Portada

UNIVERSIDAD POLITÉCNICA
SALESIANA SEDE GUAYAQUIL
CARRERA DE
TELECOMUNICACIONES

Desarrollo e implementación de un prototipo de robot explorador de inspección horizontal con control manual y visión artificial para detección de fallas en infraestructuras cableadas

Trabajo de titulación previo a la obtención del

Título de ingeniero en telecomunicaciones

AUTORES: MAYLI ADRIANA CABAY VARGAS
GELENNY KARITHZI LEÓN RODRIGUEZ

TUTOR: ING. JOSÉ CÓRDOVA LEÓN

Guayaquil - Ecuador

2026

II. Certificado de responsabilidad y autoría del trabajo de titulación

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros Mayli Adriana Cabay Vargas con documento de identificación N° 0956339626, y Gelenny Karithzi León Rodríguez con documento de identificación N° 0954701280 manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Guayaquil, a los 31 días del mes de enero del año 2026.

Atentamente,

Mayli Adriana Cabay Vargas
0956339626

Gelenny Karithzi León Rodríguez
0954701280

III. Certificado de cesión de derechos de autor del trabajo de titulación a la Universidad Politécnica Salesiana.

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA.

Nosotros, Mayli Adriana Cabay Vargas, con C.I. 0956339626 y Gelenny Karithzi León Rodríguez, con C.I. 0954701280, expresamos nuestra voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana del Ecuador la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Artículo Científico "DESARROLLO E IMPLEMENTACIÓN DE UN PROTOTIPO DE ROBOT EXPLORADOR DE INSPECCIÓN HORIZONTAL CON CONTROL MANUAL Y VISIÓN ARTIFICIAL PARA DETECCIÓN DE FALLAS EN INFRAESTRUCTURAS CABLEADAS", el cual ha sido desarrollado para optar por el título de: Ingeniero en Telecomunicaciones, en la Universidad Politécnica Salesiana, quedando la Institución facultada para ejercer plenamente los derechos concedidos.

En concordancia con lo manifestado, suscribo este documento en el momento que hago la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Guayaquil, a los 31 días del mes de enero del año 2026.

Atentamente,

Mayli Adriana Cabay Vargas
0956339626

Gelenny Karithzi León Rodríguez
0954701280

IV. Certificado de Dirección del Trabajo de Titulación.

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN.

Yo, José Renato Córdova León, con documento de identificación N° 0919123067 docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: DESARROLLO E IMPLEMENTACIÓN DE UN PROTOTIPO DE ROBOT EXPLORADOR DE INSPECCIÓN HORIZONTAL CON CONTROL MANUAL Y VISIÓN ARTIFICIAL PARA DETECCIÓN DE FALLAS EN INFRAESTRUCTURAS CABLEADAS, realizado por Mayli Adriana Cabay Vargas con documento de identificación N° 09533926, y Gelenny Karithzi León Rodríguez con documento de identificación N° 0954701280, obteniendo como resultado final el trabajo de titulación bajo la opción de Artículo académico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Guayaquil, a los 31 días del mes de enero del año 2026.

Atentamente,

José Renato Córdova León
0919123067

V. Dedicatoria

Dedico el presente de trabajo de titulación a mis padres, quienes han sido mi guía y han forjado la persona que soy en la actualidad. Muchos de mis logros se los debo a ustedes, por su confianza, motivación constante y por enseñarme a levantarme cada vez que me equivoco. Este logro también refleja su ardua dedicación, esfuerzo y sacrificio, los cuales han sido fundamentales para alcanzar esta meta

El presente trabajo lo dedico a mi madre Jennifer, cuyo amor, sacrificio y apoyo incondicional han sido el motor que ha sostenido cada uno de mis pasos. Todo lo que soy y todo lo que logro lleva un pedacito de ella, y así mismo es reflejo de su dedicación y del inmenso corazón con el que siempre me ha acompañado.

VI. Agradecimiento

A lo largo de este proceso he comprendido que ningún logro se construye en soledad. Agradezco a Dios y a mi familia por brindarme fortaleza, guía y apoyo constante en los momentos de dificultad, permitiéndome no desistir en el camino hacia el cumplimiento de mis metas, en especial a mis padres, quienes han sido el motor fundamental día a día, impulsándome incondicionalmente. Agradezco también a mi enamorado, Guillermo Falconí, por su acompañamiento y motivación constante . De igual manera, agradezco a mi compañera de tesis, Gelenny León, por su colaboración, apoyo y los momentos compartidos que hicieron más llevadero este proceso. Finalmente agradezco a mi tutor de tesis y a los docentes de la carrera, quienes con sus enseñanzas y orientación contribuyeron al desarrollo óptimo del presente trabajo de titulación.

Al final, llevamos dentro un pedacito de todas las personas que caminaron a nuestro lado. Me reconforta saber que algunas de ellas marcaron de forma profunda quién soy hoy. En especial, agradezco a mis abuelos, Benjamín y Nelly; a mi madre, Jennifer, y a Leonidas, cuyo cariño, apoyo y enseñanzas han sido el impulso que ha acompañado cada paso en mi corta vida. Asimismo, expreso mi sincero agradecimiento a Anthony García, a Guillermo Falconí y a mi compañera de tesis, Mayli Cabay, por su acompañamiento, colaboración y apoyo constante durante el desarrollo de este trabajo, aportando motivación, compromiso y confianza en cada etapa del proceso académico.

VII. Resumen

Este documento trata sobre el uso de un robot explorador para la inspección de desplazamiento horizontal de cableado eléctrico y de telecomunicaciones, ubicado en espacios confinados. El prototipo propuesto se enfoca en zonas de difícil acceso, como tumbados falsos, donde la inspección manual suele ser lenta y limitada por el entorno. El sistema utiliza una plataforma robótica móvil controlada de forma manual mediante comunicación inalámbrica LoRa y una arquitectura basada en microcontroladores ESP32. Para la inspección visual se trabajó con un sensor HuskyLens entrenado con distintos tipos de fallas en cableado. Durante el recorrido del robot se capturaron imágenes y video usando un módulo ESP32-CAM. La información obtenida se almacenó en un servidor local implementado en Node.js.

Las pruebas se realizaron dentro de una vivienda y en uno de los laboratorios de la Universidad Politécnica Salesiana, en un tumbado falso. El entorno presentó condiciones habituales de este tipo de espacios, como poca iluminación, presencia de polvo y elementos estructurales que dificultan el movimiento. En esta etapa se consideraron seis tipos de fallas en cableado. Para la evaluación del sistema se realizaron pruebas a tres distancias (15 cm, 30 cm y 45 cm), utilizando un total de 360 imágenes obtenidas durante los ensayos.

Los resultados evidenciaron que la precisión del sistema disminuye conforme aumenta la distancia de inspección, alcanzando un promedio del 85% a 15 cm, del 75% a 30 cm y del 65% a 45 cm. Asimismo, el sistema mantuvo una transmisión estable de los datos, permitiendo una correcta recepción y almacenamiento de la información. Durante este proceso se detectó un mayor número de fallas, lo que demuestra la efectividad del sistema en condiciones reales de operación.

Palabras clave

1; Robot explorador 2; Inspección de cableado 3; Visión artificial 4; ESP32-CAM

5; Comunicación LoRa 6; HuskyLens

VIII. Abstract

This research focuses on the use of an exploratory robot for the horizontal inspection of electrical and telecommunications wiring located in confined spaces. The proposed prototype is aimed at hard-to-access areas, such as false ceilings, where manual inspection is usually slow and limited by environmental conditions. The system uses a mobile robotic platform manually controlled through LoRa wireless communication and an architecture based on ESP32 microcontrollers. For visual inspection, a HuskyLens sensor trained with different types of wiring faults was used. During the robot's operation, images and video were captured using an ESP32-CAM module. The collected information was stored on a local server implemented in Node.js.

The tests were carried out inside a residential building and in one of the laboratories of the Universidad Politécnica Salesiana, specifically in a false ceiling. The environment presented typical conditions of this type of space, such as low lighting, dust presence, and structural elements that hinder movement. At this stage, six types of wiring faults were considered. For system evaluation, tests were conducted at three distances (15 cm, 30 cm, and 45 cm), using a total of 360 images obtained during the experiments.

The results showed that system accuracy decreases as the inspection distance increases, reaching an average of 85% at 15 cm, 75% at 30 cm, and 65% at 45 cm. In addition, the system maintained stable data transmission, allowing proper reception and storage of information. During this process, a higher number of faults were detected, demonstrating the effectiveness of the system under real operating conditions.

Keywords

1; Explorer robot 2; Cable inspection 3; Artificial vision 4; ESP32-CAM

5; LoRa communication 6; HuskyLens

IX. Índice de Contenido

X. Introducción

Las infraestructuras que soportan las redes de telecomunicaciones y de energía constituyen un pilar fundamental para el desarrollo de las sociedades modernas, ya que permiten la transmisión de información y el suministro continuo de servicios básicos. Estas infraestructuras suelen encontrarse ubicadas en entornos de difícil acceso, tales como canalizaciones subterráneas, conductos horizontales, techos industriales y estructuras urbanas elevadas, lo que convierte a las labores de inspección y mantenimiento en procesos complejos que demandan tiempo, esfuerzo físico y un alto nivel de precaución. Estas condiciones incrementan la probabilidad de fallas técnicas y accidentes laborales, especialmente cuando las inspecciones se realizan de forma manual.

Además de los riesgos presentes para el personal técnico especializado, los accidentes eléctricos también afectan a trabajadores que realizan labores de mantenimiento sin la capacitación adecuada, constituyendo un problema recurrente a nivel internacional. En este contexto, diversos organismos de seguridad laboral han documentado incidentes graves asociados a la manipulación inadecuada de instalaciones eléctricas.

De acuerdo con un medio digital que recopila reportes internacionales de seguridad laboral, específicamente la base de datos de accidentes de la Occupational Safety and Health Administration (OSHA), dentro de la recopilación de incidentes reportados, se destaca un caso ocurrido durante trabajos en un techo suspendido. Según la información publicada en la página oficial de OSHA, un trabajador que se encontraba sobre un andamio móvil retirando baldosas y luminarias procedió a cortar conductores energizados de 277 V sin realizar el aislamiento previo del circuito ni contar con la asistencia de un electricista. Durante la intervención, al cortar un segundo conductor mientras se encontraba en contacto con la estructura metálica del techo, el trabajador sufrió una descarga eléctrica fatal. El reporte indica que, pese a la aplicación de maniobras de reanimación, el accidente tuvo un desenlace mortal [1] .

En este contexto, los robots exploradores han surgido como una alternativa tecnológica para la inspección de infraestructuras complejas. Diversos estudios han demostrado que estas plataformas automatizadas permiten realizar inspecciones en áreas de difícil acceso, reduciendo riesgos y optimizando recursos. Investigaciones relacionadas con robots orientados a la detección de fallas eléctricas han evidenciado que el análisis en tiempo real incrementa la precisión y la seguridad en el diagnóstico de irregularidades en líneas y conexiones eléctricas [2]. Asimismo, se ha demostrado que la utilización de plataformas de inspección automatizadas puede disminuir el consumo promedio de energía entre un 6.5 % y un 17.8 %, validando su eficiencia, autonomía y sostenibilidad frente a los métodos tradicionales [3].

Entre las propuestas más relevantes se encuentra un robot explorador diseñado para operar en ambientes mineros, el cual integra sistemas de navegación autónoma mediante SLAM y sensores LiDAR, además de un sistema de tracción tipo oruga que le permite adaptarse a diferentes tipos de terreno sin comprometer la recolección de datos. El uso de herramientas de software libre y control remoto por Wi-Fi permitió que esta plataforma se consolide como una alternativa económica y versátil para tareas de inspección, demostrando que es posible mejorar la seguridad y la eficiencia en entornos de alto riesgo [4]. De igual forma, se han desarrollado robots exploradores capaces de detectar metales en áreas mineras, incorporando sistemas de tracción, suspensión y sensores inductivos que evidencian la viabilidad de construir plataformas funcionales con componentes económicos y personalizables [5].

Adicionalmente, estudios recientes han abordado la eficiencia energética en robots móviles de inspección, logrando reducciones significativas en el consumo de energía sin comprometer el rendimiento operativo. Estos avances refuerzan la importancia de diseñar plataformas robóticas con autonomía prolongada y uso eficiente de los recursos energéticos [6]. Paralelamente, el uso de robots aéreos, terrestres y submarinos ha ampliado el campo de aplicación de estas tecnologías, permitiendo inspeccionar edificaciones, túneles, plantas eléctricas y estructuras sumergidas, mejorando la seguridad, precisión y continuidad de los servicios [7].

A pesar de estos avances, en el contexto ecuatoriano las inspecciones en infraestructuras cableadas eléctricas y de telecomunicaciones continúan realizándose principalmente de manera manual. Los técnicos deben acceder a espacios reducidos, oscuros y confinados, lo que incrementa los riesgos laborales y prolonga los tiempos de diagnóstico[8].

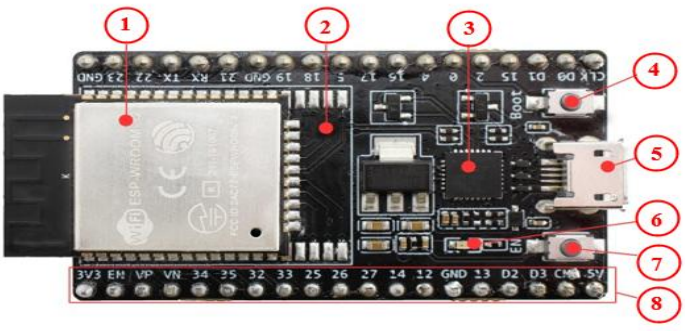
Ante esta problemática, surge la necesidad de incorporar tecnologías avanzadas que permitan una inspección más eficiente, segura y precisa. En este sentido, el desarrollo de un prototipo de robot explorador de inspección horizontal con control manual y visión artificial responde a la necesidad de optimizar las tareas de mantenimiento en canalizaciones subterráneas y conductos horizontales, disminuyendo la exposición humana y los costos de operación [9].

En el ámbito nacional, normativas como el Decreto Ejecutivo 2393, la Guía Básica de Prevención de Riesgos Laborales 2024 y el Código Eléctrico Ecuatoriano INEN establecen protocolos estrictos para trabajos en entornos eléctricos, los cuales incrementan la logística y los costos operativos. Estudios realizados por la Universidad Católica de Cuenca (UCACUE) evidencian que las labores de mantenimiento en infraestructuras elevadas en ciudades como Quito, Guayaquil y Cuenca presentan altos índices de siniestralidad laboral, especialmente cuando son ejecutadas por personas no capacitadas [10], [11], [12]. Esto evidencia la necesidad de promover soluciones tecnológicas que reduzcan la exposición directa de las personas a entornos eléctricos peligrosos.

Por lo tanto, el presente proyecto no solo busca mejorar la seguridad y eficiencia en las inspecciones de infraestructuras cableadas, sino también promover la aplicación de tecnologías innovadoras y de bajo costo en el ámbito de las telecomunicaciones, aportando una solución adaptable que contribuya al desarrollo tecnológico y a la prevención de riesgos laborales en el Ecuador [13].

La presente investigación se enfocará en el desarrollo de un prototipo de robot explorador destinado a la inspección de infraestructuras cableadas en contextos eléctricos y de telecomunicaciones, especialmente en espacios donde el acceso humano resulta limitado, como tumbados, falsos techos y superficies horizontales estrechas. En este contexto, un falso techo o techo suspendido es un sistema no estructural instalado por debajo de la losa estructural superior, utilizado para ocultar instalaciones eléctricas, iluminación y sistemas de ventilación. La separación entre la losa y el falso techo suele ser de entre 7,6 cm y 20 cm, lo que permite el paso y mantenimiento de los servicios técnicos El sistema propuesto incorporará control manual, transmisión de video en tiempo real y herramientas básicas de visión artificial, con el fin de facilitar la identificación visual de posibles deterioros, anomalías o fallas en el cableado.

El propósito central del estudio es mejorar los procedimientos tradicionales de inspección, buscando reducir tanto la exposición del personal técnico a zonas confinadas o elevadas como el tiempo requerido para el diagnóstico de fallas. De manera complementaria, se evaluará el desempeño del prototipo como una solución tecnológica de bajo costo, capaz de aportar mejoras en términos de seguridad operativa, eficiencia del proceso y precisión en las labores de mantenimiento preventivo de infraestructuras cableadas.



1. MÓDULO ESP-WROOM-32
2. ESPACIO PARA WROVER
3. CONVERSOR USB-SERIAL ENTRADA/SALIDA
4. BOTÓN BOOT
5. PUERTO MICRO USB
6. LED DE ENCENDIDO
7. BOTÓN RESET
8. PINES DE

Figura 1 Partes del ESP-32 Fuente: Autores

El ESP32, que se muestra en la Figura 1, es un microcontrolador de doble núcleo utilizado en aplicaciones de Internet de las cosas (IoT) que requieren buena capacidad de procesamiento y bajo consumo energético. Incorpora conectividad inalámbrica mediante Wi-Fi 802.11 b/g/n y Bluetooth/BLE, permitiendo la comunicación directa con redes y dispositivos sin módulos adicionales. Asimismo, el dispositivo dispone de numerosos pines de propósito general (GPIO), compatibles con protocolos de comunicación SPI, I2C, UART y PWM, los cuales facilitan la conexión con sensores, actuadores y periféricos de un sistema, como se observa en la Figura 2 [14].

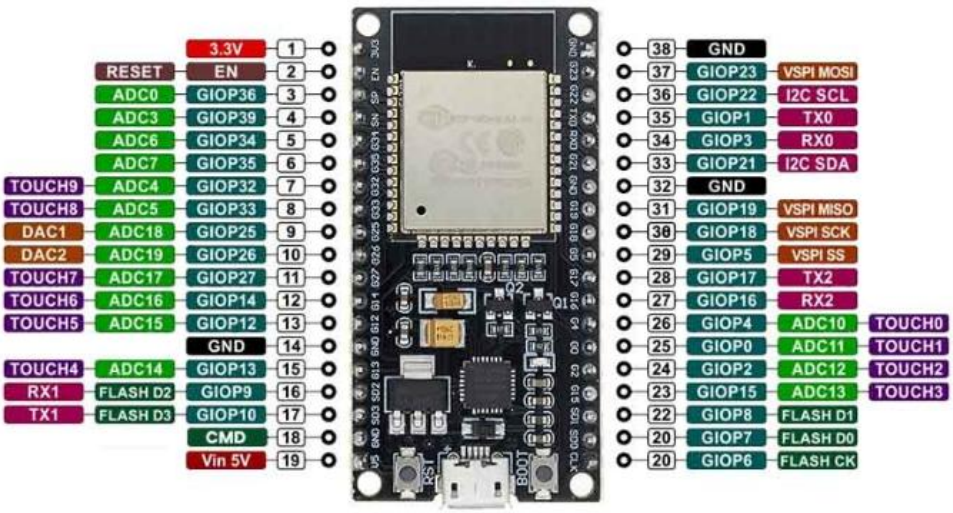


Figura 2 Diagrama de pines del ESP-32 Fuente:[15]

a. ESP32-CAM

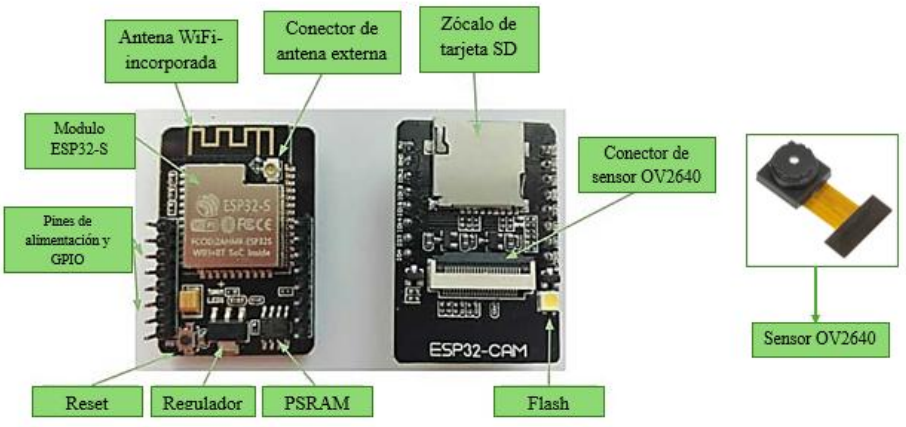


Figura 3 Partes del módulo Esp32-Cam Fuente: Autor

El ESP32-CAM, mostrado en la Figura 3 integra el sensor OV2640, capaz de trabajar desde QVGA (320×240) hasta UXGA (1600×1200), lo que permite ajustar la resolución según la velocidad requerida para el reconocimiento facial. El sensor utiliza un formato 1/4”, consume alrededor de 125 mW y opera con voltajes entre 2.5–3.0 V en la sección analógica y 1.7–3.3 V en I/O. El módulo incorpora además un conector UFL, que permite usar una antena externa para mejorar la calidad de transmisión inalámbrica [16]. Para enviar los datos capturados por la cámara, el ESP32-CAM emplea el esquema OFDM, que divide la señal en múltiples subportadoras para optimizar el canal inalámbrico. Sobre ellas se aplican modulaciones digitales como BPSK, QPSK, 16-QAM y 64-QAM. Gracias a este método, el dispositivo adapta automáticamente la modulación según la calidad del enlace [17]. La asignación y distribución de pines del módulo, utilizada para su correcta conexión con el resto de los componentes del sistema, se presenta en la Figura 4.

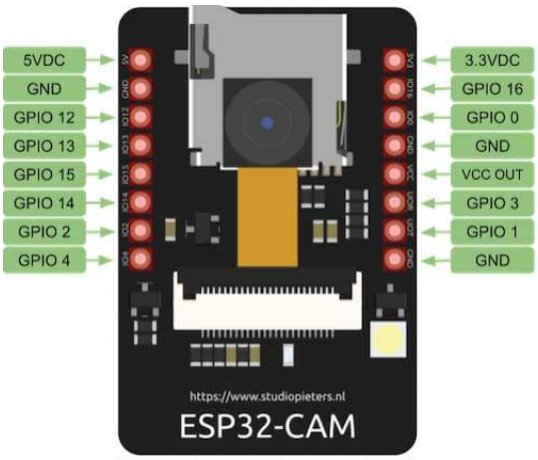


Figura 4 Pines del módulo ESP-32 CAM Fuente: [18]

b. HuskyLens

La HuskyLens es un sensor de visión con algoritmos de reconocimiento y seguimiento integrados (visión AI on-board) que procesa en tiempo real la imagen de su propia cámara y entrega al microcontrolador datos de la detección, como la posición del objetivo en ejes x–y y el tamaño de la caja delimitadora (útil para estimar distancia). Se integra de forma directa mediante interfaces[19]. En el contexto de vigilancia autónoma, el HuskyLens actúa como el responsable de identificar y seguir un objeto específico, mediante el procesamiento directo de la imagen captada por su cámara integrada. El sensor aplica sus algoritmos internos para localizar el objetivo dentro del campo visual y como se explicó previamente obtener su posición en los ejes x y. Lo reportado en proyectos relacionado (detección de rostros) evidencia que el HuskyLens es capaz de mantener un seguimiento estable bajo condiciones de operación típicas, lo que confirma su utilidad como solución de visión embebida para tareas de reconocimiento y seguimiento en plataformas móviles de bajo costo [20]. En la Figura 5 se evidencia el sensor HuskyLens.

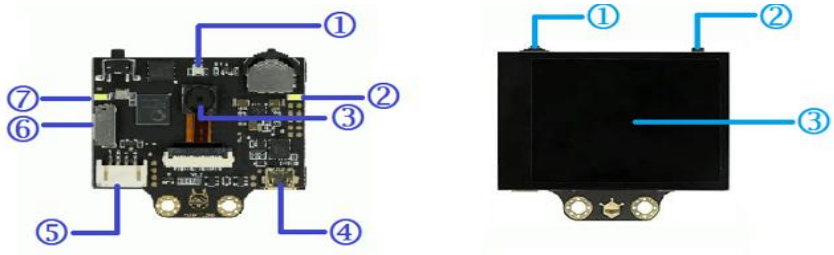


Figura 5 HuskyLens AI Machine Vision Sensor Fuente: [21]

c. Modulo LORA E32 433T20D (Antena)

El módulo LoRa analizado se estructura a partir del transmisor Semtech SX1278, diseñado para operar en la banda de 433 MHz y capaz de alcanzar una potencia máxima de 20 dBm. Su configuración permite establecer enlaces de hasta 3000 m cuando existe línea de visión directa, lo que lo convierte en una opción viable para aplicaciones de comunicación de largo alcance. Este diseño se complementa con un receptor tipo AQD, empleado para la transmisión de datos ambientales mediante protocolo LoRa [22]. En la Figura 6 se ilustra el módulo Lora.

Figura 6 Modulo LoRa SX1278 de 433 MHz Fuente:[22]

d. Joystick

El joystick analógico, funciona como un pequeño sistema mecatrónico, el cual es capaz de transformar cada movimiento físico del usuario en una señal eléctrica que el equipo interpreta, para de esta forma ajustar dirección, velocidad o cualquier otra acción requerida. En diversos proyectos relacionados se lo ha utilizado como interfaz háptica bastante versátil: combina sensores de fuerza, actuadores lineales y motores paso a paso que detectan con precisión las instrucciones y envían esa información de manera inalámbrica [23]. En la Figura 7 aparece el joystick analógico a usar.

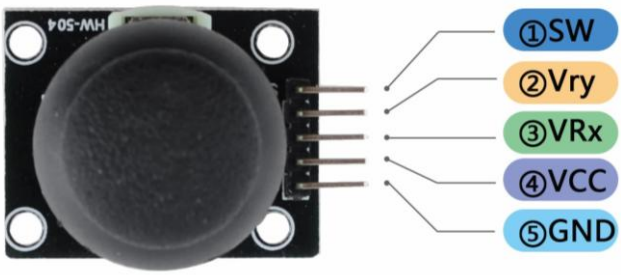


Figura 7 Modulo Joystick analógico MOD-046 con pulsador Fuente: [24]

e. Driver de motor TB6612FNG

El driver TB6612FNG es un pequeño modulo para mover motores DC de manera sencilla. Funciona con un tipo de circuito que aprovecha mejor la energía y evita que el componente se caliente, su tamaño reducido hace que no requiera muchos elementos extra para funcionar. También incluye protecciones integradas que ayudan a que el motor y el propio modulo trabajen con mayor estabilidad y seguridad durante su uso. Gracias a su control PWM independiente por canal, ofrece un manejo preciso del movimiento en plataformas móviles [25]. En la Figura 8 se presenta la distribución de los pines del driver que nos ayudará en el proyecto.

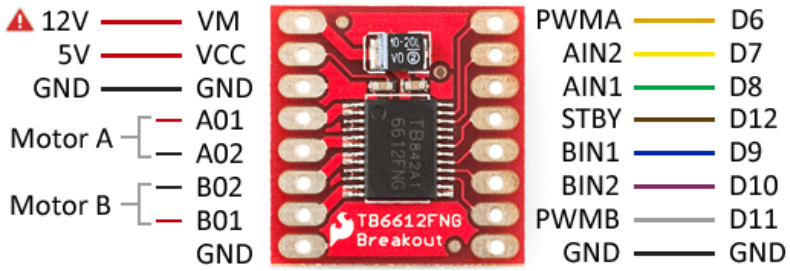


Figura 8 Distribución de pines del driver de motores TB6612FNG Fuente: [26]

f. Protoboard

El protoboard, también conocido como breadboard, es una superficie de trabajo diseñada para el montaje temporal de circuitos sin recurrir a soldadura. Su estructura interna se compone de tiras metálicas que enlazan grupos de orificios, lo que permite insertar componentes y realizar conexiones eléctricas de forma rápida y ordenada. En los extremos suelen incorporar rieles destinados a la distribución del voltaje y el retorno, mientras que la zona central presenta una separación que evita que los pines de los circuitos integrados se unan accidentalmente [27]. En la Figura 9 se visualiza un protoboard con sus respectivas partes.

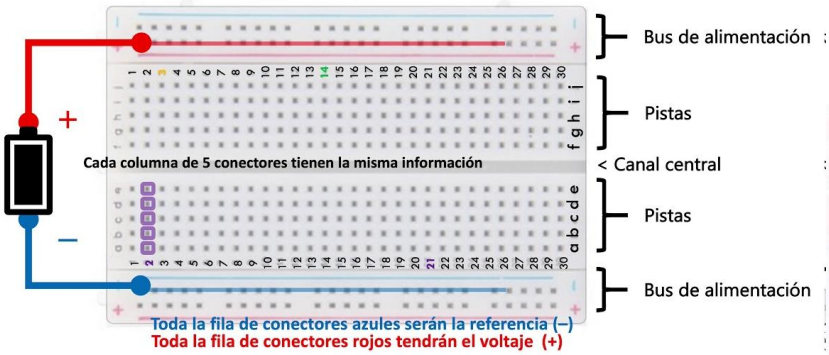


Figura 9 Estructura y distribuciones de conexiones en una protoboard Fuente: [28]

g. Kit robot Arduino o carrito chasis

El carrito chasis Arduino es una plataforma móvil de uso comercial empleada en prototipos robóticos a pequeña escala. Se trata de una base mecánica sencilla que permite montar motores, ruedas y la electrónica necesaria para el desplazamiento. Este tipo de chasis incorpora cuatro motores de corriente continua con engranajes, uno para cada rueda. Las ruedas incluidas son estándar y están diseñadas para acoplarse de manera directa a los motores. Para asegurar la fijación, el conjunto utiliza soportes de montaje que permiten sujetar cada motor con su respectiva rueda al chasis de forma adecuada [29]. En la Figura 10 se visualiza el kit completo del carrito chasis utilizado en este proyecto.

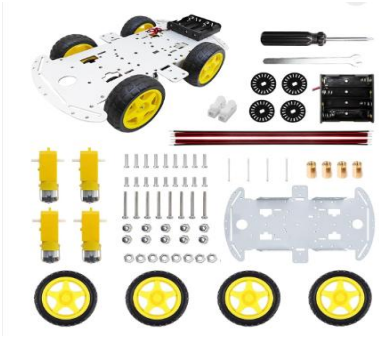


Figura 10 Plataforma móvil 4WD con chasis metálico y encoders. Fuente:[30]

h. Portapilas

El Portapilas o battery holder funciona como sistema de alimentación se requiere adquirir por separado, ya que no viene integrado en las placas o proyectos electrónicos[31]. En la Figura 11 se visualiza el portapilas.

Figura 11 Portapilas para 4 baterías AA Fuente:[32]

i. Pilas

Las pilas se entienden como celdas electroquímicas capaces de almacenar una cantidad limitada de energía y liberarla de forma directa para sostener el funcionamiento de un sistema electrónico. En el contexto robótico en varios estudios cumplen el mismo principio general que las baterías descritas en el artículo: proporcionan la reserva interna de energía que compensa las pérdidas propias de cualquier mecanismo y permiten que el robot opere sin conexión permanente a una fuente externa [33]. En la Figura 12 se muestra pilas parecidas a las que se usarán.

Figura 12 Pilas AA aisladas. Fuente:[33]

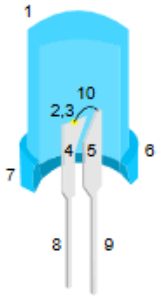
j. Policarbonato

El policarbonato se presenta como un polímero termoplástico adecuado para funcionar como base estructural en un robot debido a la combinación poco común de transparencia óptica, alta tenacidad, resistencia al impacto y estabilidad dimensional, propiedades que permiten soportar esfuerzos mecánicos sin deformaciones significativas y sin comprometer la integridad del montaje electrónico [34]. En la Figura 13 se aprecia un pedazo de policarbonato.

Figura 13 Plataforma de base en policarbonato Fuente:[34]

k. Leds

Los LED se utilizan típicamente como emisores de luz que emite fotones cuando se recibe una corriente eléctrica de muy baja intensidad. El LED por lo general se encierra en un material plástico de color que acentúa la longitud de onda generada por el diodo y ayuda a enfocar la luz en un haz, pero también pueden utilizarse como fotosensores, siendo sensibles a longitudes de onda iguales o inferiores a la longitud de onda predominante que emiten. Además, un LED utilizado como detector es sensible a longitudes de onda iguales o inferiores a la longitud de onda predominante que emite. Esta capacidad de los LED ofrece la posibilidad de desarrollar una herramienta sencilla capaz de demostrar el efecto fotoeléctrico [35]. .En la Figura 14 se puede ver el led y sus respectivas partes.



1. Encapsulado de resina epoxy.
2. Diodo semiconductor emisor de luz.
3. Copa reflectora o cavidad reflectante
4. Yunque
5. Poste
6. Base
7. Base Plana (identifica la terminal negativa o cátodo).
8. Terminal de conexión (Negativo o cátodo)
9. Terminal de conexión (positivo o ánodo)
10. Cable de unión o bigote (Alambre muy fino)

Figura 14 Partes de un Led Fuente:[36]

l. Hotspot

Un hotspot es un punto de acceso Wi-Fi que un dispositivo crea, para compartir su conexión a Internet. En los teléfonos, esta función usa los datos móviles para permitir que otros equipos se conecten [37]. A continuación en la Figura 15 se evidencia dispositivos que actúan como hotspots.

Figura 15 Dispositivos que actúan como Hotspots Fuente [37]

m. Arduino IDE

El entorno Arduino IDE se distingue por ofrecer una plataforma unificada para la programación de distintas tarjetas de desarrollo, lo que permite que un mismo proyecto pueda adaptarse a varios dispositivos mediante la simple selección del modelo de placa. Esta característica facilita la migración del código y reduce la dependencia del hardware específico. No obstante, las diferencias inherentes entre microcontroladores pueden generar variaciones en el tiempo de ejecución y en el rendimiento general, incluso cuando se emplea el mismo código; por ello, antes de iniciar el desarrollo, es necesario verificar que la tarjeta seleccionada cuente con las funcionalidades requeridas por el proyecto [38].

El Arduino IDE es ampliamente utilizado para la implementación de sistemas embebidos de manera sencilla y eficiente, siendo especialmente compatible con placas ESP32, lo que lo convierte en una herramienta idónea para aplicaciones que requieren conectividad y procesamiento en tiempo real. En el presente proyecto, el Arduino IDE se emplea para la carga del firmware y la programación de los módulos ESP32 y ESP32-CAM, permitiendo la configuración del sistema de comunicación inalámbrica LoRa, el control de motores y la integración del sensor de visión artificial HuskyLens. En la Figura 16 se observa el logotipo oficial del entorno de desarrollo utilizado.

