la navegación autónoma en entornos reales. A continuación, se presenta una tabla comparativa de estos.

Tabla 2.1: Tabla comparativa de sistemas de visión de diferentes robots bípedos

Robot bípedo	**Imagen**	**Sistema de visión**	**Características**	**Ventajas**	**Desventajas**
ASIMO (2000)		Cámaras estéreos	Dos cámaras para estimar profundidad	Buena percepción espacial y navegación	Requiere calibración y mayor procesamiento
NAO (2008)		Cámaras RGB	Cámaras frontales para reconocimiento visual	Bajo consumo y fácil integración	No proporciona información de profundidad

Atlas (2013)		Sensores LiDAR más cámaras	Combina visión con sensores de distancia	Alta precisión para detección de obstáculos	Sistema complejo y alto costo
Pepper (2014)		Cámaras RGB más sensores 3D	Combina cámaras con sensores de profundidad	Mejor interacción con el entorno	Mayor consumo de recursos
Cassie (2017)		Cámaras RGB-D y sensores de navegación	Percepción visual para locomoción autónoma	Buena detección del entorno	Mayor consumo computacional
Digit (2019)		Cámaras RGB-D más LiDAR	Navegación autónoma en entornos complejos	Alta precisión de percepción	Sistema de gran complejidad
BIPO (2026)		Cámara RGB-D	Imagen RGB más mapas de profundidad.	Permite detectar objetos y estimar distancia.	Dependencia de iluminación y capacidad de procesamiento

2.4. Cámara Intel RealSense D435i

Esta cámara es un sensor de visión avanzado, el cual está diseñado para aplicaciones robóticas y tiene percepción tridimensional [6]. Es de tipo RGB-D, lo cual permite capturar imágenes a color (RGB) e información de profundidad al mismo tiempo. Además, cuenta con un sistema de visión estéreo activa, para generar mapas de profundidad en tiempo real [7]. Asimismo, incorpora una unidad de medición inercial (IMU). Todo esto la hace una herramienta especialmente valiosa para el uso en aplicaciones de robótica móvil, que necesitan información visual y espacial de manera simultánea. Finalmente, es posible obtener no solo información visual bidimensional, sino además datos tridimensionales del ambiente, fundamentales para la navegación y control.

Figura 2.3: Cámara Intel RealSense D435i

A continuación, se muestra una tabla con sus características principales:

Tabla 2.2: Características generales de la cámara [6]

Características	**Descripción**
Uso del ambiente	Tanto interior como exterior
Tecnología de profundidad	Estereoscópica
Resolución de profundidad	Hasta 1280 x 720
Resolución RGB	1920 x 1080
Velocidad de fotogramas de profundidad	Hasta 90 fps
Campo visual profundidad (FOV)	87º x 58º
Placa de Procesador de Visión	Visión Processor D4
Interfaz	USB 3.1

Esta cámara de profundidad pertenece a la serie RealSense D400, la cual utiliza tecnología de visión estereoscópica para el cálculo de la profundidad. En particular, esta implementación consta de un sensor izquierdo, un sensor de imagen derecho y un proyector infrarrojo opcional [8]. El proyector infrarrojo proyecta un patrón infrarrojo estático no visible para mejorar la precisión de la profundidad en escenas con poca textura. Los sensores de imagen izquierdo y derecho capturan la escena y envían

los datos al procesador de imágenes de profundidad, que calcula los valores de profundidad para cada pixel de la imagen mediante la correlación de los puntos de la imagen izquierda con la imagen derecha y por medio del desplazamiento entre un punto de cada imagen. Los valores de los pixeles de profundidad se procesan para generar un fotograma de profundidad, donde los fotogramas de profundidad subsiguientes crean una secuencia de video de profundidad.

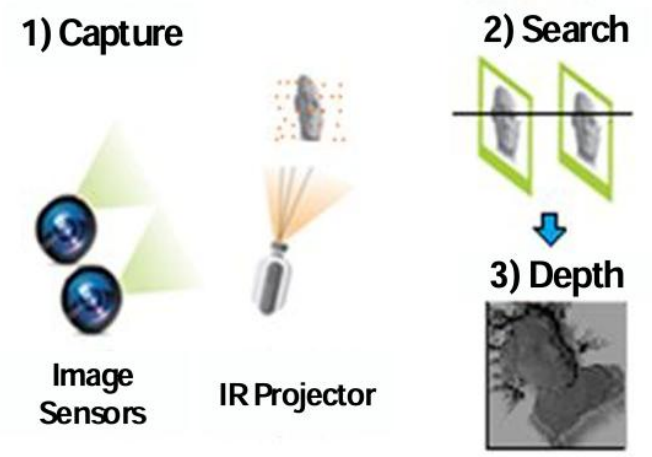


Figura 2.4: Tecnología de visión estéreo infrarroja activa

El valor del pixel de profundidad es una medición de la distancia al plano paralelo de los sensores de imagen Z y no la distancia absoluta R, como se ilustra en la imagen.

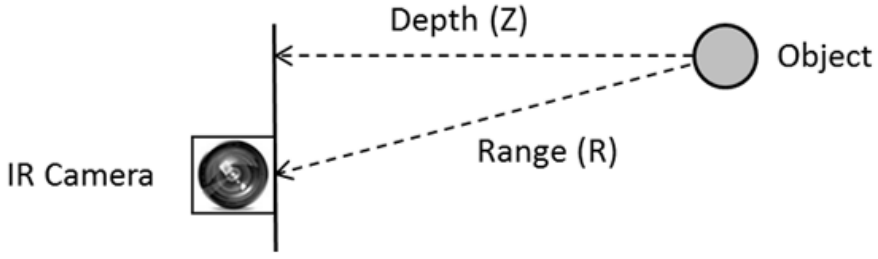


Figura 2.5: Medición de profundidad (Z) vs. Rango (R)

Este tipo de cámara integra diversos sensores que permiten la captura de información visual y la estimación de profundidad en tiempo real. En la tabla 2.5 se presentan los principales componentes del sistema de visión, junto con sus funciones y el tipo de información que generan [6]. La figura 2.6 muestra la ubicación de estos componentes.

Tabla 2.3: Descripción de los componentes del sistema de visión de la cámara

Componente	**Función principal**	**Información generada**	**Uso en el sistema**
Sensor RGB	Captura imágenes a color del entorno	Imagen RGB	Permite detectar señales colores y formas
Sensores estéreo izquierdo y derecho	Capturan imágenes infrarrojas desde dos perspectivas distintas	Imagen estéreo	Permiten calcular la distancia de los objetos mediante triangulación
Proyector infrarrojo	Proyecta un patrón de luz infrarroja sobra la escena	Patrón estructurado	Mejora la detección de profundidad en superficies con poca textura
Procesador de visión D400	Procesa la información de los sensores	Mapa de profundidad	Genera la información de distancia utilizada por el sistema de navegación

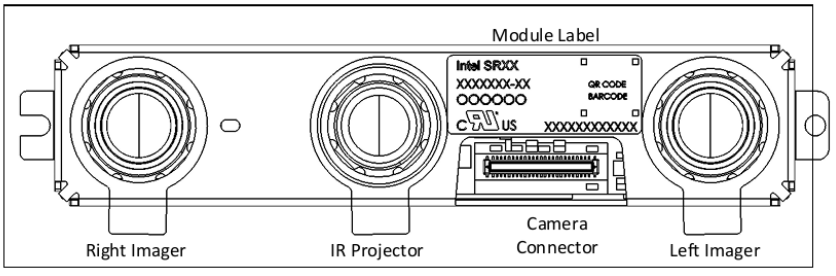


Figura 2.6: Ubicación de los sensores [6]

En relación al movimiento y orientación contamos con una IMU, donde es posible obtener información adicional sobre el movimiento del sistema, como aceleraciones y rotaciones. Esta capacidad resulta especialmente útil cuando la cámara se encuentra en movimiento, debido a que mejora la estimación de la posición y orientación del sensor.

Asimismo, la integración de datos visuales e inerciales permiten trabajar con estimaciones de profundidad más robustas facilitando la implementación de técnicas como SLAM (Simultaneous Localization and Mapping), capaces de construir mapas del entorno junto con la estimación de la localización del sistema [15].

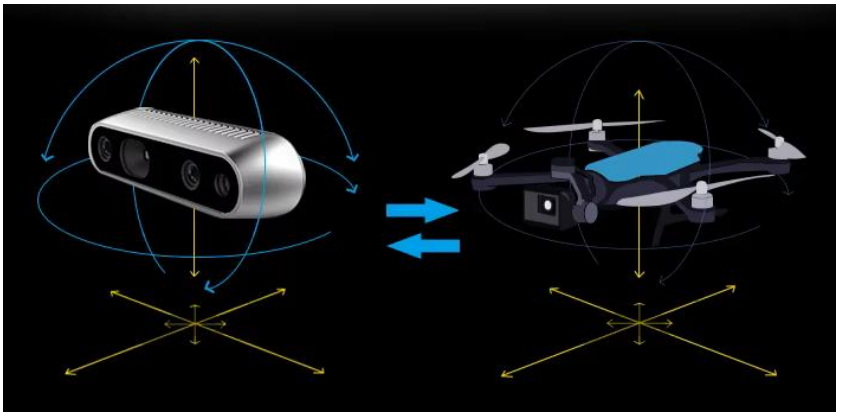


Figura 2.7: Uso de una unidad de memoria inercial (IMU)

La unidad de medición inercial se utiliza para la detección de movimientos y rotaciones en 6 grados de libertad (6DoF), y corresponde al modelo Bosch BMI055, el cual se puede observar en la siguiente imagen y mediante la tabla 2.3 se aprecian sus especificaciones.

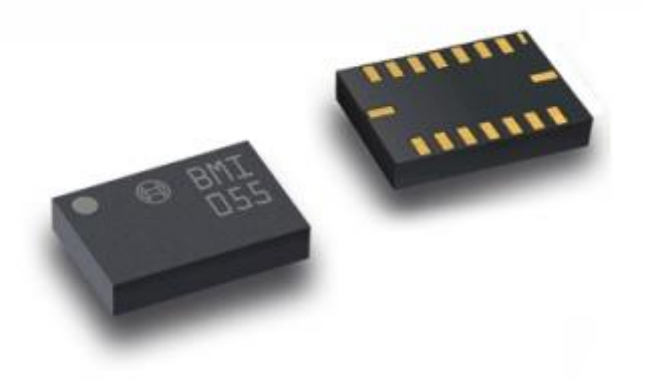


Figura 2.8: IMU modelo Bosch BMI055

Tabla 2.4: Especificaciones del IMU BMI055

Parámetros	**Propiedades**
Grados de Libertad	6
Rango de Aceleración	± 4 g
Frecuencia de Muestreo del Acelerómetro	100, 200 (Hz)
Alcance del Giroscopio	±1000 deg/s
Frecuencia del Muestreo del Giroscopio	200, 400 (Hz)
Muestra de Precisión de la Marca de tiempo	50 µsec

Esta combina una variedad de sensores con giroscopios para detectar tanto la rotación como el movimiento en 3 ejes, así como el Paso (Pitch) movimiento hacia arriba o hacia abajo respecto al eje transversal, la Guiñada (Yaw) movimiento hacia la izquierda o derecha alrededor del eje vertical y el Balanceo (Roll) rotación de adelante hacia atrás alrededor del eje longitudinal.

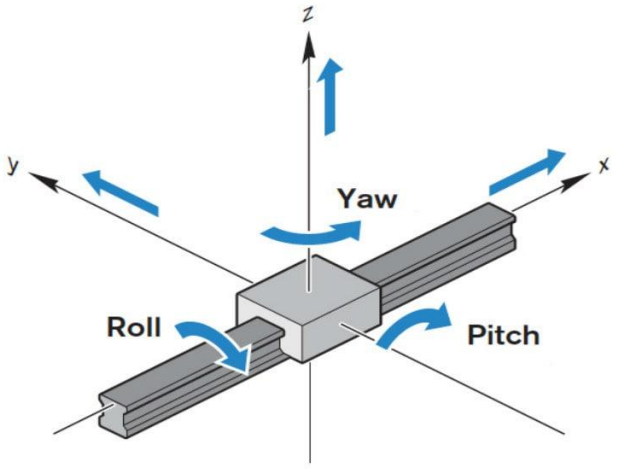


Figura 2.9: Conceptos básicos de movimiento

En cuanto a la arquitectura interna la cámara se compone principalmente de un módulo de profundidad y un procesador de visión dedicado (D4), encargado de procesar la información capturada por los sensores. El módulo de profundidad incluye los sensores estéreo y el proyector infrarrojo, mientras que el procesador se encarga de generar los mapas de profundidad en tiempo real. La comunicación con el sistema de procesamiento se realiza mediante interfaz USB.

En relación a las características físicas y eléctricas de la cámara tenemos la siguiente imagen y una tabla describiéndolas.

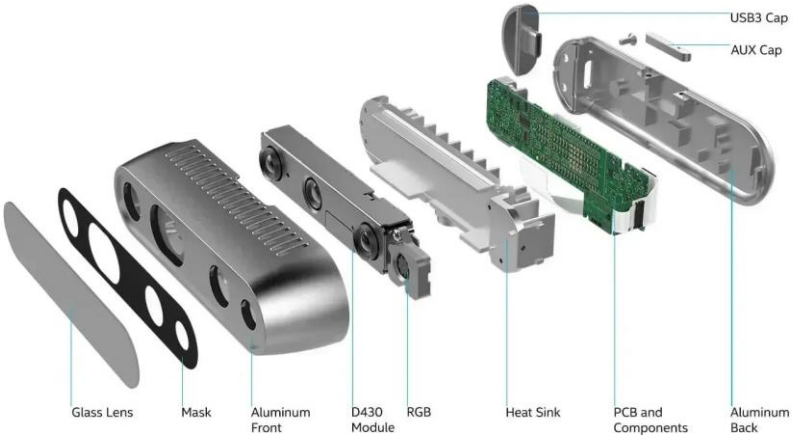


Figura 2.10: Estructura y materiales de la cámara

Tabla 2.5: Características físicas y eléctricas de la cámara

Dimensiones	**Valor**
Dimensiones	90 mm x 25 mm x 25 mm
Peso	120 g
Conector	USB-C 3.1
Voltaje de operación	5 V
Consumo	~1.5 – 2 W
Temperatura optima	0º - 35º C
Sistema de montaje	Rosca UNC ¼ -20 o Tornillos M3

También es indispensable el campo visual (FOV) del sensor que determina la región del entorno que logra ser capturada en cada imagen. Por lo que, si el campo visual es grande, el robot puede detectar señales y objetos que están en diferentes lugares dentro del ambiente del robot.

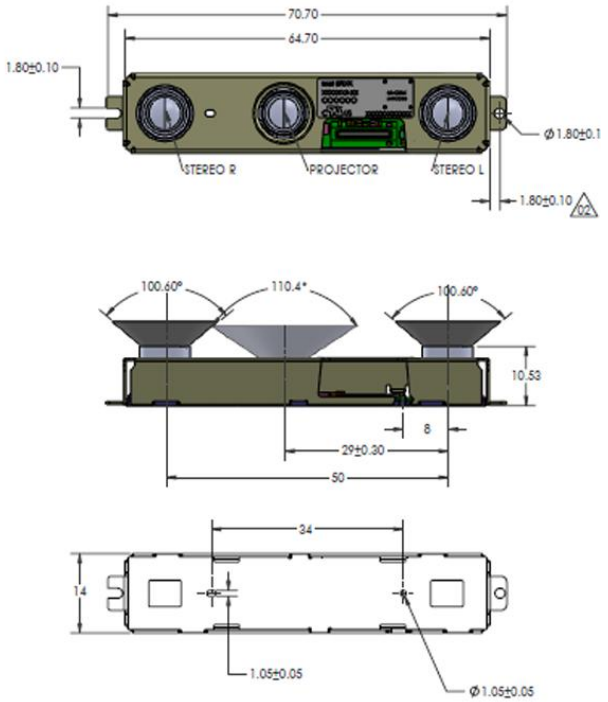


Figura 2.11: Campo visual de la cámara [6]

Tabla 2.6: Campo visual de la cámara [6]

FOV	**Estéreo**	**Proyector**
Vertical	65.5º	69º
Horizontal	91.2º	100.4º
Diagonal	100.6º	100.4º
Inclinación del Eje Óptico	±1º	±1º

En este proyecto, la cámara funciona como el principal sensor visual para la apreciación del medio industrial que lo rodea, la elección de esta cámara se justifica por su capacidad de proporcionar información visual y espacial en tiempo real, contribuyendo derechamente a la autonomía, seguridad y robustez del sistema robótico desarrollado.

