UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE GUAYAQUIL

CARRERA DE ELECTRÓNICA Y AUTOMATIZACIÓN

DISEÑO E IMPLEMENTACIÓN DE UN ROBOT CUADRÚPEDO PARA COMPETENCIAS CON CONTROL AUTÓNOMO O MANUAL MEDIANTE UNA APLICACIÓN MÓVIL.

Trabajo de titulación previo a la obtención del

Título de Ingeniero en Electrónica

AUTORES: ISRAEL MARCELO MOYA HERRERA

ARLENA OMAYRA ZAMBRANO CUADRO

TUTOR: ING. RAFAEL CHRISTIAN FRANCO REINA, MSc

Guayaquil – Ecuador

2025

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, Israel Marcelo Moya Herrera con documento de identificación N° 0940347206 y Arlena Omayra Zambrano Cuadro con documento de identificación N° 0942064916, manifestamos que:

Somos las 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, 29 de enero del año 2025.

Atentamente,

Israel Marcelo Moya Herrera
0940347206

Arlena Omayra Zambrano Cuadro
0942064916

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

Nosotros, Israel Marcelo Moya Herrera con documento de identificación N° 0940347206 y Arlena Omayra Zambrano Cuadro con documento de identificación N° 0942064916, manifestamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Proyecto Técnico: "DISEÑO E IMPLEMENTACIÓN DE UN ROBOT CUADRÚPEDO PARA COMPETENCIAS CON CONTROL AUTÓNOMO O MANUAL MEDIANTE UNA APLICACIÓN MÓVIL", el cual ha sido desarrollado para optar por el título de: Ingeniero en Electrónica, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

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

Guayaquil, 29 de enero del año 2025.

Atentamente,

Israel Marcelo Moya Herrera
0940347206

Arlena Omayra Zambrano Cuadro
0942064916

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Rafael Christian Franco Reina con documento de identificación N° 0923328629, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: "DISEÑO E IMPLEMENTACIÓN DE UN ROBOT CUADRÚPEDO PARA COMPETENCIAS CON CONTROL AUTÓNOMO O MANUAL MEDIANTE UNA APLICACIÓN MÓVIL", realizado por Israel Marcelo Moya Herrera con documento de identificación N° 0940347206 y Arlena Omayra Zambrano Cuadro con documento de identificación N° 0942064916, obteniendo como resultado final el trabajo de titulación bajo la opción de Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Guayaquil, 31 de enero del año 2025.

Atentamente,

Ing. Rafael Franco Reina, MSc
0923328629

DEDICATORIA

Quiero dedicar esta tesis con todo mi cariño y gratitud a mis padres y mis abuelos, en especial a mi querido abuelito Jorge. A mis padres, por ser mi pilar inquebrantable, por su apoyo incondicional, sus valiosos consejos y por inspirarme a seguir adelante incluso en los momentos más difíciles. Su amor y sabiduría han sido la brújula que ha guiado mi camino.

A mi abuelito Jorge, quien, aunque ya no esté físicamente, sigue presente en mi corazón y en cada uno de mis logros. Su ejemplo de esfuerzo, perseverancia y amor por el aprendizaje me ha motivado a superar mis propios límites y a seguir adelante con determinación.

También quiero agradecer a los amigos que, aunque pocos, han sido invaluables en esta etapa de mi vida. Su compañía, apoyo y aliento han hecho que este camino sea más llevadero y significativo.

A todos ustedes, gracias por ser parte de mi historia y por ayudarme a llegar hasta aquí.

.

Israel Moya Herrera

DEDICATORIA

A Dios, fuente de mi vida y fortaleza en cada desafío, dedico este logro con humildad y gratitud. A Jesús, mi Salvador, quien con su amor y misericordia ha guiado mis pasos, y al Espíritu Santo, cuya luz y sabiduría han sido mi inspiración en este camino. Sin su gracia y dirección, este esfuerzo no habría sido posible.

A mi familia, por ser mi refugio y apoyo incondicional, por cada palabra de aliento, por cada sacrificio y por creer en mí aun en los momentos de duda. Su amor ha sido la fuerza que me impulsó a seguir adelante.

A mis amigos, por estar a mi lado en este proceso, por su compañía, su ánimo en los días difíciles y por recordarme que cada esfuerzo tiene su recompensa.

Este trabajo es testimonio de fe, perseverancia y gratitud. Lo dedico a todos aquellos que han sido parte de este camino, confiando en que este logro es solo el inicio de nuevos retos y oportunidades.

Arlena Zambrano Cuadro

AGRADECIMIENTO

Quiero expresar mi más sincero agradecimiento a la Universidad Politécnica Salesiana y al Club de Robótica, cuyo apoyo fue fundamental para la fabricación de las piezas del robot mediante impresión 3D. Su colaboración permitió que este proyecto tomara forma y superara los desafíos técnicos del diseño estructural.

También agradezco a mi tutor, cuyo conocimiento y guía fueron clave para la correcta ejecución de este trabajo, así como a mis compañeros y amigos, por su apoyo, ideas y esfuerzo compartido durante este proceso. Finalmente, extiendo mi gratitud a mi familia, quienes me han brindado su apoyo incondicional, motivándome a seguir adelante en cada etapa de mi formación profesional.