Figura 16 Logo oficial de Arduino Fuente:[39]

n. Node.js

Node.js es un entorno de ejecución de JavaScript (JS) creado en torno a V8, el motor JS utilizado en Chromium, que es la base del navegador web de Google. Ha ganado una adopción masiva por parte de desarrolladores y organizaciones de todo el mundo debido a su facilidad de desarrollo, así como al eficiente modelo de entrada/salida (E/S) basado en eventos y sin bloqueos. Node.js se emplea ampliamente en aplicaciones web y de comunicación entre dispositivos [40]. En la Figura 17 se puede apreciar el logo de Node.js

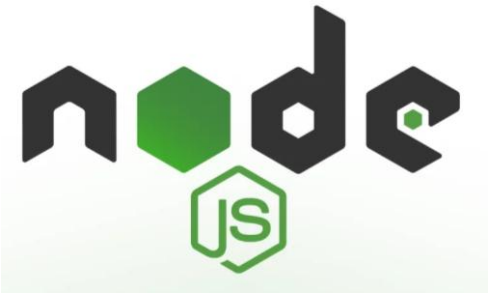


Figura 17 Logo de Node.js Fuente: [40]

o. Visual Studio Code

Visual Studio Code es un entorno pensado para hacer más sencilla la programación ya que ofrece herramientas ligeras y extensiones que cada usuario puede ajustar a su manera. Aunque su uso común está en el desarrollo de software, también puede adaptarse a tareas técnicas más específicas gracias a los módulos que se le pueden añadir. Un ejemplo de ello es la extensión diseñada para archivos CIF, que permite resaltar su sintaxis, sugerir autocompletado y detectar errores durante la edición [41]. En la Figura 18 se visualizará el logo del entorno de visual studio code, el cual nos ayudará en la programación de un servidor.

Figura 18 Logo oficial de Visual Studio Code Fuente:[41]

p. Wireshark

Wireshark es uno de los analizadores de protocolos más utilizados para inspección y diagnóstico de tráfico en redes, como tal la herramienta permite capturar, visualizar y decodificar paquetes en tiempo real, soportando protocolos como TCP/IP, UDP, HTTP y FTP, en general su interfaz gráfica y sus filtros avanzados permiten analizar comunicaciones, detectar fallos, revisar y estudiar el comportamiento de dispositivos dentro de un entorno de red [42]. La Figura 19 nos dará un ejemplo del tráfico que wireshark captura en la red.

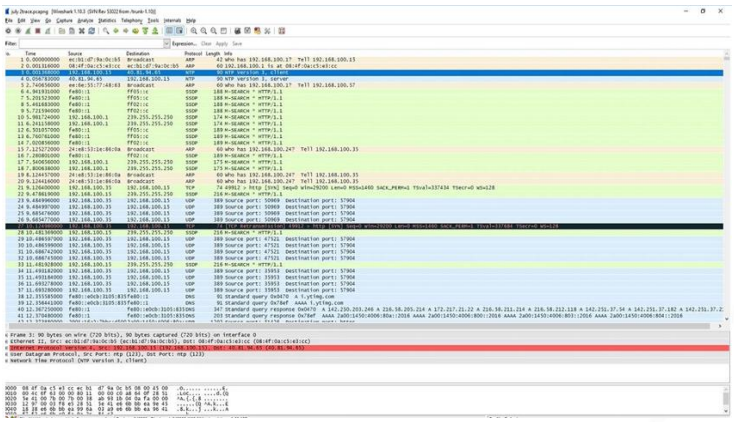


Figura 19 Tráfico de red con Wireshark Fuente: [42]

q. Protocolo I2C

La tecnología I^2C (Inter-Integrated Circuit) corresponde a un protocolo de comunicación serial ampliamente utilizado en sistemas electrónicos para la interconexión de microcontroladores con otros dispositivos integrados. Este protocolo permite establecer una comunicación eficiente entre múltiples componentes dentro de un mismo sistema. En la figura 20 se ilustra el principio de funcionamiento del bus I^2C, donde un dispositivo actúa como maestro, responsable de coordinar el intercambio de información, mientras que uno o varios dispositivos esclavos responden a las solicitudes emitidas por el dispositivo principal [43].

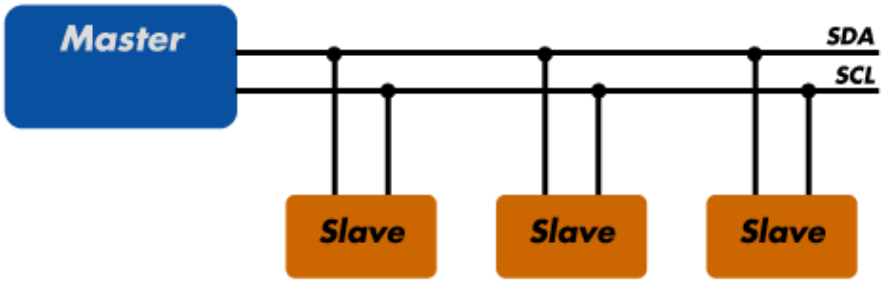


Figura 20 Funcionamiento del protocolo I^2C Fuente: [43]

r. Norma ISO 13482:2014

La norma ISO 13482:2014 establece los requisitos y directrices de seguridad para robots de cuidado personal, definiendo medidas de diseño seguro, protección y uso adecuado de estos dispositivos punto la norma clasifica los robots de cuidado personal en 3 tipos principales:

- Robot sirviente móvil
- Robot asistente físico
- Robot transportador de personas

Esos robots están destinados a mejorar la seguridad y la calidad de vida de los usuarios, independientemente de su edad o capacidad. Hizo 13482:2014 describe los peligros asociados al uso de estos robots y proporciona directrices para reducirlos a niveles aceptables, abarcando incluso aplicaciones de contacto físico entre humanos y robots. La norma se aplica a robots terrestres utilizados en entornos de servicio o asistencia y establece criterios de seguridad directamente relevantes para robots móviles de inspección que interactúan indirectamente con personas. Diversos estudios analizan la ISO 13482:2014 como norma vigente para garantizar seguridad en robots móviles de servicio destacando su utilidad para proteger a personas y operadores en entornos de riesgo [44].

s. Rango de visión del sistema

El rango de visión del sistema hace referencia a la distancia en la que el módulo de visión artificial HuskyLens puede detectar y reconocer objetos de manera estable y con un nivel aceptable de precisión, Lo cual es esencial para la navegación y percepción de un robot móvil de inspección. En aplicaciones robóticas, la integración de sensores de visión con capacidades de procesamiento inteligente, como HuskyLens, ha demostrado ser eficaz para proporcionar datos visuales del tiempo real y facilitar la identificación y seguimiento de objetos en el entorno. El sensor HuskyLens Utilizado de manera efectiva dentro de un rango operativo definido por ejemplo entre 0,3 m y 3 m para tareas de inspección, permitiendo a un robot identificar irregularidades obstáculos o zonas que sean de interés sin acercarse demasiado al objeto observado, lo que reduce riesgos de colisiones y daños durante la exploración [45].

t. Parámetro de precisión (Accuracy)

La precisión, conocida también como accuracy, es una métrica fundamental para evaluar el desempeño de un sistema de visión artificial durante la detección de objetos. Este parámetro representa la proporción de veces en que el sistema identifica correctamente un objeto o zona de interés con respecto al total de predicciones realizadas, sirviendo como un indicador general del comportamiento del sistema en pruebas de experimentales. En tareas de reconocimiento y clasificación de objetos, accuracy se utiliza ampliamente porque proporciona una medida directa de cuántas predicciones coinciden con las verdaderas etiquetas del objetivo, lo que permite estimar el nivel de confiabilidad del algoritmo desarrollado en condiciones de operaciones reales. Por esta razón, la accuracy se considera uno de los principales indicadores para evaluar la calidad del proceso de detección, ya que refleja qué tan consistente es el reconocimiento visual dentro del rango de visión establecido para la inspección [46].

u. Ganancia de antena (dBi)

El parámetro dBi es una unidad de medida estándar utilizada para cuantificar la ganancia de una antena comparada con una antena isotrópica ideal, que radia uniformemente en todas las direcciones. La ganancia de una antena en dBi se define como la proporción de la potencia radiada en la dirección de máximo radiación frente a la potencia que radiaría un radiador isotrópico bajo las mismas condiciones, lo cual es una métrica fundamental en sistemas de comunicaciones inalámbricas para caracterizar antenas y optimizar enlaces de radiofrecuencia [47].

v. Clasificación de defectos detectados

Tipo de defecto	Descripción
Defecto funcional	Equipos o elementos que no cumplen su función
Obstáculo	Elementos que bloquean el área o el acceso
Zona oscura	Áreas sin iluminación suficiente y dificultan una inspección
Área inaccesible	Zonas difíciles de alcanzar o peligrosas
Degradación superficial	Grietas, desgaste o corrosión en la superficie

Tabla 1 Clasificación de defectos en un entorno de inspección Fuente: Autor

La clasificación de defectos detectados es una etapa clave en los sistemas de visión artificial, ya que permite organizar y distinguir los distintos eventos identificados a lo largo del proceso de inspección visual como se puede observar en la Tabla 1. Este proceso de categorización facilita el análisis de las condiciones observadas al agrupar las anomalías de acuerdo con sus características visuales y estructurales, contribuyendo a una evaluación más clara y sistemática del estado de la infraestructura inspeccionada. La clasificación diferenciada de defectos no solo organiza la información visual, sino que también soporta decisiones posteriores sobre el estado de la superficie o componente inspeccionado y es una parte fundamental de los métodos de control de calidad en visión artificial aplicada [48].

w. Riesgo en inspecciones técnicas

Realización de inspecciones técnicas de infraestructuras conlleva múltiples riesgos físicos para el personal, especialmente cuando estas labores de implican el ingreso a espacios reducidos, la presencia de cableado energizado, zonas con acceso complicado o visibilidad limitada. En este tipo de entornos, lo de espectadores están expuestos a peligros que pueden resultar en lesiones accidentes o daños a la salud. Por ello la robótica y la automatización han surgido como herramientas claves para mitigar estos riesgos, ya que al estar equipados con sensores y sistemas de visión puede desempeñar tareas de inspección sin que una persona deba estar presente físicamente en zonas desglosas, según una revisión sistemática sobre el uso de robots en la inspección y el monitoreo de edificios e infraestructura la adopción de sistema robótico reduce la

exposición humana a condiciones arriesgadas, mejora la eficiencia de las inspecciones y contribuye a una gestión más segura de activos críticos. Esto posición de la robótica como una estrategia efectiva para disminuir los riesgos asociados a inspecciones técnicas tradicionales en Ambientes complejos o potencialmente peligrosos [49].

x. Fórmulas

i. Control de velocidad del motor DC mediante PWM

El movimiento del robot explorador se controla mediante motores de corriente continua, los cuales son accionados a través del driver TB6612FNG. La velocidad de cada motor se ajusta utilizando modulación por ancho de pulso (PWM), una técnica habitual en sistemas embebidos que permite un control eficiente y estable del giro de motores DC. La relación entre el ciclo de trabajo de la señal PWM y la velocidad alcanzada por el motor se presenta en la ecuación 1 [50].

$$D = \frac{t_{on}}{T} \times 100 \qquad (1)$$

Donde:

D: ciclo de trabajo PWM [%]

t_{on}: tiempo activo de la señal PWM [s]

T: periodo total de la señal PWM [s]

ii. Velocidad del motor de corriente continua

La velocidad angular del motor de corriente continua se relaciona de forma directa con el voltaje promedio que se le aplica, el cual depende del ciclo de trabajo de la señal PWM. A medida que este ciclo varía, el comportamiento del motor cambia de manera proporcional, lo que permite describir su funcionamiento a través de un modelo matemático que se presenta en la ecuación 2 [51].

$$\omega = K_v \cdot V_{avg} \qquad (2)$$

Donde:

ω: velocidad angular del motor [rad/s]

K_v: constante del motor

V_{avg}: voltaje promedio aplicado mediante PWM [V]

iii. Potencia eléctrica consumida por el robot

El consumo energético del robot explorador se calcula a partir de la relación entre el voltaje de alimentación y la corriente que demanda el sistema, considerando tanto los componentes electrónicos como los elementos mecánicos. Esta relación permite saber la energía utilizada durante el funcionamiento del prototipo y se expresa mediante la ecuación 3 [52].

$$P = V \cdot I \qquad (3)$$

Donde:

P: potencia eléctrica [W]

V: voltaje del sistema [V]

I: corriente total consumida [A]

iv. Autonomía del sistema de alimentación

La autonomía del robot explorador se calcula considerando la capacidad de las pilas utilizadas y la corriente promedio que consume el sistema durante su operación. A partir de estos valores, se puede estimar el tiempo de funcionamiento continuo del prototipo, relación que se expresa mediante la ecuación 4 [53].

$$T = \frac{C}{I} \qquad (4)$$

Donde:

T: tiempo de operación [h]

C: capacidad de las pilas [Ah]

I: corriente promedio durante la operación continua [A]

v. Resolución de imagen del sistema de visión ESP32-CAM

La resolución de imagen del sistema de visión ESP32-CAM se determina a partir de la cantidad de píxeles horizontales y verticales que conforman la imagen capturada. Este valor influye directamente en la calidad de la inspección visual remota, ya que define el nivel de detalle con el que se pueden observar los elementos analizados durante la exploración. La relación correspondiente se expresa en la ecuación 5 [54].

$$R = A_x \times A_y \qquad (5)$$

Donde:

R: resolución total [píxeles]

A_x: píxeles horizontales

A_y: píxeles verticales

vi. Tasa de transmisión de datos inalámbrica

La velocidad con la que el sistema de control remoto transmite la información está condicionada por el ancho de banda disponible y por la relación señal-ruido del canal inalámbrico. Esta dependencia se describe mediante el modelo de Shannon-Hartley, el cual se presenta en la ecuación 6 [55].

$$C = Blog\ _2(1 + SNR) \qquad (6)$$

Donde:

C: capacidad del canal [bps]

B: ancho de banda [Hz]

SNR: relación señal-ruido

vii. Longitud de onda de la señal inalámbrica

La longitud de onda de una señal se relaciona directamente con su frecuencia de operación y afecta parámetros como antenas, propagación y cobertura [56]. Se describe mediante la ecuación 7:

$$\lambda = \frac{c}{f} \qquad (7)$$

Donde:

λ = longitud de onda [m]
C= velocidad de la luz (3 x 10^8 m/s)
f = frecuencia de operación del sistema inalámbrico [Hz]

viii. Campo de visión (FOV) de la cámara

El Fov determina el área visible por la cámara del robot y permite justificar el rango y cobertura de inspección y se determina mediante las ecuaciones 8 y 9 [57]:

$$FOV_{horiz} = 2\ arctan\left(\frac{w}{2f}\right) \quad (8)$$

$$FOV_{vert} = 2\ arctan\left(\frac{h}{2f}\right) \quad (9)$$

Donde:

w, h = ancho y alto del sensor de la cámara [mm]
f = distancia focal de la cámara [mm]

ix. Distancia mínima de detección

Permite estimar que tan cerca o lejos puede detectar el robot un objeto o defecto basado en resolución y FOV, se determina mediante la ecuación 10 [58]:

$$d_{min} = \frac{R_{OBJ}}{R_{pixel}} . f \quad (10)$$

Donde:

R_{OBJ} = Tamaño del objeto [m]
R_{pixel} = Tamaño del pixel proyectado en el objeto [m/pixel]
f = distancia focal de la cámara [m]

x. Alcance máximo de transmisión (modelo Friis)

El alcance inalámbrico máximo teórico se estima considerando potencia, ganancia de antena y longitud de onda, se puede determinas mediante la ecuación 11[59]:

$$P_r = P_t G_t G_r \left(\frac{\lambda}{4\pi d}\right)^2 \quad (11)$$

Donde:

P_r = potencia recibida [W]

P_t = potencia transmitida

P_t= longitud de onda [m]

$G_t G_r$ = ganancia de las antenas [m]

λ = longitud de onda [m]

d = distancia [m]

xi. Perdida en espacio libre (FSPL)

La pérdida por espacio libre representa la atenuación que sufre una señal electromagnética cuando se propaga en un medio sin obstáculos, es decir, en "espacio libre". Este fenómeno se debe a la dispersión de la energía de la señal en todas las direcciones conforme aumenta la distancia desde la fuente. La FSPL constituye un parámetro fundamental en el análisis y diseño de sistemas de telecomunicaciones inalámbricas, ya que permite estimar la potencia recibida en función de la distancia y la frecuencia de operación. Su estudio es especialmente relevante en tecnologías como enlaces de radio, sistemas celulares, comunicaciones satelitales y redes IoT. Se expresa mediante la ecuación 12 [60]:

$$FSPL\ (dB) = 32.44 + 20log_{10}(f_{MHz}) + 20log_{10}(d_{km}) \quad (12)$$

Donde:

FSPL (dB) = perdida por espacio libre en decibelios[dB]

f_{MHz} = frecuencia de operación de la señal [MHz]

d_{km} = distancia entre transmisor y receptor [km]

XI. ARTÍCULOS RELACIONADOS

Los autores del artículo [4] presentan un robot explorador desarrollado para trabajar en entornos mineros, donde las condiciones del terreno suelen ser irregulares y representan un riesgo constante para el personal. El sistema incorpora un esquema de navegación autónoma basado en SLAM junto con sensores LiDAR, lo que permite al robot desplazarse de forma controlada sin necesidad de intervención humana directa. Este tipo de navegación resulta útil en escenarios donde el acceso es limitado y la estabilidad del entorno no siempre es predecible.

El robot cuenta con una plataforma de tracción tipo oruga, la cual mejora la estabilidad durante el desplazamiento sobre superficies complejas. Además, se implementa un sistema de control remoto mediante enlaces Wi-Fi apoyados por repetidores, lo que amplía el rango de operación y permite mantener la supervisión del equipo desde zonas seguras. Otro aspecto relevante del trabajo es el uso de software libre, ya que facilita la replicabilidad del sistema y contribuye a reducir los costos de implementación. Los resultados obtenidos muestran que este tipo de plataformas permite mejorar la eficiencia de las tareas de inspección y disminuir la exposición humana en ambientes de alto riesgo.

A partir del análisis del trabajo presentado en, los autores definen las siguientes recomendaciones técnicas, las cuales pueden servir como referencia para el diseño de sistemas robóticos exploradores con características similares:

1.- Implementación de navegación autónoma basada en SLAM, para permitir el desplazamiento seguro en terrenos inestables.

2.- Integración de sensores LiDAR para detección temprana de obstáculos físicos.
3.- Uso de plataformas de tracción tipo oruga para mejorar la adherencia y estabilidad.
4.- Empleo de enlaces Wi-Fi con repetidores para ampliar el rango operativo.
5.- Uso de software libre para facilitar la replicabilidad y reducir costos de implementación.

Para el diseño del sistema se consideraron lineamientos de seguridad operacional aplicables a maquinaria autónoma utilizada en ambientes industriales. En ese contexto, se toman como referencia los principios establecidos en la norma ISO 12100, relacionada con la seguridad de maquinaria y la evaluación de riesgos, donde se establece la necesidad de identificar peligros, evaluar riesgos y aplicar medidas de control técnico durante el proceso de diseño.

A partir de lo explicado se muestra en la Figura 21 el modelo de gestión de riesgos, los defectos que se detectaron en la inspección en la Tabla 2 y el diagrama de flujo del funcionamiento del robot en la Figura 22.

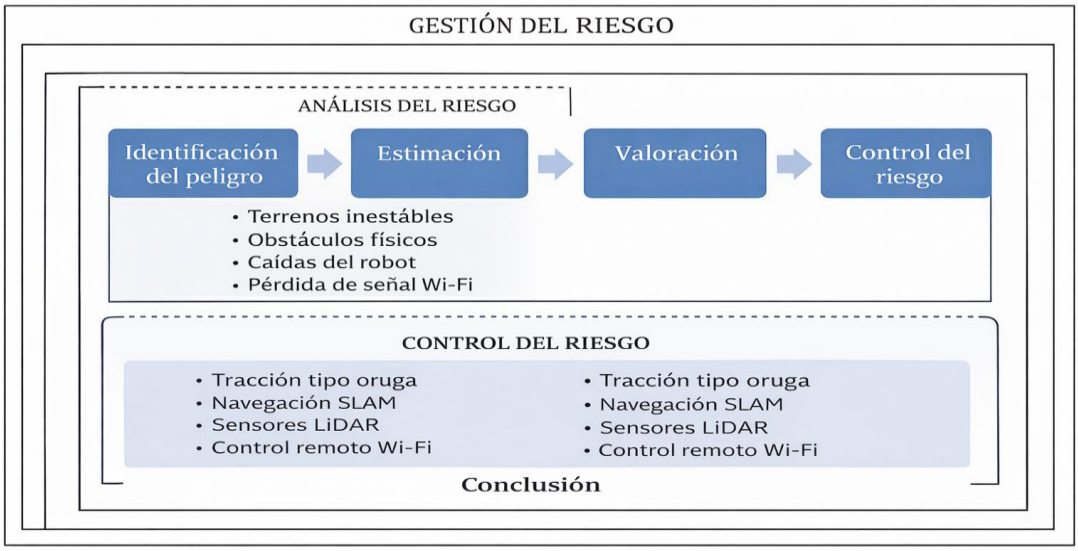


Figura 21 Modelo de gestión de riesgos del robot explorador minero Fuente: [4]

Componente	Función
Sensores LiDAR	Mapeo y detección de obstáculos
SLAM	Navegación autónoma
Tracción tipo oruga	Estabilidad
Wi-Fi	Comunicación remota
Software libre	Plataforma económica

Tabla 2 Defectos detectados en el entorno de inspección Fuente:[4]

Figura 22 Diagrama de flujo de funcionamiento del robot Fuente:[4]

El articulo concluye que el empleo de plataformas robóticas equipadas con navegación autónoma basada en SLAM, sensores LiDAR y control remoto mediante enlaces Wi-Fi constituye una alternativa técnicamente viable para la ejecución de inspecciones en ambientes mineros con condiciones de riesgo. La aplicación de modelos de gestión de riesgos contribuye a disminuir de forma directa la exposición del personal humano durante las labores de inspección, lo cual se refleja en un incremento de la seguridad operativa.

Asimismo, se observa una mejora en la eficiencia del proceso y en la consistencia de la información obtenida, aspectos que respaldan la incorporación de este tipo de sistemas en procesos de inspección industrial.

Con base en los resultados y conclusiones del trabajo presentado en, se establecen los siguientes lineamientos técnicos orientados a la implementación de plataformas robóticas para inspección en entornos peligrosos:

1.- Priorizar la navegación autónoma basada en SLAM para asegurar un desplazamiento controlado en escenarios con geometría irregular o limitada visibilidad.

2.- Incorporar sensores LiDAR como medio principal para la detección temprana de obstáculos y la construcción del entorno de operación.

3.- Implementar sistemas de control remoto mediante enlaces Wi-Fi con apoyo de repetidores, garantizando supervisión continua desde zonas seguras.

4.- Considerar modelos de gestión de riesgos durante el diseño del sistema, con énfasis en la reducción de la interacción directa entre el operador y el entorno peligroso.

5.- Evaluar la estabilidad mecánica del sistema de locomoción como factor clave para la operación segura en terrenos complejos.

En relación con el artículo [5], los autores desarrollaron un robot explorador orientado a la detección de metales en zonas mineras, especialmente en áreas con alta concentración de minerales metálicos. El sistema se basa en una plataforma de cuatro ruedas con tracción y suspensión, controlada mediante una tarjeta Arduino Mega, e incorpora módulos de comunicación inalámbrica junto con un sensor inductivo especializado en la detección de objetos metálicos. Aunque el trabajo se centró en las etapas de diseño y simulación, los resultados obtenidos muestran que es posible desarrollar plataformas robóticas funcionales empleando componentes de bajo costo y con un alto nivel de personalización, lo que refuerza la viabilidad técnica de este tipo de soluciones para tareas de inspección minera.

Entre las recomendaciones técnicas establecidas tenemos:

1.- Incorporar sensores inductivos calibrados para la detección confiable de materiales metálicos.

2.- Utilizar plataformas con suspensión y tracción para garantizar estabilidad en suelos rocosos.

3.- Emplear módulos inalámbricos que permitan transmisión remota de datos en tiempo real.

4.- Priorizar microcontroladores de arquitectura abierta como Arduino Mega para facilitar la integración de sensores.

5.- Diseñar estructuras modulares que permitan la personalización del sistema.

El diseño del sistema se apoya en principios de instrumentación electrónica aplicados a entornos industriales, con énfasis en la detección temprana de elementos metálicos presentes en zonas de riesgo. Este enfoque busca asegurar que la identificación de materiales metálicos se realice de forma oportuna y con un nivel de confiabilidad adecuado, considerando las condiciones propias del entorno minero.

De manera paralela, se adoptan criterios de seguridad operacional orientados a disminuir la exposición directa del operador durante las tareas de inspección. La automatización del proceso de detección permite que estas actividades se ejecuten sin contacto físico con áreas potencialmente peligrosas, alineándose con prácticas comunes en sistemas industriales donde la protección del personal y la continuidad operativa son prioridades durante el diseño y la operación del sistema. En la Figura 23 se presenta el diagrama de flujo de operación del robot explorador de metales y en la Tabla 3 las características técnicas del sistema:

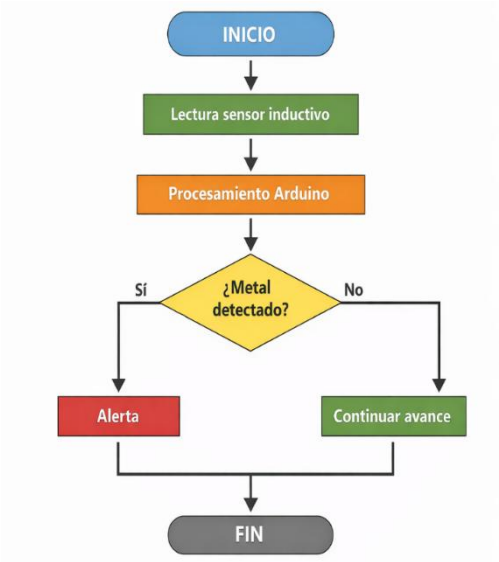


Figura 23 Diagrama de flujo de operación del robot explorador de metales Fuente: [5]

Elemento	Descripción
Plataforma	4 ruedas
Microcontrolador	Arduino Mega
Sensor	Inductivo
Comunicación	Inalámbrica
Aplicación	Detección de metales

Tabla 3. Características técnicas del sistema Fuente:[5]

Por lo cual se puede evidenciar que el desarrollo de plataformas robóticas de bajo costo equipadas con sensores inductivos permite una detección confiable de materiales metálicos en zonas mineras. Este tipo de sistemas representa una alternativa técnicamente viable que contribuye a mejorar la seguridad, reducir los tiempos de inspección y disminuir la exposición del personal humano durante las operaciones de campo .

Los autores del artículo [6] mostrado presentan un robot móvil autónomo de cuatro ruedas orientado a la optimización del consumo energético en tareas de inspección prolongadas. Los resultados reportados muestran una reducción del consumo energético desde 19.64 Wh hasta 15.8 Wh, manteniendo un desempeño estable durante la ejecución de las tareas. Si bien esta optimización generó un ligero aumento en el tiempo de recorrido, la operación general del sistema se mantuvo dentro de parámetros aceptables, priorizando un uso más eficiente de los recursos energéticos, aspecto relevante en plataformas de operación continua. Dentro de las recomendaciones técnicas que nos presentan están:

1.- Implementar sistemas de monitoreo continuo del nivel de batería.
2.- Optimizar la velocidad de desplazamiento para reducir el consumo energético.
3.- Priorizar el uso de componentes electrónicos de bajo consumo.
4.- Diseñar algoritmos de control que regulen la potencia de los actuadores.
5.- Incorporar estrategias de ahorro energético durante periodos de inactividad.

El sistema se basa en principios de eficiencia energética aplicados a plataformas móviles autónomas, con el objetivo de prolongar la autonomía durante la operación en campo. Estos criterios permiten establecer mecanismos de control orientados a reducir el consumo energético sin afectar de forma significativa el desempeño del proceso de inspección. En la Figura 24 se representa el diagrama de flujo del modelo de gestión energética aplicado, y en la Tabla 4 se detalla la comparación energética:

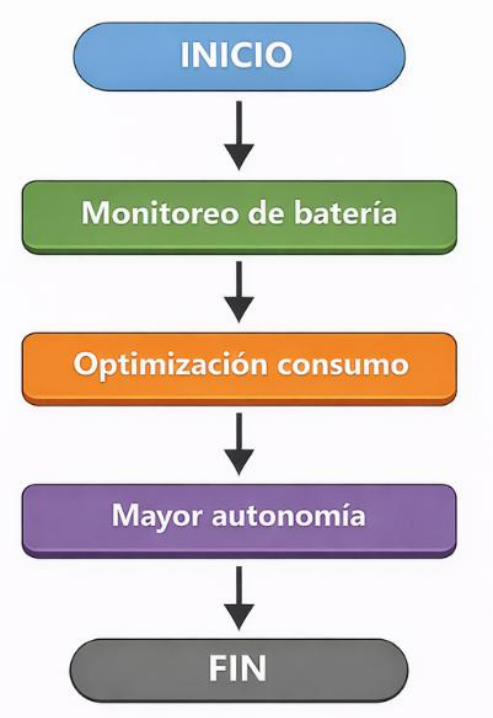


Figura 24 Diagrama de flujo del modelo de gestión energética aplicado Fuente:[6]

Consumo inicial	Consumo optimizado
19.64 Wh	15.8 Wh

Tabla 4 Comparación energética Fuente: [6]

Con base al análisis los autores del articulo previo, nos muestran que la aplicación de estrategias de optimización energética en robots móviles autónomos permite aumentar la autonomía de operación y reducir el consumo energético durante inspecciones prolongadas, favoreciendo un uso más eficiente de los recursos en tareas de campo.

Los autores del artículo [7] analizan el uso de robots aéreos, terrestres y submarinos en la inspección de infraestructuras complejas, señalando que estas plataformas representan una alternativa tecnológica eficiente para la evaluación de entornos de difícil acceso. El estudio aborda aplicaciones en edificaciones, túneles, industrias, presas y tuberías, destacando la precisión en la captura de información y la reducción del riesgo para el personal humano.

Asimismo, se evidencia que la combinación de diferentes tipos de plataformas robóticas permite cubrir una mayor variedad de escenarios de inspección. La correcta planificación de rutas y la selección adecuada del robot según el entorno influyen directamente en la calidad de los datos obtenidos y en la eficiencia del proceso de inspección. Por ello recomiendan:

1. Utilizar robots UAV para inspecciones aéreas de edificaciones y estructuras en altura.
2. Aplicar robots UGV para exploraciones terrestres en túneles y plantas industriales.
3. Emplear ROVs para inspecciones submarinas en presas, tanques y tuberías sumergidas.
4. Seleccionar la plataforma robótica adecuada según el entorno específico de inspección.
5. Programar la ruta de inspección considerando los obstáculos y condiciones del entorno.

El uso de plataformas robóticas como las que se muestran en la Tabla 5, destinadas para inspección se rige por estándares de seguridad y eficiencia, los cuales orientan la selección de la tecnología y la planificación de rutas de operación. Estos principios buscan asegurar una recolección de datos precisa y segura, considerando las particularidades de cada entorno de trabajo, ya sea aéreo, terrestre o submarino. También se puede apreciar en la Figura 25 el modelo conceptual de inspección robótica.

Plataforma	Aplicación
UAV	Inspección de edificaciones y estructuras en altura
UGV	Exploración de túneles, plantas industriales y áreas de dificil acceso
ROV	Inspección en entornos submarinos como presas, tanques y tuberias sumergidas

Tabla 5 Plataformas robóticas empleadas Fuente:[7]

Figura 25 Modelo conceptual de inspección robótica Fuente: [7]

Finalmente podemos decir que los resultados demuestran que la implementación de robots UAV, UGV y ROV per-mite realizar inspecciones en infraestructuras complejas de manera segura y precisa, optimizando el tiempo y los costos, y reduciendo significativamente la exposición humana en entornos peligrosos. Estos sistemas se consolidan como soluciones versátiles y eficaces para la evaluación de infraestructuras en difícil acceso.

XII. METODOLOGÍA

a. Prototipo de un robot explorador para inspección de infraestructuras

La investigación se desarrolló bajo un enfoque mixto, combinando métodos cuantitativos y cualitativos para evaluar el desempeño de un prototipo robótico destinado a la inspección de cableado eléctrico y de telecomunicaciones en espacios confinados. El análisis cuantitativo permitió medir la presión del sistema de visión artificial, el tiempo de respuesta y la confiabilidad de la transmisión de datos, mientras que el análisis cualitativo evalúa el comportamiento operativo del robot y la calidad de la información visual obtenida en condiciones reales de inspección.

La metodología aplicada correspondió al desarrollo tecnológico de tipo experimental mediante prototipado evolutivo, estructurado en las fases de diseño, construcción y prueba-refinamiento, lo que facilitó la integración progresiva de los componentes de hardware y software, así como la mejora continua del sistema a lo largo del proceso de desarrollo. El estudio se apoyó en los métodos experimental y analítico-sintético, permitiendo validar el funcionamiento del prototipo mediante pruebas controladas con cables dañados y el análisis individual de cada subsistema antes de su integración final.

El prototipo desarrollado consiste en una plataforma robótica móvil basado en microcontrolador ESP32 y comunicación Lora, diseñados para operar de forma remota y transmitir información visual durante recorridos de inspección, constituyéndose en una herramienta eficiente para el mantenimiento predictivo de infraestructuras eléctricas y de telecomunicaciones.

En la Figura 26 se muestra el diagrama de bloques funcional del sistema, el cual describe el flujo general de la operación del prototipo desde la entrada de información hasta la generación de salida, permitiendo identificar la disposición real de los módulos electrónicos y mecánicos.

En la Figura 27 se muestra el esquema del prototipo robótico, donde se representan los componentes físicos del sistema y su interconexión, permitiendo identificar la disposición real de los módulos electrónicos y mecánicos.