2.5. Raspberry Pi 5

Es una plataforma de computación embebida, es decir, es un dispositivo que se especializa en realizar funciones dedicadas dentro de sistemas más grandes, de tipo Single-Board Computer (SBC), esto es, un computador de una sola placa. Corresponde a la quinta versión y está diseñada para entregar capacidades de procesamiento avanzadas en un formato compacto y de bajo costo. En esta placa se integra el procesador, la memoria, las interfaces de comunicación, además de los puertos necesarios para la interacción con actuadores, sensores o dispositivos periféricos [17].

Figura 2.12: Placa Raspberry Pi 5

Son utilizados en gran medida para la creación de prototipos y sistemas integrados, debido a que su arquitectura, flexibilidad, gran variedad de herramientas de desarrollo y diseño de bajo consumo energético junto con su gran rendimiento y sus opciones de conexión, resulta ser una base perfecta para iniciativas de proyectos robóticos móviles y procesamiento de imágenes [18].

Entre sus aspectos técnicos más destacados, tenemos la siguiente tabla:

Tabla 2.7: Principales Características de la Raspberry Pi 5

Características	Descripción
Procesador	Broadcom BCM21712 Quad-Core ARM Cortex-A76
Frecuencia	2.4 GHz
Memoria RAM	4GB/8GB
GPU	VideoCore VII
Conectividad	Wi-Fi 802.11ac, Bluetooth 5.0
Red	Ethernet Gigabit

Puertos USB	2 x USB 3.0 y 2 x USB 2.0
Interfaz de cámara	2 x MIPI CSI
Almacenamiento	Tarjeta microSD
Alimentación	5V / 5A mediante USB-C
Salida de video	2 x micro-HDMI
Controlador PCIe	PCIe 2.0 x1
Controlador de E/S	RP1 I/O Controller
GPIO	40 pines

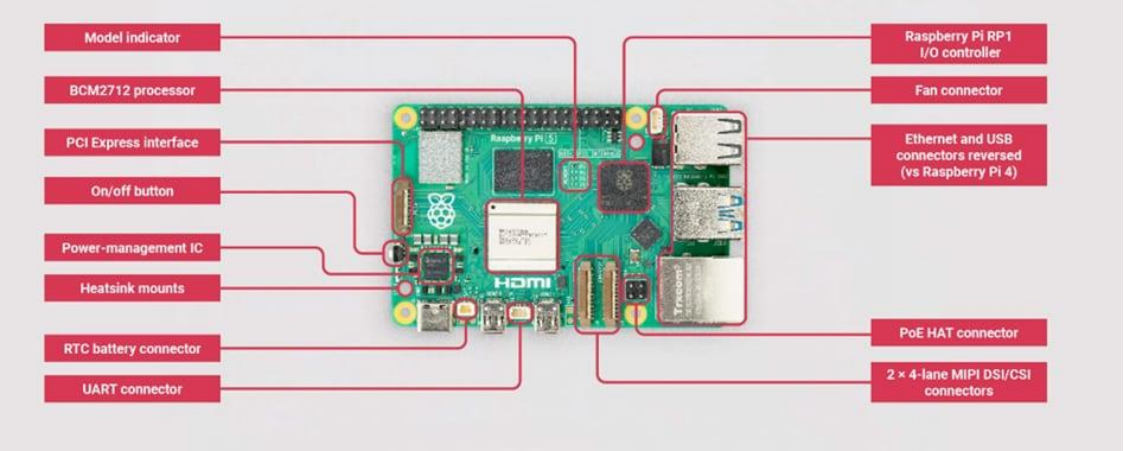


Figura 2.13: Ubicación de las características principales de la Raspberry

Ofrece una tasa de rendimiento de CPU de dos a tres veces mayor a la de la Raspberry Pi 4. Además, posee una mejora sustancial en el rendimiento gráfico, debido a que tiene una GPU VideoCore VII de 800 MHz, cuenta con doble salida de video 4Kp60 por medio de HDMI y compatibilidad con cámaras de última generación gracias a su Procesador de Señal de Imagen (ISP) de Raspberry Pi rediseñado, lo cual proporciona una fluida experiencia para los usuarios y abre la puerta a nuevas aplicaciones. Es la primera vez que Raspberry Pi consta de un ordenador de tamaño completo con componentes de silicio fabricados internamente por Raspberry. El puente sur RP1 proporciona la mayor parte de las capacidades de entrada/salida de esta placa, lo que supone un salto cualitativo en el rendimiento y la funcionalidad de los periféricos.

Es por ello que cuenta con una capacidad de procesamiento apropiada para la implementación de algoritmo de visión artificial, considerando adquisición de imágenes, preprocesamiento y análisis visual al instante, permitiendo emplear flujos de datos procedentes de cámaras y otros sensores sin afectar significativamente el desempeño del sistema. Esta resulta ser vital importancia para aplicaciones robóticas, debido a que requieren de una respuesta rápida frente a cambios en el entorno. Por lo que poder ejecutar bibliotecas y frameworks de visión artificial ampliamente utilizados permite hacer más fácil el desarrollo y la validación de soluciones de percepción, logrando mantener un equilibrio entre consumo de recursos y rendimiento.

A pesar de que es posible utilizar otras plataformas embebidas como NVIDIA Jetson Nano o NVIDIA Jetson Xavier NX, las cuales ofrecen un mayor rendimiento computacional y capacidades de procesamiento paralelo mediante GPU, siendo más adecuadas para aplicaciones de visión artificial complejas, presentan un mayor costo y consumo energético que una Raspberry, teniendo ésta última una integración más fácil.

Figura 2.14: Accesorios incluidos

Además de la placa, contiene un cable USB-HDMI, una carcasa base con ventilación, fuente de alimentación y una tarjeta microSD de 32 GB.

2.6. Raspberry Pi OS

Es un sistema operativo basado en Linux, el cual está optimizado para usarse en dispositivos Raspberry Pi. Otorga un entorno optimizado para el hardware de la placa, integrando controladores, herramientas de desarrollo y servicios fundamentales para el uso de aplicaciones en sistemas integrados. Al basarse en una arquitectura Linux, ofrece seguridad, estabilidad y flexibilidad, las cuales son cualidades esenciales para el desarrollo de aplicaciones robóticas que requieren ejecución continua y confiable [18].

Algunos de los beneficios que posee son una interfaz gráfica amigable, una alta compatibilidad con sensores y periféricos, además de un fácil acceso a herramientas de programación y depuración, también recibe actualizaciones frecuentes y soporte comunitario. Incluye soporte nativo para interfaces de cámara y dispositivos USB, favoreciendo la integración de sensores visuales sin necesidad de requerir configuraciones complejas. De igual modo, es compatible con una gran variedad de bibliotecas y frameworks de procesamiento visual, lo cual permite la implementación de algoritmos de procesamiento de imágenes y análisis visual de manera eficiente, por lo que reduce el tiempo de desarrollo y favorece la utilización de herramientas en la comunidad científica y tecnológica.

Figura 2.15: Interfaz del sistema operativo de Raspberry

2.7. Bibliotecas

Para el desarrollo de un sistema de visión artificial, es necesario el uso de bibliotecas de software especializadas que dejen obtener, procesar y analizar información visual eficazmente, debido a que facilitan la implementación de los algoritmos necesarios y aseguran la compatibilidad con la plataforma seleccionada.

2.7.1. Open CV

OpenCV (Open Source Computer Vision Library) es una biblioteca de visión artificial de código abierto, lo que le permite ser una herramienta bastante útil en el ámbito de la investigación y aplicaciones industriales [5]. Debido a que se utiliza para procesar imágenes y videos, además de realizar funciones como detección de objetos, detección de color, identificación de contornos, análisis de movimiento entre otros [4]. Esta desarrollada en C++, pero también se puede usar con Python, esto ha permitido el uso de algoritmos tales como: conversión de color (BGR<->HSV, RGB<-> Grayscale), filtrado y suavizado (Gassuian blur, median filter), detección de bordes (Canny, Sobel), segmentación por color, localización de líneas (transformada de Hough) y análisis de contornos y objetos.

2.7.2. NumPy

NumPy corresponde a una biblioteca elemental para la manipulación de matrices y arreglos numéricos, las cuales son utilizadas en gran medida en aplicaciones científicas y de ingeniería. Realiza operaciones matemáticas complejas de manera efectiva, logrando optimizar el uso de recursos computacionales. Además, se emplea para manejar matrices de imágenes, procesar datos numéricos y cálculo de operaciones necesarias en el procesamiento visual, facilitando el desarrollo de algoritmos claros, estructurados y de alto rendimiento.

2.7.3. Pyrealsense2

Pyrealsense2 corresponde a una biblioteca que pertenece a la interfaz de programación en Python del SDK de Intel RealSense, donde ayuda a obtener un acceso directo a los datos proporcionados por la cámara, incluyendo imágenes RGB y mapas de profundidad. Esto quiere decir que es posible obtener mediciones de distancias y representar el entorno en tres dimensiones, al sincronizar los datos visuales con la información espacial del ambiente, lo que resulta ser de vital importancia para la detección de obstáculos y la movilidad del robot bípedo.

2.7.4. SDK de Intel RealSense

Para interactuar con la cámara de profundidad, se utiliza el kit de desarrollo de software o SDK (Software Development Kit) que proporciona Intel para los dispositivos de la familia Intel Realsense.

El SDK utilizado corresponde al 2.0, el cual ofrece un conjunto de herramientas, controladores y bibliotecas que permiten acceder a los datos de la cámara, incluyendo imágenes en color RGB y mapas de profundidad. Para este caso, se utiliza la biblioteca pyrealsense2 para acceder a la cámara Intel Realsense D435i, la cual corresponde a la interfaz de programación en Python del SDK y permite capturar imágenes, obtener información de profundidad y sincronizar los diferentes flujos de datos que genera el sensor.

Esta combinación son las herramientas que proporcionan la base técnica para llevar a cabo un sistema de visión artificial efectivo, modular y adaptable a plataformas embebidas. Trabajan de manera complementaria para conseguir información visual, procesarla de forma correcta y recopilar datos relevantes para la toma de decisiones instantáneamente en zonas industriales.

2.8. Aplicaciones industriales

En el ámbito industrial, la existencia de riesgos es inherentes derivados de funcionamiento de la maquinaria, de la circulación de vehículos, de una superficie irregular, de la existencia de zonas prohibidas y de condiciones de visibilidad e iluminación variadas. Dado este tipo de entornos, la seguridad en la ejecución operacional es un elemento fundamental, ya sea para el ser humano como para sistemas automatizados. Es por ello que resulta necesario un esfuerzo continuo para el perfeccionamiento de los mecanismos de prevención y control; entonces es apropiado incorporar tecnologías que permitan minimizar la exposición del ser humano al entorno peligroso y que puedan ayudar a la supervisión de los procesos. En estas circunstancias, cobra sentido que los sistemas robóticos móviles sean utilizados para la ejecución de tareas de inspección, supervisión y monitoreo, tales como: la revisión visual automatizada de estructuras, procedimientos, el rastreo de rutas o señales visuales para una mejor orientación, el reconocimiento de zonas codificadas por colores, como el caso de áreas de carga o de seguridad, la interacción con trabajadores humanos mediante indicaciones visuales y el control de calidad en cadenas de producción. La integración de estas capacidades en un robot bípedo haría posible extender su desempeño a lugares de difícil acceso, donde se precise adaptación a superficies humanas, o se demande desplazamiento vertical.

Las plantas industriales utilizan diversos mecanismos de señalización visual, ya que desempeñan un papel crucial en cuanto a la comunicación de peligros, responsabilidades o advertencias en el interior de estas plantas, permitiendo comunicar e identificar rápidamente, en forma estandarizada y lo más comprensible posible, los riesgos que pueden llegar a existir, las zonas de acceso restringido, el uso obligatorio de equipos de protección personal o las condiciones especiales

de operación.

La señalización industrial es muy importante para evitar accidentes y organizar el lugar de trabajo. Estas señales utilizan colores, formas y símbolos estandarizados, para dar información clara y rápida sobre situaciones que requieren atención o acción de los trabajadores.

Uno de los referentes más utilizados es la norma ISO 7010, la cual está definida por organismos internacionales como la International Organization for Standardization, que establece un conjunto de símbolos gráficos normalizados con colores y formas que deben utilizarse en la señalización de seguridad, permitiendo comprenderlos fácilmente, sin importar el idioma o el contexto cultural. La interpretación correcta, es una manera de prevenir accidentes y asegurar un entorno laboral seguro.

Entre las principales categorías se señalización definidas en esta norma, se encuentran las siguientes:

- Señales de advertencia: que nos indican que hay peligro potencial cerca.
- Señales de prohibición: que nos dicen que acciones deben cumplirse
- Señales de seguridad o emergencia: que nos muestran rutas seguras o nos indican donde están los equipos de emergencia.
- Señales de obligación: que nos indican en qué condiciones es apropiado transitar por determinadas áreas.

En este tipo de escenarios, un robot bípedo puede ejercer un rol activo en la interpretación automática de las señales visuales que aparecen en el entorno industrial mediante un sistema de visión artificial. Debido a que el análisis de imágenes y la identificación automática de su contenido permiten al robot identificar las señales de obligación, de advertencia o de prohibición para ayudar a tomar decisiones. A la vez, la identificación de la señalización industrial permite al robot servir como guía, asegurando que su recorrido evite zonas de peligro, áreas restringidas y ajuste la forma de desplazarse en función del contexto del entorno. De este modo, el sistema de visión artificial, no solo ayuda a la autonomía del propio robot, sino que se convierte en un instrumento para la mejora de la seguridad operacional.

Así, la interpretación automática de la señalización industrial a través de sistemas robóticos es una aplicación práctica relacionada con el potencial de reducir riesgos, al mismo tiempo que aumenta la seguridad en las plantas industriales.

3. Metodología

3.1. Arquitectura del sistema

Un sistema de visión artificial está integrado en un robot bípedo y tiene como objetivo principal ser capaz de percibir, interpretar y reaccionar a elementos visuales del entorno industrial, en particular la señalización de seguridad y los obstáculos relevantes para la navegación.

Los elementos principales de la arquitectura del sistema son:

- Una cámara RGB-D, encargada de obtener información visual en color y de profundidad del entorno.
- Una unidad de procesamiento basada en Raspberry Pi 5, que ejecuta los algoritmos de visión artificial.
- Un conjunto de algoritmos desarrollados en software de procesamiento y análisis visual que se encargan de detectar señalización industrial, analizar formas, color y estimar distancias.
- Un módulo de generación de decisiones que convierte la información visual procesada en ordenes de navegación.

De manera general, el sistema desarrollado se compone de una cámara RGB-D, una unidad de procesamiento que se basa en Raspberry Pi 5 y un conjunto de algoritmos de visión artificial que tienen la función de interpretar la información visual del entorno. Con esta información, el sistema desarrollado genera órdenes de navegación que determinan el desplazamiento del robot bípedo, logrando que esta navegación logre ser segura y autónoma.

Figura 3.1: Diagrama de flujo de información

El funcionamiento del sistema de visión artificial tiene por base un flujo continuo de información, que empieza desde el instante que se capturan los datos de profundidad de la escena hasta la toma de decisiones por parte del robot durante la navegación. Esta información sobre el entorno, en primer lugar, es capturada por la cámara RGB-D, y posteriormente enviada a la unidad de procesamiento, donde se ejecutan los algoritmos de visión artificial en tiempo real. A continuación,

el sistema detecta alguna señalética y analiza elementos del entorno relevantes, empleando tanto la información visual dada por imágenes a color como los datos de profundidad para así poder estimar distancias y decidir sobre un eventual peligro. Al final de este proceso interpretativo, se imprime en pantalla la acción que debería tomar el robot frente a la señal detectada.

3.2. Hardware

La cámara Intel RealSense D435i se instaló en la parte frontal del robot bípedo, con la finalidad de tener un punto de vista en primera persona del entorno al momento de su desplazamiento. Este posicionamiento permite tomar imágenes de forma directa de las señaléticas, de los elementos que puedan interrumpir la ruta y de características que pueda tener el terreno. Mediante esta instalación la cámara fue fijada en la posición invertida, debido a las restricciones físicas del diseño del chasis y por la necesidad de optimizar el campo visual frontal del robot. Dada esta condición, fue necesario realizar el ajuste necesario y agregarlo al momento de realizar el procesamiento de imágenes.