Israel Moya Herrera

AGRADECIMIENTO

Con el corazón lleno de gratitud, elevo mi primer agradecimiento a Dios, quien ha sido mi guía, fortaleza y sustento en cada paso. Su amor infinito me ha dado la valentía para seguir adelante y la paciencia para enfrentar los desafíos. A Jesús, por su amor redentor, y al Espíritu Santo, por iluminar mi mente y corazón en este proceso.

A mi familia, mi mayor pilar, gracias por su amor incondicional, por su apoyo constante y por ser mi refugio en los momentos de incertidumbre. Sus palabras de aliento y su confianza en mí han sido el motor que me impulsó a seguir adelante.

A mis docentes y mentores, quienes con su entrega y dedicación han dejado una huella imborrable en mi formación. Gracias por su paciencia, por compartir su conocimiento y por motivarme a superar mis propios límites.

A mis amigos, compañeros de alegrías y desafíos, quienes han sido un soporte invaluable en este camino. Gracias por su compañía, su ánimo y por hacer de este proceso una experiencia más llevadera.

Y a todas aquellas personas que, de una u otra manera, contribuyeron con su apoyo, tiempo y conocimientos a la realización de este trabajo, mi más sincero agradecimiento. Que Dios les bendiga abundantemente, así como Él ha bendecido mi vida con su presencia.

Arlena Zambrano Cuadro

Resumen

Este trabajo de titulación presenta el diseño e implementación de un robot cuadrúpedo orientado a competencias, diseñado para operar con modos de control autónomo y manual mediante una aplicación móvil. La propuesta aborda la carencia de robots cuadrúpedos en el club de robótica de la Universidad Politécnica Salesiana, lo que limita su participación en eventos nacionales e internacionales. A través del diseño y desarrollo de este proyecto, se busca explorar y fomentar el conocimiento en robótica móvil, inteligencia artificial y control dinámico.

El desarrollo del robot se llevó a cabo en varias etapas: diseño estructural utilizando software CAD 3D, fabricación de piezas mediante impresión 3D, integración de componentes electrónicos como el microcontrolador ESP32 y el módulo de visión artificial Grove AI Vision V2, programación de algoritmos para navegación autónoma y control remoto, y desarrollo de una aplicación móvil que actúa como interfaz de usuario. El proyecto utilizó metodologías experimentales e investigativas, llevando a cabo pruebas para garantizar la estabilidad del sistema, la maniobrabilidad en terrenos irregulares y la precisión de los algoritmos de visión y control.

Los resultados obtenidos demuestran que el diseño propuesto cumple con los requerimientos para competencias, destacando su capacidad de adaptación a diferentes entornos y su eficiencia en la ejecución de tareas específicas. Este trabajo no solo contribuye al avance en el campo de la robótica competitiva, sino que también inspira nuevas investigaciones en aplicaciones industriales, educativas y de rescate, consolidando la robótica cuadrúpeda como una solución versátil y prometedora.

*Palabras Clave**:** Visión Artificial, Robótica, Robot insecto, ESP32, Algoritmos de Control, Robótica Móvil, Competencias de Robótica, Robots Autónomos, Control Dinámico.*

Abstract

This thesis presents the design and implementation of a quadruped robot tailored for competitions, capable of operating in both autonomous and manual control modes via a mobile application. The project addresses the lack of quadruped robots in the robotics club of Universidad Politécnica Salesiana, limiting their participation in national and international events. By developing this robot, the aim is to explore and enhance knowledge in mobile robotics, artificial intelligence, and dynamic control systems.

The robot was developed through multiple stages, including structural design using CAD 3D software, fabrication of components via 3D printing, integration of electronic components such as the ESP32 microcontroller and Grove AI Vision V2 module, programming of algorithms for autonomous navigation and remote control, and the development of a mobile application serving as the user interface. The project employed experimental and investigative methodologies, conducting rigorous tests to ensure system stability, maneuverability over uneven terrains, and precision in vision and control algorithms.

The results demonstrate that the proposed design meets the requirements for competitive robotics, excelling in adaptability to various environments and efficient task execution. Beyond its utility in competitions, the robot opens avenues for further research in industrial, educational, and rescue applications, establishing quadruped robotics as a versatile and promising solution.

Keywords: Artificial Vision, Robotics, Insect Robot, ESP32, Control Algorithms, Mobile Robotics, Robotics Competitions, Autonomous Robots, Dynamic Control.

Índice de Contenido

Índice de Figuras

Índice de Tablas

I Introducción

En un mundo donde la tecnología avanza a pasos agigantados, la robótica emerge como un campo clave que combina creatividad, ciencia y funcionalidad. En particular, los robots cuadrúpedos han captado la atención por su similitud con los movimientos de los animales y su capacidad para adaptarse a terrenos complejos. Estas características los convierten en herramientas ideales para aplicaciones que van desde el entretenimiento hasta misiones de rescate en entornos desafiantes.

Este proyecto se centra en el diseño y desarrollo de un robot cuadrúpedo orientado a competencias, una plataforma que busca demostrar la versatilidad de este tipo de dispositivos. El sistema propuesto integra un control que puede ser autónomo o manual, operado a través de una aplicación móvil. Esta dualidad en el control no solo maximiza su utilidad en diversos escenarios, sino que también refleja el compromiso con la innovación y la interacción humano-máquina.