El sistema está conformado por los siguientes componentes:

Input

A. Cableado dañado captado por el sensor HuskyLens

Procesamiento y Comunicación

B. HuskyLens (Sensor de visión AI)
C. ESP32-CAM
D. Esp32 (Receptor)
E. Plataforma robótica móvil
F. Módulo LoRa (Receptor)
G. Módulo LoRa (Transmisor)
H. ESP32 (Transmisor)
I. Dispositivo de control Joystick analógico

Output

J. Imagen (capturas tomadas por ESP32-CAM)
K. Video (Tomado por ESP32-CAM)
L. Red Wi-Fi
M. Laptop o Pc + Servidor local Node.js (Visual Studio Code)
N. Video en tiempo real
O. Alerta del sistema
P. Led (Sistema de alerta)

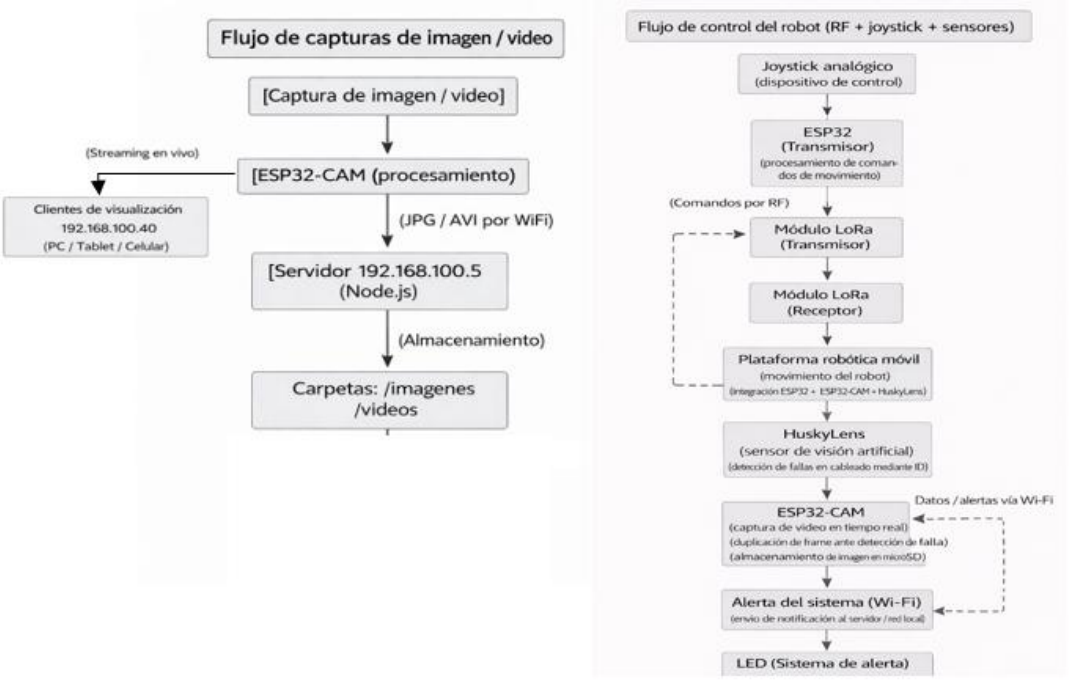


Figura 26 Diagrama de bloques funcional del sistema de robot explorador Fuente: Autor

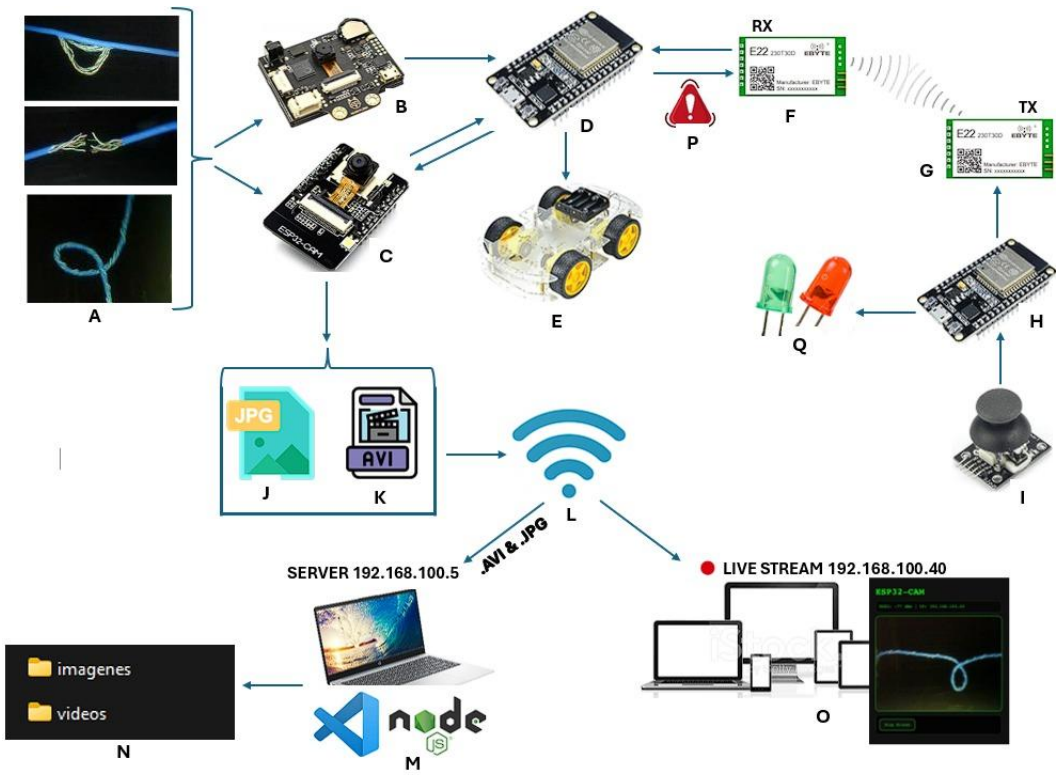


Figura 27 Esquema del prototipo de Robot explorador Fuente: Autores

Entorno para realizar pruebas

• Las pruebas se realizaron en una vivienda ubicada en la ciudadela Huancavilca Norte, además de entornos académicos como por ejemplo el laboratorio de electrónica analógica, ubicado en el bloque E de la Universidad Politécnica Salesiana, sede Guayaquil.

• Se eligieron áreas interiores con características similares a condiciones reales de operación, principalmente zonas técnicas y áreas confinadas (cielos falsos).

Recursos utilizados

•Microcontroladores ESP32

•Módulos LoRa E32

•HuskyLens

• ESP32-CAM

• Plataforma robótica

• Sistema de alimentación portátil

• Laptop para servidor local

• Cableado y accesorios de conexión

Instalación del sistema

La instalación del sistema se llevó a cabo considerando tanto el montaje físico del prototipo como la correcta configuración de los elementos de comunicación y procesamiento, con el fin de asegurar un funcionamiento estable antes de iniciar las pruebas como:

- Montaje de los módulos electrónicos sobre el chasis del robot.
- Integración de los módulos de comunicación LoRa entre el transmisor y el receptor.
- Configuración de la red Wi-Fi local.
- Instalación del servidor Python para recepción y almacenamiento de datos.
- Verificación del correcto funcionamiento de sensores, motores y módulos de visión.

Recopilación de datos

La recopilación de datos se realizó durante recorridos controlados para simular condiciones reales de operación.

- Se capturaron imágenes y video en tiempo real mediante la HuskyLens y ESP32-CAM.
- Se almacenaron evidencias visuales en el servidor local.
- Se registraron eventos de detección y alertas generadas por el sistema.

Análisis de datos

El análisis se centró en evaluar la información obtenida durante las pruebas, así como el comportamiento general del sistema de comunicación y respuesta.

- Se analizaron las imágenes y videos recolectados.
- Se identificaron zonas con posibles daños o irregularidades.
- Se evaluó la estabilidad del enlace LoRa y Wi-Fi.

b. Base estructural del Robot

Para el desarrollo del prototipo robótico se utilizó inicialmente un kit robótico educativo como base mecánica; sin embargo, debido a que la estructura original no cumplía con los requerimientos de espacio, estabilidad y distribución de peso del sistema implementado, fue necesario realizar un proceso de rediseño y mejora del chasis, mediante la construcción de una nueva estructura de policarbonato de doble nivel.

La base estructural diseñada que se muestra en la Figura 28 cumple la función de chasis principal del robot proporcionando estabilidad mecánica, protección a los circuitos electrónicos y una correcta distribución del peso lo cual es fundamental para asegurar un desplazamiento uniforme, evitar vibraciones y prevenir posibles vuelcos durante la operación.

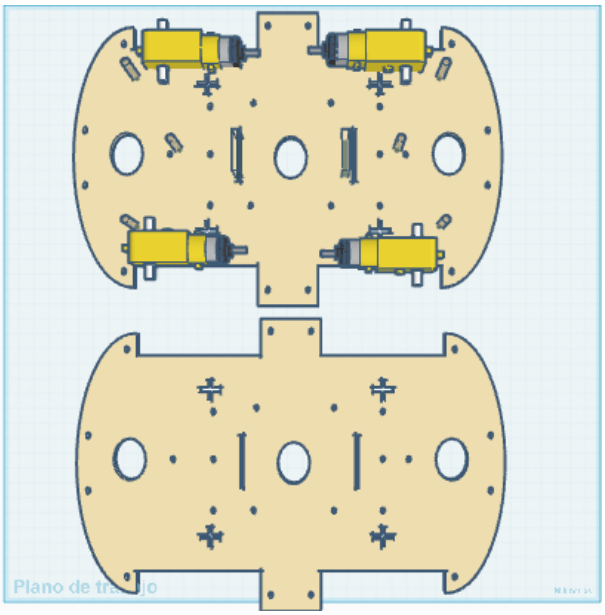


Figura 28 Base estructural diseñada Fuente: Elaboración propia (Tinkercad)

Las dimensiones de las nuevas bases estructurales fueron definidas y modeladas en el entorno Tinkercad a partir de un kit robótico existente, los ajustes se realizaron considerando el tamaño real de los componentes a instalar, los cuales se detallan en la Tabla 6 y el espacio necesario para el cableado, ventilación y fijaciones mecánicas. Para ello se realizó una medición física de los módulos principales:

Componente	Dimensión aproximada
ESP32 Cam	2.7 x 4.05 x 0.45 cm
Protoboard (Esp32, Driver TB6612FNG, antena LoRa)	9.5 x 16.5
HuskyLens	*5 × 5 × 3 cm*
Motores DC (cada uno)	**7 cm x 2.2 cm x 1.8 cm**

Tabla 6 Dimensiones de los componentes Fuente: Autores

Se diseñaron plataformas rectangulares de 23 cm X 26 cm, como se puede observar en la Figura 29 lo que permitió una adecuada distribución de los componentes, espacio para el montaje del proto Board y futuras ampliaciones del sistema.

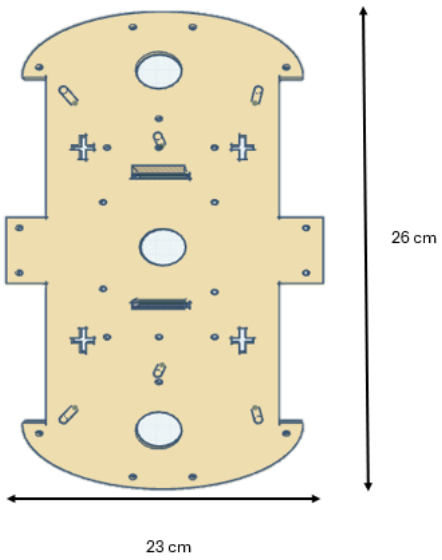


Figura 29 Diseño de plataformas rectangulares de 23 x 26 Fuente: Medición propia

Para la construcción de las nuevas bases se seleccionó policarbonato de alta densidad, debido a que presenta las siguientes ventajas:

- Peso reducido
- Alta resistencia al impacto
- Buen aislamiento eléctrico
- Facilidad de perforación y corte
- Adecuada resistencia mecánica para el prototipo

El chasis final está conformado por dos bases de policarbonato:

En la base inferior se encuentran montados:

- Motores DC y ruedas provenientes del kit
- Power Bank
- Banco de 3 pilas AA
- Soportes de fijación estructural

Como se observa en la Figura 30, el respectivo diseño de base y montaje de componentes. Esta base concentra la mayor masa del sistema, permitiendo mantener el centro de gravedad bajo y mejorar la estabilidad del robot durante su desplazamiento.

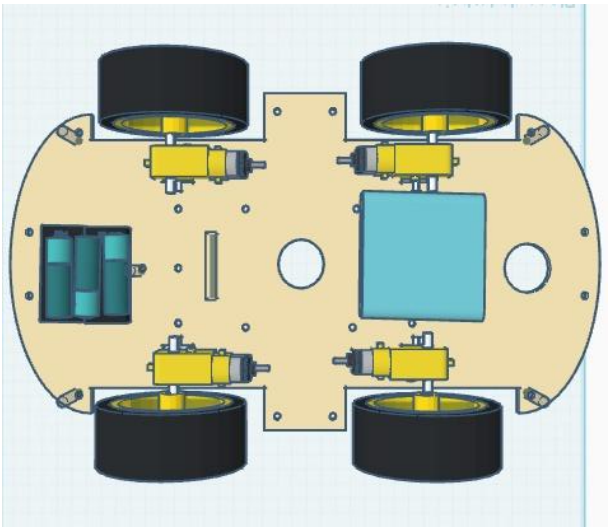


Figura 30 Diseño de base inferior y montaje de respectivos componentes Fuente: Tinkercad

La base superior que se observa en la Figura 31, fue diseñada específicamente para albergar:

- Protoboard principal del sistema (ESP 32, driver TB 6612 FNG, módulo LoRa E32)
- Módulo de visión HuskyLens
- Módulo ESP32-CAM

Por motivos de la plataforma donde fue diseñado el modelo no contaba con el módulo Lora , aún así dicho componente sí estará en el prototipo real.

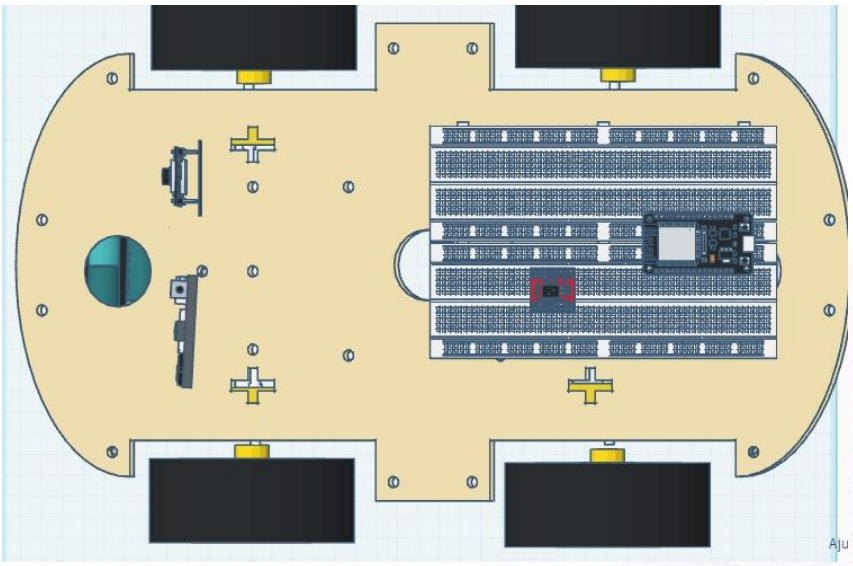


Figura 31 Diseño de base superior para albergar microcontroladores y sensores Fuente: Tinkercad

Ambas bases se encuentran Unidas mediante columnas separadoras plásticas y formando una estructura rígida de doble nivel.

La implementación del chasis personalizado permitió:

- Aumentar la capacidad de carga
- Mejorar la estabilidad
- Protege los componentes electrónicos
- Incrementar el espacio disponible para módulos
- Facilitar el mantenimiento y futuras ampliaciones del sistema

c. Sistema Transmisor

El sistema transmisor es el punto de interacción entre el usuario y el vehículo móvil, siendo responsable de la generación y transmisión de los comandos de control. Para este propósito se seleccionó el microcontrolador ESP32, debido a su arquitectura de 32 bits, capacidad de procesamiento en tiempo real, múltiples interfaces de comunicación y bajo consumo energético, características que lo hacen idóneo para aplicaciones de control remoto y sistemas embebidos.

Previo a su utilización, el microcontrolador ESP32 fue sometido a un proceso de limpieza y restauración del firmware, con el fin de eliminar configuraciones o programas previamente cargados que pudieran interferir con el correcto funcionamiento del sistema. Este procedimiento permitió garantizar un entorno de ejecución limpio y confiable; el proceso se describe con mayor detalle en el Anexo 1. Antes del ensamblaje, se realizó una verificación individual de cada uno de los componentes electrónicos, comprobando la continuidad de las conexiones, el correcto estado de los pines y la compatibilidad de niveles de voltaje.

El módulo joystick fue conectado a las entradas analógicas GPIO34 y GPIO35, permitiendo la lectura de los desplazamientos en los ejes X y Y .Para la transmisión inalámbrica de datos se integró un módulo LoRa E32, el cual emplea comunicación UART utilizando los pines GPIO16 y GPIO17 del ESP32. La tecnología LoRa fue seleccionada por su amplio alcance , alta inmunidad al ruido y bajo consumo energético, lo que permite operar el sistema en entornos abiertos o rurales donde las redes convencionales presentaban limitaciones La configuración del módulo LoRa incluyendo la selección del modo de operación y los parámetros de comunicación se detalla en el Anexo 2.

El microcontrolador procesa las lecturas del joystick mediante un algoritmo que establece un punto central y un margen de tolerancia, con el fin de filtrar pequeñas variaciones y ruido eléctrico. De este modo, se puede generar comandos de movimiento claramente definidos, los cuales son transmitidos únicamente cuando se detecta un cambio real en la posición del joystick. Este mecanismo reduce el tráfico de datos, evita la saturación del enlace LoRa y mejora la confiabilidad de la comunicación. El código fuente que implementa este algoritmo se presenta en el Anexo 4. En la Tabla 7 se muestran todas las conexiones a realizar, en la Figura 32 se muestra el diagrama de conexiones del prototipo transmisor y en la Figura 33 se muestra el montaje experimental de un sistema de control inalámbrico basado en ESP32.

Componente	Pin del dispositivo	Pin ESP32	Función
Joystick eje X	VRx	GPIO 34	Lectura analógica eje X
Joystick eje Y	VRy	GPIO 35	Lectura analógica eje Y
Pulsador joystick	SW	GPIO 14	Entrada digital (Stop)
Joystick VCC	+5 V	3.3 V	Alimentación del joystick
Joystick tierra	GND	Tierra común (Protoboard)	Referencia a tierra
LoRa E32	M0	Tierra común (Protoboard)	Configuración de modo normal
LoRa E32	M1	Tierra común (Protoboard)	Configuración de modo normal
LoRa E32	TX	GPIO 16	Recepción de datos
LoRa E32	RX	GPIO 17	Transmisión de datos
LoRa E32	AUX	GPIO 18	
LoRa E32	VCC	3.3 V	Alimentación
LoRa E32	GND	GND	Referencia a tierra
ESP32	GND	Tierra común (protoboard)	Referencia de tierra del sistema

Tabla 7 Conexiones del módulo transmisor Fuente: Autores

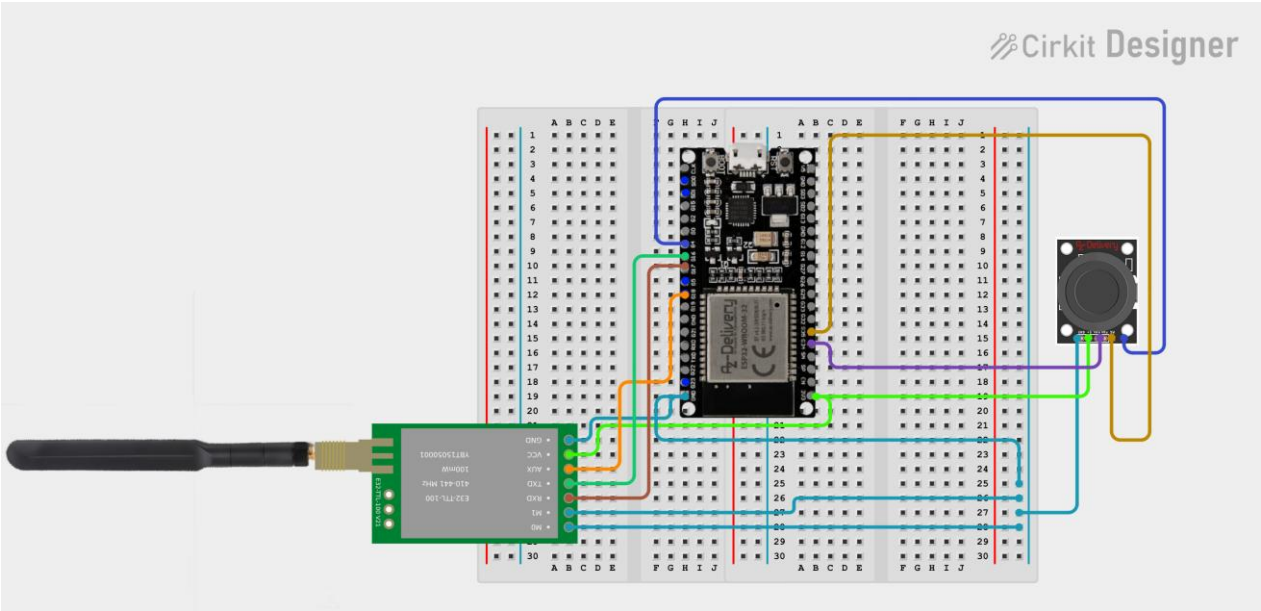


Figura 32 Diagrama de conexiones del prototipo transmisor Fuente: Autores

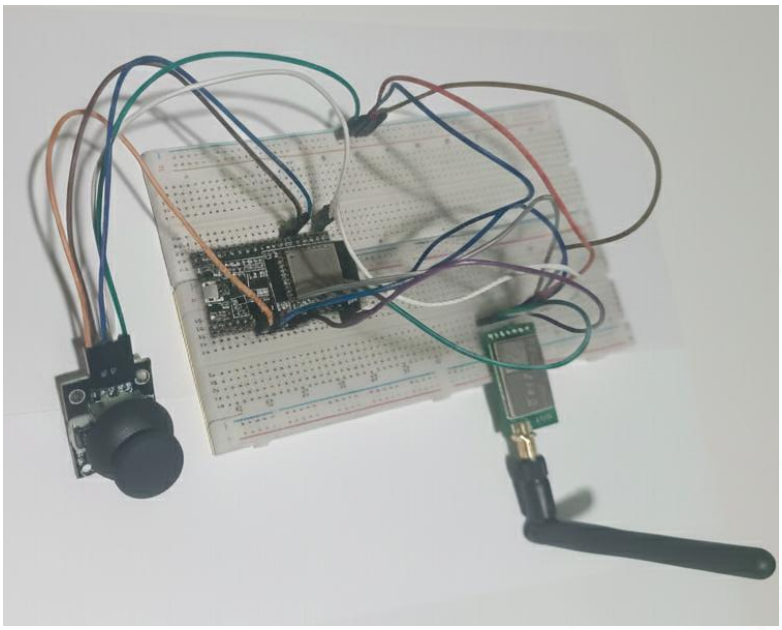


Figura 33 Montaje experimental de un sistema de control inalámbrico basado en ESP32 Fuente: Autores

d. Sistema Receptor

El sistema receptor se construyó a partir de un segundo microcontrolador ESP32 al cual también se le realiza la debida limpieza del firmware, Con el objetivo de garantizar un entorno de ejecución estable y confiable; dicho procedimiento se detalla en el Anexo 1. Este componente se encarga de recibir los comandos enviados desde el transmisor y ejecutar las acciones correspondientes sobre el sistema de tracción del prototipo y además el microcontrolador actúa como el centro de operación del robot, ya que desde él se gestionan las tareas de control del movimiento, las funciones básicas de seguridad y la comunicación interna del sistema. El código fuente implementado para el funcionamiento del sistema receptor se presenta en el Anexo 5. En la Figura 34 se puede observar la construcción del sistema receptor.

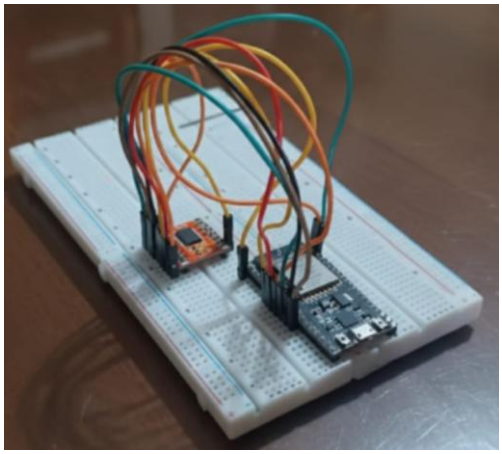


Figura 34 Construcción del sistema receptor Fuente: Autores

El control de los motores de corriente continua se realiza mediante el driver TB6612FNG que se observa en la Figura 35, el cual permite controlar dos canales de motores de forma independiente. El canal A acciona los motores del lado izquierdo y el canal B los del lado derecho, permitiendo la ejecución de movimientos como avance, retroceso, y giros diferentes.

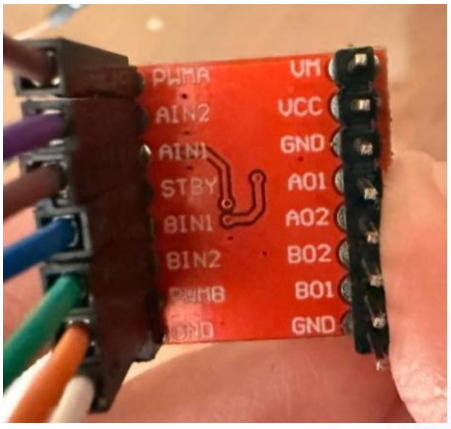


Figura 35 Driver TB6612FNG Fuente: Autores

La recepción de los comandos inalámbricos se realiza a través del módulo Lora E32, configurado para operar como receptor. Los parámetros de configuración y el modo de operación del módulo la hora se describe en el Anexo 2, manteniendo coherencia con el sistema transmisor.

La regulación de la velocidad de los motores se realiza mediante señales PWM(modulación por ancho de pulso), utilizadas para variar la potencia aplicada durante el movimiento del vehículo. Este método permite que los cambios de velocidad no sean bruscos, lo que ayuda a mantener un desplazamiento más estable y a reducir esfuerzos mecánicos innecesarios en el sistema de tracción.

Como medida adicional, se incorporó un mecanismo de seguridad tipo failsafe. Este sistema supervisa la recepción de comandos y detiene automáticamente los motores cuando no se detecta ninguna orden durante un intervalo mayor a 1500 ms. De esta manera, se evita que el vehículo continúe en movimiento ante pérdidas de señal, interferencias o fallos en el enlace de comunicación, asegurando una operación más segura del prototipo. En la Figura 36 se puede observar las conexiones del sistema receptor, y en la Tabla 8 se muestra que conexiones nomas se utilizó y en la Figura 37 se muestra el diagrama de conexiones correspondiente.

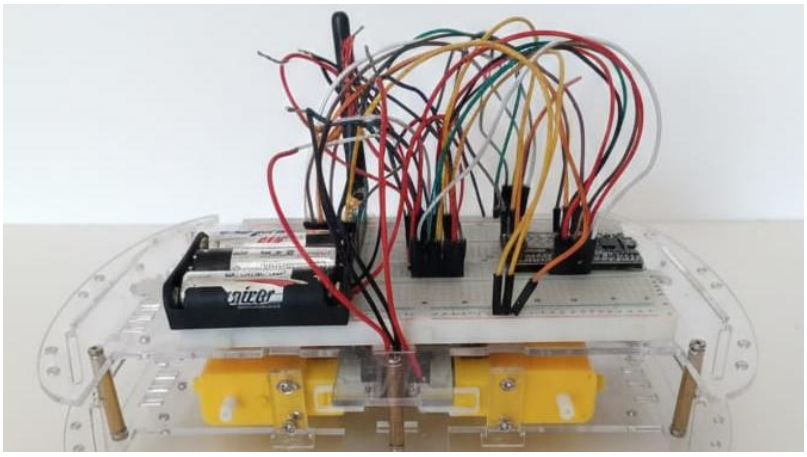


Figura 36 Montaje y conexiones del sistema receptor Fuente: Autores

Componente	Pin	Pin ESP32	Función
LoRa E32	TX	GPIO 16	Recepción de comandos
LoRa E32	RX	GPIO 17	Transmisión de comandos
LoRa E32	AUX	GPIO 18	Pin auxiliar de estado
LoRa E32	M0	Tierra común	Selección de modo normal
LoRa E32	M1	Tierra común	Selección de modo normal
LoRa E32	VCC	3.3 común	Alimentación del módulo
LoRa E32	GND	Tierra común	Referencia a tierra
TB6612	AIN1	GPIO 32	Control motor izquierdo
TB6612	AIN2	GPIO 33	Control motor izquierdo
TB6612	PWMA	GPIO 25	PWM motor izquierdo
TB6612	BIN1	GPIO 4	Control motor derecho
TB6612	BIN2	GPIO 27	Control motor derecho
TB6612	PWMB	GPIO 26	PWM motor derecho
TB6612	STBY	GPIO 23	Habilitación del driver
TB6612	VM	6–9 V	Alimentación de motores
TB6612	VCC	3.3 común	Alimentación del driver
TB6612	GND	Tierra común	Referencia a tierra
Motor izquierdo	A01	TB6612 A01	Negativo motor delantero izquierdo/ positivo motor trasero izquierdo
Motor izquierdo	A02	TB6612 A02	Negativo motor trasero izquierdo/ positivo motor delantero izquierdo
Motor derecho	B01	TB6612 B01	Negativo motor delantero derecho/ positivo motor trasero derecho
Motor derecho	B02	TB6612 B02	Positivo motor delantero derecho/ negativo motor trasero derecho

Tabla 8 Conexiones del sistema receptor Fuente: Autores

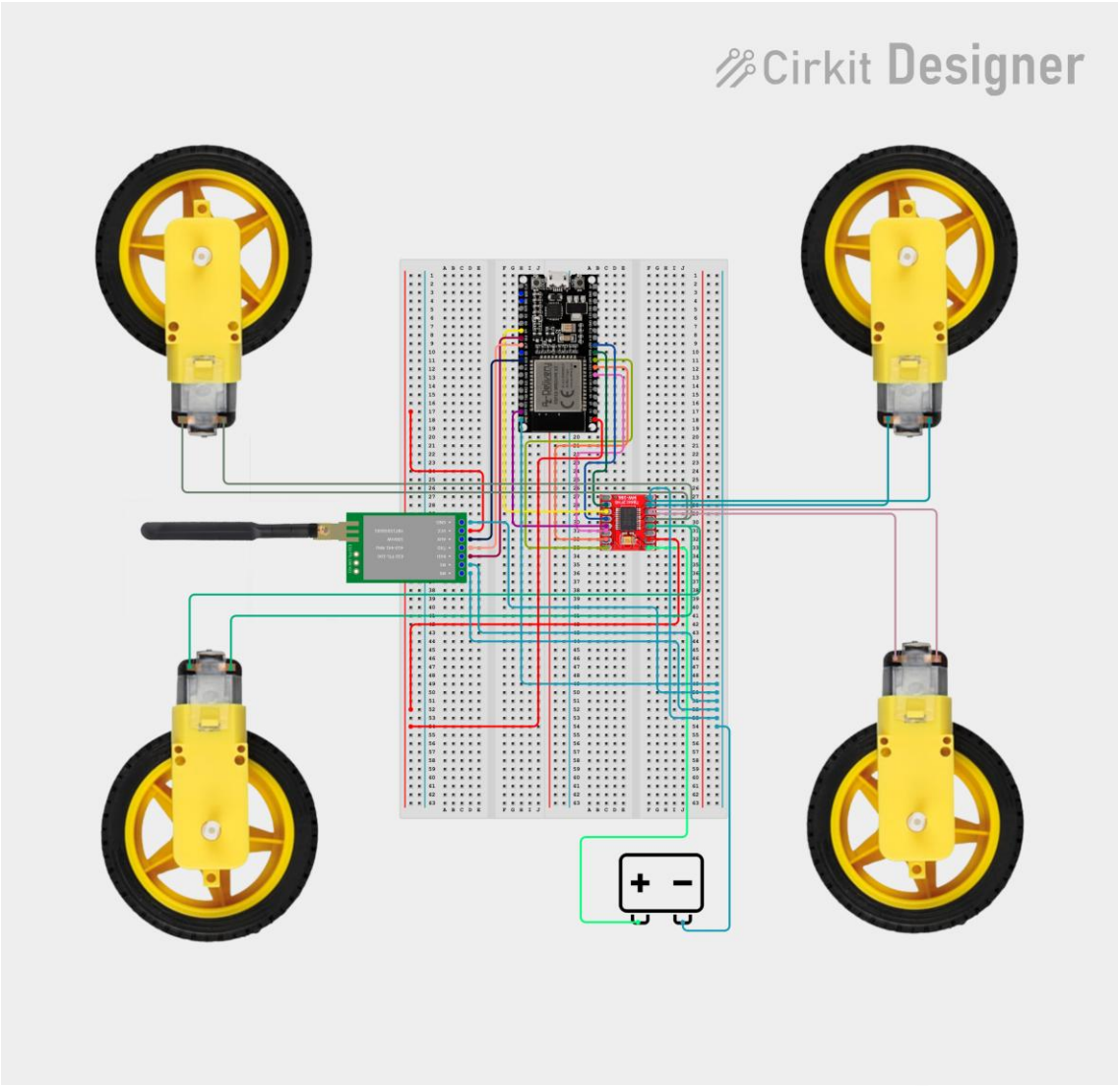


Figura 37 Diagrama de conexiones del sistema receptor Fuente: Autores

e. Sistema de visión artificial

El sistema incorpora un módulo HuskyLens para añadir capacidades de percepción visual al prototipo. Este dispositivo se emplea para la detección y seguimiento de objetos previamente entrenados, lo que permite ampliar las funciones del robot durante las tareas de inspección.

La comunicación entre el HuskyLens y el ESP32 receptor se realiza mediante el protocolo I²C, utilizando los pines GPIO 21 y GPIO 22. Esta interfaz fue seleccionada por su simplicidad y estabilidad en la transmisión de datos entre dispositivos embebidos.

Para evitar detecciones inestables, se aplicó un criterio de validación temporal, es decir que un identificador se considera válido únicamente cuando permanece constante durante un intervalo continuo de 5 segundos. Este enfoque permite descartar lecturas generadas por variaciones de iluminación, sombras o movimientos bruscos del entorno.

El proceso de entrenamiento del módulo HuskyLens, así como la definición de los identificadores de asociados a cada tipo de daño en los cables, se describen detalladamente en el Anexo 3. En la Figura 38 se muestra el diagrama de conexión del sistema de visión artificial, mientras que en la Tabla 9 se detallan las conexiones eléctricas del módulo HuskyLens con el ESP32.