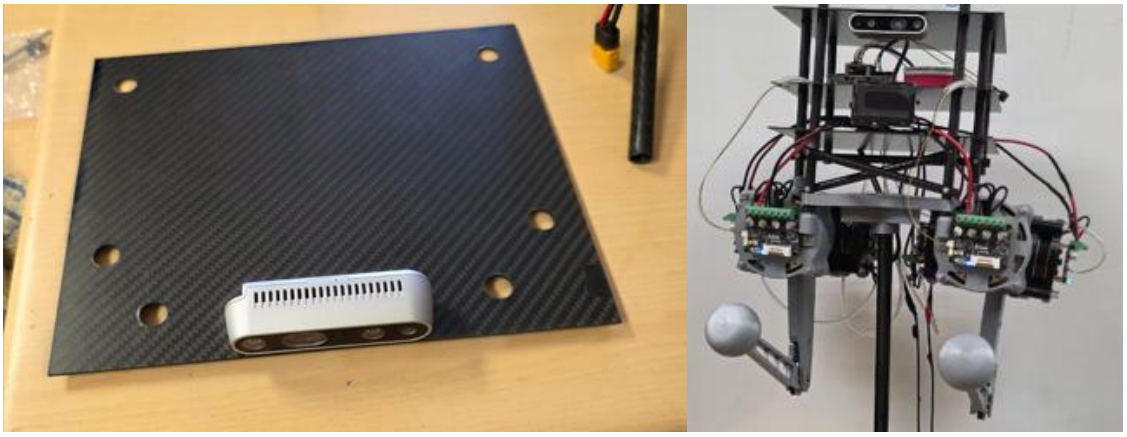


Figura 3.2: Ubicación de la cámara invertida y en la parte frontal del Robot Bípedo

La ubicación frontal de la cámara era necesaria para llevar a cabo las tareas de navegación, ya que resulta ser la manera más practica para que el robot logre anticipar los elementos presentes a su alrededor, lo cual le otorga un marco de decisiones más fiable.

La Raspberry Pi 5 fue seleccionada como la unidad central de procesamiento del sistema de visión artificial, la cual se encuentra incorporada físicamente en la estructura del robot, logrando establecer una conexión directa y compacta, también se utilizó el Case, para darle una mejor autonomía a la placa, entregándole ventilación y protección.

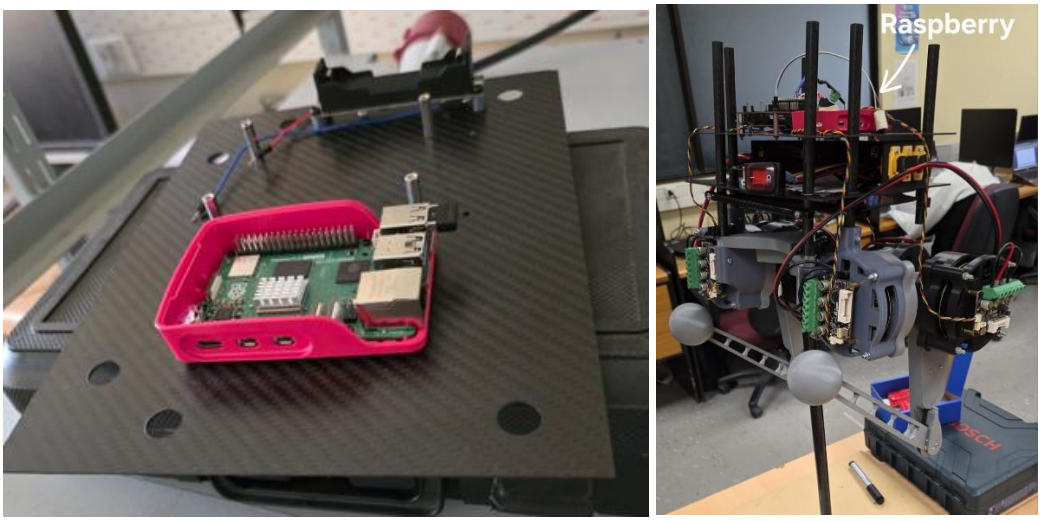


Figura 3.3: Ubicación de la Raspberry en el chasis del Robot

La comunicación de esta Raspberry con la cámara se lleva a cabo mediante una interfaz USB, a través del cual, mientras se transfieren imágenes RGB junto a la información de los datos de profundidad en tiempo real, posibilita el flujo continuo de información visual hacia la unidad mecánica de procesamiento de la Raspberry, y ejecutar los algoritmos correspondientes de visión artificial y análisis de entorno. De esta manera esta especificación arquitectónica permite integrar todo el sistema en un robot móvil, ya que reduce la cantidad de componentes externos, a la vez que favorece la portabilidad total del sistema.

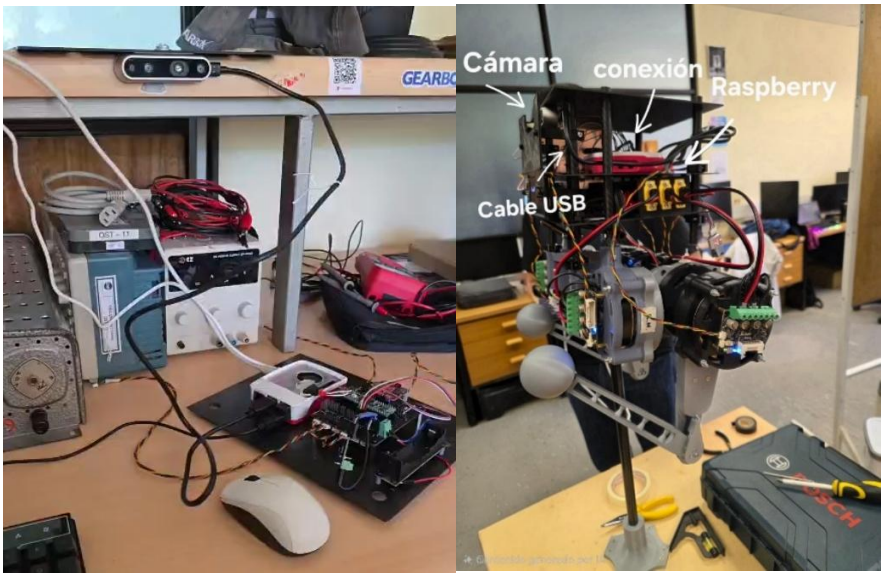


Figura 3.4: Conexión Física entre la Cámara y la Raspberry

En el transcurso de la integración física del hardware se tuvieron en cuenta múltiples consideraciones practicas propias de un sistema robótico móvil. Uno de ellos corresponde a la alimentación eléctrica, dado que tanto la cámara como la unidad de procesamiento deben operar dentro de límites de consumo energético disponibles en el robot, pero sin afectar la autonomía del mismo. La Raspberry se conecta a corriente mediante su fuente de alimentación, donde a su vez se conecta a un monitor, por medio de la entrada HDMI, junto a sus respectivos periféricos, como son mouse y teclado, usando 2 conexiones USB 2.0, con lo cual ya es posible empezar a trabajar. Por otro lado, también se realiza la conexión con la cámara mediante USB 3.0.

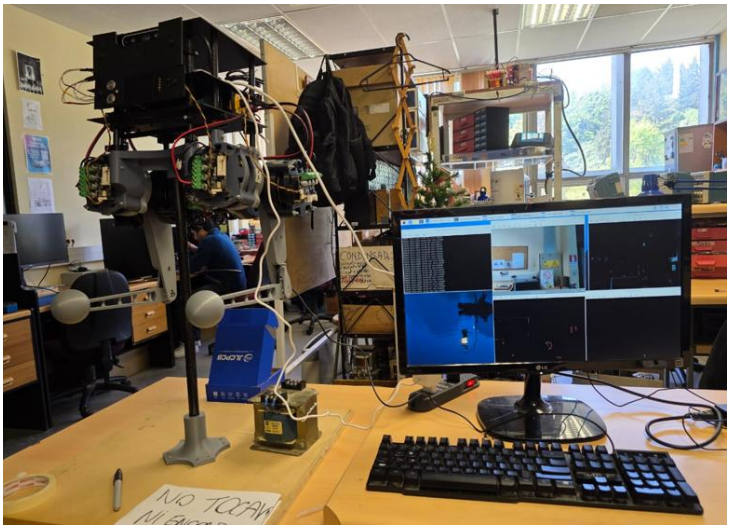


Figura 3.5: Todas las conexiones realizadas

Otro aspecto a tener en cuenta es el campo visual de la cámara, que depende de las características ópticas de la propia cámara como de la posición que tenga en el robot. La posición elegida intenta buscar la máxima cobertura visual, con la finalidad de obtener una detección anticipada de señaléticas industriales, así como de los obstáculos presentes durante la navegación.

Además, se consideraron las limitaciones físicas del montaje, como por ejemplo la disponibilidad de espacio en el chasis, la protección de diversos componentes ante vibraciones o movimientos, así como la fijación de los distintos dispositivos para evitar el desplazamiento en el caso de la locomoción.

3.3. Software

El programa informático del sistema de visión artificial fue desarrollado usando el lenguaje de programación Python, el cual es unos de los lenguajes de programación más utilizados en aplicaciones relacionadas con la robótica y la visión artificial. El optar por Python se debe a que proporciona una sintaxis clara, una gran facilidad de desarrollo y se integra perfectamente con bibliotecas de trabajo que permiten tanto la manipulación de imágenes como la interacción con los sensores. Por otro lado, el hecho de utilizar Python facilita la implementación y validación rápida de algoritmos, lo cual resulta altamente conveniente al trabajar en entornos prototipados o desarrollos experimentales.

Fue diseñado con una estructura modular la cual facilita su comprensión, escalabilidad y mantenimiento, ya que esta estructura separa las diferentes funcionalidades del sistema en módulos independientes, donde cada uno de ellos se concentra en realizar una tarea especifica dentro del flujo general de visión artificial.

Se divide en los siguientes módulos:

- Captura de imágenes: encargado de adquirir imágenes RGB y datos de profundidad provenientes de la cámara.
- Procesamiento visual: responsable del preprocesamiento de las imágenes, implicando filtrado y transformaciones útiles para el análisis.
- Detección e interpretación de señalización: identifica los elementos visuales relevantes del entrono industrial.
- Generación de acciones: traduce la información visual interpretada en decisiones de navegación para el robot bípedo.

La designación de esta estructura posibilita a cada uno de los módulos opere en coordinación dentro de un flujo de procesamiento continuo, desde la adquisición de datos hasta la emisión de ordenes de movimiento.

Al implementar un sistema de visión artificial, son necesarias diversas bibliotecas especializadas, las que permiten llevar a cabo tareas de procesamiento y análisis visual de forma eficiente. Por ende, la biblioteca OpenCV se utiliza para la manipulación de imágenes, detección de colores, análisis de contornos e identificación de formas presentes en el entorno. A su vez, NumPy es utilizada para la manipulación de matrices y arreglos numéricos. Por lo tanto, realizan los cálculos necesarios para el procesamiento de imágenes y el análisis de datos visuales de manera optimizada. Finalmente, pyrealsense2, permite la integración directa de la cámara RGB-D con el software desarrollado, lo cual nos da el acceso a la información visual provista por el sensor.

En la práctica, estas bibliotecas ofrecen la base técnica necesaria para dar marcha al sistema de visión artificial de forma eficiente, manteniendo la arquitectura de software clara y adecuada para la ejecución en una plataforma integrada.

3.4. Captura y preprocesamiento de imágenes

La captación de información visual en color se realiza mediante el uso de la cámara Intel RealSense D435i, la cual permite capturar imágenes RGB en tiempo real del entorno que lo rodea, representando la primera entrada del sistema de visión artificial, donde estas imágenes son la principal fuente de información para la detección de señaléticas o de factores visuales importantes para la navegación del robot bípedo [10]. Las imágenes son adquiridas con una resolución apropiada para el procesamiento al instante, logrando un equilibrio entre la calidad de la imagen y la carga computacional. Además, se adquieren a una frecuencia aproximadamente constante, lo suficiente para garantizar una oportuna respuesta del sistema frente a posibles variaciones del entorno en el que el robot se está desplazando.

Tabla 3.1: Características del sensor RGB

Característica	**Valor**
Resolución RGB	1920 x 1080
Sensor RGB	2 MP
Tecnología	Rolling Shutter
Velocidad de captura	Hasta 30 fps
Campo visual RGB	69º x 42º

De manera paralela, el sistema también genera mapas de profundidad, que se encuentran relacionados con la información tridimensional del ambiente. Estos mapas incluyen la estimación de la distancia entre el robot y los objetos de la escena, mediante cada pixel de la imagen. La información de profundidad se utiliza para complementar el análisis visual, ya que con ello es posible poder estimar la distancia a obstáculos y señaléticas [3]. En general, la combinación de imágenes RGB y profundidad proporciona una completa representación del entorno, lo que mejora la capacidad del sistema para interpretar adecuadamente la escena observada [11].

Tabla 3.2: Descripción de formatos de imagen

Formato	Resolución	Fotogramas por segundo (FPS)	Comentario
Z (16 bits)	1280 x 720 848 x 480 640 x 480 640 x 360 480 x 270 424 x 240	6, 15, 30 6, 15, 30, 60, 90 6, 15, 30, 60, 90 6, 15, 30, 60, 90 6, 15, 30, 60, 90 6, 15, 30, 60, 90	Profundidad
Y8 (8 bits)	1280 x 720 848 x 480 640 x 480 640 x 360 480 x 270 424 x 240	6, 15, 30 6, 15, 30, 60, 90 6, 15, 30, 60, 90 6, 15, 30, 60, 90 6, 15, 30, 60, 90 6, 15, 30, 60, 90	Luminancia del generador de imágenes izquierdo y derecho
YUY2 (16 bits)	1920 x 1080 1280 x 720 960 x 540 848 x 480 640 x 480 640 x 360 424 x 240 320 x 240 320 x 180	6, 15, 30 6, 15, 30 6, 15, 30, 60, 90 6, 15, 30, 60, 90 6, 15, 30, 60, 90 6, 15, 30, 60, 90 6, 15, 30, 60, 90 6, 30, 60 6, 30, 60	Transmisión de color desde la cámara RGB
Calibración (24 bits)	1280 x 800	15, 25	D435i
Autocalibración RealSense	256 x 144	90	Formato de Autocalibración RealSense Serie D400

La captura de imágenes se llevó a cabo mediante el SDK Intel RealSense 2.0 usando la biblioteca pyrealsense2 en Python, lo cual hace posible la transmisión simultánea de la información visual RGB y mapas de profundidad con resoluciones adecuadas para el procesamiento en tiempo real. Para este caso se utilizó la resolución 640 x 480 a 30 FPS, la cual es muy utilizada en aplicaciones robóticas.

La sincronización de los flujos se realiza usando el pipeline proporcionado por el SDK. Esto permite capturar los frames de color y profundidad al mismo tiempo, mediante la función wait_for_frames(), el cual devuelve un conjunto de imágenes que están sincronizadas y corresponden

al mismo instante de captura. De esta forma, es posible obtener un mapa de profundidad para cada imagen RGB, permitiendo que el sistema pueda estimar la distancia de los objetos detectados.

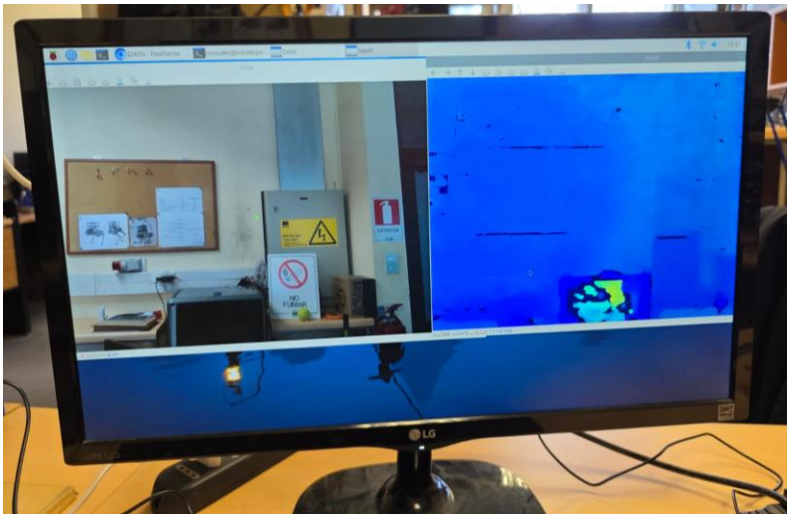


Figura 3.6: Sincronización Cámara-Stream

Previo a la aplicación de los algoritmos de detección e interpretación, las imágenes obtenidas son sometidas a una etapa de preprocesamiento, cuyo objetivo es favorecer la extracción de características visuales relevantes y lograr mejorar la robustez del sistema [2].