Al explorar el potencial de un robot cuadrúpedo, esta investigación contribuye al avance del conocimiento en robótica móvil e inteligencia artificial. Además, fomenta un enfoque práctico que responde a los desafíos reales de navegación, estabilidad y control dinámico, sentando las bases para futuras aplicaciones en competencias, educación y sectores industriales.

Desde el ámbito académico, este trabajo ayuda al desarrollo de nuevas competencias en investigación aplicada, fomentando el aprendizaje de tecnologías emergentes como la robótica y su implementación en soluciones tangibles. Al proporcionar un enfoque práctico y experimental, el proyecto no solo enriquece el conocimiento técnico, sino que también inspira a estudiantes e investigadores a innovar en un campo que conecta creatividad y tecnología con un impacto real.

II Problema

En la actualidad el ámbito de la robótica ha ido en crecimiento con el paso de los años, en la cual ya se ha visto su desempeño desde las industrias y en la medicina hasta las misiones de rescate, sin embargo, ya están apareciendo robots móviles autónomos que están tomando nuevos roles más allá de los usos industrias (Robotnik, 2024) , como por ejemplo el nuevo robot humanoide Optimus creado por Tesla, en la que busca de manera autónoma realizar actividades cotidianas como tareas domésticas (Teleamazonas, 2024).

No obstante, una de las configuraciones prometedoras, pero menos exploradas es la de los robots cuadrúpedos, ya que estos robots, basados en la estructura terrestre, imitan el movimiento de animales de cuatro patas como perros y arañas, en la que ofrecen características únicas que permiten una maniobrabilidad y estabilidad en terrenos irregulares o de difícil acceso, además, estos tipos de robots, presentan ventajas significativas para tareas complejas que requieren un movimiento versátil y el equilibrio como es el caso de los robots de rescate en situaciones de desastre (Instrumentación, 2022).

La robótica de búsqueda y rescate surge como respuesta a la necesidad de apoyar a las brigadas en tareas extremadamente riesgosas, especialmente en las primeras fases de exploración tras un desastre, como un terremoto el cual deja pérdidas humanas y materiales, por tal motivo los equipos de primeros auxilios son quienes intervienen rápidamente para localizar y rescatar a personas inconscientes o atrapadas, pero esta labor los expone a peligros considerables como lo ocurrido el 16 de abril de 2016 en el Terremoto de Ecuador en la provincia de Manabí en Pedernales la cual marco la peor tragedia para los ecuatorianos el cual cobro muchas vidas y varias zonas destruidas (London, 2022).

En el Ecuador con el paso del tiempo se ha visto un incremento en el avance de la tecnologia, la robotica y la programación esto queda en evidencia con el aumento de ferias y competencias de robótica uno de ellos es el Torneo Nacional de Robótica StarBot Planet 1.0 donde se reunen competidores de diferentes partes del país para participar en diversas categorias como mini sumo, soccer, humanoides, insecto, seguidor de linea, entre otras, siendo asi aun queda un gran camino por avanzar en los diferentes campos de estudio de los robots tipo cuadrupedo cuales su caracteristica mas importante es el de aprendizaje automático e interacción humano -robot que motivan a la investigación para su uso en diferentes campos competitivos y posteriormente su uso en la vida cotidiana (UPS, 2024).

Actualmente el club de Robótica de la Universidad Politécnica Salesiana no cuenta con un robot cuadrúpedo para las competencias universitarias en esa categoría, la mayoría de los robots participantes están diseñados como modelos bípedos o sobre ruedas, dejando en segundo plano configuraciones menos convencionales como los robots cuadrúpedos, por lo que esta carencia limita la participación de los estudiantes de la Universidad en torneos nacionales e internacionales en la categoría de robots cuadrúpedos, adicionalmente se evidencia en el repositorio de la Universidad que existen 4 trabajos de titulación relacionadas a este tipo de robot, uno de ellos enfocándose en la inspección inteligente y enseñanza en mecatrónica, más no como un área de desarrollo e investigación en el área de competencias robóticas de categoría cuadrúpedo (Salesiana, 2024).

2.1 Justificación

La Universidad Politécnica Salesiana es una institución innovadora, que está a la vanguardia en los ámbitos tecnológicos y de robótica por dicho motivo no es ilógico que los estudiantes tengan en cuenta el implementar y diseñar un robot cuadrúpedo útil tanto para el ámbito cotidiano, laboral y académico, se ve necesaria la implementación de este proyecto dentro del campo de la robótica de la Universidad ya que así mejorará el desarrollo cognitivo de los estudiantes, aplicando conocimiento previos adquiridos dentro de la carrera si no también el continuo avance de la tecnología en el mundo.

La importancia de este proyecto radica en el diseño e implementación de un robot cuadrúpedo, con modos de control manual y autónomo, representa una oportunidad para abordar áreas clave de interés y necesidad en el campo de la robótica, de modo que este robot no solo ampliará el espectro de tipos de robots en competencias universitarias, sino que también ofrecerá una plataforma innovadora para la creatividad y el desarrollo tecnológico, en vista que al centrarse en un diseño cuadrúpedo inspirado en arañas, se investigarán aspectos esenciales como la estabilidad en movimiento, el centro de masa, la eficiencia en la maniobrabilidad y la capacidad de adaptación a diferentes tipos de terreno, todos ellos factores cruciales en la robótica moderna.