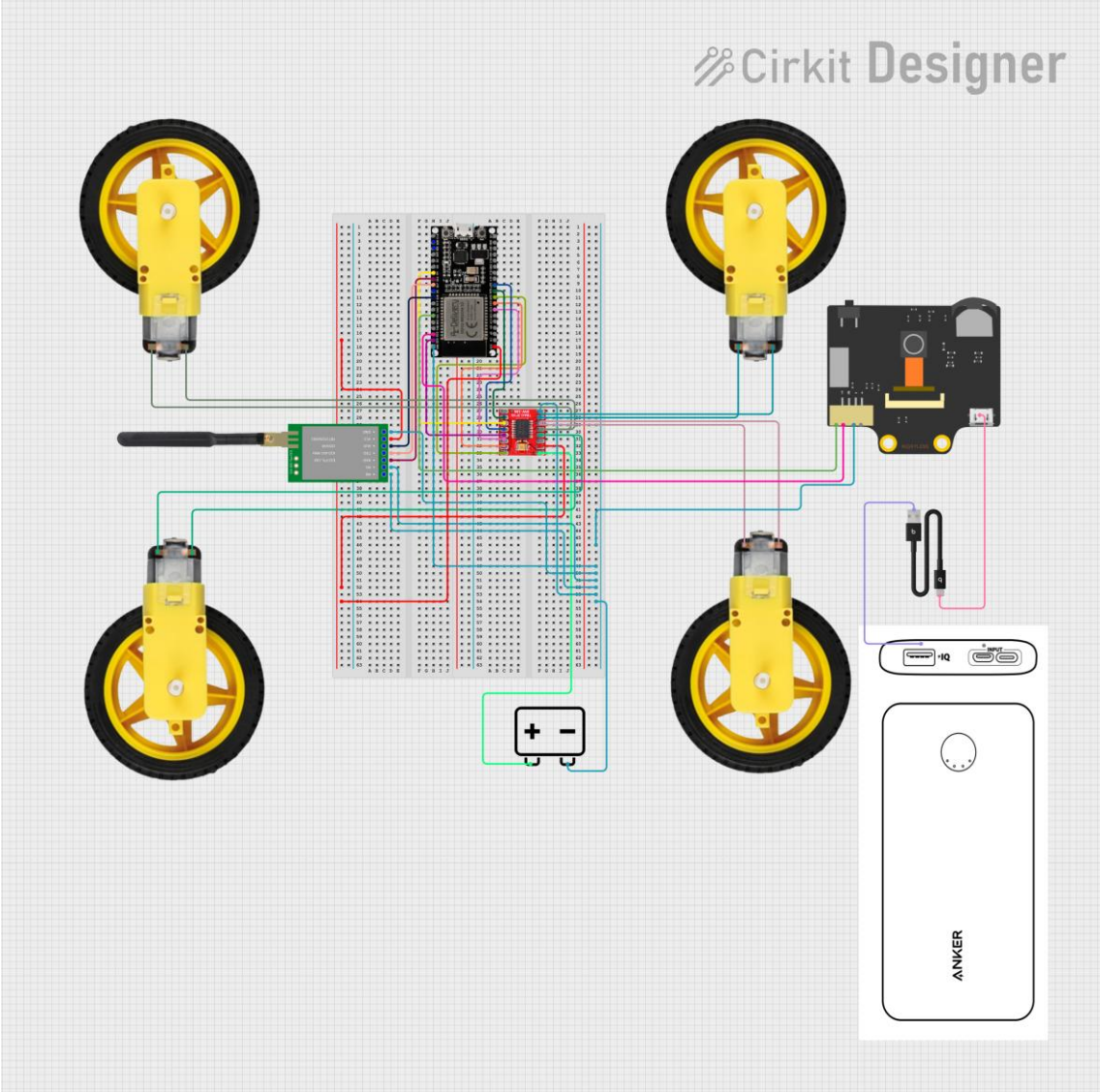


Figura 38 Diagrama de conexión del sistema de visión artificial Fuente: Autores

HuskyLens	ESP32	Función
SDA	GPIO 21	Datos I²C
SCL	GPIO 22	Reloj I²C
VCC	5 V	Alimentación
GND	GND	Tierra

Tabla 9 Conexiones del módulo HuskyLens Fuente. Autor

f. Sistema de monitoreo y almacenamiento en servidor local.

Cuando el sistema confirma una detección válida, se activa un mecanismo de alerta a través de la conexión Wi-Fi. En este punto, el ESP32 receptor se conecta a la red local y envía una solicitud HTTP al módulo ESP32-CAM, encargado de capturar imágenes y transmitir video del entorno.

Para la visualización en tiempo real, se desarrolló una página web, alojada directamente en el módulo ESP32-CAM, la cual permite acceder al flujo de video mediante un navegador web dentro de la misma red, facilitando el monitoreo remoto del área inspeccionada como se aprecia en la Figura 39. Adicionalmente, con el fin de analizar y verificar el tráfico de datos generado durante el proceso de transmisión entre los dispositivos, se empleó la herramienta Wireshark, lo que permitió observar los paquetes intercambiados, los protocolos utilizados y el comportamiento de la comunicación de red, así lo podemos observar en la Figura 40. El código fuente desarrollado para el funcionamiento del módulo ESP32-CAM encargado de la captura y transmisión de imágenes y vídeo, se presenta en el Anexo 6.

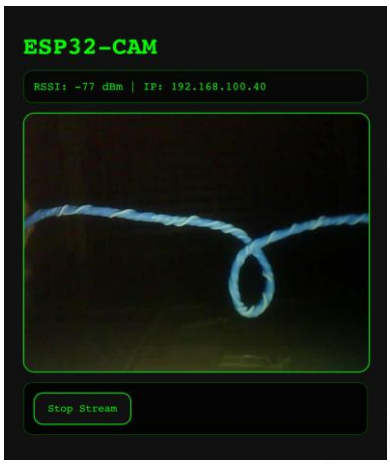


Figura 39 Visualización del flujo de video MJPEG capturado por el ESP32-CAM Fuente: Autor

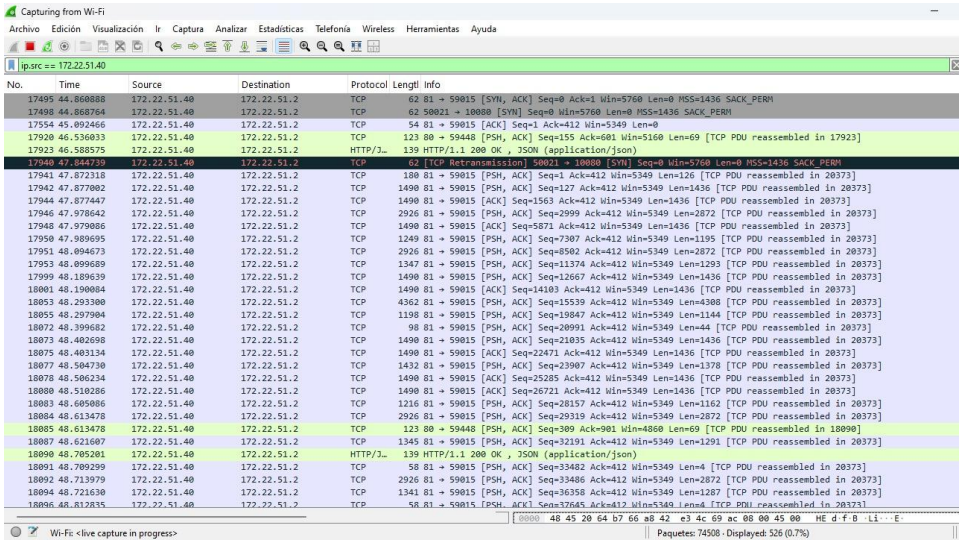


Figura 40 Tráfico de datos generado durante el proceso de transmisión entre los dispositivos Fuente: Autor

Estos datos multimedia se desarrollaron en un servidor local basado en Node.js, el cual se ejecuta sobre una red local y permanece a la escucha de solicitudes HTTP con un puerto definido (1080), el servidor se implementó utilizando el framework Express, debido a su ligereza y facilidad para manejar peticiones entrantes en aplicaciones de tiempo real.

El servidor se inicializa mediante la ejecución del archivo principal Server. JS, así se ve en la Figura 41, el cual configura las rutas necesarias para la recepción de imágenes individuales y flujos de video. Una vez levantado el servicio, este queda disponible para aceptar conexiones entrantes desde los dispositivos ESP 32 involucrados en el sistema.

Las imágenes capturadas por el ESP32- CAM se envían mediante solicitudes HTTP POST y son almacenadas automáticamente en el servidor, asignándoles un nombre único basado en una marca temporal, lo que permite identificar el momento exacto de cada captura y facilita su análisis posterior.

De manera similar, los flujos de video transmitidos en formato MJPEG Son recibidos por el servidor y redirigidos a la herramienta FFmpeg, la cual se encarga de convertirlos en archivos AVI sin aplicar compresión adicional. Este proceso permite conservar la calidad original y del video y realizar el almacenamiento de forma eficiente.

Adicionalmente el servidor incorpora un mecanismo de monitoreo del ancho de banda, el cual calcula la cantidad de datos recibidos por segundo durante la transmisión. Esta información se utiliza para observar el comportamiento del enlace de comunicación durante las pruebas experimentales y verificar que la transmisión de imágenes y vídeos se mantenga estable.

Finalmente el servidor ofrece acceso local a los archivos almacenados permitiendo la visualización y descarga de los vídeos e imágenes generadas durante las pruebas del sistema.

El proceso de levantamiento, configuración y funcionamiento interno del servidor, así como el manejo de imágenes y flujos de video, se describe con mayor detalle en el Anexo 7.

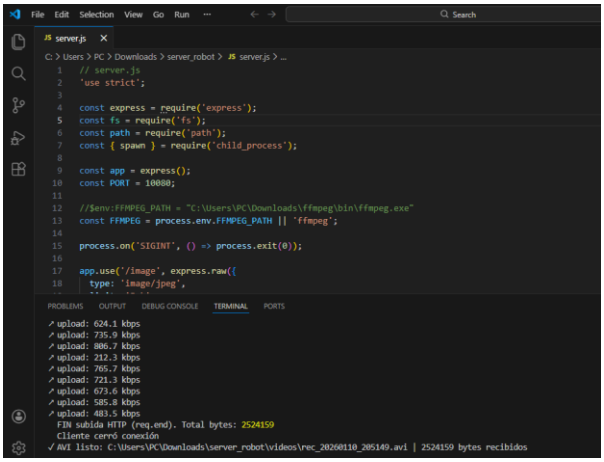


Figura 41 Servidor local Node.js para recepción y almacenamiento de imágenes y video Fuente: Autor

g. Datos obtenidos

- **Tumbado falso de vivienda – Sistema de inspección visual automatizada**

En esta sección se presentan los resultados obtenidos durante la ejecución de las pruebas del sistema de inspección visual automatizada implementado mediante HuskyLens y ESP32-CAM, aplicado a los entornos del tumbado falso de una vivienda y un laboratorio del bloque E de la Universidad Politécnica Salesiana. Las pruebas se realizaron bajo condiciones reales no controladas, caracterizadas por iluminación variable, presencia de polvo, fondos oscuros y estructuras de madera, condiciones típicas en este tipo de entornos.

El objetivo de esta etapa experimental fue evaluar el desempeño del sistema en la detección y clasificación ,implementadas mediante visión artificial, así como también la transmisión de información asociada a fallas reales de cableado eléctrico y de telecomunicaciones, considerando métricas de precisión, exactitud de clasificación, tiempo de respuesta y confiabilidad de transmisión.

Se estableció un procedimiento experimental basado en la captura controlada de imágenes en tres distancias de inspección: 15 cm, 30 cm y 45 cm. Estas distancias fueron seleccionadas debido a que, durante las pruebas preliminares, el módulo HuskyLens presentaba un desempeño adecuado en los rangos de 15 cm y 30 cm; sin embargo, a una distancia de 45 cm se evidenció una disminución significativa en la capacidad de detección, presentando un mayor número de clasificaciones erróneas.

Tipos de fallas evaluadas

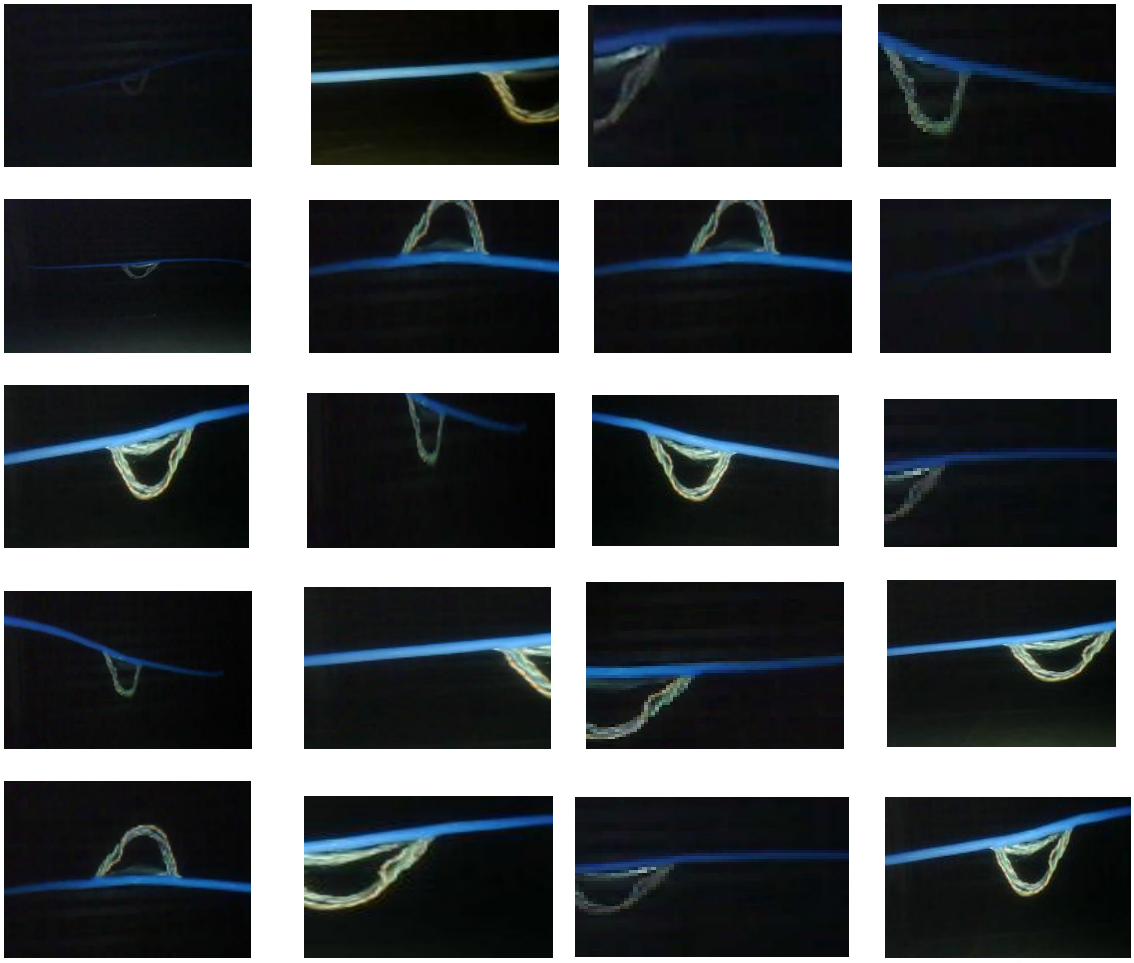
Durante la etapa de diseño del prototipo, con el fin de evaluar el desempeño del sistema en escenarios controlados y representativos. Se determinó trabajar únicamente con seis tipos de fallas en el cableado las cuales son presentadas en la Tabla 10, debido a que, durante las etapas iniciales de entrenamiento, se observó que al incrementar el número de daños el módulo HuskyLens presentaba mayores niveles de confusión, afectando negativamente la precisión del reconocimiento. Por este motivo, se seleccionaron aquellas fallas más representativas y claramente diferenciables.

ID	Tipo de falla
F1	UTP expuesto
F2	UTP seccionado
F3	UTP perforado con clavo
F4	UTP entorchado
F5	Cable eléctrico mal encintado
F6	Rotura de núcleo en cable eléctrico

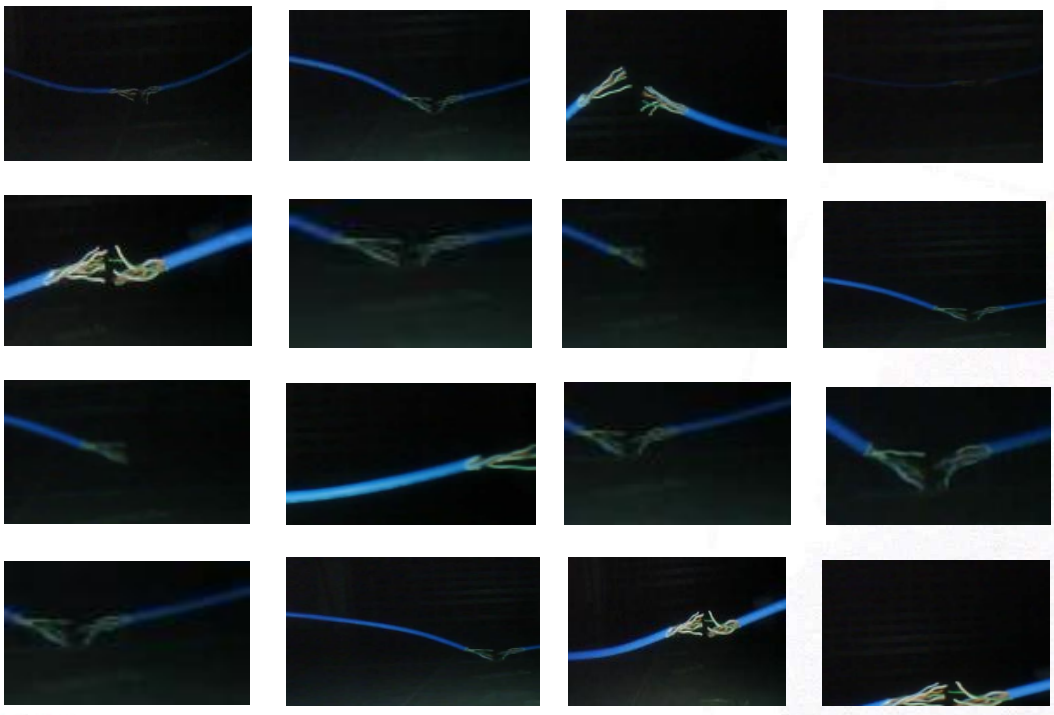
Tabla 10 Fallas identificadas Fuente: Autor

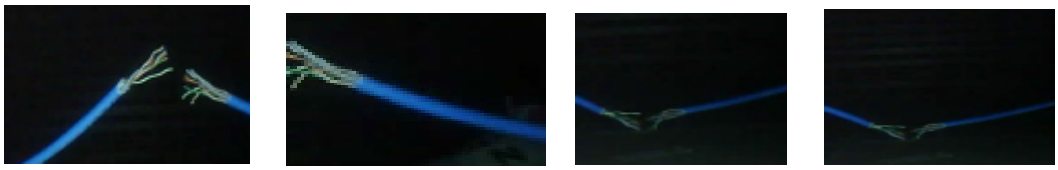
En cada distancia se analizaron seis tipos distintos de fallas, capturándose un total de 20 imágenes por tipo, lo que dio como resultado 120 imágenes por distancia y 360 imágenes en total durante el proceso experimental. Cada imagen fue procesada automáticamente por el sistema y clasificada según el tipo de falla previamente entrenado.

Pruebas realizadas Falla 1 : UTP Cableado expuesto

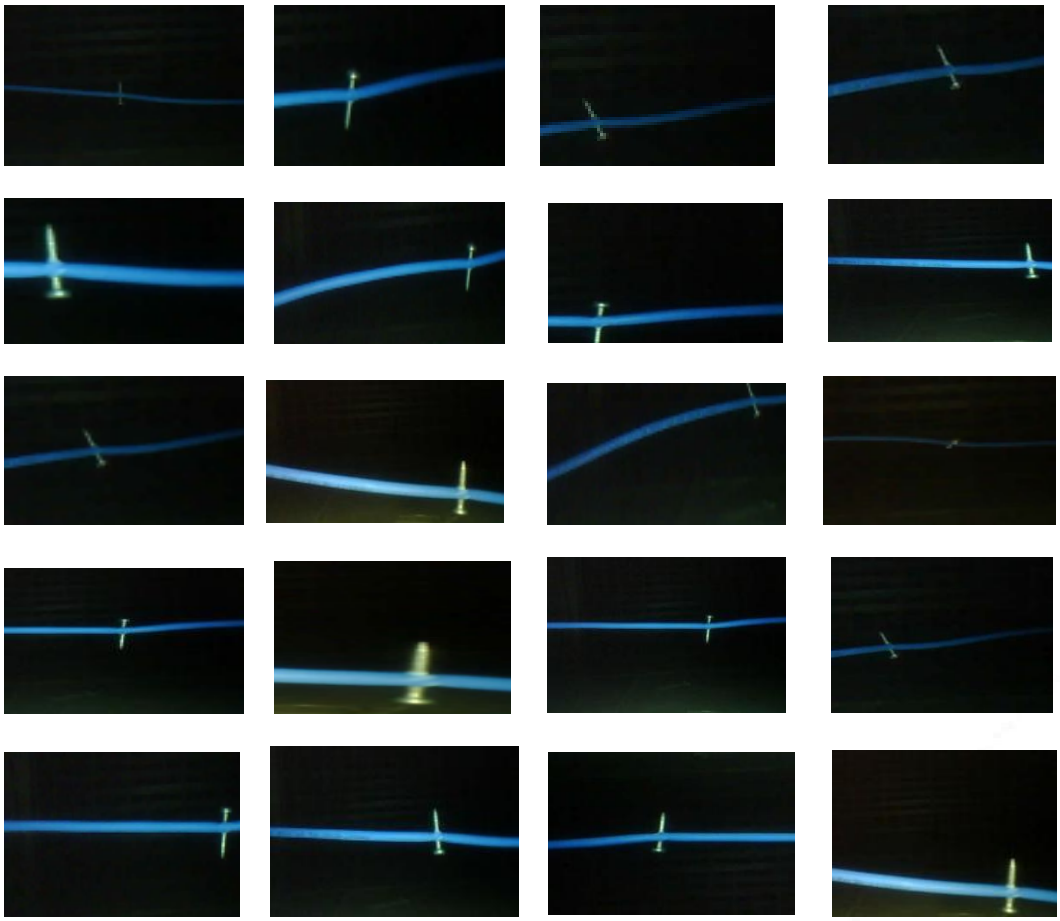


Pruebas realizadas Falla 2 : Cable UTP seccionado

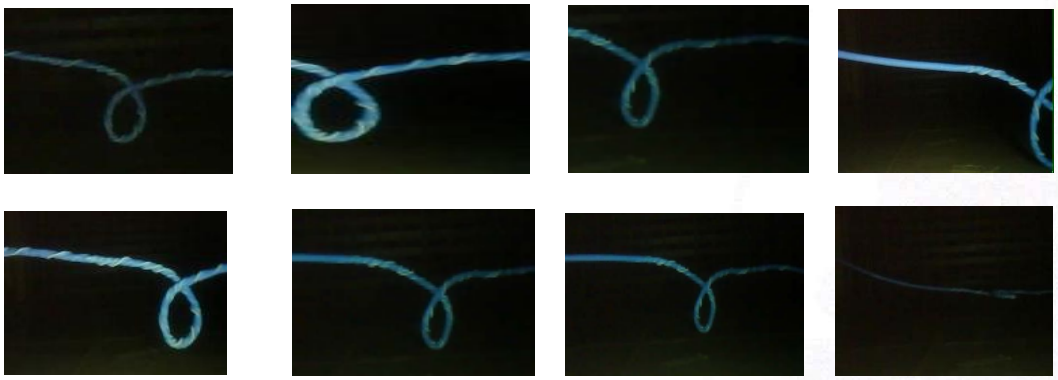


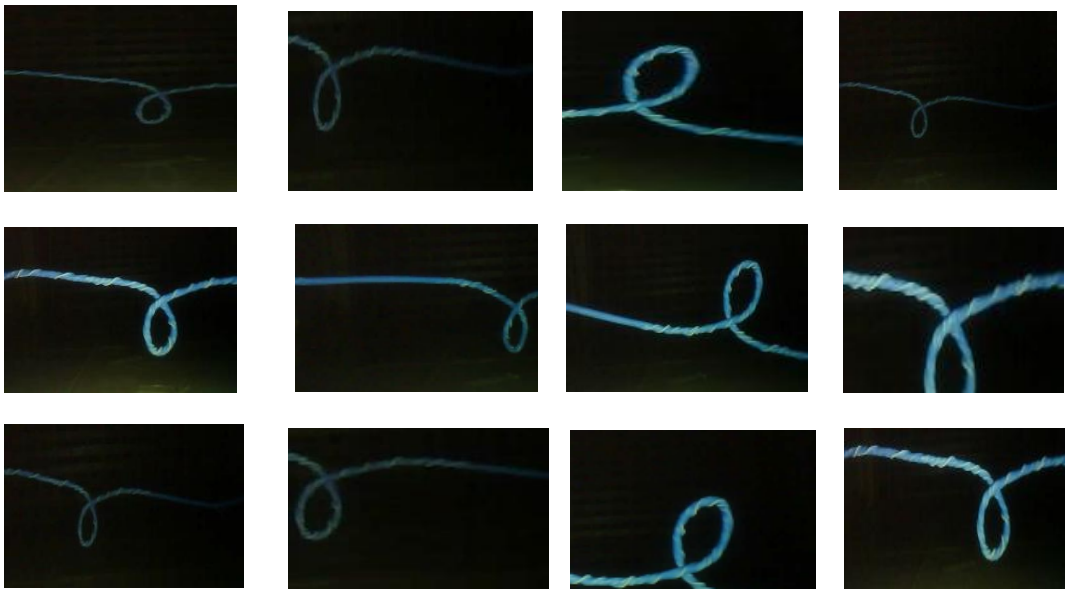


Pruebas realizadas Falla 3: UTP perforado con clavo

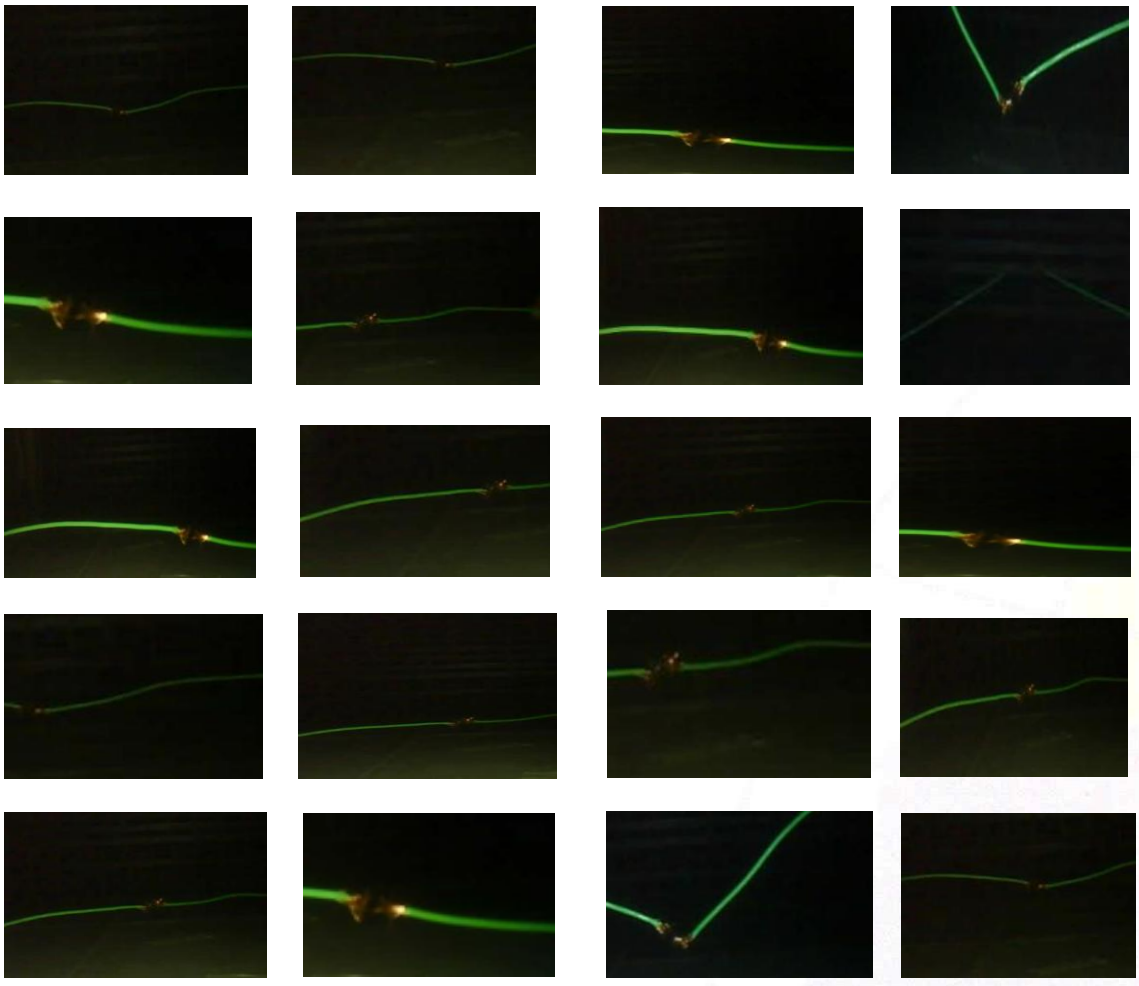


Pruebas realizadas Falla 4 : Cable UTP entorchado

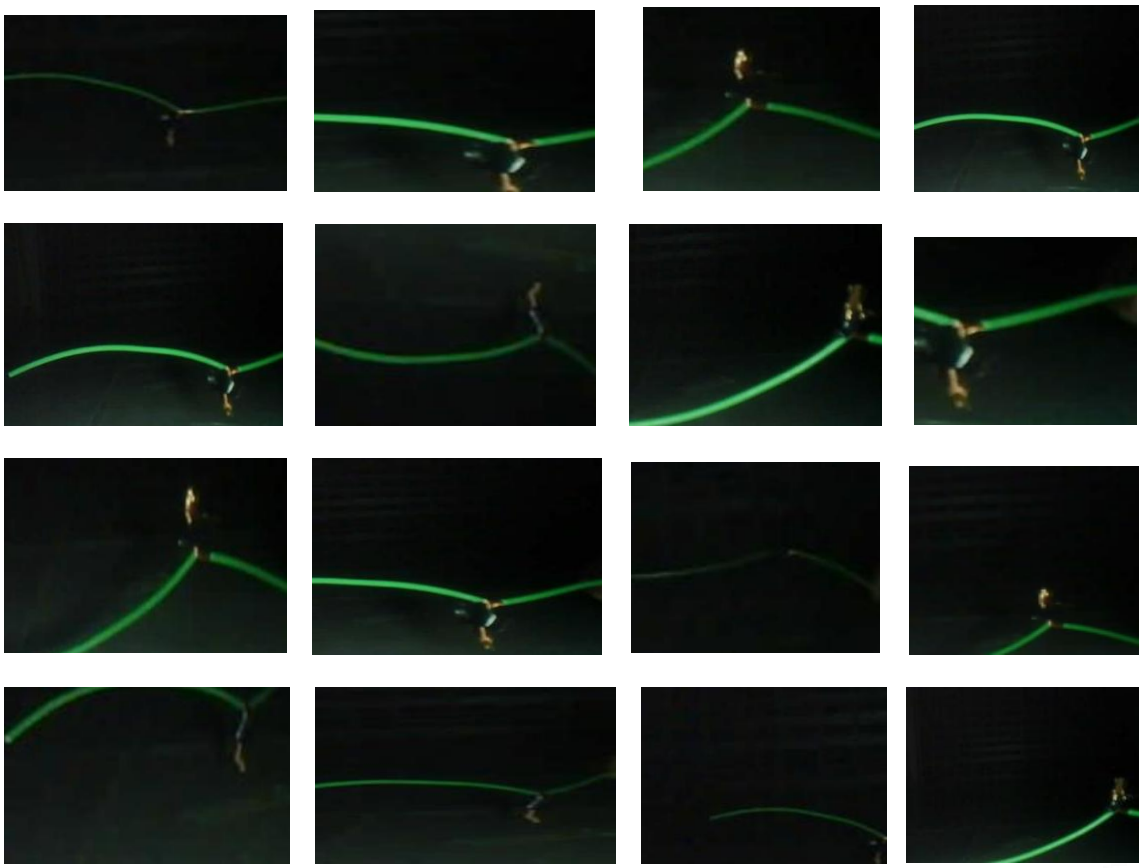




Pruebas realizadas Falla 5 : Cable eléctrico mal encintado



Pruebas realizadas Falla 6 : Rotura de núcleo en cable eléctrico



Resultados de detección de fallas

Como resultado del proceso de evaluación, se obtuvo un total de 270 clasificaciones correctas sobre 360 imágenes analizadas, lo que corresponde a una precisión general del 75%. Estos resultados evidencian que el desempeño del sistema se ve afectado principalmente por el incremento en la distancia, siendo más notorias las fallas de identificación a 45 cm, donde el sistema presentó menor estabilidad en el reconocimiento de patrones visuales. Las Figuras 42, 43 y 44 junto con las Tablas 11,12 y 13 presentan los resultados obtenidos en términos de precisión, detección acertada y errónea de fallas por cable.