Una de las principales operaciones que se llevan a cabo es la conversión del espacio de color, donde las imágenes del espacio BGR (Blue-Green-Red) se transforman al formato HSV, arrastrando como consecuencia una mejor separación de la información del color y, por lo tanto, permitiendo facilitar la detección de regiones de interés asociadas a señales industriales. Asimismo, en el caso que corresponda, se aplican filtros básicos a las imágenes, con la finalidad de disminuir el ruido presente en éstas y de proporcionar una mejor calidad de visualización en las regiones de interés, para lograr una detección más estable frente a condiciones variables de iluminación.

Este procedimiento se lleva a cabo en la unidad de cómputo del sistema, mediante la función cvtColor ubicada en la biblioteca de OpenCV, la cual permite aplicar diferentes transformaciones a las imágenes que se capturan.

Dado la orientación física de la cámara, las imágenes capturadas se encuentran dispuestas de forma invertida en su orientación, la cual es corregida dentro de esta etapa de preprocesamiento. Mediante este ajuste, se asegura que la información visual obtenida sea interpretada apropiadamente por el procesamiento posterior y mantenga la coherencia espacial del entorno observado.

3.5. Detección de características visuales

La detección primaria de las señaléticas industriales se lleva a cabo mediante técnicas de segmentación de color, usando el espacio de color HSV (Hue, Saturation, Value). El cual permite una separación más efectiva de la información del color en relación con la iluminación, ayudando a facilitar la detección de colores característicos presentes en señales de seguridad industrial.

A partir de la imagen convertida en HSV, se fijan unos determinados umbrales de color que permiten segmentar las áreas de la imagen asociadas con aquellos colores de interés, y que son, normalmente, los utilizados en señaléticas de seguridad industrial, con los cuales se obtienen unas mascaras binarias que destacan las áreas potencialmente relevantes para el posterior análisis. Por lo que, la segmentación por color corresponde a una primera etapa de filtrado, con el fin de reducir el volumen de información para procesar y ajustar el análisis a las zonas con mayor probabilidad de corresponder a señales industriales.

El uso del espacio de color HSV facilita el poder aislar los colores predominantes de las señales industriales, permitiendo que su detección sea más efectiva mediante el uso de umbrales definidos para cada color. Esto es posible implementando la función inRange de la biblioteca de OpenCV, la cual genera la máscara binaria a partir de los limites inferior y superior definidos para cada color. En particular, el color rojo requiere dos rangos de valores en el canal Hue, ya que en el modelo HSV este color se encuentra dividido en dos partes en los extremos del espectro de colores.

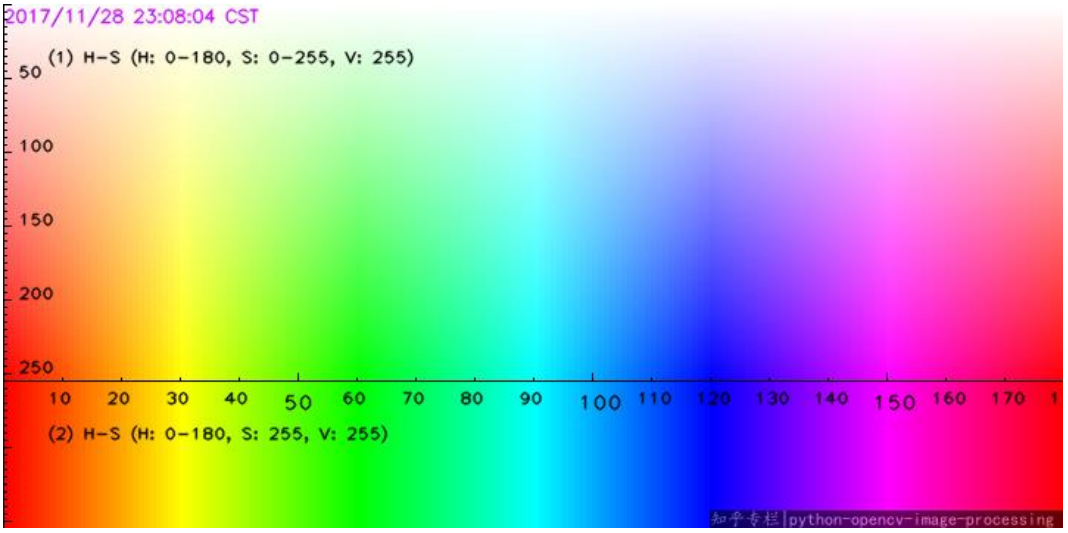


Figura 3.7: Vista de los componentes HSV

Tabla 3.3: Rangos HSV con los colores utilizados

Color detectado	Hue	Saturation	Value
Rojo 1	0-15	60-255	60-255
Rojo 2	165-179	60-255	6-255
Amarillo	18-40	70-255	70-255
Azul	80-145	60-255	60-255
Verde	35-95	50-255	50-255

Una vez realizado la segmentación de las regiones de interés mediante el color, se continua con la detección de contornos, donde se busca identificar los límites de los objetos representados en la imagen. Al conocer los contornos es posible adquirir la información geométrica de interés, como, por ejemplo: el área, perímetro y aproximación de formas [14]. Esto se realiza usando la función findContours perteneciente a la biblioteca de OpenCV, donde es posible extraer las coordenadas de los puntos de forman cada contorno presente en la imagen, formándose una lista de contornos correspondientes a todas las regiones detectadas dentro de la máscara binaria obtenida durante la etapa de segmentación.

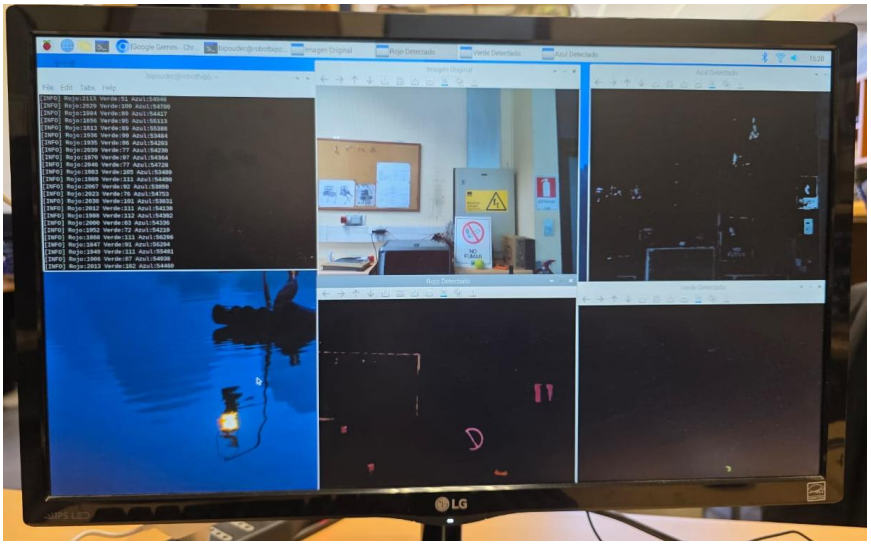


Figura 3.8: Detección de colores rojo, azul y verde.

Mediante este análisis se realiza una clasificación básica basada en características geométricas simples, tales como triángulos, círculos y rectángulos, facilitando de esta manera una identificación adecuada y efectiva, gracias a que son frecuentemente conocidas en el uso de señales de advertencia,

obligación o prohibición. Esta aproximación de contorno se realiza por medio del algoritmo de Douglas-Peucker, implementado en OpenCV, usando la función approxPolyDP, donde a partir de la cantidad de vértices obtenidos es posible estimar la forma geométrica del objeto detectado. Siendo, 3 vértices un triángulo, 4 vértices un rectángulo o cuadrado y mayor a 4 vértices una forma circular aproximada. De esta manera la combinación de ambos tipos de información ayuda a reducir los falsos positivos y a mejorar la fiabilidad del sistema de detección.

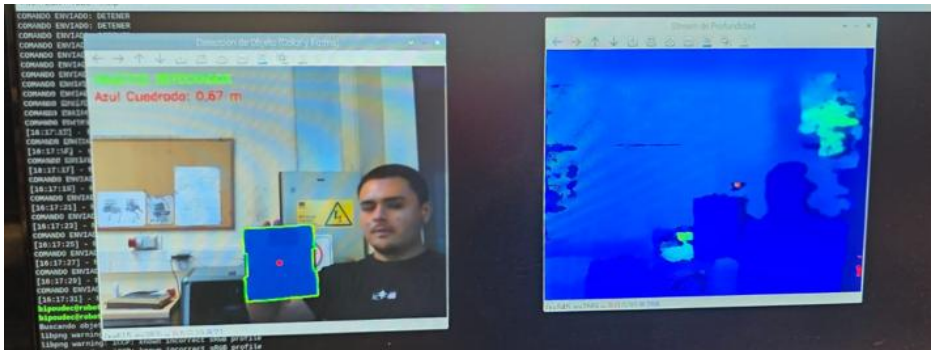


Figura 3.9: Detección de forma cuadrada y de color azul

Con el fin de lograr una mayor robustez al sistema, se incluye una etapa de selección de regiones de interés que consiste en filtrar objetos detectados teniendo en cuenta unos criterios predefinidos, de los cuales el tamaño de las regiones es uno de los más importantes, permitiendo descartar aquellas consideradas demasiado pequeñas o muy grandes para una señalética industrial válida. Asimismo, se implementan mecanismos de eliminación de ruido, situados para detectar y eliminar espurias provocadas por variaciones de iluminación, reflejos o los elementos del entorno que no son de importancia para la navegación del robot bípedo.

Este proceso de filtrado permite centrar la atención exclusivamente en regiones de la imagen con alta probabilidad de ser señaléticas industriales relevantes, aumentando la eficiencia del sistema a la vez que se disminuye la carga computacional. En consecuencia, el uso de estas técnicas de detección de color, análisis de formas y selección de regiones de interés nos facilita identificar regiones de la imagen asociadas a señaléticas industriales, siendo la base a partir de la cual se procede con las etapas de interpretación y toma de decisiones del sistema de visión artificial.

3.6. Estimación de la distancia mediante la profundidad

Junto con la información de color, la cámara del sistema proporciona un mapa de profundidad, para estimar la distancia que existe entre el robot y cualquier objeto del entorno [6]. Este mapa realiza una asignación donde cada pixel de la imagen tiene un valor, el cual representa la distancia entre la cámara y el punto asociado en el espacio. Cuando se ha detectado una posible señalética industrial mediante el análisis de color y de forma, se identifica la región de la imagen asociada a dicha detección. Este proceso es posible mediante las funciones proporcionadas por el SDK de la cámara, logrando obtener directamente la distancia asociada al pixel seleccionado, mediante el algoritmo get_distance(x, y), siendo (x, y) la posición del objeto identificado dentro de la escena. A continuación, se obtiene el valor de profundidad para unos o varios pixeles de esa región, lo que posibilita realizar la estimación de la distancia entre el robot y la señal detectada. De este modo, se puede relacionar la información visual obtenida de la imagen RGB y la información espacial proporcionada por el sensor de profundidad.

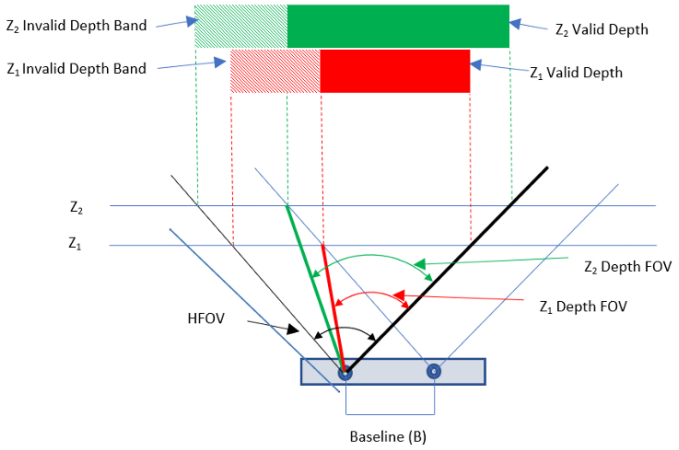


Figura 3.10: Ilustración del Campo de Visión de Profundidad a Mapa de Profundidad

La cámara de profundidad utilizada tiene un rango operacional restringido, dentro del cual las mediciones son confiables; en condiciones normales de operación, el sistema puede realizar estimaciones de distancia con éxito hasta aproximadamente 3 metros, dependiendo de factores como la iluminación del ambiente y las características de las superficies observadas. A su vez, existe una distancia mínima confiable de operación, cercana a los 0.3 metros. Por este motivo, el sistema considera principalmente las mediciones dentro del rango útil del sensor para que las estimaciones de distancia puedan ser consideradas confiables.

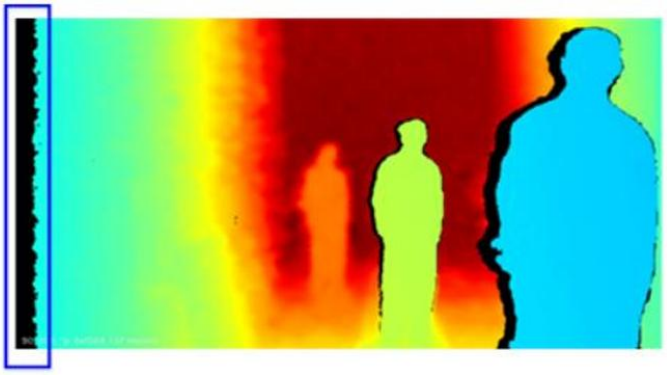


Figura 3.11: Banda de profundidad de la izquierda no válida

Los datos de profundidad generados por el procesador de visión D4 utilizan el sensor izquierdo como referencia para el algoritmo de coincidencia estereoscópica, lo que genera una región no superpuesta en el campo de visión de la cámara. Esta región ubicada en el borde izquierdo del fotograma no contiene datos, por lo que la banda de datos de profundidad no validos disminuye a medida que aumenta la distancia entre la cámara y la escena.

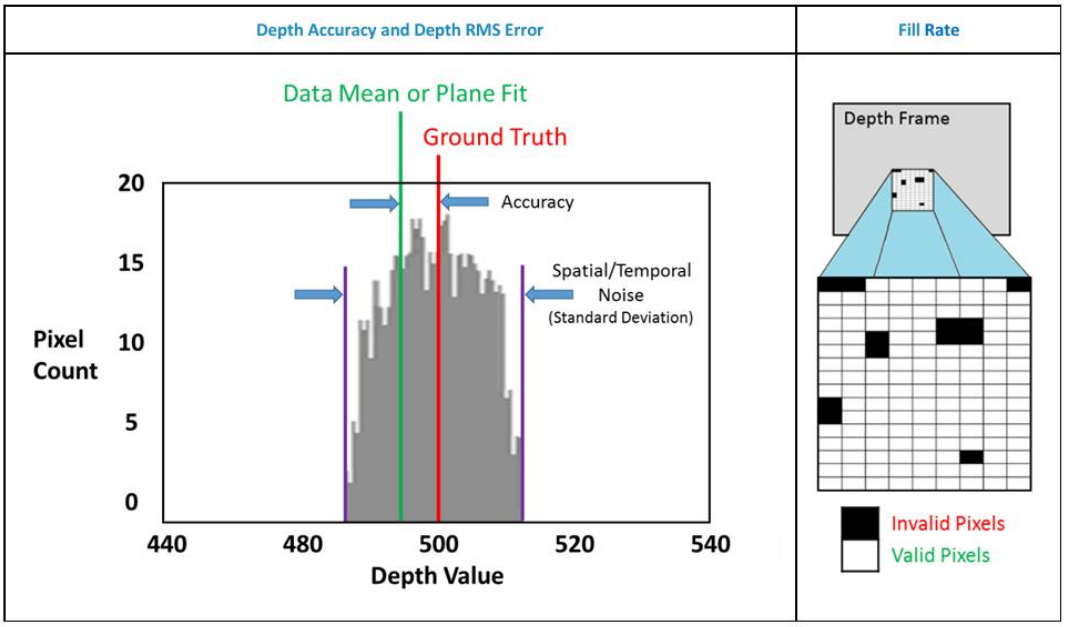


Figura 3.12: Ilustración de la Métrica de Calidad de Profundidad

La imagen nos da a entender cómo se evalúa la calidad de una cámara de profundidad usando un histograma de valores medidos. En el gráfico de la izquierda, la línea roja (Ground Truth) representa la distancia real al objeto, en tanto que la línea verde (Data Mean) representa el promedio

de las mediciones realizadas por la cámara, donde la distancia entre ambas indica la exactitud (accuracy) de la medición. Siendo las barras grises la distribución de los valores medidos por los pixeles; por lo que si se encuentran muy dispersas quiere decir que existe ruido en la medición, lo cual se mide mediante el RMS o desviación estándar. Por otro lado, la parte derecha muestra el Fill Rate, lo cual indica la cantidad de pixeles que tienen una medición de profundidad válida dentro de la imagen, siendo los pixeles blancos los datos válidos y los negros los inválidos. Un alto Fill Rate implica que la mayoría de los pixeles tienen información útil, lo cual mejora la calidad de la detección y el análisis de objetos en la escena.