La implementación de una aplicación de control específica para este robot añade una dimensión significativa al proyecto, esta aplicación permitirá tanto el control manual preciso del robot en entornos competitivos como la operación autónoma en escenarios que requieran decisiones rápidas y adaptativas, además, la capacidad del robot para operar en modo autónomo abrirá la posibilidad de implementar y probar algoritmos de inteligencia artificial y sensores avanzados, lo cual es fundamental para el desarrollo de robots con mayor autonomía y adaptabilidad en entornos dinámicos.

III Objetivos

3.1 Objetivo general

Diseñar e implementar un robot cuadrúpedo para competencias con control autónomo o manual mediante una aplicación móvil.

3.2 Objetivos específicos

- Diseñar un modelo mecánico mediante software 3D incluyendo el circuito electrónico necesario para su funcionamiento.
- Programar el algoritmo de visión autónoma o radiocontrolada del robot cuadrúpedo.
- Configurar el microcontrolador y la aplicación móvil para el funcionamiento del robot cuadrúpedo.

IV Fundamento Teórico

4.1 Robótica Móvil

4.1.1 Robótica Cuadrúpeda y principios de locomoción

La robótica se describe como la disciplina científica y técnica que abarca el diseño, fabricación y uso de robots, entre las clasificaciones de los robots se encuentra los cuadrúpedos como se muestra en la figura 1, que es un tipo de robot que está diseñado con cuatro patas, imitando la estructura y locomoción de animales cuadrúpedos como perros o caballos, donde dicha configuración permite al robot moverse de manera estable y eficiente en terrenos irregulares y difíciles, proporcionando una mayor adaptabilidad y capacidad de maniobra en comparación con robots con otras configuraciones de movilidad, como los bípedos o los de ruedas (Definicion.de, 2016).

Figura 1.

Robot Cuadrúpedo Spot Mini **(Castañeda, 2018)**.

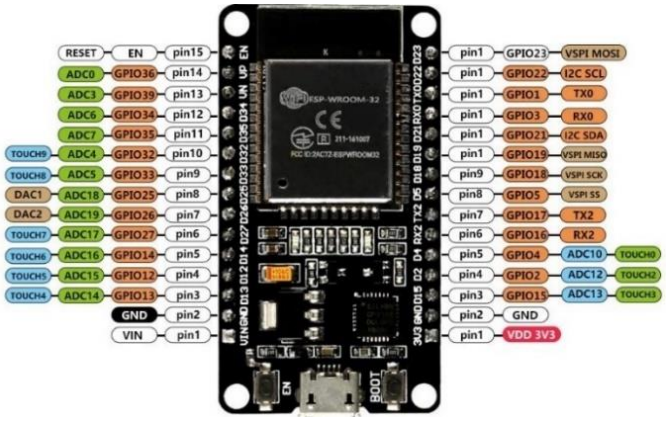
4.2 Hardware

4.2.1 Microcontroladores

El microcontrolador es un circuito integrado esencial en aplicaciones embebidas, funcionando como el componente principal de dichos sistemas, y es comparable a una pequeña computadora como el ejemplo de la figura 2, ya que incluye sistemas para gestionar elementos de entrada/salida, un procesador, y memoria (tanto flash como RAM) para almacenar programas y variables, por lo tanto, su función principal es automatizar procesos y procesar información de manera eficiente, además, los microcontroladores se utilizan en una amplia variedad de dispositivos y productos que requieren la ejecución de procesos automáticos basados en las condiciones de distintas entradas (E-Marmolejo, 2021).

Figura 2.

Ejemplo microcontrolador ESP32 con capacidades de Wi-Fi y Bluetooth integrado **(Rosario, 2018)**.



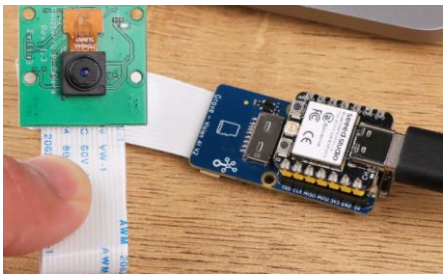
4.2.2 Modulo Cámara Grove AI Visión V2

Un módulo de cámara Grove AI visión V2 es un dispositivo compacto que proporciona capacidades avanzadas para la captura y procesamiento de imágenes en conjunto con su cámara

OV5647 como se muestra en la figura 3, facilitando la implementación de aplicaciones de visión artificial, siendo compatible con la plataforma SenseCraft que permite a los usuarios entrenar y desplegar modelos de aprendizaje automático sin necesidad de experiencia en codificación (CITRIC, 2023).

Figura 3.

Módulo Grove AI Vision junto a su cámara OV5647 **(STUDIO S. , 2024)**.



4.2.3 Servomotores

Un servomotor es un dispositivo que permite controlar con alta precisión la posición y el movimiento de su eje, y a diferencia de un motor eléctrico común como se muestra en la figura 4, un servomotor puede ajustarse en ángulo, posición y velocidad de manera específica y exacta en cada momento, proporcionando un control detallado y preciso (Egasen, 2021).