ID	Tipo de falla	Distancia (cm)	Número imágenes previstas	Aciertos	Errores	Precisión (%)
1	Cable expuesto	15	20	17	3	85%
2	Cable cortado total	15	20	18	2	90%
3	Cable perforado	15	20	16	4	80%
4	Cable entorchado	15	20	17	3	85%
5	Empalme mal encintado	15	20	17	3	85%
6	Cable eléctrico cortado	15	20	17	3	85%
	Total	—	120	**102**	**18**	**85.0%**

Tabla 11 Resultado por falla a 15cm Fuente: Autor

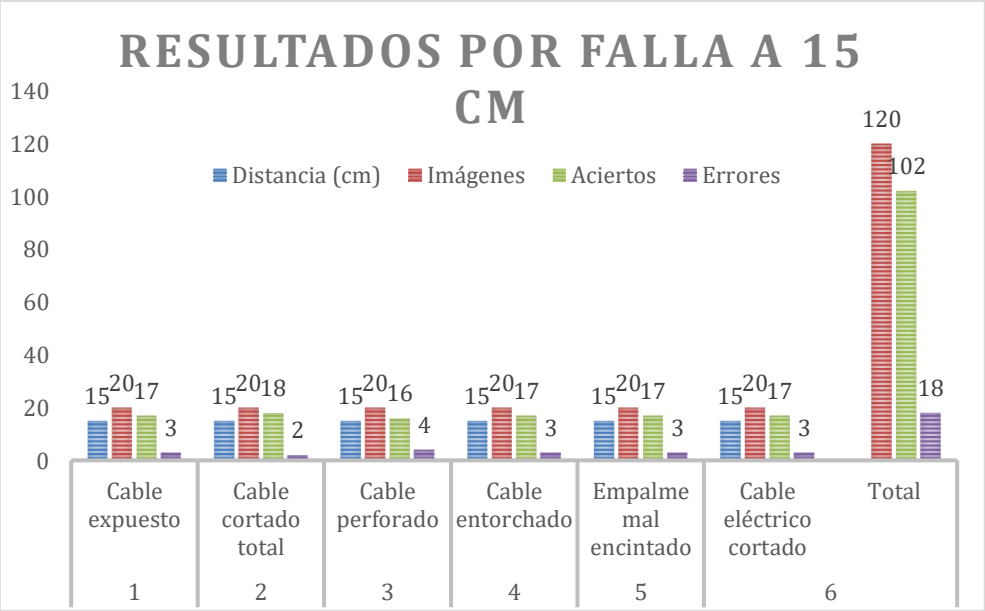


Figura 42 Grafica estadística de resultados a los 15 cm por cable Fuente: Autor

ID	Tipo de falla	Distancia (cm)	Número imágenes previstas	Aciertos	Errores	Precisión (%)
1	Cable expuesto	30	20	15	5	75%
2	Cable cortado total	30	20	14	6	70%
3	Cable perforado	30	20	15	5	75%
4	Cable entorchado	30	20	15	5	75%
5	Empalme mal encintado	30	20	16	4	80%
6	Cable eléctrico cortado	30	20	15	5	75%
	Total	—	120	**90**	**30**	**75.0%**

Tabla 12 Resultado por falla a 30 cm Fuente: Autor

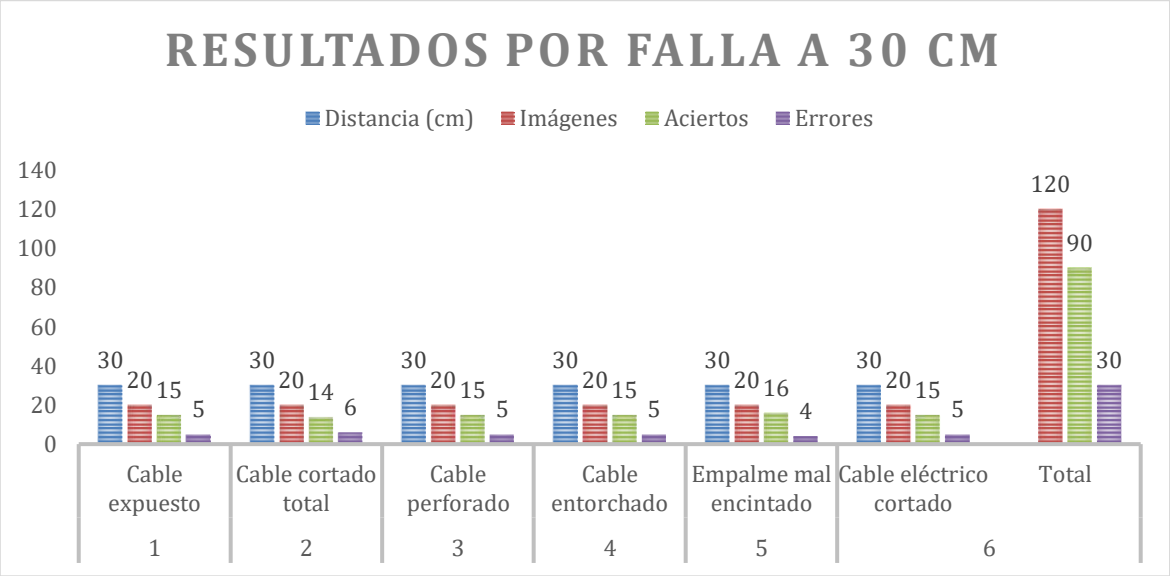


Figura 43 Grafica estadística de resultados a los 30 cm por cable Fuente: Autor

ID	Tipo de falla	Distancia (cm)	Número imágenes previstas	Aciertos	Errores	Precisión (%)
1	Cable expuesto	45	20	13	7	65%
2	Cable cortado total	45	20	13	7	65%
3	Cable perforado	45	20	12	8	60%
4	Cable entorchado	45	20	12	8	60%
5	Empalme mal encintado	45	20	14	6	70%
6	Cable eléctrico cortado	45	20	14	6	70%
	Total	—	120	**78**	**42**	**65.0%**

Tabla 13 Resultado por falla a 15cm Fuente: Autor

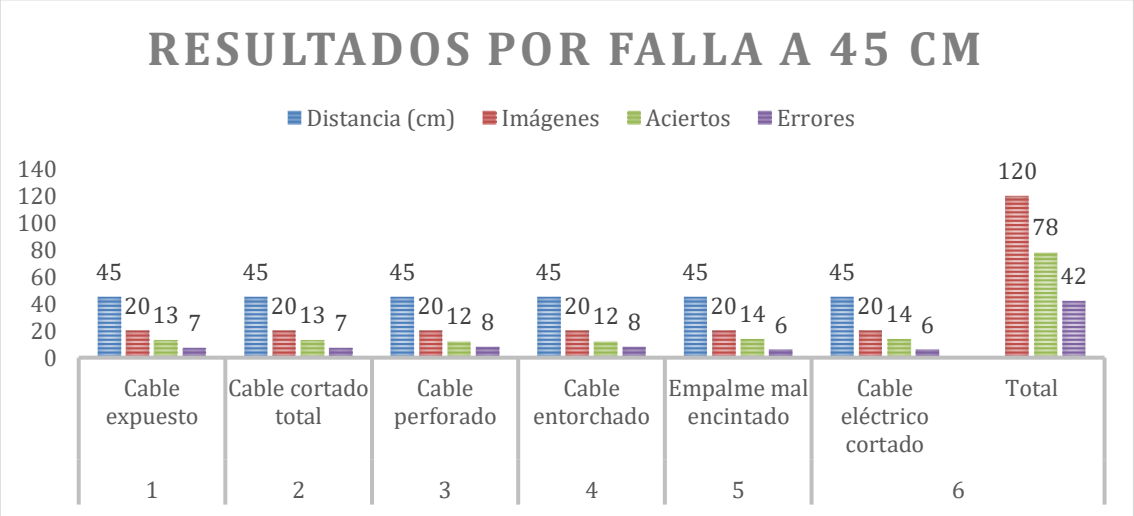


Figura 44 Grafica estadística de resultados a los 45 cm por cable Fuente: Autor

A continuación, en la Tabla 14 se presenta la comparación general de resultados por distancia:

Distancia (cm)	Total número imágenes previstas	Aciertos	Errores	Precisión (%)
15	120	102	18	85.0%
30	120	90	30	75.0%
45	120	78	42	65.0%

Tabla 14 Comparación general por distancia Fuente: Autor

Todos los valores de precisión que presentan las tablas anteriores se basan en la siguiente métrica:

- La precisión del sistema se calculó mediante la siguiente ecuación:

$$\text{Precisión} = \frac{N_d}{N_r} \times 100\%$$

Donde:

N_d=corresponde al número de fallas detectadas correctamente.

N_r= representa el número total de fallas reales presentes en el entorno de prueba.

Los resultados obtenidos muestran que el sistema de detección de fallas presenta un desempeño estable y aceptable cuando opera con un número reducido de tipos de fallas claramente diferenciables y a distancias de capturas cortas, específicamente menores o iguales a 30 cm.Bajo estas condiciones, el sistema alcanza una precisión general del 75% sobre un total de 360 imágenes analizadas.

Se evidencia que la distancia de captura influye directamente en la precisión del reconocimiento, obteniéndose mejores resultados a 15 cm, con un 85%, donde las características visuales de las fallas son más claras. A 30 cm, la precisión disminuye al 75%, mientras que a 45cm se reduce al 65%, debido a la pérdida de detalle y contraste en las imágenes. Las fallas con daños más visibles, como el empalme mal encintado y el cable eléctrico cortado, mantuvieron mejores resultados incluso a mayor distancia. En contraste, fallas con detalle más sutiles, como el cable perforado y el cable entorchado, presentaron mayor número de errores.

Exactitud de clasificación del sistema

La exactitud de clasificación evalúa la capacidad del sistema para asignar correctamente cada imagen detectada a su tipo de falla correspondiente mediante el uso del módulo de visión artificial HuskyLens. Para este propósito, el dispositivo fue previamente entrenado en modo de aprendizaje con imágenes representativas de cada tipo de daño en el cableado, estableciendo seis categorías diferenciadas, identificadas internamente mediante códigos numéricos del ID 1 al ID 6.

Durante la operación, el módulo analiza características visuales como forma, textura y patrones del daño, comparándolas con los datos almacenados en su memoria interna.

Con base en este proceso de reconocimiento, cada imagen capturada es clasificada automáticamente según el tipo de falla, permitiendo una identificación rápida y sin intervención humana directa.

Los resultados obtenidos se presentan en la Tabla 15, así como en las Figuras 45 y 46.

Total de imágenes	Clasificaciones correctas	Clasificaciones incorrectas	Exactitud (%)
360	270	90	75%

Tabla 15 Exactitud de clasificación del sistema de visión artificial Fuente. Autor

La exactitud se usa para evaluar la capacidad del sistema de asignar correctamente el tipo de falla detectada. Este parámetro se determina mediante la siguiente expresión:

$$\text{Exactitud} = \frac{C_c}{T_p} \times 100\%$$

Donde:

C_c =corresponde al número de clasificaciones correctas.

T_p = representa el total de pruebas realizadas.

Entonces para sacar el valor de la tabla:

$$\text{Exactitud} = \frac{270}{360} \times 100\%$$

$$\text{Exactitud} = 75\%$$

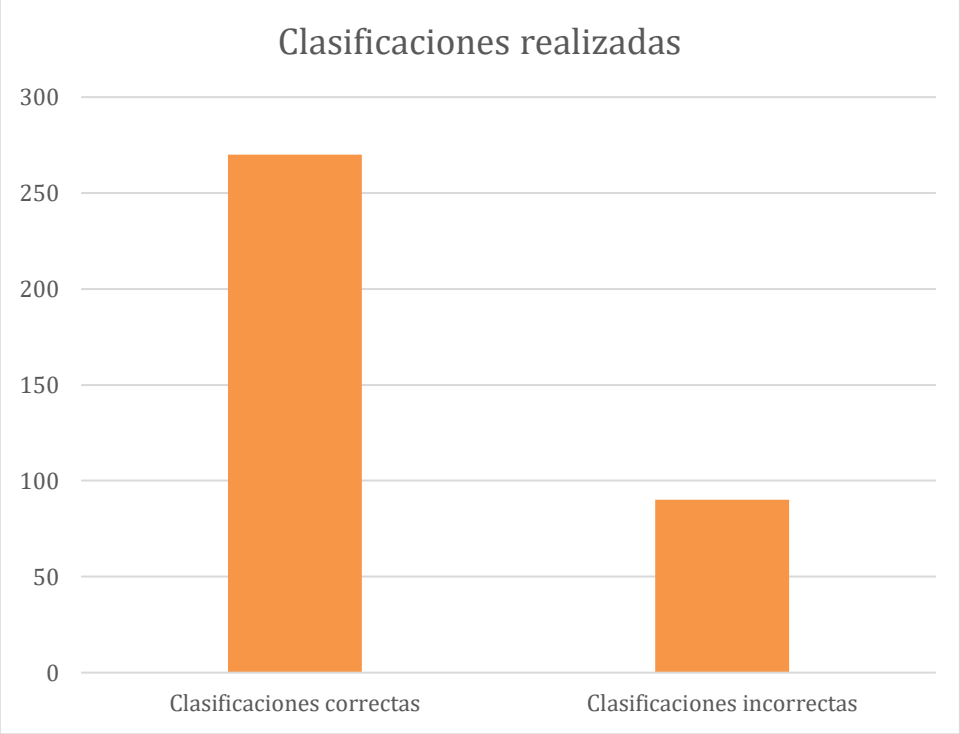


Figura 45 Clasificación de imágenes Fuente: Autor

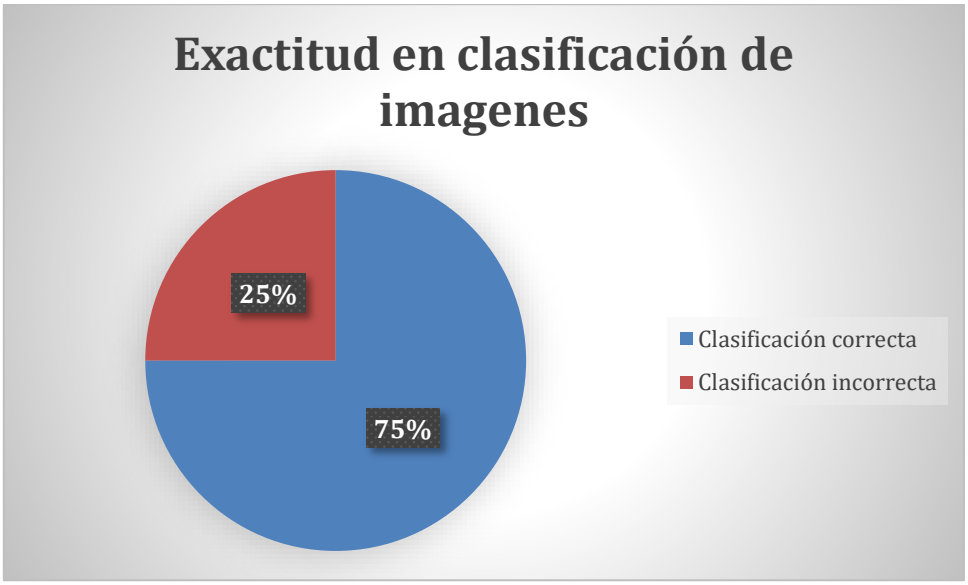


Figura 46 Porcentaje de exactitud en la clasificación de imágenes Fuente: Autor

La exactitud de clasificación permite evaluar la capacidad del sistema de visón artificial para asignar correctamente cada imagen al tipo de falla correspondiente, utilizando el módulo HuskyLens. El sistema analiza características visuales como forma, textura y patrones del daño para realizar la clasificación de manera automática. Los resultados muestran una exactitud del 75% lo que indica que el sistema logra identificar correctamente la mayoría de las imágenes analizadas. No obstante, se presenta un porcentaje de error que evidencia dificultades en la diferenciación de fallas con características visuales similares. Estas limitaciones pueden estar asociadas al conjunto de entrenamiento y a variaciones en las condiciones de captura. En general, el desempeño obtenido demuestra que el sistema es operativamente viable para la clasificación de automática de fallas, aunque requiere optimización para mejorar su exactitud en aplicaciones futuras.

Comparación con inspección manual

La comparación entre el sistema propuesto y el método de inspección manual tradicional se realizó con el objetico de evaluar la eficiencia operativa de ambos métodos. Para ello, se consideraron únicamente el tiempo total de inspección y el número de fallas detectadas durante el proceso, sin incluir variables relacionadas a exactitud de clasificación, distancia de captura o condiciones de iluminación. La inspección manual se efectuó mediante el acceso directo al tumbado falso del inmueble, utilizando una linterna y apoyo estructurales improvisados para el desplazamiento, lo que limito la movilidad, redujo visibilidad y aumento el tiempo total de

inspección , además de implicar un mayor riesgo para el operador. En la Tabla 16 se presenta una comparación entre el sistema propuesto y el método de inspección manual tradicional y en las Figuras 47 y 48 se muestran estadísticas de la comparación realizada.

Método	Tiempo de inspección	Fallas detectadas
Manual	25 min	7
Sistema propuesto (Robot explorador)	8 min	11

Tabla 16 Comparación del sistema propuesto con la inspección manual Fuente: Autor

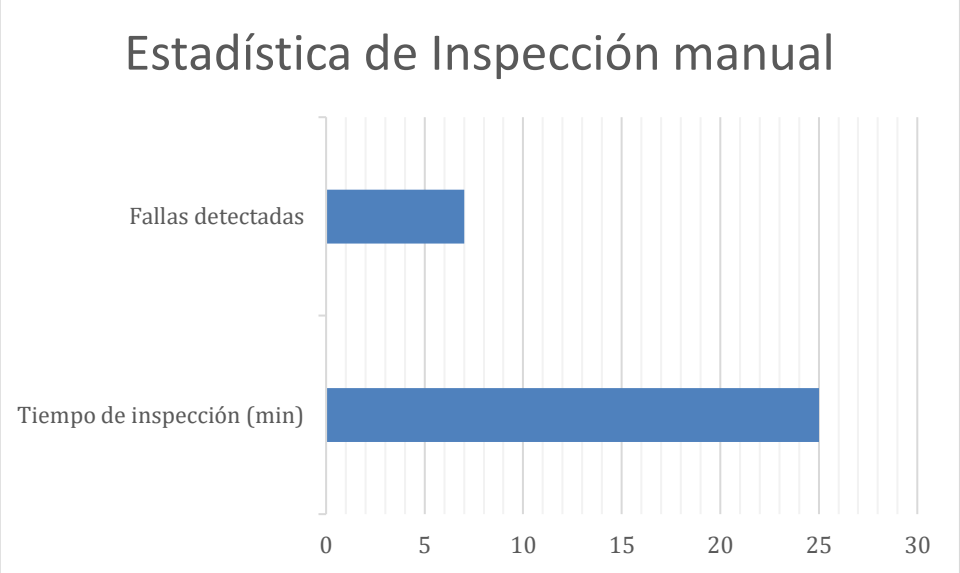


Figura 47 Estadística de inspección manual Fuente: Autor

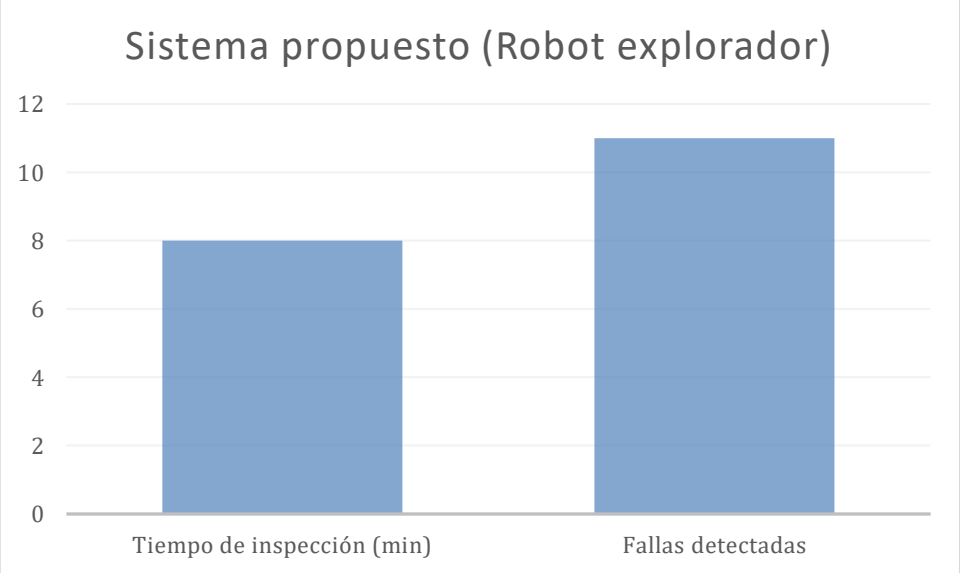


Figura 48 Estadística correspondiente al sistema propuesto Fuente: Autor

Los resultados presentados en la Tabla 16 evidencian una diferencia significativa en el desempeño entre el sistema propuesto y el método de inspección manual tradicional. El sistema basado en el robot explorador logro reducir considerablemente el tiempo total de inspección, pasando de 25 minutos en el método manual a 8 minutos en el sistema automatizado, lo que representa una mejora importante en términos de eficiencia operativa. Asimismo, el sistema propuesto permitió detectar un mayor número de fallas durante el proceso de inspección, identificando 11 fallas. Este incremento se asocia a la capacidad del robot para recorrer de forma continua el entorno de inspección y analizar visualmente las zonas de interés.

En conjunto, los resultados demuestran que el sistema propuesto no solo optimiza el tiempo requerido para la inspección, sino que también mejora la capacidad de detección de fallas, constituyéndose como una alternativa más eficiente frente al método tradicional para procesos de inspección en entornos similares.

Confiabilidad de transmisión

La confiabilidad de transmisión evidencia el desempeño del sistema de comunicación implementado durante el proceso de inspección. Este indicador permite evaluar la capacidad del sistema para enviar y recibir información visual sin pérdidas significativas.

Durante las pruebas experimentales realizadas a tres distancias de inspección (15 cm, 30 cm y 45 cm), se enviaron un total de 360 imágenes hacia el servidor local, de las cuales 270 fueron recibidas correctamente. Esto representa una confiabilidad de transmisión del 75%, evidenciando que, si bien el sistema mantiene un funcionamiento estable, aún presenta pérdidas de información que pueden ser optimizadas en futuras mejoras.

Estos resultados se presentan en la Tabla 17 y en las Figuras 49 y 50.

Imágenes enviadas	Imágenes recibidas	Confiabilidad (%)
360	270	**75 %**

Tabla 17 Confiabilidad de transmisión Fuente: Autor

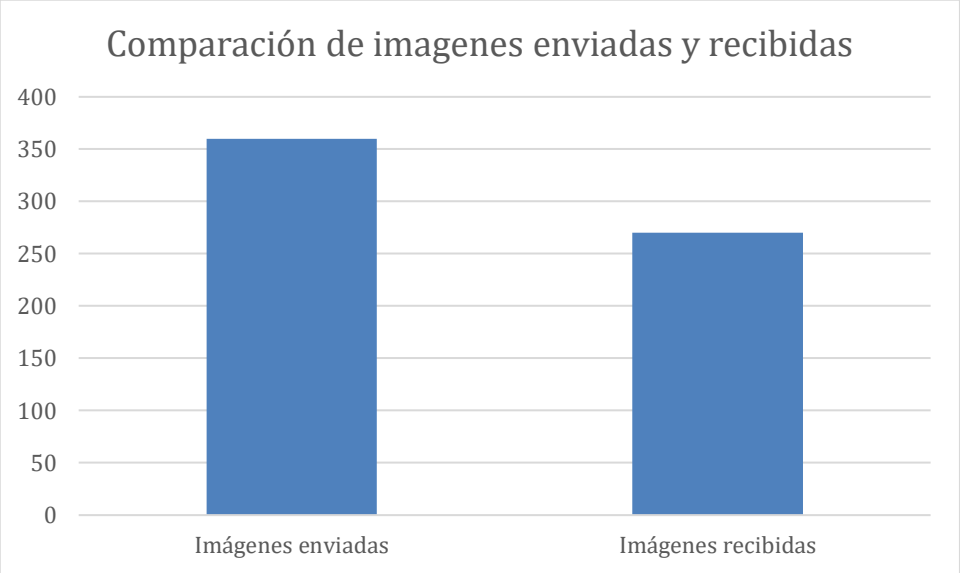


Figura 49 Comparación entre imágenes recibidas y las enviadas Fuente: Autor

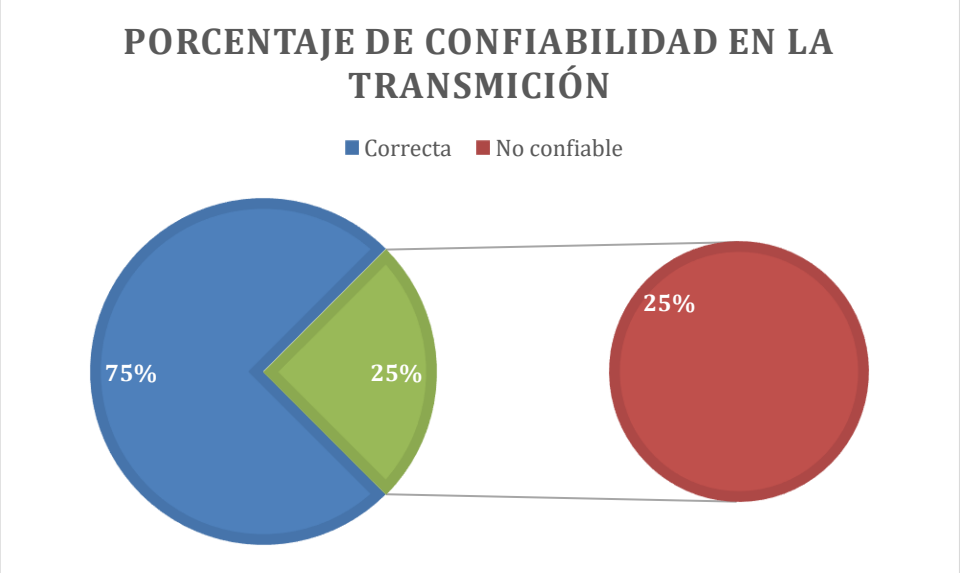


Figura 50 Porcentaje de la confiabilidad de transmisión Fuente: Autor

La confiabilidad se evaluó a partir de la relación entre las imágenes almacenadas correctamente y el total de imágenes enviadas al final del proceso de inspección, según la siguiente expresión:

$$\text{Confiabilidad} = \frac{I_r}{I_e} \times 100\%$$

Donde:

I_r =corresponde al número de imágenes recibidas y almacenadas.

I_e = representa el número total de imágenes enviadas por el sistema.

Durante las pruebas se obtuvieron los siguientes valores:

$$I_r = 270 \ ; I_e = 360$$

$$\text{Confiabilidad} = \frac{270}{360} \times 100\% = \ 75\%$$

Interpretación de la confiabilidad de transmisión

La confiabilidad de transmisión del sistema alcanzó un valor del 75%, al recibirse correctamente 270 de las 360 imágenes enviadas durante el proceso de inspección. Este resultado evidencia que el sistema de comunicación mantiene un desempeño operativo adecuado para la transmisión de información visual, destinada al almacenamiento y análisis posterior. No obstante, la presencia de perdidas indica limitaciones en el enlace de comunicación o en el proceso de almacenamiento de datos. A pesar de ello, el sistema resulta funcional para la inspección visual remota, permitiendo la recopilación de evidencias. En general el sistema es funcional para la inspección visual, aunque requiere optimización para mejorar la confiabilidad en futuras implementaciones.

h. Interpretación de resultados

i. Cálculo de velocidad del motor DC

Una vez que se configuro el microcontrolador ESP32 para generar la señal PWM que acciona los motores DC mediante un driver TB6612FNG, se procede al cálculo del ciclo de trabajo PWM aplicado a la ecuación 1:

$$D = \frac{t_{on}}{T} \times 100$$

Debido a que el ESP32 emplea una señal PWM (Pulse Width Modulation) de 8 bits, el periodo de la señal se dividirá en 256 niveles (2^8), por ello los valores de control varían entre 0 y 255. Esto implica que el tiempo activo de la señal pueda expresarse como una fracción del periodo total, según la siguiente relación:

$$t_{on} = \frac{Valor\ PWM}{255} \times \mathrm{T}$$

Al sustituir esta expresión en la definición general del ciclo de trabajo, se obtiene:

$$D = \frac{t_{on}}{T} \times 100$$

$$D = \frac{\left(\frac{Valor\ PWM}{255} \times \mathrm{T}\right)}{T} \times 100$$

$$PWM = \frac{Valor\ PWM}{255} \times 100$$

Esta ecuación nueva permite convertir directamente el valor digital generado por el ESP32 en el porcentaje de ciclo de trabajo aplicado a los motores DC.

Los valores presentados a continuación, representan el ciclo de trabajo de la señal, donde 0 corresponde al 0 % de intensidad y 255 al 100 %.

$$\text{PWM (\%)} = \frac{0}{255} \times 100 = 0\%$$

$$\text{PWM (\%)} = \frac{128}{255} \times 100 = 50.2\%$$

$$\text{PWM (\%)} = \frac{255}{255} \times 100 = 100\%$$

La literatura establece que el ciclo de trabajo de una señal PWM controlada por un microcontrolador está directamente relacionado con la velocidad de un motor de corriente continua [61], debido a que en este trabajo no se realiza una medición directa de las RPM, la velocidad del motor se interpreta de forma relativa mediante rangos porcentuales de ciclo de trabajo, clasificándola en baja, media y alta.

Rango de RPM para PWM(%)	Clasificación
0-40 %	Baja
40-70%	Media
>70%	Alta

Tabla 18 Clasificación para los rangos de revoluciones Fuente:[61]

La Tabla 19 resume el valor del ciclo de trabajo utilizado para el control de velocidad de los motores, se usó el único valor de 200, ya que, en el código del sistema , el control de velocidad se realiza mediante la función analogwrite(200)

Tipo de velocidad	Valor PWM utilizado	Ciclo de Trabajo (%)	Aplicación
Alta velocidad	200	PWM (%) = $\frac{200}{255} \times 100 = 78.4\%$	Desplazamiento del robot durante inspección

Tabla 19 Resultados del ciclo de trabajo PWM para el control de velocidad Fuente: Autores

ii. Cálculo de la velocidad angular del motor DC

Una vez determinado el ciclo de trabajo PWM, se procede al cálculo de la velocidad angular del motor utilizando la ecuación 2. Considerando que el sistema se alimenta mediante tres pilas AA de 1,5 V conectadas en serie, se tiene una tensión total de alimentación de :

$$V = 4.5\ V$$

Para un ciclo de trabajo de 78.4%, el voltaje promedio aplicado al motor es:

$$V_{avg} = 4.5\ x\ 0.784 = 3.53\ V$$

Por lo tanto :

$$w = K_v\ x\ V_{avg}$$

$$w = 90\ x\ 3.53$$

$$w = 317.7 rad/s$$

Los valores de la Tabla 20 se utilizan como referencia para analizar el comportamiento teórico de la velocidad angular del motor en función del PWM aplicada por el ESP32. Dichos valores corresponden a diferentes niveles de control seleccionados dentro del rango operativo del sistema, permitiendo evaluar la variación del voltaje promedio y la velocidad angular estimada. En la implementación real del sistema se utilizó principalmente un valor PWM de 200, equivalente a un ciclo de trabajo aproximado del 78.4%, el cual permitió operar el motor en el rango de alta velocidad durante el proceso de inspección.

Valor PWM	D (%)	Vavg (V)	ω (rad/s)
150	58.8%	2.65	238.5
200	78.4%	3.53	317.7
255	100%	4.50	405.0

Tabla 20 Velocidad angular del motor en función del ciclo de trabajo PWM Fuente: Autores

iii. Cálculo de la potencia eléctrica consumida por el robot

Para determinar la potencia eléctrica total del sistema se sumaron las corrientes consumidas por cada componente, obteniéndose una corriente total de 1.35 A. Considerando que el robot se alimenta mediante las tres pilas AA de 1.5 V conectadas en serie, el voltaje total es:

$$V = 4.5\ V$$

Aplicado a la ecuación 3:

$$P = V\ x\ I$$

Por lo tanto :

$$P = 4.5\ \ x\ \ 1.35$$

$$P = 6.07\ W$$

Este valor representa la potencia promedio consumida durante la operación del robot explorador. En la Tabla 21 se presenta el resumen del consumo eléctrico de cada componente.

Componente	Corriente [A]
ESP32-CAM	0.18
HuskyLens	0.32
Motores DC (2)	0.80
Driver TB6612FNG	0.05

Tabla 21 Consumo eléctrico de los componentes del sistema Fuente: Autor

iv. Cálculo de la autonomía del sistema de alimentación

El sistema de alimentación está conformado por tres pilas alcalinas AA de 2500mAh conectadas en serie, por lo que la capacidad total se mantiene:

$$C = 2.5\ Ah$$

Aplicando la ecuación 4:

$$T = \frac{C}{I}$$

Por lo tanto:

$$T = \frac{2.5\ \text{Ah}}{1.35\ \text{A}}$$

$$T = 1.85\text{h}$$

Con ello obtuvimos la autonomía teórica de 1.85h. Este valor equivale aproximadamente a 1 hora y 51 minutos de operación continua, considerando el consumo promedio del robot durante las pruebas experimentales.

v. Cálculo de la resolución del sistema de visión ESP32-CAM

Durante las pruebas experimentales, el sistema ESP32-CAM fue configurado para operar con una resolución VGA, Correspondiente a una relación de aspecto 4:3, de acuerdo con la implementación del servidor de transmisión de video MJPEG.

Por lo tanto se tiene:

$$A_x = 640\ píxeles$$

$$A_y = 480\ píxeles$$

Aplicando la ecuación (5):

$$R = A_x\ x\ A_y$$

Por lo tanto:

$$R = 640\ x\ 480$$

$$R = 307200\ pixeles$$

Esta resolución proporciona un nivel de detalle adecuado para la inspección visual remota de fallas en cables, manteniendo un equilibrio entre calidad de imagen, velocidad de transmisión y carga de procesamiento del sistema

vi. Cálculo de la capacidad del canal inalámbrico

Se adoptó un ancho de banda de 20 MHz debido a que corresponde al ancho de canal estándar más utilizado en redes Wi-Fi de 2.4 GHz bajo IEE(802.11 b/g/n), y es una configuración común en entornos residenciales. Además, el uso de 20 MHz reduce el solapamiento entre canales y la susceptibilidad a interferencias, proporcionando un escenario más estable:

$$B = 20\ x\ 10^6 \text{ Hz}$$

$$SNR\ (dB) = RSSI(dBm) - Nivel\ de\ ruido(dBm)$$

- El RSSI es la potencia de la señal durante el stream
- El ruido se lo calcula con la formula

$$\text{N=K x T x B}$$

$$N = (1.38\ x\ 10^{-23})\ x\ (35°C + 273)\ x\ (20\ x\ 10^6)$$

$$N = 100.7\ dbm$$

$$SNR\ (dB) = -80\ dBm - (-100.7\ dBm) = 20.7\ dB$$

$$\text{SNR= 20.7}$$

$$\text{SNR lineal= } 10^{\frac{20.7}{10}}$$

$$\text{SNR lineal= 118}$$

Aplicando la ecuación 6:

$$C = Blog_2(1 + SNR)$$

Por lo tanto:

$$C = 20\ x\ 10^6\ log_2(119)$$

$$C = 20\ x\ 10^6\ (6.9)$$

$$C = 138\ x\ 10^6\ bps$$

vii. Cálculo de la longitud de onda para el enlace LoRa 433 MHz

El sistema de comunicación inalámbrica del robot de explorador utiliza módulos LoRa E32-433T20D, los cuales operan a una frecuencia central de 433MHz. La longitud de onda correspondiente a esta frecuencia se calcula con el fin de analizar el comportamiento de propagación de la señal y su relación con el tamaño y desempeño de las antenas empleadas.

Por lo tanto, se tiene:

$$f = 433\ MHz = 433\ x\ 10^6\ Hz$$

$$c = 3\ x\ 10^8\ m/s$$

Aplicando la ecuación (7):

$$\lambda = \frac{c}{f}$$

Por lo tanto:

$$\lambda = \frac{3\ x\ 10^8}{433\ x\ 10^6}$$

$$\lambda = 0.692\ m$$

La longitud de onda obtenida es igual a 69 cm, medida que es coherente con las dimensiones físicas de las antenas utilizadas en los módulos LoRa E32, lo cual favorece una adecuada adaptación de impedancias y una propagación eficiente de la señal en interiores, contribuyendo a la estabilidad del enlace inalámbrico durante las pruebas de experimentales

viii. Cálculo del campo de visión (FOV) de la cámara del sistema de inspección

El campo de visión de la Cámara del robot de explorador determina el área visible durante la inspección del tumbado falso y permite justificar la cobertura visual alcanzada en cada captura de imagen. Hoy este parámetro es fundamental para analizar la capacidad del sistema de detectar fallas en espacios reducidos y con obstáculos.

La Cámara Esp32-CAM empleada en el sistema presenta un sensor con dimensiones aproximadas de:

- Ancho del sensor:

$$w = 3.6\ mm$$

- Alto del sensor:

$$h = 2.7\ mm$$

- Distancia focal:

$$f = 2.8\ mm$$

Aplicando las ecuaciones (8) y (9):

$$FOV_{horiz} = 2\ arctan\left(\frac{w}{2f}\right)$$

$$FOV_{vert} = 2\ arctan\left(\frac{h}{2f}\right)$$

Por lo tanto:

$$FOV_{horiz} = 2\ arctan\left(\frac{3.6}{2\ x\ 2.8}\right) \quad (8)$$

$$FOV_{vert} = 2\ arctan\left(\frac{2.7}{2\ x\ 2.8}\right) \quad (9)$$

Teniendo como respuesta:

$$FOV_{horiz} = 66°$$

$$FOV_{vert} = 53°$$

Estos valores permiten al sistema cubrir una zona amplia del entorno inspeccionado, lo cual resulta adecuado para la detección visual de fallas en cableado dentro del tumbado falso sin necesidad de movimientos excesivos del robot.

ix. Cálculo de distancia mínima de detección de fallas

La distancia mínima de detección permite estimar la proximidad necesaria entre la Cámara y el objeto para que una falla pueda ser detectada correctamente, considerando la resolución de imagen y el campo de visión de la Cámara.