Tabla 3.4: Especificaciones de la calidad de profundidad de la cámara

Métrica	**D435i <= 2 metros y 80 % de Retorno de la Inversión (ROI), Resolución HD**
Precisión Z o error absoluto	± 2%
Fill Rate	≥ 99 %
Error RMS o ruido espacial	≤ 2%
Ruido Temporal	≤ 1%
Tiempo de vida	5 años

Es vital importancia para el comportamiento del robot el poder obtener la estimación de la distancia, dado que permite evaluar la cercanía de las señales detectadas en el entorno. En un contexto de navegación, hay que tener en cuenta que no todas las señales detectadas exigen una respuesta inmediata, por ejemplo, una señal ubicada a una gran distancia puede ser considerada como información anticipada del entorno, por otro lado, que una señal a corta distancia puede requerir una reacción más inmediata por parte del robot bípedo.

Tabla 3.5: Características del Sistema de Profundidad

Características	**Datos**
Tecnología de profundidad	Estereoscópica
Rango ideal	0.3m – 3m
Distancia mínima	~28 cm
Precisión	~2% a 2 m
Velocidad	Hasta 90 fps

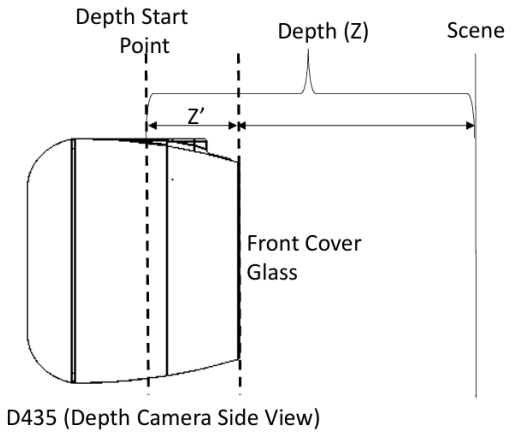


Figura 3.13: Referencia del Punto de Inicio de Profundidad de la Cámara

El punto de inicio de profundidad o la referencia cero del terreno se puede describir como el punto o plano de inicio donde la profundidad es 0. Para la cámara utilizada, este punto se toma como referencia desde la parte frontal del cristal de la cubierta de la cámara.

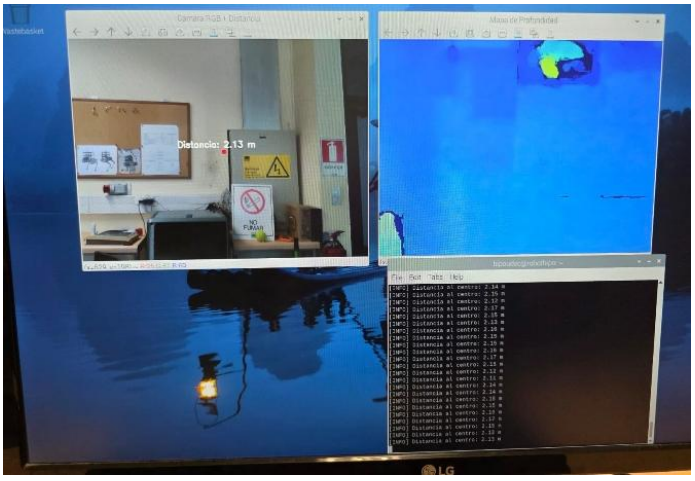


Figura 3.14: Distancia de la muralla con respecto a la cámara

En relación a la interpretación de los colores presentes en el mapa de profundidad, los colores más cálidos tienden a representar a objetos cercanos a la cámara, en cambio los colores más fríos representan los objetos ubicados a una mayor distancia. Es por ello que la información acerca de la profundidad permite al sistema no solo percibir la existencia de las señales industriales, sino que lo ayuda a situarse espacialmente, facilitando así la toma de decisiones de la navegación de una forma más apta para una locomoción segura del robot.

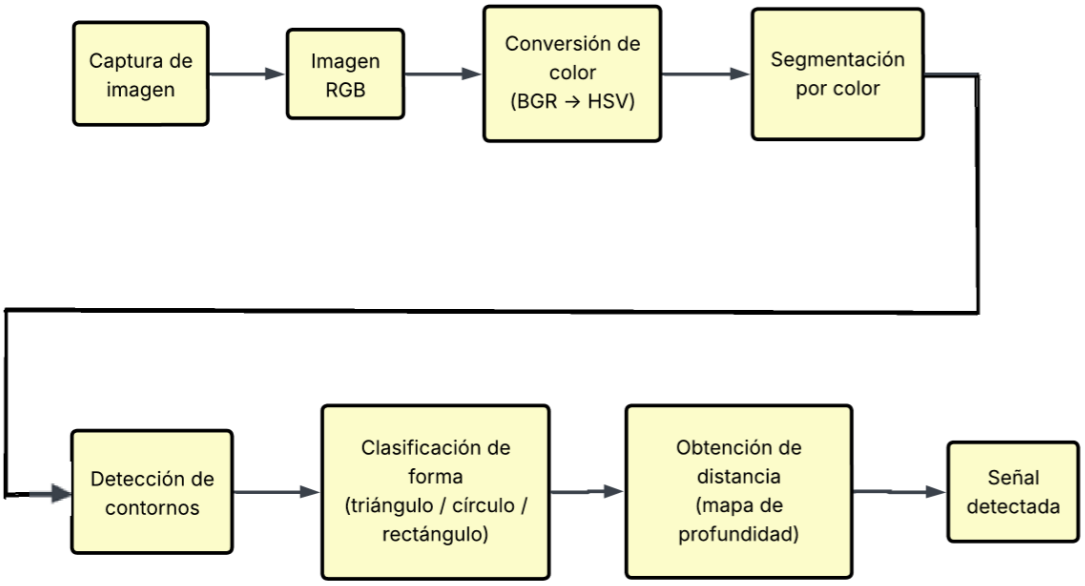


Figura 3.15: Diagrama del Pipeline del algoritmo de visión artificial

3.7. Tipo de Señales Seleccionadas

Para desarrollar el sistema de visión artificial, se seleccionaron varios tipos de señales de tráfico industrial que siguen las normas de la ISO 7010. Para la selección de estas se consideraron los siguientes aspectos: frecuencia de uso en entornos industriales, fácil identificación visual con visión artificial, una clara distinción por medio de formas y colores [16]. Estos factores ayudan a que las señales se puedan detectar de manera más segura con técnicas de procesamiento de imágenes.

El sistema puede entender diferentes tipos de señales, que se dividen en estas categorías:

3.7.1. Señales de advertencia

Estas señales indican que puede haber un peligro en el lugar donde se trabaja. Por lo general, son triángulos de color amarillo con borde negro. Cuando el sistema se encuentre con este tipo de señal, puede hacer que el robot reduzca su velocidad o tenga más cuidado durante su desplazamiento.

3.7.2. Señales de prohibición

Las señales de prohibición nos indican que acciones no deben realizarse dentro de un determinado lugar. Normalmente, estas señales se identifican con un círculo rojo con un fondo blanco y tienen un símbolo negro dentro. Cuando el sistema esté frente a este tipo de señal, el robot debe detenerse.

3.7.3. Señales de obligación

Estas señales indican que acciones debemos realizar para estar seguros dentro de nuestro entorno laboral. Suelen ser circulares y de color azul. Si se detectan estas señales, el sistema sabe que si debe transitar de manera segura por ese sector. Para este caso, debido a que el robot transita por ahí y no una persona, simplemente se le asocia la acción de continuar con precaución.

3.7.4. Señales de seguridad o emergencia

Estas señales indican donde están los elementos de seguridad o las rutas de evacuación. Suelen ser rectangulares o cuadradas y tiene un color verde. La señal relacionada con la seguridad contra incendios, se caracteriza por tener forma cuadrada o rectangular y ser de color rojo. Estas señales de seguridad o emergencia nos ayudan a encontrar zonas seguras o rutas permitidas para continuar con la navegación del robot sin dificultades.

La interpretación automática de estas señales permite que el robot pueda anticipar situaciones de riesgo, ajustar su comportamiento durante la navegación o responder de manera adecuada a las condiciones del entorno industrial.

3.8. Generación de decisiones de navegación

Una vez que se han detectado las señales existentes en el entorno y una vez calculada la ausencia de obstáculos mediante el análisis del mapa de profundidad, el sistema genera las decisiones de navegación que determinan la forma en la que el robot debería comportarse durante su desplazamiento, en relación al tipo de señal identificada [2].

Las decisiones se fundamentan en un conjunto de instrucciones que relacionan el tipo de la señal detectada con una acción determinada, permitiendo de este modo que el sistema actúe de forma coherente ante las diversas situaciones que se pueden presentar en el entorno.

Tabla 3.6: Relación entre el tipo de señal y el accionar del robot

Tipo de señal	**Forma**	**Color**	**Acción del robot**
Advertencia	Triangulo	Amarillo	Reducir velocidad
Prohibición	Circulo	Rojo	Detenerse
Obligación	Circulo	Azul	Continuar con precaución
Seguridad o emergencia	Rectángulo o cuadrado	Verde	Continuar navegación
Seguridad contra incendios	Rectángulo o cuadrado	Rojo	Evitar zona y mantener acceso libre

También se le añade la detección de obstáculos, ante los cuales el robot debe cambiar de dirección, realizando un giro hacia la izquierda o derecha, según corresponda.

Este conjunto de instrucciones permite que la información visual recibida sea más contextualizada y segura, al así tomar acciones concretas durante la navegación [3].

Las decisiones son generadas por el sistema mediante el algoritmo de visión artificial y mostradas como salida del sistema, siendo mostradas en pantalla, donde se presentan las órdenes de navegación del robot bípedo, como, por ejemplo: seguir adelante, detenerse, reducir velocidad o cambiar la dirección de movimiento.

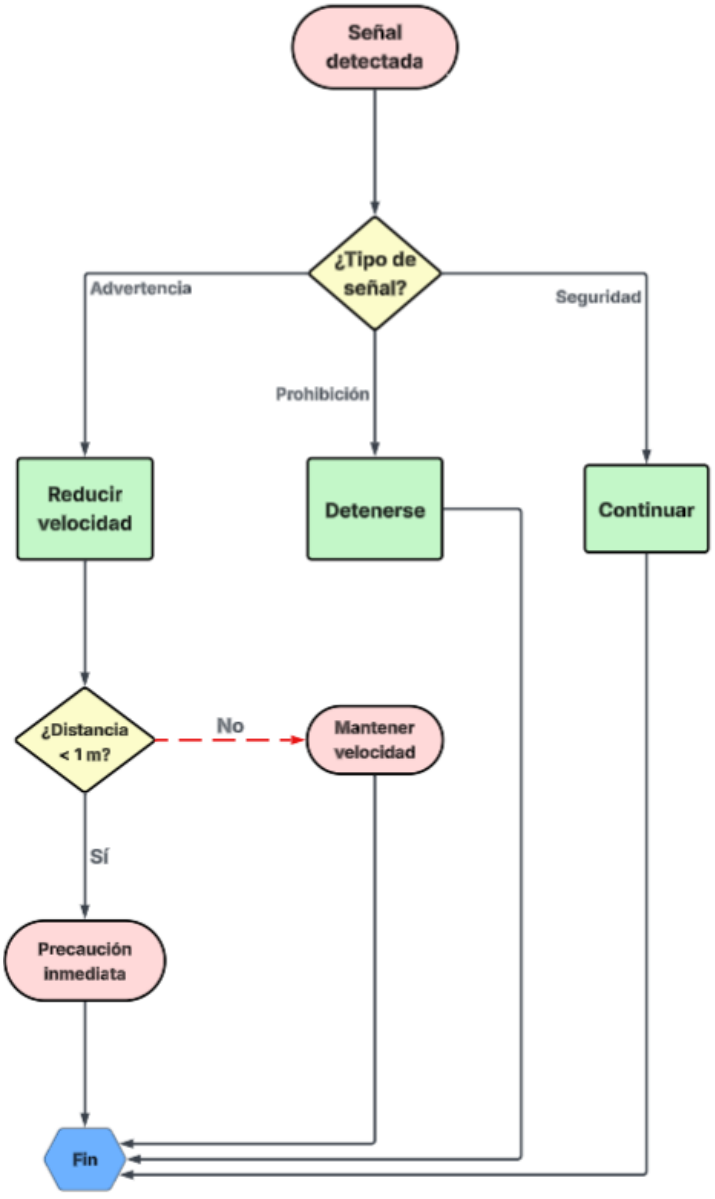


Figura 3.16: Diagrama de flujo de decisión del robot

4. Pruebas experimentales y resultados

4.1. Objetivo de las pruebas

El objetivo de las pruebas es evaluar el desempeño del sistema de visión artificial, por lo que se desarrollaron una serie de pruebas experimentales enfocadas a verificar si el robot estaba capacitado para detectar las señaléticas industriales seleccionadas previamente y tomar acción frente a ellas. Estas pruebas se diseñaron con la finalidad de analizar el comportamiento del sistema frente a distintos tipos de señales, comprendiendo tanto la detección visual de la señal como la estimación de la distancia frente a ellas.

El enfoque de estas pruebas se centró en los siguientes aspectos:

- La capacidad del sistema para reconocer las señales que se seleccionaron.
- La precisión con la que el sistema calcula la distancia entre el robot y la señal.
- La capacidad de generar ordenes de movimiento en relación a la señal detectada.
- Comportamiento del sistema frente a diferentes condiciones de observación del entorno.

Por medio de estas evaluaciones es posible apreciar si el sistema funciona correctamente y considerar su potencial ocupación en entornos industriales donde la interpretación automática de la señalización visual puede ayudar a mejorar la seguridad y navegación tanto de robots como del personal de trabajo.

4.2. Configuración experimental

Se utilizaron pruebas experimentales en un robot bípedo equipado con una cámara Intel RealSense D435i, donde ésta permite tomar imágenes RGB y mapas de profundidad del entorno, por medio de tecnología de visión estéreo [2]. La información que se obtuvo de la cámara se procesó en una Raspberry Pi 5 [18], la cual es la unidad principal de procesamiento, y que es operada mediante el sistema operativo de Raspberry Pi OS. Esto proporcionó el ambiente necesario para implementar y ejecutar el software desarrollado.

El sistema de visión artificial fue ejecutado utilizando el lenguaje de programación de Python, empleando múltiples bibliotecas relacionadas con el procesamiento de imágenes y manejo de datos. Para comunicarse con la cámara y obtener estos datos, se utilizó el SDK 2.0, específicamente la biblioteca librealsense, con la cual es posible acceder a los flujos de imagen RGB y a la información de profundidad que genera el sensor. Asimismo, se emplearon bibliotecas como OpenCV para procesar imágenes y NumPy para manejar datos y matrices numéricas [4], [5].

4.3. Escenarios de pruebas

Se realizaron pruebas del sistema en un entorno controlado que simula las condiciones de un lugar de trabajo industrial básico, donde las señales visuales cumplen un rol importante para la seguridad y orientación de los trabajadores. Para la elaboración de estas pruebas, se colocaron varias señales de seguridad industrial en el área de prueba, situándolas a diferentes distancias y posiciones respecto al robot, con el objetivo de poder evaluar si el sistema está capacitado para detectar y reconocer las señales identificadas dentro del campo visual de la cámara.