Figura 4.

Ejemplo servomotor marca Tower Pro **(Pardo, 2018)**.

4.2.4 Baterías de Lipo

Las baterías LiPo, o polímero de litio como se puede observar en la figura 5, son baterías recargables conocidas por su alta capacidad de almacenamiento, ligereza y capacidad de descarga rápida, lo que las hace muy populares en el ámbito de los drones, además, estas baterías se componen de múltiples celdas; por ejemplo, una batería 4S está formada por cuatro celdas conectadas en serie, mientras que una batería 6S consta de seis celdas en serie (Eduardo, 2019).

Figura 5.

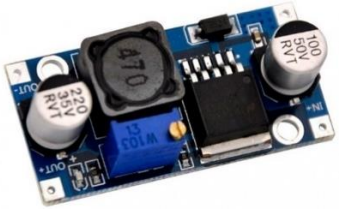
Batería de LiPo Tattu con sus características **(AMAZON, 2017)**.

4.2.5 Regulador de Voltaje

El regulador de voltaje, o estabilizador de voltaje como se muestra en la figura 6, es un dispositivo diseñado para proteger los equipos eléctricos al mantener un voltaje constante y seguro, ya que su función principal es evitar que las variaciones de tensión perjudiciales lleguen a los dispositivos, asegurando que reciban un voltaje estable dentro de un rango operativo seguro, además, el regulador convierte un voltaje inestable en uno uniforme y limpio (Avtek, 2022).

Figura 6.

Módulo regulador de voltaje step down LM2596 **(Todomicro, 2022)**.



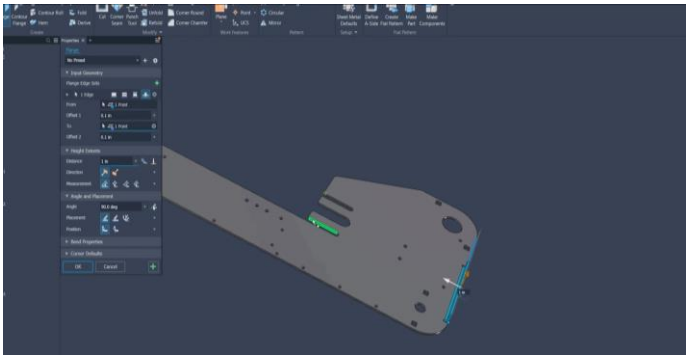
4.3 Software de Desarrollo

4.3.1 Autodesk Inventor

Autodesk Inventor es un software de diseño CAD mecánico 3D ampliamente utilizado a nivel mundial para desarrollar prototipos digitales en el ámbito del diseño, visualización y simulación de productos mecánicos como se muestra en la figura 7, además, emplea técnicas de modelado sólido paramétrico, lo que lo convierte en una herramienta ideal para diseñar mecanismos de ingeniería, esto facilita modificaciones rápidas y la adaptación de las características del diseño mediante un proceso de trabajo muy intuitivo (Marta, 2023).

Figura 7.

Entorno de desarrollo de Autodesk Inventor **(Autodesk, 2024)**.

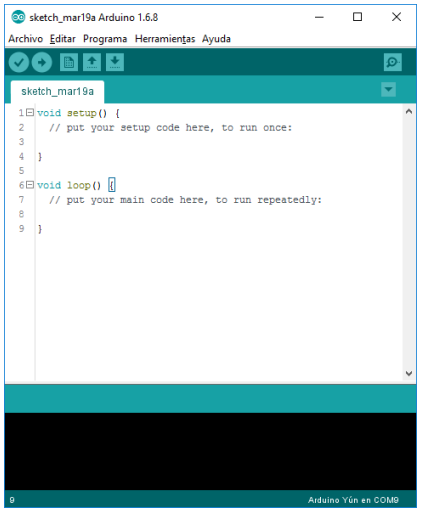


4.3.2 Arduino

El IDE de Arduino es una suite de herramientas de software diseñada para ayudar a los programadores a crear y cargar el código necesario para operar un Arduino según sus especificaciones según lo indica la figura 8, además, este entorno de desarrollo integrado facilita la escritura, depuración, edición y carga de programas (denominados “sketches” en el ecosistema Arduino), lo que contribuye significativamente a la popularidad de Arduino debido a su facilidad de uso y accesibilidad (Arduino, 2019).

Figura 8.

Entorno de desarrollo del IDLE de Arduino **(aprendiendoarduino, 2016)**.

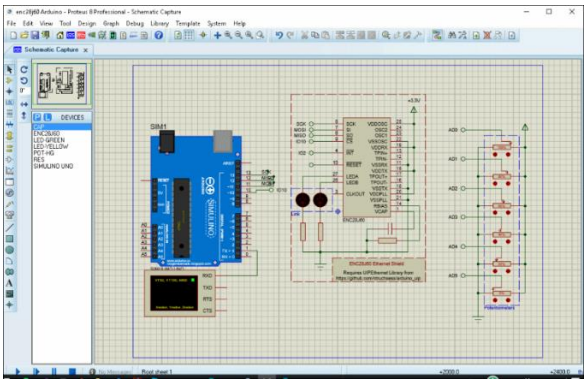


4.3.3 Proteus 7

PROTEUS es un software integral como se muestra en la figura 9 que facilita el diseño y la simulación de circuitos electrónicos de manera práctica y accesible, además, ofrece una amplia gama de funciones para el trabajo con circuitos electrónicos, como la generación automática de pistas de cobre, y la herramienta ISIS dentro del programa permite crear circuitos reales y verificar su funcionamiento en una placa de circuito impreso (PCB) (Enerxia, 2014).