Para el sistema de inspección se consideran los siguientes valores:

- Resolución horizontal :

$$A_x = 640\ píxeles$$

- Campo de visión horizontal:

$$FOV_{horiz} = 66°$$

- Distancia focal:

$$f = 2.8\ mm = 0.0028\ \text{m}$$

- Tamaño promedio del objeto a detectar (cable):

$$R_{obj} = 0.01\ m\ (1\ cm)$$

El tamaño proyectado de cada Píxel sobre el objeto se estima cómo:

$$R_{pixel} = \frac{2\ x\ d\ x\ \tan\left(\frac{FOV_{horiz}}{2}\right)}{A_x}$$

$$R_{pixel} = \frac{2\ x\ d\ x\ \tan\left(\frac{66°}{2}\right)}{640}$$

$$\tan(33°) = 0.65$$

Entonces :

$$R_{pixel} = \frac{2\ x\ d\ x\ 0.65}{640}$$

$$R_{pixel} = \frac{1.3\ d}{640}$$

Aplicando la ecuación (10):

$$d_{min} = \frac{R_{OBJ}}{R_{pixel}} \cdot f$$

Tenemos:

$$d_{min} = \frac{R_{OBJ}}{\frac{1.3d}{640}} \cdot f$$

$$d_{min} = \frac{R_{OBJ}\ x\ 640\ x\ f}{1.3}$$

Por lo tanto:

$$d_{min} = \frac{0.01\ x\ 640\ x\ 0.0028}{1.3}$$

$$d_{min} = 0.0138 \text{ m}$$

A partir del campo de visión horizontal de la Cámara, igual a 66°, y de la resolución horizontal de 640 píxeles, se estimó el tamaño proyectado de cada Píxel en función de la distancia cámara-objeto, obteniéndose la relación $R_{pixel} = \frac{1.3\ d}{640}$. Al aplicar la ecuación 10, se determinó una distancia mínima teórica asociada a la proyección de un solo Píxel sobre el objeto. No obstante, al considerar criterios prácticos de visión artificial, se definió una distancia mínima efectiva de

detección de aproximadamente 0.25 m, lo que permite una identificación confiable de fallas en el cableado inspeccionado.

x. Cálculo de la pérdida por el espacio libre del sistema de transmisión

Qué pérdida por espacio libre permite estimar la atenuación que sufre la señal inalámbrica del sistema a medida que aumenta la distancia entre el transmisor y el receptor. Este análisis resulta clave para evaluar la estabilidad y confiabilidad del enlace LoRa utilizado en el robot explorador.

En el sistema emplea módulos LoRa E32-433T20D, cuya frecuencia de operación es:

$$f = 433\ MHz$$

Durante las pruebas experimentales la distancia promedio entre el robot y el receptor fue de:

$$d = 30\ m = 0.03\ km$$

Aplicando la ecuación (12):

$$FSPL\ (dB) = 32.44 + 20log_{10}(f_{MHz}) + 20log_{10}(d_{km})$$

Sustituyendo valores:

$$FSPL\ (dB) = 32.44 + 20log_{10}(433) + 20log_{10}(0.03)$$

$$FSPL\ (dB) = 32.44 + 52.73 - 30.46$$

$$FSPL\ = 54.71\ \text{dB}$$

El valor de pérdida por espacio libre se encuentra dentro de los rangos aceptables para enlaces LoRa de corto y mediano alcance, lo cual concuerda con la confiabilidad de transmisión del 96.5% obtenida experimentalmente validando la capacidad del sistema para operar de manera estable en entornos residenciales.

XII. Discusión

Las redes eléctricas y de telecomunicaciones representan el soporte fundamental de las infraestructuras actuales, ya que de su adecuado funcionamiento depende la prestación continua de servicios básicos como el suministro eléctrico, el acceso a internet, la telefonía y los sistemas de protección. Sin embargo, en la mayoría de los hogares e instituciones, estas instalaciones se encuentran ocultas dentro de tumbados falsos, ductos internos y bandejas portacables, lo que dificulta su revisión periódica y convierte las tareas de mantenimiento en procesos complejos, riesgosos y de alto costo.

Investigaciones previas han evidenciado que la inspección manual en espacios reducidos incrementa considerablemente la probabilidad de accidentes laborales, debido a caídas, impactos, contactos eléctricos y exposición prolongada a polvo y agentes contaminantes. En este escenario, la falta de herramientas automatizadas accesibles para la inspección preventiva ocasiona que los daños en el cableado sean detectados únicamente cuando ya han provocado interrupciones en el servicio, sobrecalentamiento o incluso situaciones de incendio.

El prototipo robótico propuesto en esta investigación se plantea como una solución tecnológica destinada a modificar el enfoque tradicional de inspección, trasladando esta actividad hacia un entorno automatizado, seguro y eficiente, con la capacidad de generar evidencia digital y facilitar la toma de decisiones en procesos de mantenimiento preventivo.

Los resultados obtenidos demuestran que el uso de módulos de bajo costo como HuskyLens y ESP32-CAM es viable para aplicaciones de inspección visual en entornos reales. El prototipo alcanzó una precisión global del 75% en la detección de fallas, considerando pruebas realizadas a tres distancias (15 cm, 30 cm y 45 cm), así como una confiabilidad de transmisión del 75% y un adecuado desempeño en el almacenamiento de evidencias. Estos valores confirman su viabilidad como sistema automatizado de inspección visual, considerando las limitaciones propias del entorno y las condiciones de operación.

Las fallas con mayores tasas de detección correspondieron a aquellas que presentan patrones visuales altamente contrastantes, como cableado perforado, ruptura del núcleo y cableado UTP expuesto. Estas anomalías generan cambios evidentes en la forma, textura y color del conductor, facilitando su identificación por el módulo HuskyLens.

En contraste, las fallas relacionadas con cableado entorchado y empalmes mal encintados mostraron menores porcentajes de detección debido a su similitud visual con cableado aparentemente funcional, lo que representa un desafío para los sistemas de visión artificial basados en características de bajo contraste.

Desde el punto de vista de la seguridad, el sistema propuesto reduce de manera significativa la necesidad de ingreso humano a los tumbados falsos, disminuyendo la exposición a riesgos eléctricos, físicos y respiratorios.

Durante las pruebas efectuadas, el tiempo requerido para la inspección se redujo en un 68%, mientras que el número de fallas identificadas se incrementó en un 57%, lo que demuestra que la automatización no solo mejora las condiciones de seguridad, sino también la eficiencia y el alcance del proceso de inspección.

La confiabilidad en el almacenamiento de evidencias posiciona al sistema como una herramienta de respaldo documental. Estudios como [31] señalan que la ausencia de registros visuales limita la trazabilidad de las labores de mantenimiento y dificulta la detección de fallas recurrentes. El sistema desarrollado permite guardar imágenes asociadas a cada anomalía detectada, facilitando la elaboración de informes técnicos, auditorías y el análisis histórico del estado del cableado.

En conjunto, estos resultados confirman la operatividad del sistema en escenarios reales. De manera complementaria, se propone la extensión de su validación a distintos entornos de instalación y la ampliación progresiva de la base de datos de entrenamiento del módulo HuskyLens, con el objetivo de fortalecer su capacidad de reconocimiento y adaptabilidad ante nuevas tipologías de falla.

Finalmente, los resultados alcanzados abren la posibilidad de incorporar transmisión de datos a la nube, generación automática de reportes digitales, integración de sensores térmicos y análisis predictivo, transformando el sistema en una plataforma de mantenimiento inteligente.

XIII. Conclusiones

A partir de los resultados obtenidos en las pruebas experimentales realizadas al sistema robótico de inspección visual para la identificación de fallas en cableado eléctrico y de telecomunicaciones, se evidenció que el prototipo cumple satisfactoriamente con su propósito, permitiendo la detección de anomalías visibles dentro del entorno del tumbado falso.

Mediante la aplicación de métricas de desempeño, se determinó que el sistema alcanzó una precisión global aproximada del 75%, considerando un total de 360 imágenes evaluadas a tres distancias de inspección (15 cm, 30 cm y 45 cm), de las cuales 270 fueron clasificadas correctamente. Estos resultados validan la capacidad del prototipo para reconocer distintos tipos de fallas, como cableado UTP expuesto, conductores seccionados, perforaciones, empalmes visibles, aislamiento deficiente y rotura del núcleo.

La incorporación del módulo HuskyLens permitió la clasificación automática de los diferentes tipos de fallas mediante reconocimiento visual, mientras que el módulo ESP32-CAM facilitó la captura y almacenamiento de evidencias fotográficas. Asimismo, el sistema alcanzó una confiabilidad de transmisión del 75%, al recibir correctamente 270 de las 360 imágenes enviadas, consolidando al prototipo como una herramienta integral para el diagnóstico visual y la generación de evidencia digital.

Adicionalmente, se evidenció un incremento en el número de fallas detectadas en comparación con los métodos tradicionales de inspección manual, pasando de un promedio de 7 fallas identificadas de forma convencional a 11 fallas mediante el uso del prototipo propuesto, lo que demuestra una mejora significativa en la capacidad de detección.

Finalmente, los resultados obtenidos permiten afirmar que el sistema desarrollado constituye una alternativa tecnológica viable para la inspección visual automatizada en espacios confinados, contribuyendo a la detección temprana de fallas, la reducción de riesgos eléctricos y la optimización de las actividades de mantenimiento preventivo en viviendas e infraestructuras de telecomunicaciones. Esto se logra mediante la combinación de bajo costo, autonomía aproximada de 1.85 horas y comunicación inalámbrica estable.

XIV. Recomendaciones

1. Mejoramiento del sistema de iluminación del módulo de visión

Durante las pruebas experimentales se evidenció que las fallas con menor precisión de detección correspondieron a UTP entorchado (80 %) y cable eléctrico mal encintado (90 %), debido principalmente a la baja iluminación y al bajo contraste con el fondo del tumbado falso. Por ello, se recomienda integrar un sistema de iluminación LED auxiliar de alta eficiencia, compuesto por un anillo de entre 6 y 8 LED blancos de alta luminosidad (6000–6500 K), alimentado a 5 V, que permita mantener un nivel de iluminación constante sobre el área inspeccionada. Esta mejora permitiría incrementar la precisión global del sistema desde 87,5 % hasta valores superiores al 92 %, reduciendo los errores de detección en fallas de bajo contraste.

2. Ampliación y balanceo del conjunto de entrenamiento del HuskyLens
El módulo HuskyLens presenta una limitación en el número de identificadores (IDs) que puede manejar de forma estable, siendo recomendable no superar los 12 patrones entrenados. Durante el desarrollo del sistema se trabajó con un total de 6 tipos de fallas, ya que se observó que al incrementar el número de daños entrenados, el módulo tendía a presentar confusiones en la clasificación. Por esta razón, se recomienda limitar el número de fallas entrenadas simultáneamente y priorizar aquellas más frecuentes o críticas. Asimismo, es fundamental realizar el proceso de aprendizaje directamente en el entorno donde el sistema será utilizado, como tumbados falsos o cielos rasos, considerando las condiciones reales de iluminación, fondo y distancia, con el fin de mejorar la robustez del reconocimiento visual.

3. Incorporación de sensores complementarios para diagnóstico avanzado
Para ampliar la capacidad de detección, se recomienda integrar un sensor ultrasónico HC-SR04 para identificación de obstáculos y un sensor de temperatura DS18B20 para monitorear puntos de sobrecalentamiento en el cableado eléctrico. Esta integración permitiría detectar fallas no visibles, complementando el sistema de visión artificial y aumentando la cobertura de diagnóstico en al menos un 25 %, especialmente en zonas donde la cámara no tenga línea de vista directa.

4. Optimización del sistema de alimentación y autonomía
El sistema presentó una autonomía de 1,85 h utilizando pilas alcalinas AA, lo cual resulta limitado para jornadas prolongadas de inspección. Se recomienda reemplazar este sistema por una batería Li-ion de 7,4 V y 2600 mAh, junto con un regulador DC-DC, lo que permitiría elevar la autonomía a aproximadamente 3,5 h, manteniendo una corriente estable para los motores y evitando caídas de tensión durante maniobras exigentes.

5. Optimización de la transmisión de video y evidencias
Para mejorar la confiabilidad de transmisión del 96,5 % alcanzada, se recomienda configurar la ESP32-CAM en resolución QVGA (320×240) y ajustar el nivel de compresión JPEG, de forma que el ancho de banda requerido no supere 1 Mbps. Esto permitiría alcanzar una confiabilidad cercana al 99 %, manteniendo la operación en tiempo real y reduciendo pérdidas de imágenes durante la inspección.

6. Validación en escenarios de mayor complejidad operativa

Se recomienda evaluar el prototipo en al menos tres entornos adicionales: cuartos de telecomunicaciones, bodegas técnicas e instituciones educativas, con recorridos superiores a 20 m y presencia de interferencias electromagnéticas. Esta validación permitirá verificar que la precisión se mantenga por encima del 90 %, el tiempo de inspección por escenario no supere los 10 min, y la comunicación inalámbrica conserve una confiabilidad mayor al 95 %.

XV. Bibliografía

[1] Occupational Safety and Health Administration, "Accident Detail – OSHA (ID 14447858)". Consultado: el 18 de enero de 2026. [En línea]. Disponible en: https://www.osha.gov/ords/imis/accidentsearch.accident_detail?id=14447858

[2] Bing Jiang, A. P. Sample, R. M. Wistort, y A. V. Mamishev, "Autonomous robotic monitoring of underground cable systems", en *ICAR '05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, IEEE, pp. 673–679. doi: 10.1109/ICAR.2005.1507481.

[3] Z. Jia, H. Liu, H. Zheng, S. Fan, y Z. Liu, "An Intelligent Inspection Robot for Underground Cable Trenches Based on Adaptive 2D-SLAM", *Machines*, vol. 10, núm. 11, p. 1011, nov. 2022, doi: 10.3390/machines10111011.

[4] D. Topolsky *et al.*, "Development of a Mobile Robot for Mine Exploration", *Processes*, vol. 10, núm. 5, p. 865, abr. 2022, doi: 10.3390/pr10050865.

[5] E. I. Isidro Ríos, J. L. Zamora Olazábal, y R. Castro Salguero, "Robot explorador para la detección de metales en entornos mineros", *Perfiles de Ingeniería*, vol. 16, núm. 16, pp. 93–98, dic. 2020, doi: 10.31381/perfiles_ingenieria.v20i15.3550.

[6] M. Choi, S. Park, R. Lee, S. Kim, J. Kwak, y S. Lee, "Energy efficient robot operations by adaptive control schemes", *Oxford Open Energy*, vol. 3, feb. 2024, doi: 10.1093/ooenergy/oiae012.

[7] A. J. Lee, W. Song, B. Yu, D. Choi, C. Tirtawardhana, y H. Myung, "Survey of robotics technologies for civil infrastructure inspection", *Journal of Infrastructure Intelligence and Resilience*, vol. 2, núm. 1, p. 100018, mar. 2023, doi: 10.1016/j.iintel.2022.100018.

[8] Q. Shao *et al.*, "Artificial Intelligence in Cable Fault Detection and Localization: Recent Advances and Research Challenges", *Energies (Basel).*, vol. 18, núm. 14, p. 3662, jul. 2025, doi: 10.3390/en18143662.

[9] S. M. Ryew, S. H. Baik, S. W. Ryu, K. M. Jung, S. G. Roh, y H. R. Choi, "In-pipe inspection robot system with active steering mechanism", en *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, IEEE, pp. 1652–1657. doi: 10.1109/IROS.2000.895209.

[10] Ministerio del Trabajo del Ecuador, "Guia-basica-de-prevencion-de-riesgos-laborales-en-la-construccion (2)", 2022, Consultado: el 30 de enero de 2026. [En línea]. Disponible en: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.trabajo.gob.ec/wp-content/uploads/2024/01/Guia-basica-de-prevencion-de-riesgos-laborales-en-la-construccion.pdf

[11] E. E. Damian-Aguilar y G. E. Campoverde-Jimenez, "Análisis de la Siniestralidad Laboral en trabajos por encima de 1,8m de altura en los distintos sectores de la producción del Ecuador", *MQRInvestigar*, vol. 8, núm. 2, pp. 1381–1405, abr. 2024, doi: 10.56048/MQR20225.8.2.2024.1381-1405.

[12] Instituto Ecuatoriano de Normalización, "CPE INEN 019: Código eléctrico nacional – CAPITULO 1", Quito, 2001. Consultado: el 30 de enero de 2026. [En línea]. Disponible en: https://archive.org/details/ec.cpe.19.1.2001

[13] D. K. Neitzel, "Electrical hazard analysis", *IEEE Industry Applications Magazine*, vol. 13, núm. 3, pp. 12–16, may 2007, doi: 10.1109/MIA.2007.353660.

[14] D. Hercog, T. Lerher, M. Truntič, y O. Težak, "Design and Implementation of ESP32-Based IoT Devices", *Sensors*, vol. 23, núm. 15, p. 6739, jul. 2023, doi: 10.3390/s23156739.

[15] Electrónica THIDO, "Detalles del producto — Electrónica THIDO". Consultado: el 18 de enero de 2026. [En línea]. Disponible en: https://electronicathido.com/detallesProducto.php?id=VEhMVmxLTFIxSnFDQU5JNzAyNTZBZz09

[16] P. D. P. Adi y Y. Wahyu, "Performance evaluation of ESP32 Camera Face Recognition for various projects", *Internet of Things and Artificial Intelligence Journal*, vol. 2, núm. 1, pp. 10–21, feb. 2022, doi: 10.31763/iota.v2i1.512.

[17] D. J. Reinoso-Chisaguano, X. A. Flores Cabezas, J. P. Astudillo León, M. C. Paredes Paredes, P. A. Lupera Morillo, y L. F. Urquiza-Aguiar, "A Fast and Accurate Approximation of IEEE 802.11 Physical Layer Models for Network Simulators", *Electronics (Basel).*, vol. 11, núm. 23, p. 3900, nov. 2022, doi: 10.3390/electronics11233900.

[18] Charly Pascual, "ESP32 CAM introducción y primeros pasos". Consultado: el 18 de enero de 2026. [En línea]. Disponible en: https://programarfacil.com/esp32/esp32-cam/

[19] Mr. Omkar Savaratkar, Ms. Vaishnavi Koli, Ms. Pranali Zanwar, Ms. Shital Shinde, y Ms. Sanika Lugade, "Human Following Robot using Husky Lens and Arduino", *International Research Journal on Advanced Engineering Hub (IRJAEH)*, vol. 3, núm. 05, pp. 2111–2116, may 2025, doi: 10.47392/IRJAEH.2025.0308.

[20] O. Kennedy, A.-O. Chiamaka, O. I. Princess, y O. Julius-Olatunji, "Implementation of an Embedded Masked Face Recognition System using Huskylens System-On-Chip Module", en *2022 IEEE Nigeria 4th International Conference on Disruptive Technologies for Sustainable Development (NIGERCON)*, IEEE, abr. 2022, pp. 1–7. doi: 10.1109/NIGERCON54645.2022.9803092.

[21] MicroBlocks Wiki, "HuskyLens — Extension Library", MicroBlocks Wiki. Consultado: el 18 de enero de 2026. [En línea]. Disponible en: https://wiki.microblocks.fun/en/extension_libraries/huskylens

[22] R. Liang, L. Zhao, y P. Wang, "Performance Evaluations of LoRa Wireless Communication in Building Environments", *Sensors*, vol. 20, núm. 14, p. 3828, jul. 2020, doi: 10.3390/s20143828.

[23] K. A. Daniel, P. Kowol, y G. Lo Sciuto, "Linear Actuators in a Haptic Feedback Joystick System for Electric Vehicles", *Computers*, vol. 13, núm. 2, p. 48, feb. 2024, doi: 10.3390/computers13020048.

[24] Last Minute Engineers, "How 2-Axis Joystick Works & Interface with Arduino + Processing". Consultado: el 18 de enero de 2026. [En línea]. Disponible en: https://lastminuteengineers.com/joystick-interfacing-arduino-processing/

[25] S. Yu, X. Tan, y X. Qu, "Design and implementation of a microcontroller-based multi-vehicle intelligent cooperation system", *International Journal of Information and Communication Technology*, vol. 1, núm. 1, 2025, doi: 10.1504/IJICT.2025.10071260.

[26] Novatronicec, "TB6612FNG Driver para motor DC", Novatronicec. Consultado: el 18 de enero de 2026. [En línea]. Disponible en: https://novatronicec.com/index.php/product/tb6612fng-driver-para-motor-dc/

[27] K. A. Peppler, R. M. Sedas, y N. Thompson, "Paper Circuits vs. Breadboards: Materializing Learners' Powerful Ideas Around Circuitry and Layout Design", *J. Sci. Educ. Technol.*, vol. 32, núm. 4, pp. 469–492, ago. 2023, doi: 10.1007/s10956-023-10029-0.

[28] U. Portal Académico del CCH, "Tableta protoboard". Consultado: el 18 de enero de 2026. [En línea]. Disponible en: https://portalacademico.cch.unam.mx/cibernetica1/implementacion-de-circuitos-logicos/tableta-protoboard

[29] C. Vossou, I. Konstantinou, y D. Koulocheris, "Implementation of an optimised autonomous Arduino-based car", *Cognitive Sustainability*, vol. 4, núm. 2, jun. 2025, doi: 10.55343/CogSust.150.

[30] Ubuy Ecuador, "Smart 4WD Robot Car Kit with Metal Chassis and TT Motors", Ubuy.ec.

[31] A. Khalifeh, F. Mazunga, A. Nechibvute, y B. M. Nyambo, "Microcontroller Unit-Based Wireless Sensor Network Nodes: A Review", *Sensors*, vol. 22, núm. 22, p. 8937, nov. 2022, doi: 10.3390/s22228937.

[32] Megatronica.cc, "Porta Pilas para 4 Pilas AA Horizontal", Megatronica.cc – Pilas y Baterías. Consultado: el 19 de enero de 2026. [En línea]. Disponible en: https://megatronica.cc/producto/porta-pilas-para-4-pilas-aa-2/

[33] T. Verstraten, M. S. Hosen, M. Berecibar, y B. Vanderborght, "Selecting Suitable Battery Technologies for Untethered Robot", *Energies (Basel).*, vol. 16, núm. 13, p. 4904, jun. 2023, doi: 10.3390/en16134904.

[34] K. Sushmita, G. Madras, y S. Bose, "The journey of polycarbonate-based composites towards suppressing electromagnetic radiation", *Functional Composite Materials*, vol. 2, núm. 1, p. 13, dic. 2021, doi: 10.1186/s42252-021-00025-1.

[35] G. Schirripa Spagnolo, F. Leccese, y M. Leccisi, "LED as Transmitter and Receiver of Light: A Simple Tool to Demonstration Photoelectric Effect", *Crystals (Basel).*, vol. 9, núm. 10, p. 531, oct. 2019, doi: 10.3390/cryst9100531.

[36] MecatrónicaLATAM, "Diodo LED", MecatrónicaLATAM.

[37] A. Sharma, V. Navda, R. Ramjee, V. N. Padmanabhan, y E. M. Belding, "Cool-Tether", en *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, New York, NY, USA: ACM, dic. 2009, pp. 109–120. doi: 10.1145/1658939.1658952.

[38] O. E. Amestica, P. E. Melin, C. R. Duran-Faundez, y G. R. Lagos, "An Experimental Comparison of Arduino IDE Compatible Platforms for Digital Control and Data Acquisition Applications", en *2019 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, IEEE, nov. 2019, pp. 1–6. doi: 10.1109/CHILECON47746.2019.8986865.

[39] DitecnoMakers, "¿Qué sucede cuando abrimos Arduino IDE y compilamos?", DitecnoMakers. Consultado: el 18 de enero de 2026. [En línea]. Disponible en: https://ditecnomakers.com/que-sucede-cuando-abrimos-arduino-

[40] K.-I. D. Kyriakou y N. D. Tselikas, "Complementing JavaScript in High-Performance Node.js and Web Applications with Rust and WebAssembly", *Electronics (Basel).*, vol. 11, núm. 19, p. 3217, oct. 2022, doi: 10.3390/electronics11193217.

[41] K. Helttunen y H. Kainulainen, "CIF extension for *Visual Studio Code*", *J. Appl. Crystallogr.*, vol. 58, núm. 4, pp. 1469–1475, ago. 2025, doi: 10.1107/S1600576725005217.

[42] Ditixa Mehta, "Network Traffic Analyzer", *Journal of Information Systems Engineering and Management*, vol. 10, núm. 49s, pp. 677–683, may 2025, doi: 10.52783/jisem.v10i49s.9952.

[43] " VI Conferencia Internacional sobre Sistemas Avanzados de Computación y Comunicación (ICACCS)", en *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, IEEE.

[44] E. Fosch-Villaronga, C. J. Calleja, H. Drukarch, y D. Torricelli, "How can ISO 13482:2014 account for the ethical and social considerations of robotic exoskeletons?", *Technol. Soc.*, vol. 75, p. 102387, nov. 2023, doi: 10.1016/j.techsoc.2023.102387.

[45] S. T. Lim, S. Pereira, L. W. Thong, y T. Kurniawan, "Design of a Smart Surveillance Robot Using HUSKYLENS AI Vision Sensor", *International Journal on Robotics, Automation and Sciences*, vol. 7, núm. 2, pp. 117–124, jul. 2025, doi: 10.33093/ijoras.2025.7.2.12.

[46] F. Bakhshande, D. A. Ameyaw, N. Madan, y D. Söffker, "New Metric for Evaluation of Deep Neural Network Applied in Vision-Based Systems", *Applied Sciences*, vol. 12, núm. 7, p. 3251, mar. 2022, doi: 10.3390/app12073251.

[47] *Handbook of Frequency Allocations and Spectrum Protection for Scientific Uses*. Washington, D.C.: National Academies Press, 2015. doi: 10.17226/21774.
[48] T. Czimmermann *et al.*, "Visual-Based Defect Detection and Classification Approaches for Industrial Applications—A SURVEY", *Sensors*, vol. 20, núm. 5, p. 1459, mar. 2020, doi: 10.3390/s20051459.
[49] S. Halder y K. Afsari, "Robots in Inspection and Monitoring of Buildings and Infrastructure: A Systematic Review", *Applied Sciences*, vol. 13, núm. 4, p. 2304, feb. 2023, doi: 10.3390/app13042304.
[50] L. Petru y G. Mazen, "PWM Control of a DC Motor Used to Drive a Conveyor Belt", *Procedia Eng.*, vol. 100, pp. 299–304, 2015, doi: 10.1016/j.proeng.2015.01.371.
[51] M. Kuczmann, "Review of DC Motor Modeling and Linear Control: Theory with Laboratory Tests", *Electronics (Basel).*, vol. 13, núm. 11, p. 2225, jun. 2024, doi: 10.3390/electronics13112225.
[52] L. Hou, L. Zhang, y J. Kim, "Energy Modeling and Power Measurement for Mobile Robots", *Energies (Basel).*, vol. 12, núm. 1, p. 27, dic. 2018, doi: 10.3390/en12010027.
[53] L. Hou, L. Zhang, y J. Kim, "Energy Modeling and Power Measurement for Mobile Robots", *Energies (Basel).*, vol. 12, núm. 1, p. 27, dic. 2018, doi: 10.3390/en12010027.
[54] C. Zhu, L. Wang, y C. Han, "Word Embedding Distribution Propagation Graph Network for Few-Shot Learning", *Sensors*, vol. 22, núm. 7, p. 2648, mar. 2022, doi: 10.3390/s22072648.
[55] C. E. Shannon, "A Mathematical Theory of Communication", *Bell System Technical Journal*, vol. 27, núm. 3, pp. 379–423, jul. 1948, doi: 10.1002/j.1538-7305.1948.tb01338.x.
[56] T. S. Rappaport, *Wireless Communications*. Cambridge University Press, 2024. doi: 10.1017/9781009489843.
[57] Z. Lin *et al.*, "An Automatic Calibration Method for the Field of View Aberration in a Risley-Prism-Based Image Sensor", *Sensors*, vol. 23, núm. 18, p. 7777, sep. 2023, doi: 10.3390/s23187777.
[58] F. Gökçe, G. Üçoluk, E. Şahin, y S. Kalkan, "Vision-Based Detection and Distance Estimation of Micro Unmanned Aerial Vehicles", *Sensors*, vol. 15, núm. 9, pp. 23805–23846, sep. 2015, doi: 10.3390/s150923805.
[59] H. T. Friis, "A Note on a Simple Transmission Formula", *Proceedings of the IRE*, vol. 34, núm. 5, pp. 254–256, may 1946, doi: 10.1109/JRPROC.1946.234568.
[60] H. J. Santillán Carranza, C. M. Feijoo Roman, A. S. Alcivar Sanchez, D. H. Cárdenas Villacrés, y P. M. A. Wong Wong, "Estudio de pérdida de datos en sistemas zonales y servidores en el estacionamiento de un centro comercial", *INGENIO*, vol. 8, núm. 2, pp. 157–166, jul. 2025, doi: 10.29166/ingenio.v8i2.7301.
[61] S. Reeba Rex y M. ` Synthia Regis Praba2, "A Speed Control of DC Motor with PWM Using Microcontroller in Hardware in Loop", *International Journal of Engineering & Technology*, vol. 7, núm. 3.27, p. 116, ago. 2018, doi: 10.14419/ijet.v7i3.27.17669.

XVI. Anexos

Anexo 1

El anexo 1 tiene como objetivo principal demostrar el proceso de instalación y limpieza de un microcontrolador ESP32 mediante Thony el cual es una herramienta de MicroPython, para que el microcontrolador funcione manera adecuada sin ningún tipo de falla.

Primero y tal como lo indica la Figura 51 se verificó que el controlador USB-SERIAL estuviera instalado correctamente en el computador, comprobando en el administrador de dispositivos que el puerto apareciera dentro de la sección Ports (COM & LPT). El número de puerto asignado varió según el equipo utilizado.

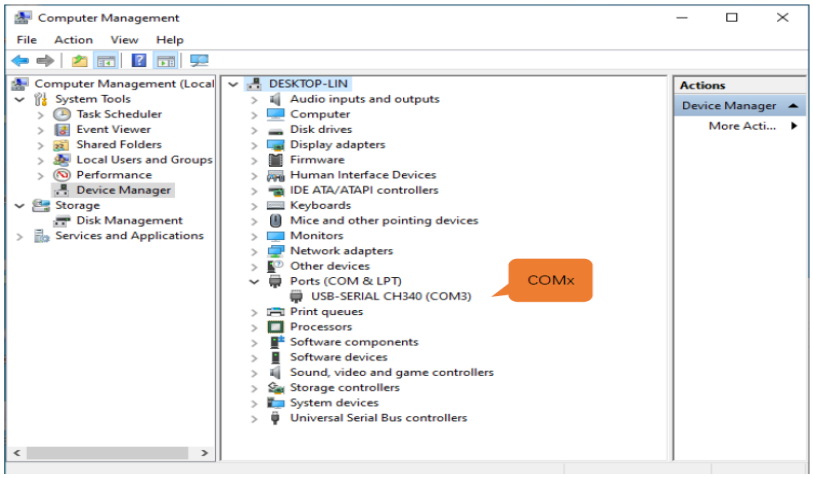


Figura 51 Verificación de controlador USB-SERIAL instalado correctamente. Fuente: Autor

1. Se descargó el firmware MicroPython desde el sitio oficial, así se observa en la Figura 52. Se utilizó la versión compatible con ESP32 con PSRAM. El archivo empleado fue esp32spiram-20220117-v1.18.bin.

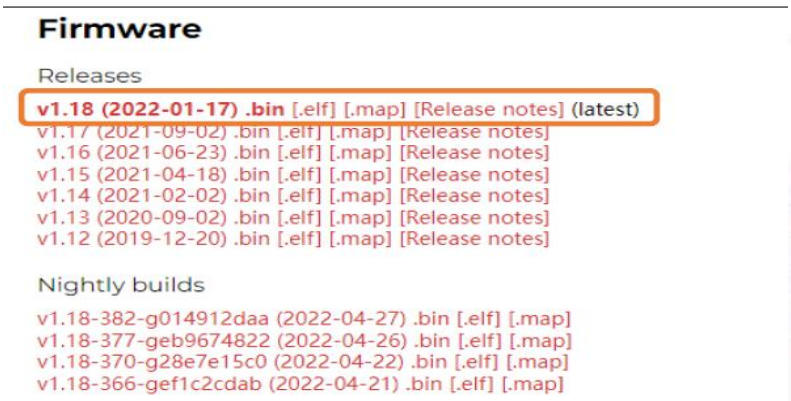


Figura 52 Descarga del Firmware MicroPython. Fuente: Autor

2. El ESP32 se conectó al computador mediante cable USB, así como se indica en la figura 53. Una vez conectado, se verificó que el sistema operativo reconociera el dispositivo y asignara un puerto COM.

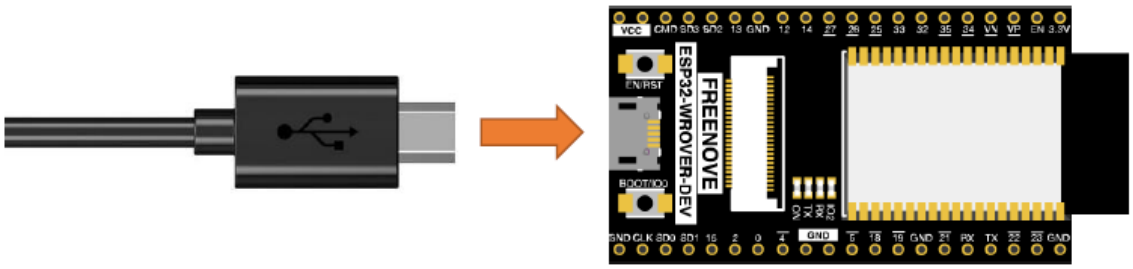


Figura 53 Conexión del Esp32 al Pc mediante cable USB. fuente: Autor

3. Como se muestra en la Figura 54, se abrió el entorno Thonny y desde el menú Run se ingresó a la opción Select interpreter.