En el transcurso de las pruebas, el robot fue orientado hacia cada una de las señales, donde se analizó si el sistema de visión artificial era capaz de identificarlas y calcular la distancia a la que estaban, mediante el uso de la información de profundidad obtenida.

Este escenario fue llevado a cabo en el laboratorio de Control Digital Aplicado (LCDA), ubicado en el segundo piso del edificio Tecnológico Mecánico de la Universidad de Concepción, donde fue posible simular las condiciones básicas de operación.

Se establecieron los siguientes escenarios:

4.3.1. Escenario 1: Detección a corta a distancia

Las señaléticas se distribuyeron hasta una distancia de aproximadamente de 1 metro del robot, permitiendo evaluar si el sistema era capaz de detectar las señales cercanas y calcular correctamente la distancia usando el mapa de profundidad de la cámara.

4.3.2. Escenario 2: Detección a distancia media

En este caso las señaléticas fueron situadas hasta aproximadamente 2 metros de distancia, con la finalidad de evaluar el comportamiento del sistema frente a las señales a esa distancia, dentro del rango operativo de la cámara.

4.3.3. Escenario 3: detección a mayor distancia

Las señales se colocaron a una distancia de hasta los 3 metros aproximadamente, siendo la distancia máxima a la que el sistema debería operar de manera confiable y efectiva.

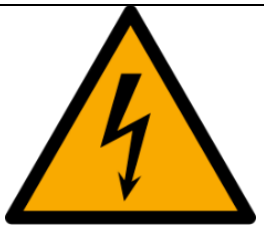
4.3.4. Escenario 4: Presencia de obstáculos

Además, se llevaron a cabo pruebas añadiendo objetos u obstáculos en el campo de visión de la cámara, para evaluar si el sistema era capaz de identificar estos elementos en el entorno y poder tomar decisiones adecuadas frente a este tipo de situación.

4.4. Señaléticas utilizadas bajo la norma ISO 7010

Con la finalidad de poder evaluar el funcionamiento del sistema de visión artificial se realizó una selección de señalizaciones industriales, las cuales están basadas en la norma ISO 7010 [16]. Estas señaléticas fueron escogidas a causa de su frecuente uso en entornos industriales y a sus características visuales definidas claramente, como se muestran a continuación mediante la siguiente tabla:

Tabla 4.1: Descripción de las señaléticas utilizadas por la norma ISO 7010

Tipo de señal	**Numero norma ISO**	**Descripción**	**Forma**	**Color**	**Imagen**
Advertencia	W012	Peligro – Alta tensión	Triangular	Amarillo	
Advertencia	W011	Piso resbaladizo	Triangular	Amarillo	
Advertencia	W021	Material inflamable	Triangular	Amarillo	
Advertencia	W003	Material radiactivo	Triangular	Amarillo	
Prohibición	P004	Prohibido el paso	Circular	Rojo	

Obligación	M003	Uso de protección auditiva	Circular	Azul	
Obligación	M014	Uso de obligatorio de casco	Circular	Azul	
Equipo contra incendios	F001	Extintor	Cuadrado	Rojo	
Seguridad	E001	Salida de evacuación	Cuadrado	Verde	
Seguridad	E003	Primeros auxilios	Cuadrado	Verde	
Seguridad	E005	Flecha direccional	Cuadrado	Verde	

4.5. Metodología de evaluación

En las pruebas la cámara Intel RealSense D435i capturó imágenes que fueron procesadas por el sistema de visión artificial. De esta manera, fue posible identificar y estimar cuán lejos se situaban las señales por medio del mapa de profundidad y gracias a esta información, el sistema generó una acción en relación al tipo de señal detectada. Luego estos resultados fueron registrados con el fin de analizar el comportamiento del sistema.

Considerando las características del sensor de profundidad de la cámara, se definieron distintos rangos de prueba dentro del rango operativo confiable del sensor. Dichas pruebas fueron realizadas desde los 0.3 m hasta los 3 m de distancia aproximadamente. Adicionalmente se consideraron los siguientes criterios:

- Detección de la señalética: el sistema debe identificar correctamente las señales que se encuentran dentro del campo visual de la cámara.
- Estimación de distancia: el sistema debe estimar la distancia aproximada entre el robot y la señalética detectada, utilizando la información del mapa de profundidad.
- Generación de decisiones de navegación: se verifica que el sistema tome la acción correcta según el tipo de señal detectada y la distancia estimada.
- Estabilidad de detección: se evalúa si el sistema detecta señales de manera consistente durante la ejecución continua del algoritmo.

Para poder utilizar las señaléticas seleccionadas, se imprimieron considerando un tamaño estándar de 20x20 cm, ya que varían desde 0.5 cm a 70 cm, dependiendo del lugar, siendo las medidas más frecuentes 15x15 cm, 30x15 cm o 40x40 cm. Luego para darles una mayor firmeza al momento de usarlas se les añadió cartón piedra y mediante cinta adhesiva se fijaron en un lienzo blanco donde se ubicó a distintas distancias del robot.

Finalmente, para el desarrollo de cada una de estas pruebas se consideró el movimiento típico de un robot bípedo, es decir, el desplazamiento que se produce cada vez que el robot simula el avance de cada pierna al realizar la acción de movimiento, como si estuviera caminando.

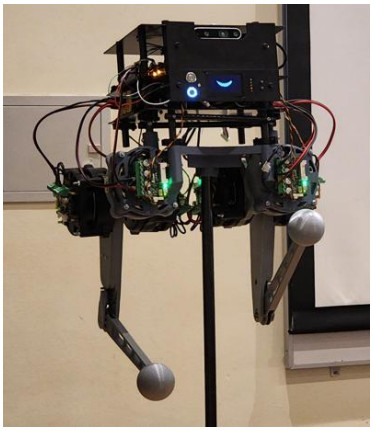


Figura 4.1: Caminata bípeda del robot

4.6. Resultados obtenidos

En esta sección, se presentan los resultados que se obtuvieron al utilizar el sistema de visión artificial que se ha desarrollado, los cuales permitieron evaluar cómo el sistema detecta las señaléticas industriales, y como estima la distancia a los objetos detectados y genera decisiones de navegación basándose en la información visual que obtiene.

Es importante tener en cuenta que las decisiones generadas por el sistema corresponden a instrucciones u ordenes calculadas por el algoritmo de visión artificial, las cuales se muestran como salida del sistema cuando se ejecuta el programa, es decir, no existe una conexión directa con el sistema de control de locomoción del robot para que en relación a la información visual que le brindamos realice los movimientos requeridos como reducir la velocidad, esto es, que las piernas se muevan más lento, que las piernas dejen de moverse o frente a un obstáculo se mueva en alguna dirección para encontrar una vía de escape. Por lo tanto, estas decisiones propuestas por el sistema no resultan en movimientos ejecutados físicamente por el robot.

Para cada señal evaluada se registró la distancia aproximada a la que fue detectada, así como su reconocimiento y la acción que generó el algoritmo de navegación, donde algunos de los resultados obtenidos se pueden muestran en la siguiente tabla:

Tabla 4.2: Resultados obtenidos

Forma detectada	**Color detectado**	**Tipo de señal interpretada**	**Distancia aproximada (m)**	**Detección del sistema**	**Acción generada por el sistema**
Triangulo	Amarillo	Advertencia	0.25	No tan exacta	Ninguna
Triangulo	Amarillo	Advertencia	0.30	Correcta	Reducir velocidad
Triangulo	Amarillo	Advertencia	0.44	Correcta	Reducir velocidad
Círculo	Azul	Obligación	0.31	Correcta	Continuar con precaución
Circulo	Azul	Obligación	0.84	Correcta	Continuar con precaución
Cuadrado	Rojo	Emergencia contra incendios	0.55	Correcta	Evitar zona y mantener acceso libre
Cuadrado	Verde	Seguridad	0.67	Correcta	Continuar desplazamiento
Círculo	Rojo	Prohibición	0.99	Correcta	Detener movimiento
Triangulo	Amarillo	Advertencia	0.98	Correcta	Reducir velocidad
Circulo	Azul	Obligación	1.00	Correcta	Continuar con precaución
Circulo	Rojo	Prohibición	1.18	Correcta	Detener movimiento
Cuadrado	Verde	Seguridad	1.30	Correcta	Continuar desplazamiento
Cuadrado	Rojo	Emergencia contra incendios	1.55	Correcta	Evitar zona y mantener acceso libre
Triangulo	Amarillo	Advertencia	1.87	Correcta	Reducir velocidad
Triangulo	Amarillo	Advertencia	1.96	Correcta	Reducir velocidad
Cuadrado	Verde	Seguridad	2.10	Correcta	Continuar desplazamiento
Círculo	Azul	Obligación	2.33	Correcta	Continuar con precaución
Círculo	Rojo	Prohibición	2.55	Correcta	Detener movimiento
Triangulo	Amarillo	Advertencia	2.69	Correcta	Reducir velocidad

Los resultados muestran que el sistema detectó correctamente las señaléticas industriales dentro del rango de operación del sensor de profundidad de la cámara utilizada, considerado durante el desarrollo del proyecto. A continuación, se presentarán imágenes de las pruebas que se realizaron en cada escenario. Se grabaron múltiples cuadros de video durante estas pruebas y se eligieron algunos ejemplos que ilustran el comportamiento del sistema de visión a diferentes distancias.

4.6.1. Primeras pruebas con una distancia cercana al límite de la cámara 0.3 m

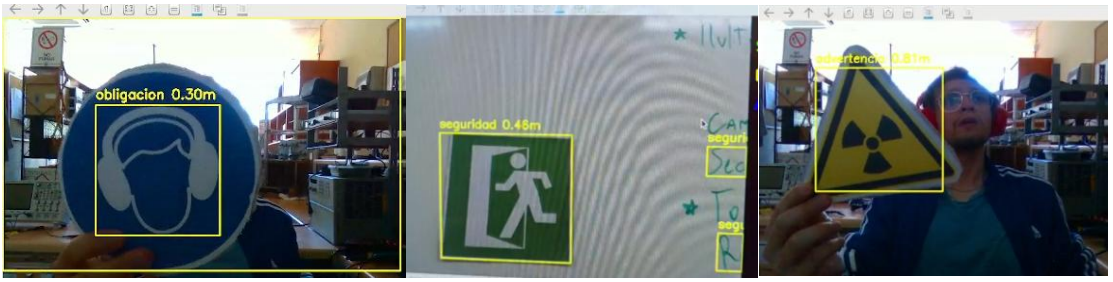


Figura 4.2: Detección de las señaléticas cercanas a 0.3 m

Figura 4.3: Detección de obstáculo cerca

En un inicio se realizaron las pruebas desde un lugar más cercano a la cámara hasta el punto que detectara si había un obstáculo. Producto de ello fue posible entender que en el entorno donde se hicieron estas evaluaciones contenía muchas distracciones, por lo que la cámara detectaba más un objeto a la vez, como el objetivo era que identificara las señales de manera efectiva, se utilizó una manta blanca siendo un color neutral y permitiendo tener una mejor iluminación. A pesar de que se consideró el nombre de la señal al momento de detectarla, resulta mucho más prácticos asignarle una letra característica a cada señal. Siendo:

Tabla 4.3: Asignación de una letra para cada tipo de señal

Letra característica	Tipo de señal
“W”	Advertencia (Warning)
“P”	Prohibición (Prohibition)
“E”	Seguridad (Evacuation)
“F”	Contra Incendio (Fire)
“M”	Obligatorio (Mandatory)

4.6.2. Pruebas hasta 1 metro de distancia

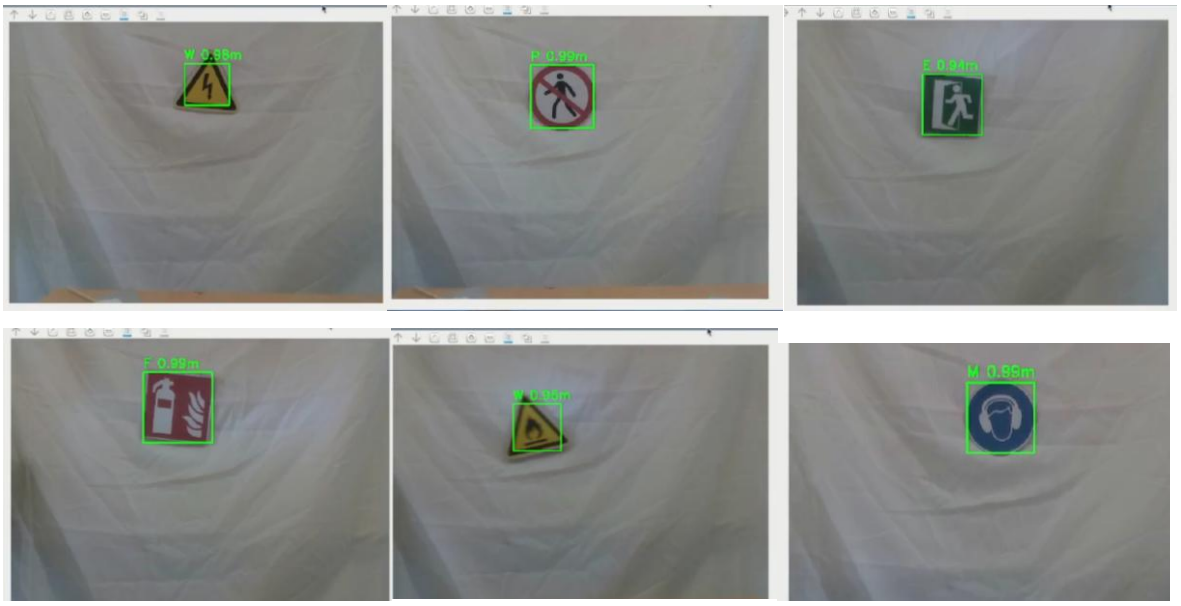


Figura 4.4: Detección de señales hasta 1 metro de distancia

4.6.3. Pruebas de hasta 2 metro de distancia

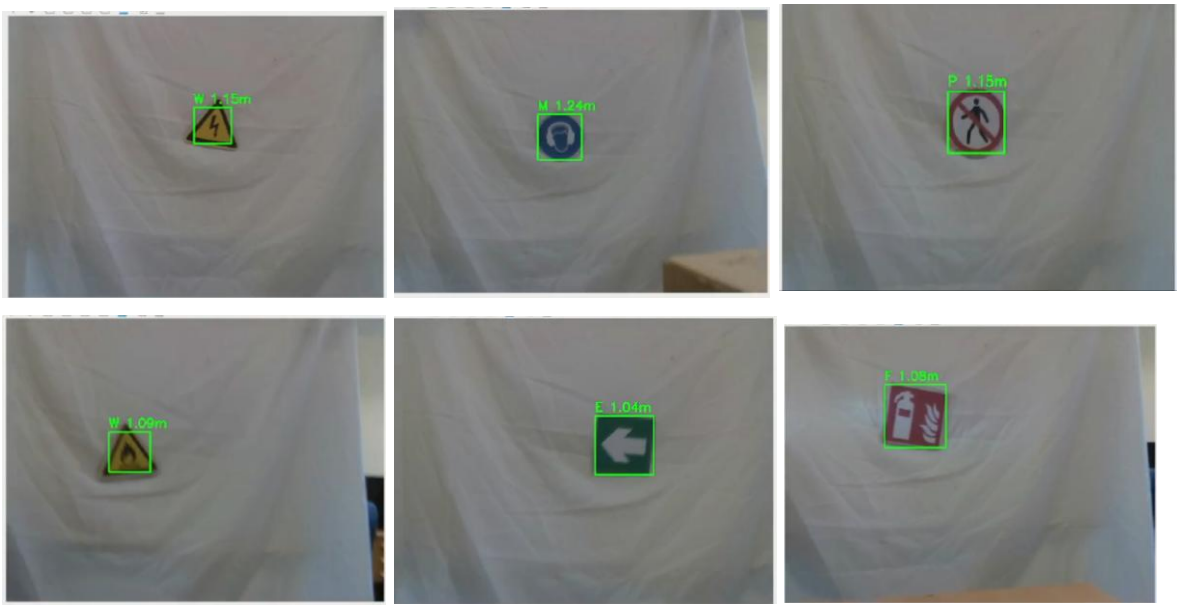


Figura 4.5: Detección de señales hasta 2 metros de distancia

4.6.4. Pruebas cercanas a 3 metros

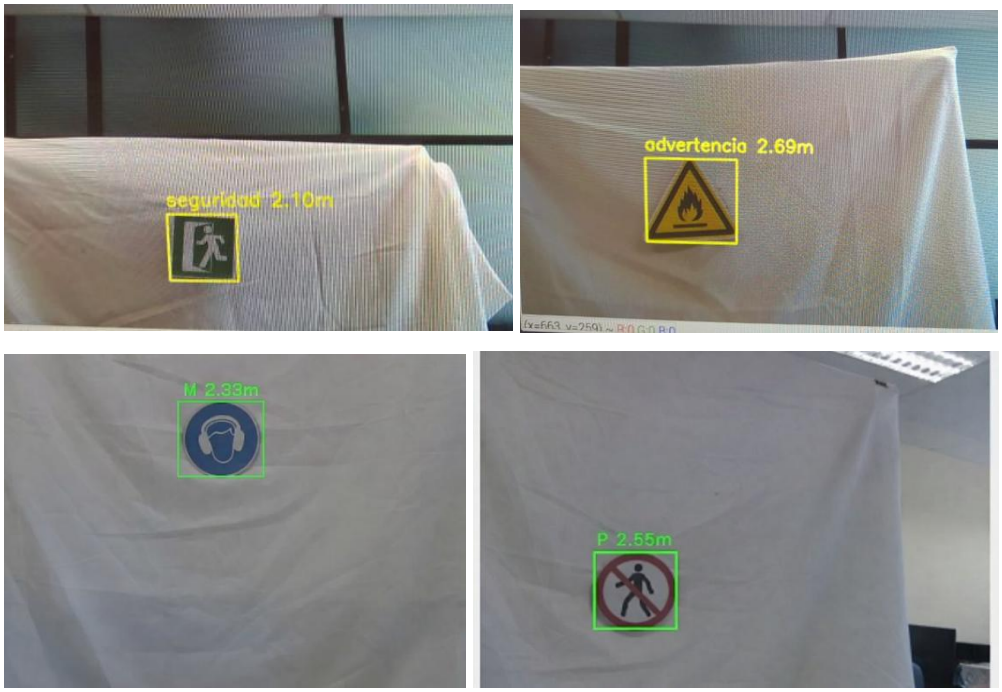


Figura 4.6: Detección de señales cercanas a 3 metros

Algunas fueron realizadas dentro de las primeras pruebas, mientras las otras corresponden a la modalidad mejorada.