Figura 9.

Entorno de desarrollo en Proteus 7 **(Anibalismo, 2016)**.

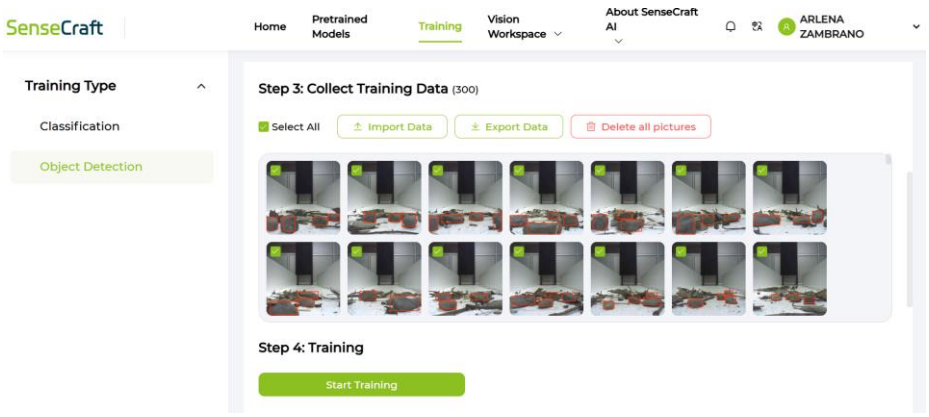


4.3.4 SenseCraft

SenseCraft es una plataforma integral para entrenar visión artificial que facilita a desarrolladores y creadores la construcción y despliegue de proyectos de inteligencia artificial de manera sencilla como se muestra en la figura 10, ofrece una gama de herramientas y funcionalidades como la recopilación de datos, entrenamiento para clasificación y detección de objetos, utilizando un interfaz web intuitiva que elimina la necesidad de configuraciones complejas (STUDIO S. , 2024)

Figura 10.

Plataforma SenseCraft



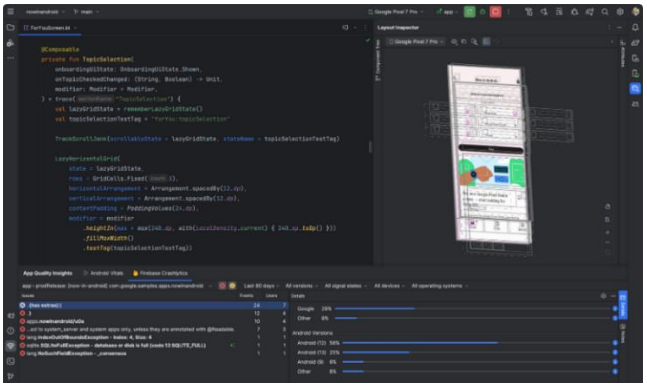
4.3.5 Android Studio

Como se indica en la figura 11, Android Studio es el entorno de desarrollo integrado (IDE) oficial para la creación de aplicaciones Android, desarrollado y distribuido por Google (Developers E. d., 2024). El control mediante aplicación móvil ha revolucionado la interacción entre los usuarios y los sistemas robóticos, permitiendo un manejo más intuitivo, versátil y remoto de los robots, aquellos robots móviles que se controlan de manera remota son sistemas sofisticados que constan de tres componentes principales: la red de comunicación como Internet, la plataforma

robótica móvil y el software que se emplea en todos los niveles para asegurar el funcionamiento integrado del sistema (MAPIR, 2014).

Figura 11.

Entorno de desarrollo de Android Studio para crear una aplicación **(Developers, s.f.)**.



4.4 Competencias: Carrera de Insectos con Obstáculos

Esta categoría trata de un robot con características similares a la de un insecto, donde su única locomoción solo debe de ser articulada, y en la que tendrá que realizar un recorrido en línea recta hasta llegar a la meta final, en algunos casos el camino puede tener obstáculos como se indica en la figura 12, algunas características principales en el diseño de un robot para la categoría insecto (Copol, 2022):

- El robot insecto autónomo será capaz de avanzar por el área de competencia.
- El mecanismo de movimiento del robot debe ser diferente al de tracción por ruedas, orugas o algún tipo de desplazamiento por saltos.
- Las dimensiones máximas son 20 cm x 20 cm, sin restricción de altura.
- El robot puede tener mecanismos desplegables, pero al término de la competencia deberá regresar a las dimensiones indicadas anteriormente. Esta acción debe hacerla de manera autónoma al momento de apagar el robot.

Figura 12.

Ejemplo carrera de insectos RobochallengeMx **(Piedra, 2020)**.