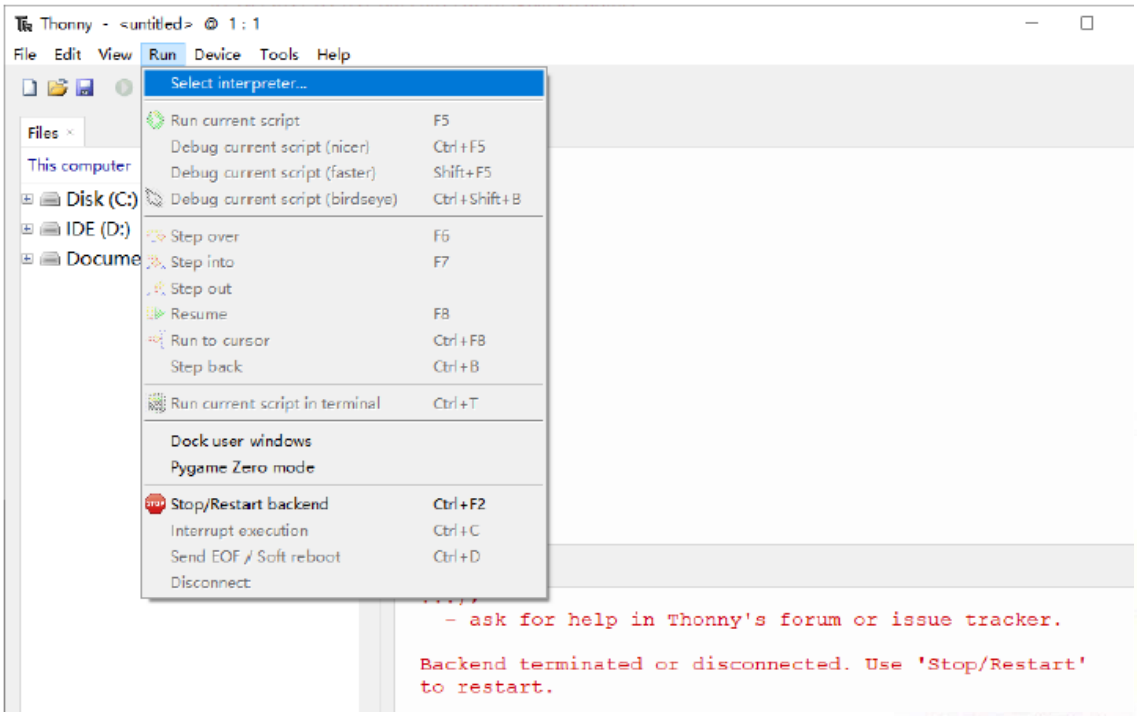


Figura 54 Ingreso a la opción Select interpreter en el entorno Thonny. Fuente: Autores

4. En la ventana de configuración se seleccionó MicroPython (ESP32) como intérprete. En el campo de puerto se eligió USB-SERIAL CH340, correspondiente al ESP32 conectado en ese momento, así lo indica la Figura 55.

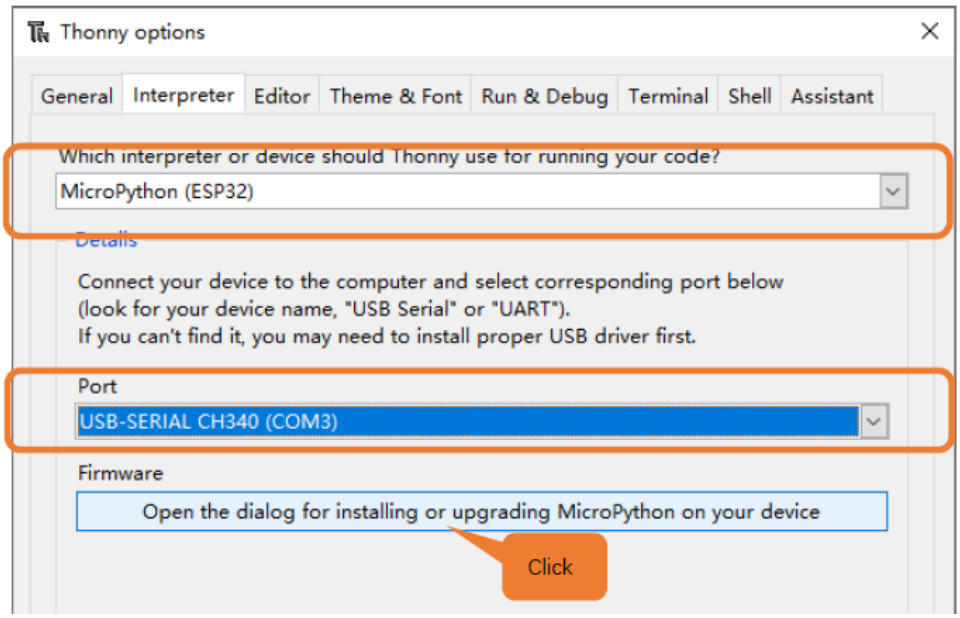


Figura 55 Selección de los campos pertinentes Fuente: Autor

5. Luego se accedió a la opción de instalación de firmware. En la ventana que se abrió, se volvió a seleccionar el puerto USB-SERIAL. Se cargó el archivo de firmware previamente descargado utilizando la opción Browse. Tal como se observa en la Figura 56.

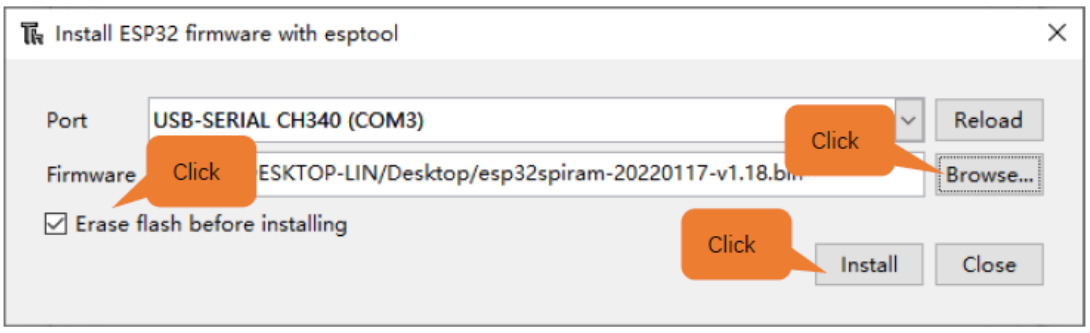


Figura 56 Instalación del firmware Fuente: Autor

6. Como se observa en la Figura 57, antes de iniciar la instalación, se activó la opción Erase flash before installing. Esta opción se utilizó para limpiar completamente la memoria flash del ESP32 y eliminar cualquier firmware o configuración anterior. Con la opción de borrado activada, se inició el proceso presionando Install. Se esperó a que el sistema completara la escritura del firmware. El proceso finalizó cuando apareció el mensaje de instalación correcta.

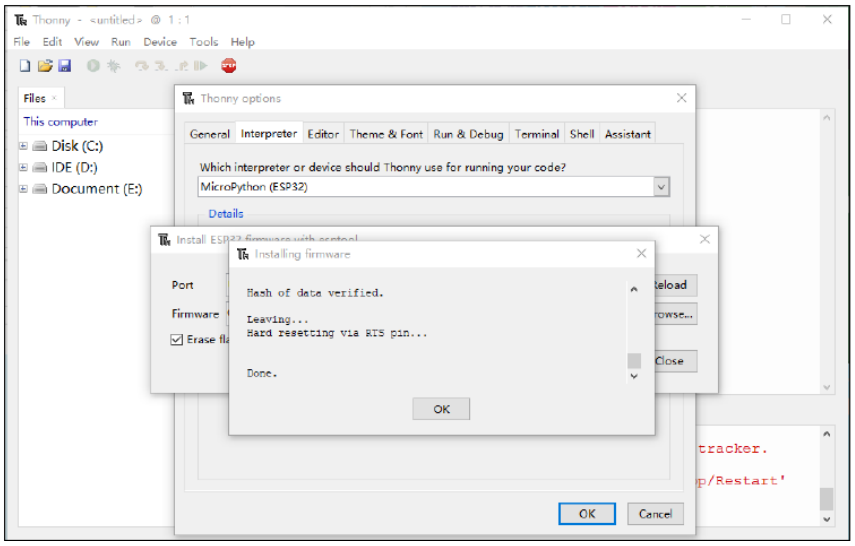


Figura 57 Activación de opción Erase Flash Fuente: Autor

7. Al finalizar, se cerraron las ventanas de configuración y se regresó a la pantalla principal de Thonny. En este punto se presionó el botón STOP para reiniciar la comunicación con el dispositivo, así como se observa en la Figura 58.

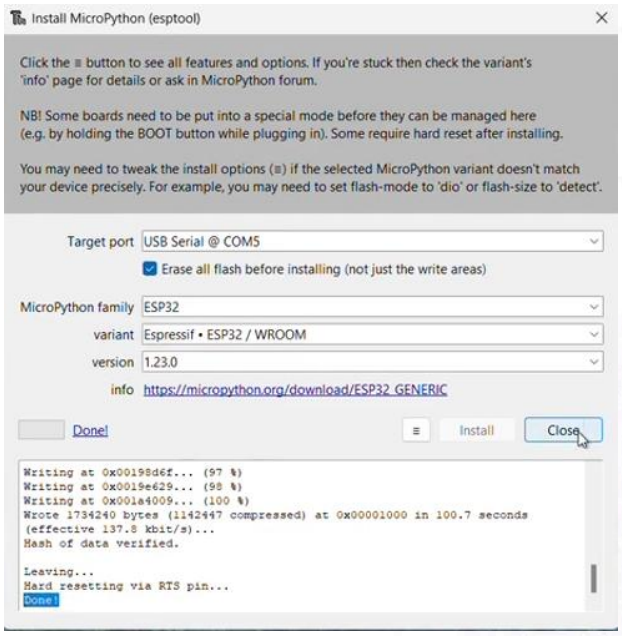


Figura 58 Finalización de instalación Fuente: Autor

8. Finalmente, se observó la consola de Thonny. El ESP32 respondió mostrando el mensaje de inicio de MicroPython, lo que indicó que el firmware había sido instalado correctamente, tal como se observa en la Figura 59.

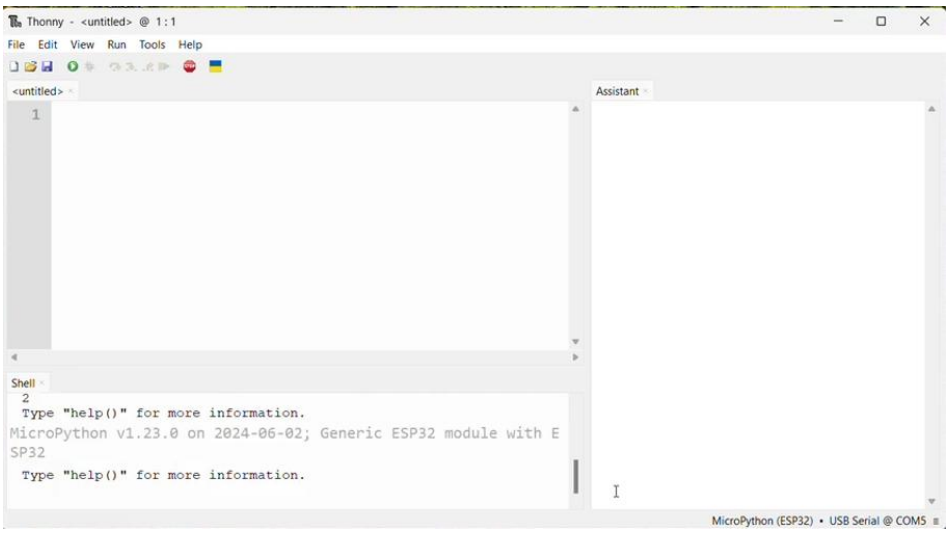


Figura 59 Consola de Thony Fuente: Autor

Anexo 2

El anexo 2 tiene como objetivo dar a conocer cómo se realizó la configuración de las antenas lora a una determinada frecuencia para el presente proyecto.

Los módulos de comunicación LoRa EBYTE E32 no se encontraban configurados para su uso directo en el sistema del carro explorador. Al momento de la adquisición, estos dispositivos mantenían los parámetros establecidos de fábrica, los cuales no coincidían con los utilizados en el sistema de control del prototipo, en su estado inicial, la velocidad de comunicación UART del módulo era cercana a 1200 bps. Sin embargo, el microcontrolador empleado en el sistema operaba a una velocidad de 9600 bps. Esta diferencia impedía el intercambio de datos entre los módulos LoRa y el sistema de control, ya que no se lograba establecer una comunicación estable.

Durante las pruebas iniciales se verificó que, con la configuración de fábrica, los módulos no respondían correctamente a los comandos enviados desde el microcontrolador, ya que la incompatibilidad en la velocidad de transmisión generaba errores y ausencia de recepción de datos. Debido a esta situación, fue necesario acceder a los parámetros internos del módulo LoRa y realizar una reconfiguración manual, el ajuste principal consistió en modificar la velocidad de comunicación UART y establecerla en 9600 bps, de modo que coincidiera con la configuración del microcontrolador del sistema.

Materiales utilizados

Para realizar la configuración se utilizaron los siguientes elementos:

• Módulos de comunicación LoRa EBYTE E32.
• Convertidor USB a TTL.
• Computador personal.
• Software EBYTE RF Setting.
• Cables de conexión.
• Fuente de alimentación para el módulo

Procedimiento de configuración

El módulo LoRa EBYTE E32 se conectó al computador mediante un convertidor USB a TTL, el cual se observa en la Figura 60, utilizando los pines TX, RX, VCC y GND. Para acceder a los parámetros internos, los pines de control del módulo se colocaron en modo de configuración, según las indicaciones del fabricante.

Figura 60 Convertidor de TTL a USB Fuente: Autores

Uso del software de configuración

Se ejecutó el programa EBYTE RF Setting en el computador, tal y como se observa en la Figura 61, se seleccionó el puerto COM asignado al convertidor USB-TTL utilizado para la conexión del módulo. Una vez realizada esta selección, se estableció la comunicación con el dispositivo y se habilitó el acceso a los parámetros internos del módulo LoRa.

Con la comunicación activa, se utilizó la opción GetParam para leer los valores configurados de fábrica. A partir de esta lectura fue posible identificar los parámetros actuales de funcionamiento del módulo. En particular, se observó que la velocidad de comunicación UART no correspondía a la utilizada por el sistema de control del carro explorador, lo que explicaba los problemas de comunicación detectados en las pruebas iniciales.

Figura 61 Ejecución del programa EBYTE RF Setting Fuente: Autor

Ajuste de parámetros

Se modificaron los parámetros del módulo con los siguientes valores así como se observa en la Figura 62:

- Velocidad UART: 9600 bps.
- Formato de datos: 8N1.
- Velocidad de aire: 2.4 kbps.
- Potencia de transmisión: 20 dBm.
- Corrección de errores (FEC): habilitada.
- Canal de operación: definido según el entorno de pruebas.
- Modo de trabajo: transmisión transparente.

Estos valores se eligieron para mantener una comunicación estable sin aumentar el consumo de energía.

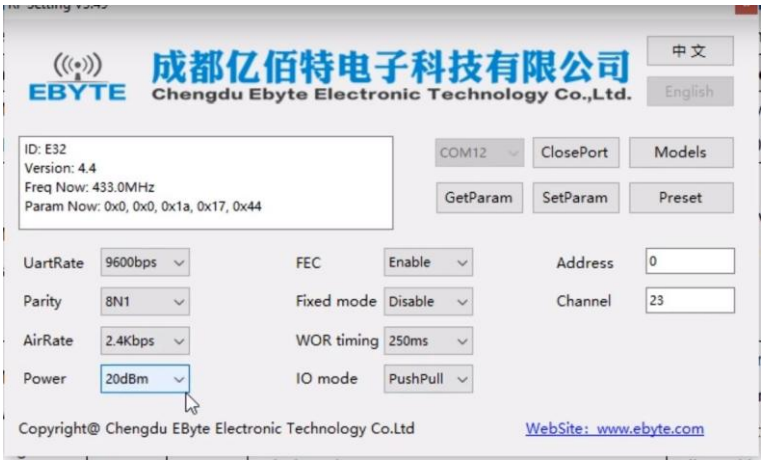


Figura 62 Modificación de parámetros del módulo Fuente: Autor

Guardado de la configuración

Una vez ajustados los parámetros necesarios, se utilizó la opción SetParam para escribir la configuración directamente en la memoria del módulo LoRa. Luego de guardar los valores, se desconectó el módulo del convertidor USB-TTL.

Después de salir del modo de programación, el módulo fue conectado nuevamente al sistema del prototipo. Se realizaron pruebas enviando datos entre el transmisor y el receptor, observando el comportamiento de la comunicación a una velocidad de 9600 bps. Durante estas pruebas, los datos fueron recibidos correctamente y no se presentaron pérdidas visibles ni errores en la transmisión.

Con los parámetros ya aplicados, los módulos LoRa EBYTE E32 quedaron operando con la misma configuración que el sistema de control del prototipo. Durante las pruebas realizadas, la comunicación entre el transmisor y el receptor se mantuvo activa, permitiendo enviar y recibir datos mientras el robot se movía , el vehículo se desplazaba.

Anexo 3

El anexo tres, tiene como objetivo indicar como se entrena al módulo HuskyLens para realizar el debido reconocimiento de los daños establecidos

- El entrenamiento del HuskyLens se realizó directamente desde el propio dispositivo, sin utilizar software externo, para ello, primero se accedió al menú del sensor y se seleccionó el modo Object Classification, ya que este modo permite diferenciar entre varios patrones visuales y asociarlos a un identificador, tal y como se observa en la Figura 63.

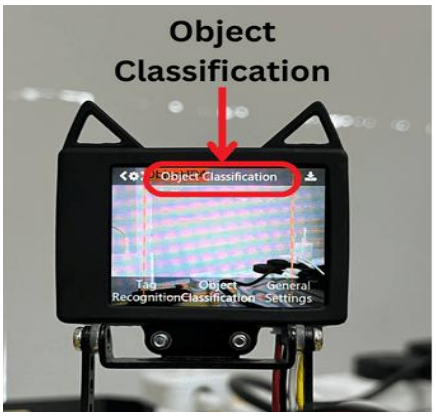


Figura 63 Modo Object Classification. Fuente: Autor

Una vez seleccionado este modo, se activó la opción Learning, en este estado, el sensor quedó listo para registrar nuevos objetos, el proceso no fue automático, ya que cada tipo de falla tuvo que ser presentado de forma individual frente a la cámara.

El cable con la falla visible se colocó frente al HuskyLens a corta distancia, así como se indica en la Figura 64. En algunos casos fue necesario mover ligeramente el sensor como se muestra en la Figura 65 hasta que la zona dañada quedara claramente visible en la pantalla. No siempre el primer encuadre funcionó, en varias pruebas fue necesario repetir la captura porque el área de interés no quedaba bien definida.

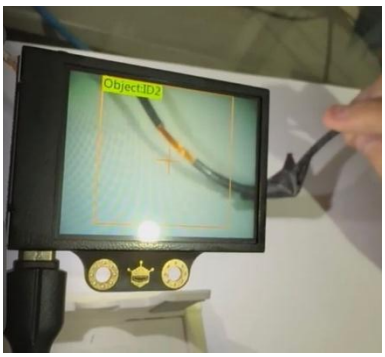


Figura 64 Falla visible en el sensor HuskyLens. Fuente: Autor

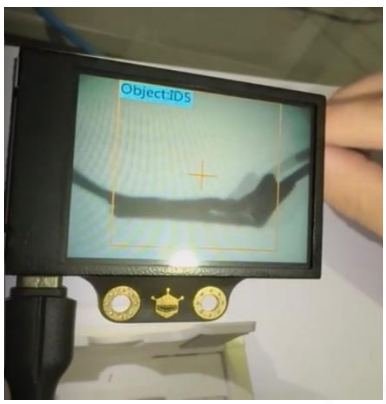


Figura 65 Movimiento del sensor para reconocimiento de daño. Fuente. Autor

Cuando la imagen fue estable, se presionó el botón de aprendizaje del dispositivo. El HuskyLens capturó el patrón y lo almacenó en memoria asignándole un identificador. Este procedimiento se repitió varias veces para el mismo tipo de falla. Se hicieron capturas con distintos ángulos y con cambios leves en la iluminación. Después de completar el entrenamiento de un tipo de falla, se continuó con el siguiente, clasificándolo en la Tabla 10 ya presentada, pero para mayor entendimiento la repetimos:

ID	Tipo de falla
ID1	UTP expuesto
ID2	UTP seccionado
ID3	UTP perforado con clavo
ID4	UTP entorchado
ID5	Cable eléctrico mal encintado
ID6	Rotura de núcleo en cable eléctrico

Durante el proceso se observó que el identificador podía cambiar cuando la imagen variaba bruscamente, sobre todo por cambios de luz o movimiento del cable, durante las pruebas se estableció que una detección solo sería válida cuando el mismo ID permanecía visible durante varios segundos de manera continua , así como se observa en la Figura 66, una vez finalizado el aprendizaje, el HuskyLens se dejó en modo de reconocimiento. En este estado, el sensor compara continuamente la imagen capturada con los patrones almacenados. Cuando se encuentra coincidencia, el identificador correspondiente es enviado al microcontrolador del sistema para su registro y tratamiento.

Figura 66 Detección con ID Fuente: Autores

Anexo 4

El anexo cuatro presenta el código fuente desarrollado en Arduino IDE para el control del prototipo robótico propuesto. Dicho programa se encarga de procesar las señales provenientes del joystick permitiendo la generación de órdenes de movimiento para los motores del sistema garantizando un control manual adecuado durante la operación del prototipo

Códigos para control remoto del robot

Joystick.ino

```
#include <Arduino.h>

// ------------------------------
// Pines LoRa (UART)
// ------------------------------
#define PIN_RX 16   // LoRa TX -> ESP32 RX
#define PIN_TX 17   // LoRa RX -> ESP32 TX

// ------------------------------
// Joystick
// ------------------------------
#define JOY_X 34
#define JOY_Y 35
#define JOY_SW 4


// LEDS
#define LED_ROJO  14     //
#define LED_VERDE 12    //

String lastCmd = "";

bool verdeActivo = false;
unsigned long verdeHasta = 0;
```

Joystick.ino

```
void setup() {
  Serial.begin(115200);

  // UART hacia el E32 (ya configurado por software)
  Serial2.begin(9600, SERIAL_8N1, PIN_RX, PIN_TX);

  pinMode(JOY_SW, INPUT_PULLUP);

  pinMode(LED_ROJO, OUTPUT);
  pinMode(LED_VERDE, OUTPUT);
  digitalWrite(LED_ROJO, LOW);
  digitalWrite(LED_VERDE, LOW);

  Serial.println("TX listo - Control del carrito");
}
```

Joystick.ino

```
41
42  void loop() {
43    // --- Recibir eventos desde el carrito ---
44    while (Serial2.available()) {
45      char ev = Serial2.read();
46
47      if (ev == '\n' || ev == '\r') continue;
48
49      if (ev == 'r') {
50        // Parpadeo rojo 1 vez (rápido)
51        digitalWrite(LED_ROJO, HIGH);
52        delay(80);
53        digitalWrite(LED_ROJO, LOW);
54      }
55      else if (ev == 'g') {
56        // Verde 3 segundos (sin bloquear demasiado)
57        digitalWrite(LED_VERDE, HIGH);
58        verdeActivo = true;
59        verdeHasta = millis() + 3000;
60      }
61    }
```

```
62
63    int xValue = analogRead(JOY_X);
64    int yValue = analogRead(JOY_Y);
65
66    int centro = 2048;
67    int margen = 400;
68
69    String cmd = "S";  // Stop por defecto
70
71    if (yValue > centro + margen)       cmd = "F";  // Forward
72    else if (yValue < centro - margen)  cmd = "B";  // Back
73    else if (xValue > centro + margen)  cmd = "R";  // Right
74    else if (xValue < centro - margen)  cmd = "L";  // Left
75
76    // Enviar SOLO si cambia el comando
77    if (cmd != lastCmd) {
78      Serial2.println(cmd);   // envío LoRa
79      Serial.print("Enviado: ");
80      Serial.println(cmd);
81      lastCmd = cmd;
82    }
83
84    delay(60);  // control suave sin saturar LoRa
85
86    if (verdeActivo && millis() > verdeHasta) {
87      digitalWrite(LED_VERDE, LOW);
88      verdeActivo = false;
89    }
90  }
```

Anexo 5

El anexo 5 presenta el código fuente desarrollado en Arduino IDE para el control inalámbrico del prototipo robótico propuesto. El programa permite la recepción de comandos enviados de forma remota y su posterior procesamiento para la ejecución de las acciones de movimiento del robot garantizando una comunicación estable durante su operación.

Código para robot inalámbrico

motores.ino

```
#include <Arduino.h>
#include <Wire.h>
#include <HUSKYLENS.h>
#include <WiFi.h>

// ------------------------------
// Pines LoRa (UART)
// ------------------------------
#define PIN_RX   16   // LoRa TX -> ESP32 RX
#define PIN_TX   17   // LoRa RX -> ESP32 TX

// ------------------------------
// DRIVER TB6612 - UN SOLO DRIVER
// ------------------------------

// Canal A → Motores IZQUIERDOS
#define AIN1 32
#define AIN2 33
#define PWMA 25

// Canal B → Motores DERECHOS
#define BIN1 4
#define BIN2 27
#define PWMB 26

#define STBY 23

```

motores.ino

```
28  const char* WIFI_SSID = "NETLIFE-CABAY";
29  const char* WIFI_PASS = "MAYLI2002";
30  // ----------------------------
31  unsigned long lastCmdTime = 0;
32  const unsigned long TIMEOUT_MS = 1500;
33  unsigned long lastAlertTime = 0;
34  const unsigned long ALERT_COOLDOWN_MS = 10000; // 10 segundos
35  unsigned long detectionStartTime = 0;
36  int stableID = -1;
37  const unsigned long DETECTION_TIME_MS = 5000; // 10 s
38  int lastDetectedID = -1;
39  HUSKYLENS husky;
40
41  // --- Notificación al control (LEDs en el joystick) ---
42  unsigned long lastTickSent = 0;
43  int tickCount = 0;
44
45
46  static unsigned long t = 0;
47  // ----------------------------
48  // SETUP
49  // ----------------------------
```

motores.ino

```
49  // ----------------------------
50  void setup() {
51    delay(500); // estabilizar E32
52
53    Serial.begin(115200);
54
55    WiFi.mode(WIFI_STA);
56    WiFi.begin(WIFI_SSID, WIFI_PASS);
57    Serial.print("Conectando a WiFi");
58    while (WiFi.status() != WL_CONNECTED) {
59      delay(500);
60      Serial.print(".");
61    }
62
63    Serial.println("\nWiFi conectado");
64    Serial.print("IP ESP32 normal: ");
65    Serial.println(WiFi.localIP());
66
67    Serial2.begin(9600, SERIAL_8N1, PIN_RX, PIN_TX);
68
69    Serial.println("RX listo - esperando comandos");
70
71    /*Wire.begin(21, 22);
72
73    while (!husky.begin(Wire)) {
74      Serial.println("HuskyLens no detectada");
75      delay(500);
76    }
77
78    Serial.println("HuskyLens lista");*/
```

motores.ino

```
79
80    // Limpiar basura inicial
81    while (Serial2.available()) Serial2.read();
82
83    // Pines TB6612
84    pinMode(AIN1, OUTPUT);
85    pinMode(AIN2, OUTPUT);
86    pinMode(PWMA, OUTPUT);
87
88    pinMode(BIN1, OUTPUT);
89    pinMode(BIN2, OUTPUT);
90    pinMode(PWMB, OUTPUT);
91
92    pinMode(STBY, OUTPUT);
93    digitalWrite(STBY, HIGH);
94
95    parar();
96    lastCmdTime = millis();
97
98  }
99
100 // ------------------------------
101 // LOOP
102 // ------------------------------
```

motores.ino

```
103 void loop() {
104
105   if (Serial2.available()) {
106     char cmd = Serial2.read();   // leer 1 byte
107
108     // Ignorar saltos de línea
109     if (cmd != '\n' && cmd != '\r') {
110
111     // Solo aceptar comandos válidos
112       bool valido = (cmd=='F' || cmd=='B' || cmd=='L' || cmd=='R' || cmd=='S');
113
114       if (valido) {
115         lastCmdTime = millis();
116
117         Serial.print("Comando recibido: ");
118         Serial.println(cmd);
119
120         switch (cmd) {
121           case 'F': avanzar(200); break;
122           case 'B': retroceder(200); break;
123           case 'L': girarIzquierda(200); break;
124           case 'R': girarDerecha(200); break;
125           case 'S': parar(); break;
126         }
127       } else {
128         // Ignorar cualquier otro byte (ruido o eventos)
129       }
130     }
131   }
132
```

motores.ino

```
132
133   // Failsafe: si se pierde enlace
134   if (millis() - lastCmdTime > TIMEOUT_MS) {
135     parar();
136   }
137
138   /*if (husky.request()) {
139     if (husky.available()) {
140       HUSKYLENSResult r = husky.read();
141
142       if (r.ID <= 0 || r.ID == 7 || r.ID == 8 || r.ID == 9) {
143         stableID = -1;
144         detectionStartTime = 0;
145
146         // reset ticks del conteo
147         tickCount = 0;
148         lastTickSent = 0;
149
150         return;
151       }
152
153       if (r.ID != stableID) {
154         stableID = r.ID;
155         detectionStartTime = millis();
156
157         // reset ticks al iniciar nuevo conteo
158         tickCount = 0;
159         lastTickSent = 0;
160
161         Serial.print("⏱ Iniciando conteo ID: ");
162         Serial.println(r.ID);
```

motores.ino

```
163
164         return;
165       }
166
167       if (tickCount < 5 && millis() - lastTickSent >= 900) {  // ~1s
168         sendToController('r');    // el control hace 1 parpadeo rojo
169         lastTickSent = millis();
170         tickCount++;
171       }
172
173       if (millis() - detectionStartTime >= DETECTION_TIME_MS) {
174         Serial.print("✅ ID estable por 5s: ");
175         Serial.println(r.ID);
176
177         enviarAlertaCam();
178
179         // avisar al control: encender verde 3s
180         sendToController('g');
181
182         // reset para evitar spam
183         stableID = -1;
184         detectionStartTime = 0;
185         tickCount = 0;
186         lastTickSent = 0;
187       }
188     }
189   }*/
190
```

motores.ino

```
191
192     if (millis() - t > 2000) {
193       t = millis();
194       sendToController('r');
195       Serial.println("TEST: mando r");
196     }
197   }
198
199   // ----------------------------
200   // FUNCIONES MOTOR
201   // ----------------------------
202   void avanzar(int speed) {
203     digitalWrite(AIN1, HIGH); digitalWrite(AIN2, LOW);
204     analogWrite(PWMA, speed);
205
206     digitalWrite(BIN1, HIGH); digitalWrite(BIN2, LOW);
207     analogWrite(PWMB, speed);
208   }
209
210   void retroceder(int speed) {
211     digitalWrite(AIN1, LOW);  digitalWrite(AIN2, HIGH);
212     analogWrite(PWMA, speed);
213
214     digitalWrite(BIN1, LOW);  digitalWrite(BIN2, HIGH);
215     analogWrite(PWMB, speed);
216   }
```

motores.ino

```
217
218   void girarDerecha(int speed) {
219     // Izquierda adelante
220     digitalWrite(AIN1, HIGH); digitalWrite(AIN2, LOW);
221     analogWrite(PWMA, speed);
222
223     // Derecha atrás
224     digitalWrite(BIN1, LOW);  digitalWrite(BIN2, HIGH);
225     analogWrite(PWMB, speed);
226   }
227
228   void girarIzquierda(int speed) {
229     // Izquierda atrás
230     digitalWrite(AIN1, LOW);  digitalWrite(AIN2, HIGH);
231     analogWrite(PWMA, speed);
232
233     // Derecha adelante
234     digitalWrite(BIN1, HIGH); digitalWrite(BIN2, LOW);
235     analogWrite(PWMB, speed);
236   }
237
238   void parar() {
239     analogWrite(PWMA, 0);
240     analogWrite(PWMB, 0);
241     digitalWrite(AIN1, LOW); digitalWrite(AIN2, LOW);
242     digitalWrite(BIN1, LOW); digitalWrite(BIN2, LOW);
243   }
```

motores.ino

```
244
245  void asegurarWiFi() {
246    if (WiFi.status() == WL_CONNECTED) return;
247
248    Serial.println("⚠️ WiFi caído, reconectando...");
249    WiFi.disconnect();
250    WiFi.begin(WIFI_SSID, WIFI_PASS);
251
252    unsigned long t0 = millis();
253    while (WiFi.status() != WL_CONNECTED && millis() - t0 < 5000) {
254      delay(500);
255      Serial.print(".");
256    }
257
258    if (WiFi.status() == WL_CONNECTED) {
259      Serial.println("\n✅ WiFi reconectado");
260      Serial.print("IP ESP32 normal: ");
261      Serial.println(WiFi.localIP());
262    } else {
263      Serial.println("\n❌ No se pudo reconectar WiFi");
264    }
265  }
266
```

motores.ino

```
267  /*void enviarAlertaCam() {
268    asegurarWiFi();
269
270    if (WiFi.status() != WL_CONNECTED) {
271      Serial.println("❌ WiFi no disponible, alerta cancelada");
272      return;
273    }
274
275    WiFiClient c;
276    c.setTimeout(1000);
277
278    IPAddress camIP(192,168,100,40);
279
280    if (!c.connect(camIP, 80)) {
281      Serial.println("❌ No conecta con ESP32-CAM");
282      return;
283    }
284
285    c.print(
286      "GET /alert HTTP/1.1\r\n"
287      "Host: 192.168.100.40\r\n"
288      "Connection: close\r\n\r\n"
289    );
290
291    c.stop();
292    Serial.println("⚠️ Alerta enviada al ESP32-CAM");
293  }*/
294
```

```
294
295   void sendToController(char code) {
296     // Enviamos 1 byte + salto de línea para que sea fácil leerlo al otro lado
297     Serial2.write(code);
298     Serial2.write('\n');
299   }
```

Anexo 6

El Anexo 6 contiene el código fuente completo del módulo ESP32-CAM, organizado en múltiples archivos de implementación y configuración. El sistema desarrollado en Arduino IDE integra la inicialización de la cámara, la gestión de la conectividad wifi mediante un portal de configuración y el manejo de los servicios de transmisión de imágenes empleadas en el prototipo robótico

Código de la esp32 cam

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | cam

```
1   #include "esp_camera.h"
2   #include <WiFi.h>
3   #include <WiFiManager.h>   // tzapu/WiFiManager
4   #include "FS.h"
5   #include "SD_MMC.h"
6   // ===========================
7   // Select camera model in board_config.h
8   // ===========================
9   #include "board_config.h"
10
11  bool sd_ok = false;
12  const char* ALERT_DIR = "/alerts";
13
14  void startCameraServer();
15  static void initSD();
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera

```
17  void setup() {
18    Serial.begin(115200);
19    Serial.setDebugOutput(true);
20    Serial.println();
21
22    // ---------- Cámara ----------
23    camera_config_t config;
24    config.ledc_channel = LEDC_CHANNEL_0;
25    config.ledc_timer   = LEDC_TIMER_0;
26    config.pin_d0       = Y2_GPIO_NUM;
27    config.pin_d1       = Y3_GPIO_NUM;
28    config.pin_d2       = Y4_GPIO_NUM;
29    config.pin_d3       = Y5_GPIO_NUM;
30    config.pin_d4       = Y6_GPIO_NUM;
31    config.pin_d5       = Y7_GPIO_NUM;
32    config.pin_d6       = Y8_GPIO_NUM;
33    config.pin_d7       = Y9_GPIO_NUM;
34    config.pin_xclk     = XCLK_GPIO_NUM;
35    config.pin_pclk     = PCLK_GPIO_NUM;
36    config.pin_vsync    = VSYNC_GPIO_NUM;
37    config.pin_href     = HREF_GPIO_NUM;
38    config.pin_sccb_sda = SIOD_GPIO_NUM;
39    config.pin_sccb_scl = SIOC_GPIO_NUM;
40    config.pin_pwdn     = PWDN_GPIO_NUM;
41    config.pin_reset    = RESET_GPIO_NUM;
42    config.xclk_freq_hz = 20000000;
43    config.frame_size   = FRAMESIZE_VGA;         // ajusta si quieres
44    config.pixel_format = PIXFORMAT_JPEG;         // streaming
45    config.grab_mode    = CAMERA_GRAB_LATEST;
46    config.fb_location  = CAMERA_FB_IN_PSRAM;
47    config.jpeg_quality = 12;
48    config.fb_count     = 1;
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h | ci.json

```
49
50    esp_err_t err = esp_camera_init(&config);
51    if (err != ESP_OK) {
52      Serial.printf("Camera init failed with error 0x%x\n", err);
53      return;
54    }
55
56
57
58    // ---------- WiFi con WiFiManager (sin credenciales en firmware) ----------
59    WiFi.mode(WIFI_STA);
60    WiFi.setTxPower(WIFI_POWER_19_5dBm);  // potencia TX más alta permitida en Arduino-ESP32
61    WiFi.setSleep(false);
62    WiFi.disconnect(true, true);  // limpia estado previo para primer arranque
63    delay(200);
64
65    WiFiManager wm;
66    wm.setRemoveDuplicateAPs(true);
67    wm.setConfigPortalBlocking(true);
68
69    Serial.println("[WM] autoConnect(\"ESP32CAM-Setup\") ...");
70    // Si no hay credenciales guardadas, abre el portal "ESP32CAM-Setup" (clave "setup1234")
71    bool ok = wm.autoConnect("ESP32CAM-Setup", "setup1234");
72
73    if (!ok) {
74      Serial.println("[WM] Portal cerrado sin guardar. Reiniciando...");
75      delay(1500);
76      ESP.restart();
77      return;
78    }
```

```
79
80      // Conectado a la red elegida -> IP dinámica
81      Serial.print("[STA] Conectado. IP: ");
82      Serial.println(WiFi.localIP());
83
84      // ---------- Servidor de la cámara ----------
85      startCameraServer();
86
87      Serial.print("Camera Ready! Use 'http://");
88      Serial.print(WiFi.localIP());
89      Serial.println("' to connect");
90    }
91
92    static void initSD() {
93      if (!SD_MMC.begin("/sdcard", true)) { // true = 1-bit (más estable)
94        log_e("[SD] No montó SD_MMC");
95        sd_ok = false;
96        return;
97      }
98      if (SD_MMC.cardType() == CARD_NONE) {
99        log_e("[SD] No hay SD");
100       sd_ok = false;
101       return;
102     }
103     sd_ok = true;
104     if (!SD_MMC.exists(ALERT_DIR)) SD_MMC.mkdir(ALERT_DIR);
105     log_i("[SD] OK");
106   }
107
108   void loop() {
109   }
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h | ci.json

```
1   #include "esp_http_server.h"
2   #include "esp_timer.h"
3   #include "esp_camera.h"
4   #include "sdkconfig.h"
5   #include "board_config.h"
6   #include <WiFi.h>
7   #include <Client.h>
8   #include "FS.h"
9   #include "SD_MMC.h"
10
11  extern bool sd_ok;
12  extern const char* ALERT_DIR;
13
14  #define SERVER_IP_FALLBACK IPAddress(192,168,100,5)
15  #define SERVER_PORT_FALLBACK 10080
16
17  #if defined(ARDUINO_ARCH_ESP32) && defined(CONFIG_ARDUHAL_ESP_LOG)
18  #include "esp32-hal-log.h"
19  #endif
20
21  // Ruta para subida chunked al servidor
22  static const char* PC_PATH = "/upload";
23
24  // CONTENIDO STREAM (MJPEG multipart)
25  #define PART_BOUNDARY "123456789000000000000987654321"
26  static const char *_STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY;
27  static const char *_STREAM_BOUNDARY     = "\r\n--" PART_BOUNDARY "\r\n";
28  static const char *_BOUNDARY_FIRST      = "--" PART_BOUNDARY "\r\n";
29  static const char *_STREAM_PART         = "Content-Type: image/jpeg\r\nContent-Length: %u\r\nX-Timestamp: %d.%06d\r\n\r\n";
30
31  httpd_handle_t stream_httpd = NULL;
32  httpd_handle_t camera_httpd = NULL;
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_

```
33
34  // Filtro rolling-average para métricas de frame time
35  typedef struct {
36    size_t size;
37    size_t index;
38    size_t count;
39    int sum;
40    int *values;
41  } ra_filter_t;
42
43  static ra_filter_t ra_filter;
44
45  // Estado de duplicación: contador de frames críticos a capturar
46  static volatile bool alert_pending = false;
47
48  // Utilidad: escribir todo el buffer
49  static bool writeAll(Client &c, const uint8_t* p, size_t n) {
50    while (n) {
51      int w = c.write(p, n);
52      if (w <= 0) return false;
53      p += w;
54      n -= (size_t)w;
55    }
56    return true;
57  }
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | can

```
58
59  // Utilidad: escribir chunk HTTP
60  static bool writeChunk(Client& c, const uint8_t* p, size_t n) {
61    char head[16];
62    int hl = snprintf(head, sizeof(head), "%X\r\n", (unsigned)n);
63    if (c.write((const uint8_t*)head, hl) != hl) return false;
64    if (!writeAll(c, p, n)) return false;
65    if (c.write((const uint8_t*)"\r\n", 2) != 2) return false;
66    return true;
67  }
68
69  static bool endChunks(Client& c) {
70    return c.print("0\r\n\r\n") > 0;
71  }
72
73  static bool saveJpgToSD(const uint8_t* buf, size_t len) {
74    if (!sd_ok || !buf || !len) return false;
75
76    char path[64];
77    static uint32_t alert_seq = 0;
78    snprintf(path, sizeof(path), "%s/%lu_%lu.jpg", ALERT_DIR,
79             (unsigned long)millis(), (unsigned long)alert_seq++);
80
81    File f = SD_MMC.open(path, FILE_WRITE);
82    if (!f) return false;
83
84    size_t w = f.write(buf, len);
85    f.close();
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h

```
86
87    if (w != len) {
88      SD_MMC.remove(path);
89      return false;
90    }
91
92    log_i("[SD] Guardado %s (%u bytes)", path, (unsigned)len);
93    return true;
94  }
95
96  static bool postFileNoAck(const char* path) {
97    if (!sd_ok) return false;
98
99    File f = SD_MMC.open(path, FILE_READ);
100   if (!f) {
101     log_e("[SD->SERVER] No pude abrir %s", path);
102     return false;
103   }
104
105   size_t len = f.size();
106   WiFiClient c;
107   c.setNoDelay(true);
108
109   if (!c.connect(SERVER_IP_FALLBACK, SERVER_PORT_FALLBACK)) {
110     log_e("[SD->SERVER] No conectó al server (%s:%u)",
111           SERVER_IP_FALLBACK.toString().c_str(), SERVER_PORT_FALLBACK);
112     f.close();
113     return false;
114   }
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h

```
116   char hdr[320];
117   snprintf(hdr, sizeof(hdr),
118     "POST /image HTTP/1.1\r\n"
119     "Host: %s:%u\r\n"
120     "Content-Type: image/jpeg\r\n"
121     "Content-Length: %u\r\n"
122     "X-Filename: %s\r\n"
123     "Connection: close\r\n\r\n",
124     SERVER_IP_FALLBACK.toString().c_str(),
125     SERVER_PORT_FALLBACK,
126     (unsigned)len,
127     path
128   );
129
130   // Header
131   if (!writeAll(c, (const uint8_t*)hdr, strlen(hdr))) {
132     log_e("[SD->SERVER] Falló header %s", path);
133     f.close(); c.stop();
134     return false;
135   }
136
137   // Body (archivo)
138   uint8_t buf[1024];
139   while (f.available()) {
140     int r = f.read(buf, sizeof(buf));
141     if (r <= 0) break;
142     if (!writeAll(c, buf, (size_t)r)) {
143       log_e("[SD->SERVER] Falló body %s", path);
144       f.close(); c.stop();
145       return false;
146     }
147   }
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | cam

```
148
149     f.close();
150     c.stop();
151     return true; // éxito = "se envió completo"
152   }
153
154   static void uploadAlertsFromSD_NoAck() {
155     if (!sd_ok) {
156       log_w("[SD] SD no montada, no subo alerts");
157       return;
158     }
159
160     File dir = SD_MMC.open(ALERT_DIR);
161     if (!dir || !dir.isDirectory()) {
162       log_w("[SD] No existe %s", ALERT_DIR);
163       return;
164     }
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h | ci.jso

```
166     File f = dir.openNextFile();
167     while (f) {
168       if (!f.isDirectory()) {
169         String name = f.name();
170         f.close();
171
172         String fullPath;
173         if (name.startsWith("/")) {
174           fullPath = name; // ya viene absoluto
175         } else if (name.startsWith(String(ALERT_DIR).substring(1))) {
176           fullPath = "/" + name; // "alerts/x.jpg" -> "/alerts/x.jpg"
177         } else {
178           fullPath = String(ALERT_DIR) + "/" + name; // "x.jpg" -> "/alerts/x.jpg"
179         }
180
181         log_i("[SD->SERVER] Enviando %s", fullPath.c_str());
182         bool sent = postFileNoAck(fullPath.c_str());
183
184         if (sent) {
185           SD_MMC.remove(fullPath.c_str());
186           log_i("[SD] Borrado %s", fullPath.c_str());
187         } else {
188           log_w("[SD->SERVER] Falló envío, no borro %s", fullPath.c_str());
189         }
190       } else {
191         f.close();
192       }
193       f = dir.openNextFile();
194     }
195
196     dir.close();
197   }
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h | ci.json

```
198
199  // Abrir POST chunked para subir el stream al servidor
200  static bool openPcUpload(WiFiClient &pc) {
201    IPAddress ip = SERVER_IP_FALLBACK;
202    uint16_t port = SERVER_PORT_FALLBACK;
203    String host = ip.toString();
204    pc.setNoDelay(true);
205    if (!pc.connect(ip, port)) {
206      log_e("[UPLOAD] connect() falló a %s:%u", ip.toString().c_str(), port);
207      return false;
208    }
209
210    char hdr[256];
211    snprintf(hdr, sizeof(hdr),
212             "POST %s HTTP/1.1\r\n"
213             "Host: %s:%u\r\n"
214             "Content-Type: %s\r\n"
215             "Transfer-Encoding: chunked\r\n"
216             "Connection: keep-alive\r\n\r\n",
217             PC_PATH, host.c_str(), port, _STREAM_CONTENT_TYPE);
218
219    pc.write((const uint8_t*)hdr, strlen(hdr));
220    log_i("[UPLOAD] POST %s abierto (HTTP/1.1 + chunked)", PC_PATH);
221
222    // 4) Lee respuesta inicial del server (status line)
223    pc.setTimeout(2000);
224    String status = pc.readStringUntil('\n'); // "HTTP/1.1 200 OK\r\n"
225    status.trim();
226    log_i("[UPLOAD] Status: %s", status.c_str());
227    if (!status.startsWith("HTTP/1.1 200") && !status.startsWith("HTTP/1.1 204") &&
228        !status.startsWith("HTTP/1.0 200") && !status.startsWith("HTTP/1.0 204")) {
229      log_w("[UPLOAD] Status no exitoso (el server podría no crear archivo).");
230    }
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h

```
231    // Opcional: consumir headers del server
232    while (true) {
233      String h = pc.readStringUntil('\n');
234      if (h == "\r" || h.length()==0) break;
235    }
236
237    return true;
238  }
239
240  static ra_filter_t *ra_filter_init(ra_filter_t *filter, size_t sample_size) {
241    memset(filter, 0, sizeof(ra_filter_t));
242    filter->values = (int *)malloc(sample_size * sizeof(int));
243    if (!filter->values) {
244      return NULL;
245    }
246    memset(filter->values, 0, sample_size * sizeof(int));
247    filter->size = sample_size;
248    return filter;
249  }
250
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | ca

```
250
251 #if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
252 static int ra_filter_run(ra_filter_t *filter, int value) {
253   if (!filter->values) {
254     return value;
255   }
256   filter->sum -= filter->values[filter->index];
257   filter->values[filter->index] = value;
258   filter->sum += filter->values[filter->index];
259   filter->index++;
260   filter->index = filter->index % filter->size;
261   if (filter->count < filter->size) {
262     filter->count++;
263   }
264   return filter->sum / filter->count;
265 }
266 #endif
267
268 // Handler de alerta: activa contador de frames críticos
269 static esp_err_t alert_handler(httpd_req_t *req) {
270   alert_pending = true;
271   log_i("[ALERT] Se capturarán 1 frame crítico");
272   httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
273   return httpd_resp_send(req, "OK", 2);
274 }
275
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | came

```
275
276 // Info WiFi (para el panel)
277 static esp_err_t wifi_handler(httpd_req_t *req) {
278   char json[160];
279   snprintf(json, sizeof(json),
280            "{\"ssid\":\"%s\",\"rssi\":%d,\"ip\":\"%s\"}",
281            WiFi.SSID().c_str(), WiFi.RSSI(),
282            WiFi.localIP().toString().c_str());
283   httpd_resp_set_type(req, "application/json");
284   httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
285   return httpd_resp_send(req, json, strlen(json));
286 }
287
288 // STREAM MJPEG
289 static esp_err_t stream_handler(httpd_req_t *req) {
290   camera_fb_t *fb = NULL;
291   struct timeval _timestamp;
292   esp_err_t res = ESP_OK;
293   size_t _jpg_buf_len = 0;
294   uint8_t *_jpg_buf = NULL;
295   char part_buf[128];
296
297   static int64_t last_frame = 0;
298   if (!last_frame) last_frame = esp_timer_get_time();
299
300   // Respuesta al navegador (start stream)
301   res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
302   if (res != ESP_OK) return res;
303   httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
304   httpd_resp_set_hdr(req, "X-Framerate", "10");
305
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h

```
305
306   // Abrir subida al servidor (chunked)
307   WiFiClient pc;
308   bool upl_ok = openPcUpload(pc);
309   bool upl_first = true;
310
311   while (true) {
312     fb = esp_camera_fb_get();
313     if (!fb) {
314       log_e("Camera capture failed");
315       res = ESP_FAIL;
316     } else {
317       _timestamp.tv_sec  = fb->timestamp.tv_sec;
318       _timestamp.tv_usec = fb->timestamp.tv_usec;
319
320       if (fb->format != PIXFORMAT_JPEG) {
321         bool ok = frame2jpg(fb, 80, &_jpg_buf, &_jpg_buf_len);
322         esp_camera_fb_return(fb); fb = NULL;
323         if (!ok) { log_e("JPEG compression failed"); res = ESP_FAIL; }
324       } else {
325         _jpg_buf_len = fb->len;
326         _jpg_buf = fb->buf;
327       }
328
329       // Duplicación de frames críticos mientras el contador esté activo
330       if (alert_pending && _jpg_buf && _jpg_buf_len) {
331         alert_pending = false;              // importantísimo: solo una vez
332         saveJpgToSD(_jpg_buf, _jpg_buf_len);
333         log_i("[ALERT] Frame guardado en SD (1/1)");
334       }
335     }
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h | ci.json

```
336
337   // 1) Enviar al navegador
338   if (res == ESP_OK) res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(_STREAM_BOUNDARY));
339   if (res == ESP_OK) {
340     size_t hlen = snprintf(part_buf, sizeof(part_buf), _STREAM_PART,
341                            (unsigned)_jpg_buf_len,
342                            (int)_timestamp.tv_sec, (int)_timestamp.tv_usec);
343     res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
344   }
345   if (res == ESP_OK) res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
346
347   // 2) Enviar al servidor (chunked)
348   if (upl_ok && pc.connected()) {
349     const char* b = upl_first ? _BOUNDARY_FIRST : _STREAM_BOUNDARY;
350     upl_first = false;
351     if (!writeChunk(pc, (const uint8_t*)b, strlen(b))) { upl_ok=false; goto pc_fail; }
352
353     size_t hlen = snprintf(part_buf, sizeof(part_buf), _STREAM_PART,
354                            (unsigned)_jpg_buf_len,
355                            (int)_timestamp.tv_sec, (int)_timestamp.tv_usec);
356     if (!writeChunk(pc, (const uint8_t*)part_buf, hlen)) { upl_ok=false; goto pc_fail; }
357
358     if (!writeChunk(pc, (const uint8_t*)_jpg_buf, _jpg_buf_len)) { upl_ok=false; goto pc_fail; }
359
360     if (!writeChunk(pc, (const uint8_t*)"\r\n", 2)) { upl_ok=false; goto pc_fail; }
361   }
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h

```
362
363     pc_fail:
364       if (!upl_ok && pc.connected()) {
365         endChunks(pc);
366         pc.stop();
367         log_e("[UPLOAD] uplink cortado: cerré POST (EOF para server)");
368       }
369
370       // liberar buffers
371       if (fb) {
372         esp_camera_fb_return(fb);
373         fb = NULL;
374         _jpg_buf = NULL; // si fb->format == JPEG, buffer pertenece al driver
375       } else if (_jpg_buf) {
376         free(_jpg_buf);  // si se creó con frame2jpg
377         _jpg_buf = NULL;
378       }
379
380       if (res != ESP_OK) {
381         log_e("Send frame failed");
382         break;
383       }
384
385       // métricas
386       int64_t now = esp_timer_get_time();
387       int64_t frame_time_ms = (now - last_frame) / 1000;
388       last_frame = now;
389       #if ARDUHAL_LOG_LEVEL >= ARDUHAL_LOG_LEVEL_INFO
390       int avg = ra_filter_run(&ra_filter, (int)frame_time_ms);
391       log_i("MJPG: %uB %lldms (%.1ffps) | AVG %dms (%.1ffps)",
392             (uint32_t)_jpg_buf_len, (long long)frame_time_ms,
393             1000.0f/(float)frame_time_ms, avg, 1000.0f/(float)avg);
394       #endif
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h | ci.json

```
395
396       vTaskDelay(100 / portTICK_PERIOD_MS);
397     }
398
399     if (pc.connected()) {
400       endChunks(pc);
401       pc.stop();
402       log_i("[UPLOAD] cerrado (final chunked enviado)");
403     }
404     uploadAlertsFromSD_NoAck();
405
406     return res;
407   }
408
409   // Página simple para control del stream
410   static esp_err_t index_handler(httpd_req_t *req) {
411     httpd_resp_set_type(req, "text/html");
412     httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
413     const char *index_html =
414       "<!doctype html>"
415       "<html><head>"
416       "<meta name=\"viewport\" content=\"width=device-width,initial-scale=1\">"
417       "<style>"
418       "body{margin:0;padding:20px;background:#111;color:#0f0;font-family:monospace}"
419       "h1{margin:0 0 10px 0}"
420
421       ".status{padding:10px;background:rgba(0,0,0,0.5);margin:10px 0;font-size:12px;border:1px solid rgba(0,255,0,0.25);border-radius:10px}"
422
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h | ci.json

```
422
423   /* ===== Lienzo/Contenedor del stream ===== */
424   ".streamWrap{max-width:520px;margin:0 auto 10px auto}"            /* ancho del panel */
425   ".streamFrame{width:100%;aspect-ratio:4/3;background:#000;"       /* 4:3 para VGA */
426   "border:1px solid #0f0;border-radius:12px;overflow:hidden;"
427   "display:flex;align-items:center;justify-content:center}"
428
429   /* El stream se adapta al lienzo, NO a la pantalla */
430   "#stream{width:100%;height:100%;display:none;object-fit:contain}"
431
432   ".controls{padding:10px;background:rgba(0,0,0,0.8);margin:10px 0;"
433   "border:1px solid rgba(0,255,0,0.25);border-radius:10px;max-width:520px;margin-left:auto;margin-right:auto}"
434   "button{padding:10px 14px;border-radius:10px;border:1px solid #0f0;background:#111;color:#0f0;font-family:monospace;cursor:pointer}"
435   "button:active{transform:scale(0.99)}"
436   "</style>"
437   "</head><body>"
438   "<div class=\"streamWrap\">"
439   "  <h1>ESP32-CAM</h1>"
440   "  <div class=\"status\">"
441   "    <span id=\"rssi\">RSSI: -- dBm</span> | "
442   "    <span id=\"ip\">IP: --.--.--.--</span>"
443   "  </div>"
444
445   "  <div class=\"streamFrame\">"
446   "    <img id=\"stream\" crossorigin>"
447   "  </div>"
448   "</div>"
449
450   "<div class=\"controls\">"
451   "  <button id=\"streamBtn\" onclick=\"toggleStream()\">Start Stream</button>"
452   "</div>"
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h | ci.json

```
453
454   "<script>"
455   "let streaming=false;"
456   "const img=document.getElementById('stream');"
457   "const btn=document.getElementById('streamBtn');"
458
459   "function toggleStream(){"
460   "  if(streaming){"
461   "    img.src='about:blank';"
462   "    img.style.display='none';  "
463   "    btn.textContent='Start Stream';"
464   "    streaming=false;"
465   "  }else{"
466   "    img.style.display='block';"
467   "    img.src='/stream?x=' + Date.now();"   /* cache-buster */
468   "    btn.textContent='Stop Stream';"
469   "    streaming=true;"
470   "  }"
471   "}"
472
473   "async function updateStatus(){"
474   "  try{"
475   "    let r=await fetch('/wifi');"
476   "    let j=await r.json();"
477   "    document.getElementById('rssi').textContent = `RSSI: ${j.rssi} dBm`;"
478   "    document.getElementById('ip').textContent   = `IP: ${j.ip}`;"
479   "  }catch(e){}"
480   "  setTimeout(updateStatus,2000);"
481   "}"
482   "updateStatus();"
483   "</script>"
484   "</body></html>";
485
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera

```
485
486     return httpd_resp_send(req, index_html, strlen(index_html));
487   }
488
489   void startCameraServer() {
490     httpd_config_t config = HTTPD_DEFAULT_CONFIG();
491     config.stack_size = 12288;   // prueba 8192, si aún cae, 12288
492     config.max_uri_handlers = 16;
493
494     httpd_uri_t index_uri = {
495       .uri = "/",
496       .method = HTTP_GET,
497       .handler = index_handler,
498       .user_ctx = NULL
499     };
500
501     httpd_uri_t wifi_uri = {
502       .uri = "/wifi",
503       .method = HTTP_GET,
504       .handler = wifi_handler,
505       .user_ctx = NULL
506     };
507
508     httpd_uri_t alert_uri = {
509       .uri = "/alert",
510       .method = HTTP_GET,
511       .handler = alert_handler,
512       .user_ctx = NULL
513     };
```

```
514
515   httpd_uri_t stream_uri = {
516     .uri = "/stream",
517     .method = HTTP_GET,
518     .handler = stream_handler,
519     .user_ctx = NULL
520   };
521
522   ra_filter_init(&ra_filter, 20);
523   log_i("Starting web server on port: '%d'", config.server_port);
524
525   if (httpd_start(&camera_httpd, &config) == ESP_OK) {
526     httpd_register_uri_handler(camera_httpd, &index_uri);
527     httpd_register_uri_handler(camera_httpd, &wifi_uri);
528     httpd_register_uri_handler(camera_httpd, &stream_uri);
529     httpd_register_uri_handler(camera_httpd, &alert_uri);
530
531     // Segundo servidor para stream (opcional, según tu arquitectura)
532     config.server_port += 1;
533     config.ctrl_port += 1;
534     log_i("Starting stream server on port: '%d'", config.server_port);
535     if (httpd_start(&stream_httpd, &config) == ESP_OK) {
536       httpd_register_uri_handler(stream_httpd, &stream_uri);
537       httpd_register_uri_handler(stream_httpd, &alert_uri);
538     }
539   }
540 }
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h

```
#ifndef BOARD_CONFIG_H
#define BOARD_CONFIG_H

#define CAMERA_MODEL_AI_THINKER // Has PSRAM

#include "camera_pins.h"

#endif  // BOARD_CONFIG_H
```

ESP32CAMPRUEBAS.ino | app_httpd.cpp | board_config.h | camera_index.h | camera_pins.h

```

#if defined(CAMERA_MODEL_WROVER_KIT)
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM  21
#define SIOD_GPIO_NUM  26
#define SIOC_GPIO_NUM  27

#define Y9_GPIO_NUM    35
#define Y8_GPIO_NUM    34
#define Y7_GPIO_NUM    39
#define Y6_GPIO_NUM    36
#define Y5_GPIO_NUM    19
#define Y4_GPIO_NUM    18
#define Y3_GPIO_NUM    5
#define Y2_GPIO_NUM    4
#define VSYNC_GPIO_NUM 25
#define HREF_GPIO_NUM  23
#define PCLK_GPIO_NUM  22


#elif defined(CAMERA_MODEL_AI_THINKER)
#define PWDN_GPIO_NUM  32
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM  0
#define SIOD_GPIO_NUM  26
#define SIOC_GPIO_NUM  27

#define Y9_GPIO_NUM    35
#define Y8_GPIO_NUM    34
#define Y7_GPIO_NUM    39
#define Y6_GPIO_NUM    36
#define Y5_GPIO_NUM    21
#define Y4_GPIO_NUM    19
#define Y3_GPIO_NUM    18
#define Y2_GPIO_NUM    5
#define VSYNC_GPIO_NUM 25
#define HREF_GPIO_NUM  23
#define PCLK_GPIO_NUM  22

// 4 for flash led or 33 for normal led
#define LED_GPIO_NUM   4

#else
#error "Camera model not selected"
#endif
```

Anexo 7

El anexo 7 tiene como objetivo explicar el levantamiento de un servidor en visual estudio para la implementación del sistema de recepción de vídeos se desarrolló un servidor local utilizando la plataforma Node.js, el cual permite recibir imágenes y flujos de video enviados desde una Cámara ESP32-CAM Mediante conexión wifi.

El servidor funciona como un punto central de recepción, encargado de escuchar las peticiones provenientes de la Cámara y procesar la información multimedia recibida.

La Cámara ESP32-CAM transmite imágenes de un formato JPEG de manera continua a través del protocolo HTTP. Este conjunto de imágenes consecutivas conforma un flujo de video del tipo MJPEG (Motion JPEG).

El servidor recibe este flujo mediante una ruta específica de recepción. Cada imagen enviada por la Cámara es capturada por el servidor y transferida a una herramienta de procesamiento de video cómo que se encarga de unirlas y convertirlas en un archivo de video con formato AVI.

De esta manera el video no se almacena como un archivo único desde la cámara, sino que se reconstruye en el servidor a partir de múltiples imágenes recibidas en tiempo real.

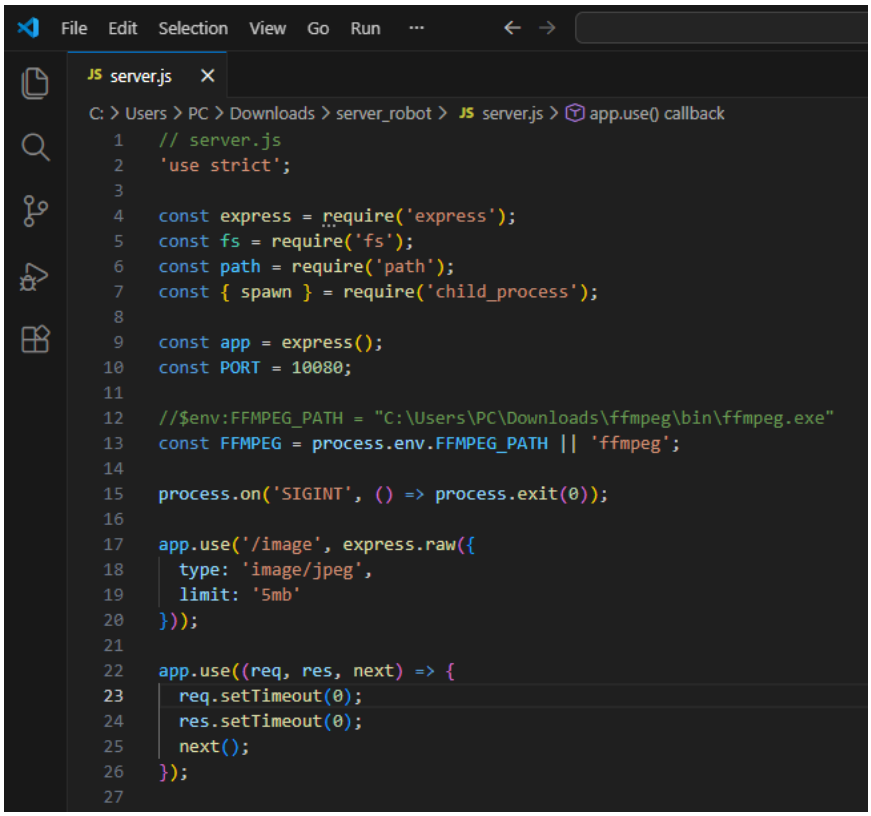
El servidor integra una herramienta de procesamiento multimedia que permite unir automáticamente las imágenes recibidas garantizando que el video se genere sin pérdidas de calidad y de forma eficiente.

Cada grabación es guardada con un nombre que incluya la fecha y la hora lo que facilita la organización y la identificación de los archivos. Los vídeos generados se almacenan en una carpeta específica de servidor, permitiendo su posterior visualización, descarga o análisis.

Una vez almacenados coma los archivos de video pueden ser consultados desde cualquier dispositivo conectado a la misma red local como accediendo mediante un navegador web a la dirección del servidor. Esto permitirá visualizar y descargar los vídeos generados sin necesidad de retirar la tarjeta de memoria del sistema.

El servidor cumple un rol fundamental dentro del sistema y actúa como intermediario entre la Cámara y el almacenamiento gracias a su implementación se evita sobrecargar la memoria del SP 32 CAM y se garantiza una mejor administración de los vídeos, permitiendo un funcionamiento más estable seguro y organizado del sistema de monitoreo

Código de levantamiento de server en java + visual studio code




```
// server.js
'use strict';

const express = require('express');
const fs = require('fs');
const path = require('path');
const { spawn } = require('child_process');

const app = express();
const PORT = 10080;

//$env:FFMPEG_PATH = "C:\Users\PC\Downloads\ffmpeg\bin\ffmpeg.exe"
const FFMPEG = process.env.FFMPEG_PATH || 'ffmpeg';

process.on('SIGINT', () => process.exit(0));

app.use('/image', express.raw({
  type: 'image/jpeg',
  limit: '5mb'
}));

app.use((req, res, next) => {
  req.setTimeout(0);
  res.setTimeout(0);
  next();
});
```

```
// Nombre con timestamp
function tsName(ext = 'avi') {
  const d = new Date();
  return `rec_${d.getFullYear()}${String(d.getMonth()+1).padStart(2,'0')}${String(d.getDate()).padStart(2,'0')}_` +
         `${String(d.getHours()).padStart(2,'0')}${String(d.getMinutes()).padStart(2,'0')}${String(d.getSeconds()).padStart(2,'0')}.${ext}`;
}

app.post('/image', (req, res) => {
  if (!req.body || !req.body.length) {
    return res.status(400).send('No image received');
  }

  const outDir = path.join(__dirname, 'imagenes');
  if (!fs.existsSync(outDir)) {
    fs.mkdirSync(outDir, { recursive: true });
  }

  const imgPath = path.join(outDir, tsName('jpg'));

  try {
    fs.writeFileSync(imgPath, req.body);
    console.log(`🖼 Imagen guardada: ${imgPath}`);
    res.status(200).send('ok');
  } catch (err) {
    console.error('Error guardando imagen:', err.message);
    res.status(500).send('error');
  }
});
```

```
57  app.post('/upload', (req, res) => {
58    // Carpeta de salida
59    const outDir = path.join(__dirname, 'videos');
60    if (!fs.existsSync(outDir)) fs.mkdirSync(outDir, { recursive: true });
61
62    const aviPath = path.join(outDir, tsName('avi'));
63
64      // === CONTADOR DE ANCHO DE BANDA (kbps) ===
65    let bytes = 0;      // total recibidos en esta subida
66    let win   = 0;      // acumulado del último segundo
67    let tick  = setInterval(() => {
68      const kbps = (win * 8) / 1000; // kbps aprox. último segundo
69      console.log(`↗ upload: ${kbps.toFixed(1)} kbps`);
70      win = 0;
71    }, 1000);
72
73    req.on('data', chunk => { bytes += chunk.length; win += chunk.length; });
74    req.on('end',  () => console.log('  FIN subida HTTP (req.end). Total bytes:', bytes));
75    req.on('close', () => { clearInterval(tick); console.log('  Cliente cerró conexión'); });
76    req.on('error', (e) => { clearInterval(tick); console.error('[req error]', e.message); });
77
78    // Lanza ffmpeg
79    const ff = spawn(FFMPEG, [
80      '-hide_banner', '-loglevel', 'error',
81      '-f', 'mpjpeg',    // demuxer para multipart/x-mixed-replace; boundary=...
82      '-i', 'pipe:0',    // stdin
83      '-an',             // sin audio
84      '-c:v', 'copy',    // copia MJPEG directo -> AVI (rápido, sin pérdida)
85      aviPath
86    ]);
87
88    ff.on('error', (e) => {
89      console.error('No pude lanzar FFmpeg:', e.message);
90      // Responder algo al cliente para que no quede colgado
91      try { res.status(500).end('ffmpeg spawn error'); } catch(_) {}
92    });
93
94    // Pipe de la subida hacia ffmpeg
95    req.pipe(ff.stdin);
96
97    ff.stdin.on('error', (e) => console.error('[ffmpeg stdin]', e.message));
98    ff.stderr.on('data', (d) => console.error('[ffmpeg]', d.toString().trim()));
99
100   // Cuando ffmpeg termina, respondemos
101   ff.on('close', (code) => {
102     if (code === 0) {
103       console.log(`✓ AVI listo: ${aviPath} | ${bytes} bytes recibidos`);
104       res.status(200).end('ok');
105     } else {
106       console.error(`X ffmpeg terminó con código ${code}`);
107       res.status(500).end('ffmpeg error');
108     }
109   });
110
111   // Cortes de conexión
112   req.on('close', () => { try { ff.stdin.end(); } catch(_) {} });
113   req.on('error', (e) => { console.error('[req error]', e.message); try { ff.kill('SIGKILL'); } catch(_) {} });
114 });
115
116   // Listado / descarga de videos y ping simple
117   app.use('/videos', express.static(path.join(__dirname, 'videos')));
118   app.get('/', (_req, res) => res.send('OK'));
119
120   app.listen(PORT, '0.0.0.0', () => {
121     console.log(`Listening on http://0.0.0.0:${PORT}`);
122     console.log('Archivos AVI en: ./videos');
123     console.log('📷 Imágenes en: ./imágenes');
124   });
```