4.6.5. Acciones en pantalla

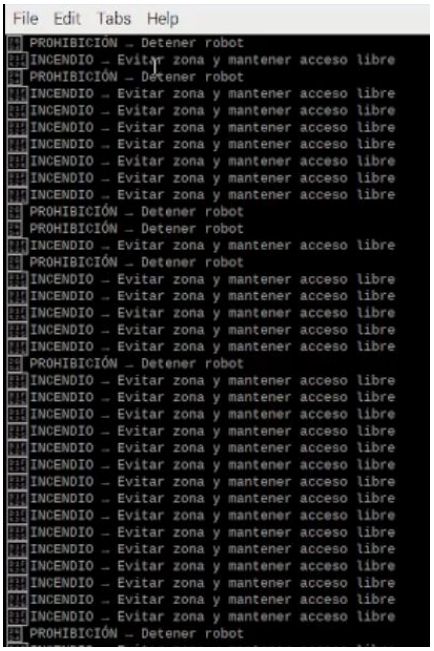


Figura 4.7: Algunas acciones mostradas en pantalla

4.7. Discusión de los resultados

A partir de los resultados obtenidos durante las pruebas experimentales, es posible realizar el análisis general del comportamiento del sistema de visión artificial desarrollado para la detección de las señaléticas industriales.

En general, fue capaz de identificar las características visuales de las señales que se evaluaron, ya que pudo detectar las formas geométricas y los colores predominantes de estas. Además, debido a que el sistema desarrollado no realiza reconocimiento especifico de pictogramas individuales, sino que identifica características visuales asociadas a las señaléticas industriales, fue posible clasificar las señales en diferentes categorías como: advertencia, prohibición, obligación y seguridad.

Los resultados muestran que el sistema logró detectar las señales en la mayoría de los casos evaluados dentro del campo visual de la cámara. La segmentación por color en el espacio HSV ayudó a identificar las regiones de interés vinculadas a los colores principales de la señalización industrial, entre tanto que el análisis de contornos facilitó la identificación de las formas geométricas presentes en las señales [11].

La distancia se estimó usando la información proporcionada por el mapa de profundidad generado por la cámara utilizada en el sistema, donde cabe considerar que a pesar de que las señaléticas fueron identificadas correctamente a estas distancias, el hecho que se generara movimiento al simular la caminata del robot, existía instantes en que se desenfocaba y volvía enfocar la detección de las señales. Esto se evidenció más concretamente a medida que el robot se alejaba de éstas. Esto se puede deber al hecho de que se va reduciendo el tamaño de la señal, por lo que ocupan un menor número de pixeles dentro de la imagen, lo cual puede llegar afectar la segmentación de colores y la detección de contornos. Otro punto a considerar son las variaciones de iluminación, por lo que el rango más óptimo fue de 1 a 2 metros.

Una vez realizado la detección de las señales y estimación de distancia, el sistema toma decisiones para navegar acordes al tipo de señal identificada. Estas decisiones son instrucciones que indican al robot como actuar en el entorno frente a determinadas situaciones. No obstante, estas acciones fueron simuladas únicamente mediante mensajes visualizados en pantalla, por lo que no existe una integración directa con el sistema de control de movimiento del robot.

Durante el desarrollo y las pruebas del sistema, se demostró que el sistema de visión implementado establece una base funcional y útil para que los robots bípedos puedan entender e interpretar su entorno, haciendo posible identificar las señales importantes que podrían afectar el desplazamiento y toma de decisiones a lo largo de la navegación de estos.

5. Conclusiones

5.1. Conclusiones generales

El objetivo de este trabajo fue desarrollar e implementar un sistema de visión capaz de detectar señaléticas industriales, por medio de una cámara Intel RealSense D435i, integrada en un robot bípedo. El sistema se ejecutó en una Raspberry Pi 5, mediante el sistema operativo Raspberry Pi OS, permitiendo el procesamiento de imágenes y la identificación de señales por medio de su forma y color.

Durante el desarrollo de este proyecto, se logró crear un sistema capaz de capturar imágenes del entorno y procesarlas para detectar diferentes tipos de señales y por medio de las pruebas experimentales se evaluó el funcionamiento del sistema frente a diferentes escenarios y distancias, logrando demostrar que es posible identificar estas señales de forma confiable dentro de determinados rangos óptimos para la cámara de profundidad.

Los resultados conseguidos muestran que el uso de sensores de visión, junto con plataformas de procesamiento embebido como la Raspberry Pi 5, es una alternativa altamente viable para dotar a los robots bípedos capacidades básicas de percepción del entorno. Esto es de vital importancia para aplicaciones en las que el robot necesita interactuar con su entorno y tomar decisiones basadas en la información visual que esté disponible.

También se comprobó que la cámara proporciona información de alta calidad, lo que hace que sea más fácil detectar objetos y señaléticas por medio de técnicas de procesamiento de imágenes. Al utilizar el SDK de este sensor, fue posible simplificar la adquisición de datos y la integración con el sistema de procesamiento que se ha implementado.

A pesar de que el sistema desarrollado nos permite detectar señales y mostrar en pantalla las acciones que el robot debería hacer frente a ellas, la integración completa con el sistema de control de locomoción del robot bípedo no fue implementada en esta etapa del proyecto, debido a que esto implica desafíos adicionales, como el control de equilibrio dinámico, planificación de trayectorias y coordinación de múltiples grados de libertad, los cuales pertenecen al subsistema de locomoción. Siendo la principal dificultad realizar una correcta traducción de las decisiones hacia acciones físicas seguras y estables en el robot.

Se abordo el problema de manera modular, separando el sistema de visión del sistema de control. Para asegurar que la información entregada por el sistema de percepción sea confiable antes de integrarla con los mecanismos de locomoción. De esta forma, establece una base sólida para futuras implementaciones en las que el robot pueda reaccionar de manera autónoma frente a distintos estímulos del entorno.

En conclusión, el desarrollo de este sistema de visión contribuye al avance en la integración de sensores y sistemas de percepción en robots bípedos, demostrando la viabilidad de utilizar cámaras de profundidad y plataformas integradas para mejorar las capacidades de navegación e interacción con el entorno de manera más efectiva.

5.2. Trabajo futuro

A partir de los resultados obtenidos a lo largo de este trabajo, se identificaron diversas líneas de crecimiento que podrían ser abordadas en futuras etapas de desarrollo e investigaciones.

Una de las principales mejoras consiste en la integración directa del sistema de detección de señales con el sistema de control de locomoción del robot bípedo, lo cual haría posible que el robot reaccione de forma autónoma ante la detección de determinadas señaléticas, logrando ejecutar acciones tales como, reducir la velocidad, cambiar de dirección o detenerse, de manera efectiva.

Otra línea de desarrollo a considerar está relacionada con la incorporación de algoritmos de visión más avanzados, como podría ser el caso de técnicas basadas en aprendizaje automático o redes neuronales, para clasificar las señales de manera más precisa en entornos más complejos o incluso en condiciones de iluminación variable. De hecho, sería un gran avance el poder abordar temas como el reconocimiento facial, ya que resultaría un gran avance tecnológico en la seguridad que puede brindar el uso de este robot bípedo, logrando reconocer a las personas que tienen acceso a determinados lugares o plantas. De igual modo, la identificación tridimensional de objetos como la de elementos diferentes que no sean necesariamente señales, como por ejemplo puertas, ventanas, fuentes de acceso, entre otros.

También, sería factible tener una mayor variedad de señales, incorporando nuevas categorías de estas que permitan que el robot obtenga una interacción más completa de su entorno e incluso el hecho de poder diferenciar dos señales del mismo tipo, lo cual sería muy útil en fábricas, empresas o incluso en la asistencia, que brindaría el robot al personal tanto del trabajo como visitas presentes en estos lugares.

Finalmente, la cámara Intel RealSense D435i nos da información sobre la profundidad, lo que nos permite desarrollar sistemas de percepción tridimensional más avanzados. Esto mejoraría la detección de obstáculos y la estimación de distancias, lo que a su vez facilita la creación de sistemas de navegación más confiables, seguros y eficientes para estos tipos de robot.

Referencias

[1] R. Szeliski, *Computer Vision: Algorithms and Applications*. London, U.K.: Springer, 2011. Dirección: https://szeliski.org/Book/

[2] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. New York, NY, USA: Pearson, 2018.

[3] S. Prince, *Computer Vision: Models, Learning, and Inference*. Cambridge, U.K.: Cambridge University Press, 2012.

[4] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. Sebastopol, CA, USA: O'Reilly Media, 2008.

[5] S. Kaehler and G. Bradski, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. Sebastopol, CA, USA: O'Reilly Media, 2017.

[6] Intel Corporation, *Intel RealSense Depth Camera D435i Datasheet*, 2025. Dirección: https://www.intelrealsense.com/depth-camera-d435i/

[7] Intel Corporation, *Intel RealSense SDK 2.0 Documentation*, 2024. Dirección: https://dev.intelrealsense.com/docs

[8] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge, U.K.: Cambridge University Press, 2004.

[9] D. Scaramuzza and F. Fraundorfer, "Visual odometry: Part I—The first 30 years and fundamentals," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[10] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Cham, Switzerland: Springer, 2016.

[11] R. Siegwart, I. Nourbakhsh and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.

[12] J. Pratt, J. Carff, S. Drakunov and A. Goswami, "Capture point: A step toward humanoid push recovery," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, Genova, Italy, 2006.

[13] S. Kajita et al., *Humanoid Robotics: A Reference*. Berlin, Germany: Springer, 2014.

[14] S. Thrun, W. Burgard and D. Fox, *Probabilistic Robotics*. Cambridge, MA, USA: MIT Press, 2005.

[15] A. Davison et al., "MonoSLAM: Real-Time Single Camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[16] ISO 7010, *Graphical symbols — Safety colours and safety signs — Registered safety signs*, International Organization for Standardization, 2019.

[17] Raspberry Pi 5 Documentation, Raspberry Pi Foundation, 2024. Dirección: https://www.raspberrypi.com/documentation/

[18] E. Upton and G. Halfacree, *Raspberry Pi User Guide*, 4th ed. Hoboken, NJ, USA: Wiley, 2016.

Anexos A. Códigos elaborados en Python

A.1. Código: Sincronización Cámara-Stream-Profundidad

```
import pyrealsense2 as rs
import numpy as np
import cv2, time

W, H, FPS = 640, 480, 30

pipe = rs.pipeline()
cfg  = rs.config()
cfg.enable_stream(rs.stream.depth, W, H, rs.format.z16, FPS)
cfg.enable_stream(rs.stream.color, W, H, rs.format.bgr8, FPS)
profile = pipe.start(cfg)

align = rs.align(rs.stream.color)
depth_scale = profile.get_device().first_depth_sensor().get_depth_scale()
print("Depth scale (m per unit):", depth_scale)

t0, frames_count = time.time(), 0
try:
    while True:
        frames = pipe.wait_for_frames()
        frames = align.process(frames)

        depth = frames.get_depth_frame()
        color = frames.get_color_frame()
        if not depth or not color:
            continue

        depth_np = np.asanyarray(depth.get_data())
        color_np = np.asanyarray(color.get_data())

        # Volteando verticalmente la imagen de color y la de profundidad
        color_np = cv2.flip(color_np, -1)
        depth_np = cv2.flip(depth_np, -1)
```

```
        # Distancia en el centro
        cx, cy = W//2, H//2
        dist_m = depth_np[cy, cx] * depth_scale
        cv2.circle(color_np, (cx, cy), 4, (0,255,0), -1)
        cv2.putText(color_np, f"{dist_m:.2f} m", (10,30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2)

        # Visualización de la profundidad
        depth_vis = cv2.convertScaleAbs(depth_np, alpha=0.03)
        depth_vis = cv2.applyColorMap(depth_vis, cv2.COLORMAP_JET)

        cv2.imshow("Color", color_np)
        cv2.imshow("Depth", depth_vis)

        if cv2.waitKey(1) == 27:  # ESC para salir
            break
finally:
    pipe.stop()
    cv2.destroyAllWindows()
```

A.2. Código: Detección de Colores

```
import pyrealsense2 as rs
import numpy as np
import cv2

print("Iniciando pipeline RealSense...")

pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
pipeline.start(config)
print("Stream de color iniciado.")

# Definiendo rangos de colores en HSV
# Rojo
lower_red1 = np.array([0, 120, 70])
upper_red1 = np.array([10, 255, 255])
lower_red2 = np.array([170, 120, 70])
upper_red2 = np.array([180, 255, 255])

# Verde
lower_green = np.array([36, 100, 100])
upper_green = np.array([86, 255, 255])

# Azul
lower_blue = np.array([94, 80, 2])
upper_blue = np.array([126, 255, 255])

try:
    while True:
        frames = pipeline.wait_for_frames()
        color_frame = frames.get_color_frame()
        if not color_frame:
            print("[WARN] No se recibió frame de color")
            continue

        color_image = np.asanyarray(color_frame.get_data())
```

```
            # Corrigiendo orientación de la cámara
            color_image = cv2.flip(color_image, 0)   # Para voltear verticalmente
            color_image = cv2.flip(color_image, 1) # Para voltear horizontalmente
            # color_image = cv2.rotate(color_image, cv2.ROTATE_180) #Rotar 180°

            hsv = cv2.cvtColor(color_image, cv2.COLOR_BGR2HSV)

            # Para crear máscaras
            mask_red    =    cv2.inRange(hsv,    lower_red1,    upper_red1)    |
cv2.inRange(hsv, lower_red2, upper_red2)
            mask_green = cv2.inRange(hsv, lower_green, upper_green)
            mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)

            # Para contar los píxeles detectados
            red_count = cv2.countNonZero(mask_red)
            green_count = cv2.countNonZero(mask_green)
            blue_count = cv2.countNonZero(mask_blue)
            print(f"[INFO]          Rojo:{red_count}          Verde:{green_count}
Azul:{blue_count}")

            # Para mostrar resultados
            cv2.imshow("Imagen Original", color_image)
            cv2.imshow("Rojo      Detectado",      cv2.bitwise_and(color_image,
color_image, mask=mask_red))
            cv2.imshow("Verde      Detectado",      cv2.bitwise_and(color_image,
color_image, mask=mask_green))
            cv2.imshow("Azul      Detectado",      cv2.bitwise_and(color_image,
color_image, mask=mask_blue))

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

    except Exception as e:
        print("[ERROR]", e)

    finally:
        pipeline.stop()
        cv2.destroyAllWindows()
```

```
        print("Pipeline detenido. Ventanas cerradas.")
```