V Marco Metodológico

El desarrollo de este proyecto empleará metodologías inductiva, experimental y sistemática, diseñadas para abordar las necesidades específicas del diseño, construcción y programación de un robot cuadrúpedo autónomo y controlado manualmente mediante una aplicación móvil. La metodología inductiva permitirá recopilar y analizar información relevante sobre la locomoción y control de robots cuadrúpedos, así como el entrenamiento de algoritmos de visión artificial para evitar obstáculos. La metodología experimental se enfocará en realizar pruebas continuas del sistema, evaluando su rendimiento en distintas configuraciones y condiciones de operación, como la estabilidad en movimiento y la precisión del algoritmo entrenado. Finalmente, la metodología sistemática organizará el desarrollo del proyecto de manera estructurada y eficiente, asegurando que cada etapa del diseño y la implementación se ejecute con base en objetivos específicos y claros.

5.1 Diseño de la Estructura del Robot

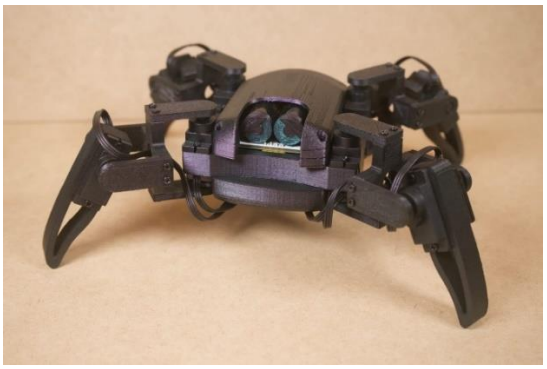
5.1.1 Referencias para el Diseño Estructural del Robot

El diseño estructural del robot cuadrúpedo se fundamenta en hallazgos provenientes de investigaciones previas, así como en el análisis de modelos existentes en el ámbito de la robótica móvil. En particular, se consideraron referencias inspiradas en la anatomía de las arañas, dadas sus ventajas en cuanto a un patrón de locomoción eficiente y la capacidad de mantener la estabilidad en superficies irregulares. Además, se revisaron diversos proyectos de robots cuadrúpedos para identificar características funcionales relevantes, como la distribución de peso, la articulación de las extremidades y la integración de los componentes electrónicos. Por último, se encontró en un repositorio el modelo base creado por “jason-workshop”, cuya propuesta de robot araña cuadrúpedo sirvió como punto de partida para el desarrollo del proyecto, tal como se ve en la figura

13. La incorporación de estas referencias resultó esencial para establecer un modelo que equilibre la estabilidad, movilidad y eficiencia en el uso de recursos, con el objetivo de lograr un desempeño óptimo durante las pruebas y competencias contempladas en el proyecto.

Figura 13.

Modelo de referencia robot cuadrúpedo **(jason-workshop, 2018)**

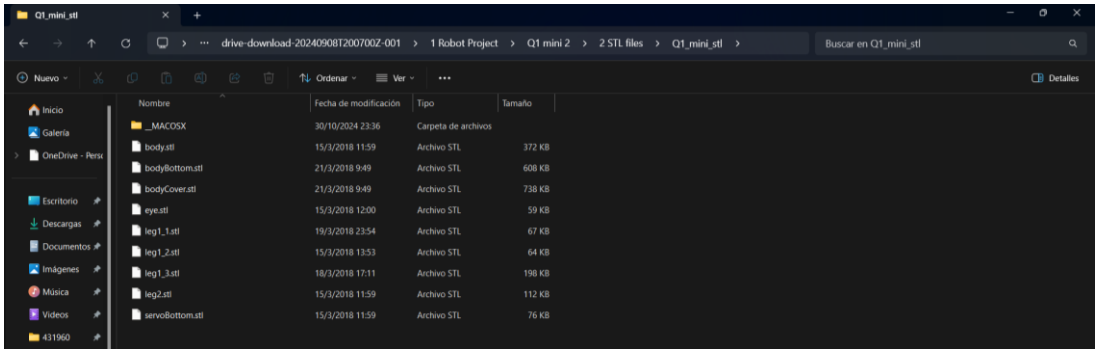


5.2 Diseño de las Piezas en Autodesk Inventor

Para el modelado de las diferentes piezas del robot, se utilizó el software Autodesk Inventor, con el propósito de adaptar el diseño base a las necesidades específicas del proyecto. En la figura 14 se muestran los archivos de piezas en 3D que sirvieron como punto de partida para la construcción del prototipo.

Figura 14.

Lista de todas las piezas del repositorio del modelo original.



Entre estos archivos, solo dos se mantuvieron sin modificaciones, debido a que cumplen de manera correcta con los requerimientos de estabilidad y funcionalidad en el diseño final. En la figura 15 se presenta la pieza denominada “servoBottom”, mientras que la figura 16 ilustra la pieza “leg2”. Ambos componentes mostraron un buen desempeño mecánico y se ajustaron a las dimensiones requeridas, por lo que no fue necesario realizar cambios. Por otro lado, el archivo “eyes” no se incluyó en la versión definitiva del prototipo, por no ser relevante para su operación. Esta decisión contribuyó a reducir tanto el peso como la complejidad general de la estructura.

Figura 15.