A.3. Código: Detección de Formas

```
    import pyrealsense2 as rs
    import numpy as np
    import cv2
    import time

    # Configuración de la cámara
    pipeline = rs.pipeline()
    config = rs.config()

    config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
    config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)

    try:
        profile = pipeline.start(config)
    except Exception as e:
        print(f"Error al iniciar la cámara RealSense: {e}")
        print("Asegurarse de que la cámara esté conectada y que el driver esté
cargado (rs-enumerate-devices).")
        exit()

    depth_sensor = profile.get_device().first_depth_sensor()
    depth_scale = depth_sensor.get_depth_scale()

    # Filtros para mejorar la calidad de la imagen de profundidad
    spatial_filter = rs.spatial_filter()
    temporal_filter = rs.temporal_filter()

    # Definiendo los rangos de colores
    COLOR_RANGES = {
        "Rojo": {
            "lower": np.array([0, 100, 100]),
            "upper": np.array([10, 255, 255]),
        },
        "Azul": {
```

```
            "lower": np.array([100, 150, 50]),
            "upper": np.array([130, 255, 255]),
        },
        "Verde": {
            "lower": np.array([35, 100, 100]),
            "upper": np.array([85, 255, 255]),
        },
    }

    # Objetivo de detección
    PRIORITY_COLORS_FOR_DETECTION = ["Azul", "Rojo", "Verde"]


    # Como funciona la detección
    def get_shape(contour):
        """Determina la forma del contorno basándose en el número de vértices."""
        perimeter = cv2.arcLength(contour, True)
        approx = cv2.approxPolyDP(contour, 0.04 * perimeter, True)
        num_vertices = len(approx)

        if num_vertices == 3:
            return "Triangulo"
        elif num_vertices == 4:
            x, y, w, h = cv2.boundingRect(approx)
            aspect_ratio = float(w)/h
            if 0.9 <= aspect_ratio <= 1.1:
                return "Cuadrado"
            else:
                return "Rectangulo"
        elif num_vertices > 4 and num_vertices < 10:
            return "Poligono"
        elif num_vertices >= 10:
            return "Circulo"
        return "Desconocido"

    def detect_object_by_color_and_shape(hsv_image, color_name, color_range):
        """Detecta un objeto de un color específico y devuelve su centro, área,
forma y contorno."""
        mask = cv2.inRange(hsv_image, color_range["lower"],
```

```
color_range["upper"])

        # Erosion y Dilatacion para eliminar ruido y mejorar los contornos
        kernel = np.ones((5, 5), np.uint8)
        mask = cv2.erode(mask, kernel, iterations=1)
        mask = cv2.dilate(mask, kernel, iterations=1)

        contours,      _      =      cv2.findContours(mask,      cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

        found_objects = []
        if contours:
            for contour in contours:
                area = cv2.contourArea(contour)
                if area > 500: # Área mínima para filtrar objetos pequeños
                    shape = get_shape(contour)
                    M = cv2.moments(contour)
                    if M["m00"] != 0:
                        cX = int(M["m10"] / M["m00"])
                        cY = int(M["m01"] / M["m00"])
                        found_objects.append({
                            "center": (cX, cY),
                            "area": area,
                            "name": color_name,
                            "shape": shape,
                            "contour": contour
                        })
        return found_objects

    # Bucle principal de detección
    print(f"Detectando objetos de colores. Presionar 'q' para salir.")
    try:
        while True:
            frames = pipeline.wait_for_frames()
            depth_frame = frames.get_depth_frame()
            color_frame = frames.get_color_frame()
            if not depth_frame or not color_frame:
                continue
```

```
            depth_image = np.asanyarray(depth_frame.get_data())
            color_image = np.asanyarray(color_frame.get_data())

            color_image_flipped = cv2.flip(color_image, -1)
            depth_image_flipped = cv2.flip(depth_image, -1)

            depth_frame = spatial_filter.process(depth_frame)
            depth_frame = temporal_filter.process(depth_frame)

            hsv_image = cv2.cvtColor(color_image_flipped, cv2.COLOR_BGR2HSV)

            all_detected_objects = []
            for color_name, color_range in COLOR_RANGES.items():
                if color_name in PRIORITY_COLORS_FOR_DETECTION:

all_detected_objects.extend(detect_object_by_color_and_shape(hsv_image,
color_name, color_range))

            # Para dibujar y mostrar los resultados
            if all_detected_objects:
                cv2.putText(color_image_flipped, "OBJETOS DETECTADOS", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2, cv2.LINE_AA)

                for i, obj in enumerate(all_detected_objects):
                    cX, cY = obj["center"]
                    distance = depth_image_flipped[cY, cX] * depth_scale

                    cv2.drawContours(color_image_flipped, [obj["contour"]], 0,
(0, 255, 0), 2)
                    cv2.circle(color_image_flipped, (cX, cY), 5, (0, 0, 255), -
1)

                    info_text = f"{obj['name']} {obj['shape']}: {distance:.2f}
m"
                    cv2.putText(color_image_flipped, info_text, (10, 60 + i *
30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
            else:
                cv2.putText(color_image_flipped, "BUSCANDO OBJETOS...", (10,
30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2, cv2.LINE_AA)
```

```
            # Para stream de profundidad
            depth_colormap                                                    =
cv2.applyColorMap(cv2.convertScaleAbs(depth_image_flipped,          alpha=0.03),
cv2.COLORMAP_JET)

            cv2.imshow("Deteccion de Objeto Color y Forma)", color_image_flipped)
            cv2.imshow("Stream de Profundidad", depth_colormap)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

    finally:
        pipeline.stop()
        cv2.destroyAllWindows()
```

A.4. Código: Detección de Distancia

```
import pyrealsense2 as rs
import numpy as np
import cv2

print("Iniciando pipeline RealSense...")

# Configuración de la cámara
pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)

# Iniciando cámara
profile = pipeline.start(config)
align_to = rs.stream.color
align = rs.align(align_to)

print("Cámara inicializada con color + profundidad.")

try:
    while True:
        # Para capturar frames
        frames = pipeline.wait_for_frames()
        aligned_frames = align.process(frames)

        depth_frame = aligned_frames.get_depth_frame()
        color_frame = aligned_frames.get_color_frame()

        if not depth_frame or not color_frame:
            continue

        # Para convertir a arrays
        depth_image = np.asanyarray(depth_frame.get_data())
        color_image = np.asanyarray(color_frame.get_data())

        # Corrigiendo orientación de la cámara
```

```
        color_image = cv2.rotate(color_image, cv2.ROTATE_180)

        # Para tomar el centro de la imagen
        h, w = depth_image.shape
        centro_x, centro_y = w // 2, h // 2
        dist_centro = depth_frame.get_distance(centro_x, centro_y)

        # Para dibujar un círculo en el centro
        cv2.circle(color_image, (centro_x, centro_y), 5, (0, 0, 255), -1)
        cv2.putText(color_image, f"Distancia: {dist_centro:.2f} m",
                    (centro_x - 100, centro_y - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)

        # Para colorear el mapa de profundidad
        depth_colormap = cv2.applyColorMap(
            cv2.convertScaleAbs(depth_image, alpha=0.03),
            cv2.COLORMAP_JET
        )

        # Para mostrar en ventanas
        cv2.imshow("Cámara RGB + Distancia", color_image)
        cv2.imshow("Mapa de Profundidad", depth_colormap)

        # Para imprimir en consola
        print(f"[INFO] Distancia al centro: {dist_centro:.2f} m")

        # Para salir con tecla "q"
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

finally:
    pipeline.stop()
    cv2.destroyAllWindows()
    print("Pipeline detenido. Ventanas cerradas.")
```

A.5. Código: Detección Obstáculo Cerca

```
import pyrealsense2 as rs
import numpy as np
import cv2

print("Iniciando pipeline RealSense...")

# Configuración de la cámara
pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)

profile = pipeline.start(config)
align_to = rs.stream.color
align = rs.align(align_to)

print("Cámara inicializada con color + profundidad.")

# Parámetro de alerta ante un posible obstáculo
DISTANCIA_ALERTA = 0.30  # metros

try:
    while True:
        # Para capturar frames
        frames = pipeline.wait_for_frames()
        aligned_frames = align.process(frames)

        depth_frame = aligned_frames.get_depth_frame()
        color_frame = aligned_frames.get_color_frame()

        if not depth_frame or not color_frame:
            continue

        # Para convertir a arrays
        depth_image = np.asanyarray(depth_frame.get_data())
        color_image = np.asanyarray(color_frame.get_data())
```

```
            # Corrigiendo orientación de la cámara
            color_image = cv2.rotate(color_image, cv2.ROTATE_180)
            depth_image = cv2.rotate(depth_image, cv2.ROTATE_180)

            # Para medir distancia en el centro
            h, w = depth_image.shape
            centro_x, centro_y = w // 2, h // 2
            dist_centro = depth_frame.get_distance(centro_x, centro_y)

            # Para buscar el punto más cercano en toda la imagen
            depth_np = np.array(depth_image, dtype=np.float32)
            depth_np[depth_np == 0] = np.inf  # ignorar valores inválidos (0)
            min_idx = np.unravel_index(np.argmin(depth_np), depth_np.shape)
            min_y, min_x = min_idx
            dist_min = depth_frame.get_distance(min_x, min_y)

            # Para dibujar rectángulo alrededor del objeto más cercano
            box_size = 30
            x1 = max(0, min_x - box_size)
            y1 = max(0, min_y - box_size)
            x2 = min(w, min_x + box_size)
            y2 = min(h, min_y + box_size)
            cv2.rectangle(color_image, (x1, y1), (x2, y2), (0, 255, 0), 2)

            # Para mostrar las distancias
            cv2.putText(color_image, f"Centro: {dist_centro:.2f} m",
                        (20, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255),
2)
            cv2.putText(color_image, f"Minimo: {dist_min:.2f} m",
                        (20, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

            # Para mostrar alerta visual si algo está demasiado cerca
            if 0 < dist_min < DISTANCIA_ALERTA:
                cv2.putText(color_image, "OBSTACULO CERCA!", (20, 100),
                            cv2.FONT_HERSHEY_DUPLEX, 0.9, (0, 0, 255), 3)
                print(f"[ALERTA] Objeto a {dist_min:.2f} m")

            # Para colorear el mapa de profundidad
```

```
        depth_colormap = cv2.applyColorMap(
            cv2.convertScaleAbs(depth_image, alpha=0.03),
            cv2.COLORMAP_JET
        )

        # Para mostrar en ventanas
        cv2.imshow("Camara RGB + Alerta", color_image)
        cv2.imshow("Mapa de Profundidad", depth_colormap)

        # Para salir con 'q'
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

finally:
    pipeline.stop()
    cv2.destroyAllWindows()
    print("Pipeline detenido. Ventanas cerradas.")
```

A.6. Código Final: Detección de Señales por Norma 7010

```
import cv2
import numpy as np
import pyrealsense2 as rs

# Configuración de la cámara
pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
profile = pipeline.start(config)

depth_scale = profile.get_device().first_depth_sensor().get_depth_scale()
align = rs.align(rs.stream.color)

# Para mostrar en ventana
cv2.namedWindow("Detección Señales ISO", cv2.WINDOW_NORMAL)
cv2.resizeWindow("Detección Señales ISO", 900, 600)

# Determinando los rangos de HSV
COLOR_RANGES = {
    "W": ([18, 70, 70], [40, 255, 255]),       # Amarillo
    "M": ([80, 60, 60], [145, 255, 255]),      # Azul
    "P": ([0, 60, 60], [15, 255, 255]),        # Rojo 1
    "P2": ([165, 60, 60], [179, 255, 255]),    # Rojo 2
    "E": ([35, 50, 50], [95, 255, 255]),       # Verde
}

# Para las acciones que tomará el robot
def ejecutar_accion(tipo, distancia):
    if tipo == "P":
        print("PROHIBICIÓN → Detenerse")

    elif tipo == "W":
        print("ADVERTENCIA → Reducir velocidad")

    elif tipo == "M":
```

```
            print("OBLIGACIÓN → Continuar con precaución")

        elif tipo == "E":
            print("EVACUACIÓN → Continuar navegación")

        elif tipo == "F":
            print("INCENDIO → Evitar zona y mantener acceso libre")

    # Bucle principal de detección
    while True:

        frames = pipeline.wait_for_frames()
        frames = align.process(frames)

        color_frame = frames.get_color_frame()
        depth_frame = frames.get_depth_frame()

        if not color_frame:
            continue

        img = np.asanyarray(color_frame.get_data())
        depth_img = np.asanyarray(depth_frame.get_data()) * depth_scale

        # Corrigiendo orientación de la cámara si es necesario
        img = cv2.rotate(img, cv2.ROTATE_180)
        depth_img = cv2.rotate(depth_img, cv2.ROTATE_180)

        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

        for signal, (lower, upper) in COLOR_RANGES.items():

            mask = cv2.inRange(hsv, np.array(lower), np.array(upper))
            mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, np.ones((5,5),
np.uint8))

            contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

            for c in contours:
```

```
if cv2.contourArea(c) < 1500:
    continue

approx = cv2.approxPolyDP(c, 0.03 * cv2.arcLength(c, True), True)
x, y, w, h = cv2.boundingRect(approx)

cx = x + w // 2
cy = y + h // 2
distancia = depth_img[cy, cx]

n = len(approx)
tipo = None

# Clasificación ISO en relación a la letra asignada
if n == 3 and signal == "W":
    tipo = "W"

elif n >= 8:
    if signal in ["P", "P2"]:
        tipo = "P"
    elif signal == "M":
        tipo = "M"

elif n in [4, 5]:
    if signal in ["P", "P2"]:
        tipo = "F"
    elif signal == "E":
        tipo = "E"

if tipo:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0,255,0), 2)
    cv2.putText(
        img,
        f"{tipo} {distancia:.2f}m",
        (x, y-8),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.7,
        (0,255,0),
```

```
                    2
                )

                ejecutar_accion(tipo, float(distancia))

    cv2.imshow("Detección Señales ISO", img)

    if cv2.waitKey(1) & 0xFF == 27:
        break

pipeline.stop()
cv2.destroyAllWindows()
```

UNIVERSIDAD DE CONCEPCION – FACULTAD DE INGENIERIA
RESUMEN DE MEMORIA DE TITULO

Departamento **:** Departamento de Ingeniería Eléctrica
Carrera : Ingeniería civil electrónica
Nombre del memorista : Nicolás Arturo Vergara Pasten
Título de la memoria **:** Diseño e Implementación de un Sistema de Visión Artificial mediante una Cámara RGB-D para la Interpretación de Señalización Industrial en un Robot Bípedo
Fecha de la presentación oral : Día/mes/año

Profesor(es) guía : Mario Rubén Medina Carrasco
Profesor(es) revisor(es) : Nombres
Concepto **:** Aprobado/Bueno/Muy bueno/Sobresaliente
Calificación **:** Nota

Resumen (máximo 200 palabras)

Esta memoria se centra en diseñar e implementar un sistema de visión artificial para un robot bípedo con la finalidad de que sea capaz de detectar eficazmente señales visuales, en específico señales de seguridad en entornos industriales. Para poder llevar a cabo esta tarea, se utilizó una cámara de profundidad Intel RealSense D435i y una plataforma de procesamiento llamada Raspberry Pi 5, con lo cual fue posible capturar imágenes y mapas de profundidad de manera simultánea.

Para poder procesar la información visual, se utilizaron algoritmos escritos en Python con la ayuda de la biblioteca OpenCV, donde estos algoritmos permitieron mejorar la calidad de las imágenes, dividiéndolas en colores y detectando los contornos, para conseguir dar con la señal buscada. Además, se realizó la estimación de la distancia que existe entre el robot y los objetos detectados, lo cual ayudo a entender y percibir mejor su entorno.

Considerando el conjunto tanto de la información visual como la distancia estimada, fue posible para el sistema generar instrucciones que debería hacer el robot para que tenga una navegación segura. Es por ello que los resultados obtenidos muestran que el sistema puede identificar señales visuales y estimar su distancia de manera efectiva para aplicaciones robóticas.

Finamente, se consideran posibles trabajos futuros, para ir mejorando este proyecto, siendo uno de los más importantes el unificar el sistema de control con el sistema de movimiento del robot.