Pieza ServoButtom del modelo original

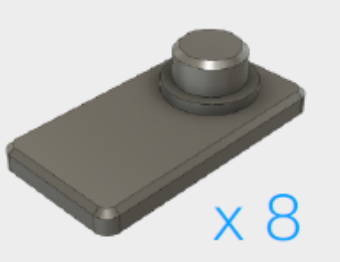
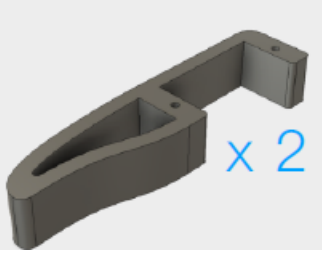


Figura 16.

Pieza pata de la araña del modelo original



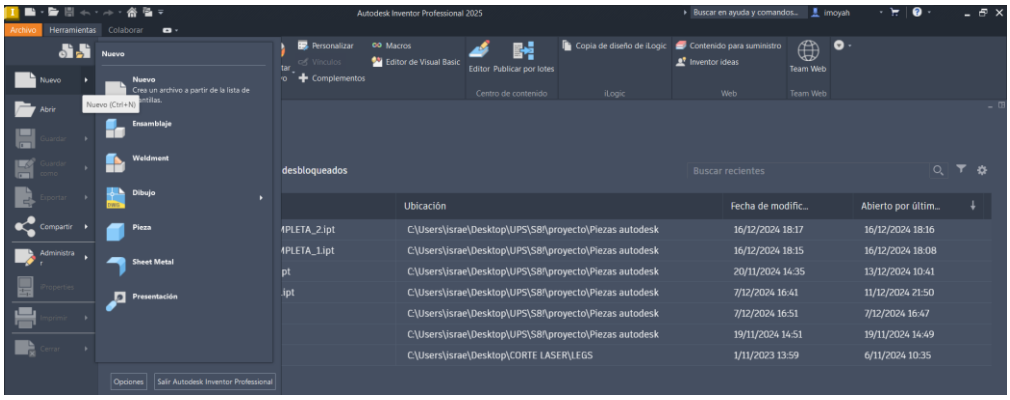
5.2.1 Creación del Archivo

El primer paso para diseñar cada pieza en Autodesk Inventor consiste en la creación de un archivo nuevo a partir de una hoja en blanco. Para ello, se accede al botón “Archivo” y,

posteriormente, se selecciona la pestaña “Nuevo”, tal como se observa en la figura 17. Este procedimiento se repite cada vez que se desea iniciar el diseño de una nueva pieza.

Figura 17.

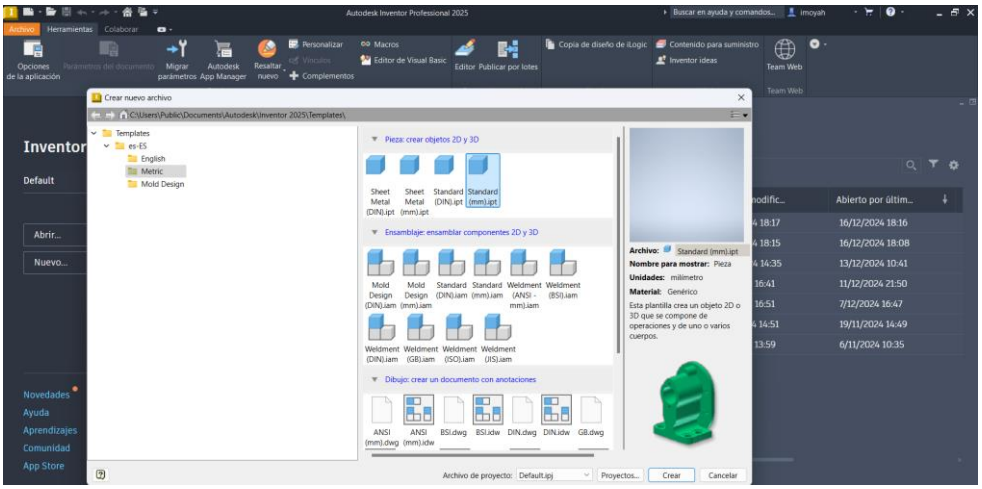
Página inicial del programa Autodesk Inventor para crear un archivo



Una vez dentro del menú “Nuevo”, se elige la plantilla llamada Standard (mm), con el fin de trabajar en el sistema métrico. Luego, se hace clic en Crear, tal como se muestra en la figura 18.

Figura 18.

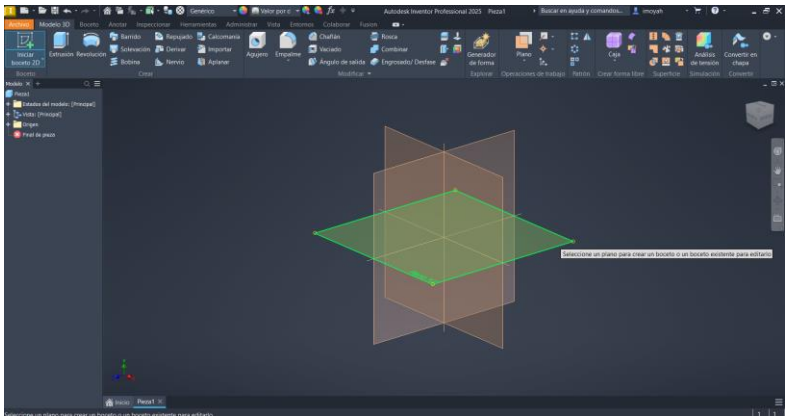
Menú para seleccionar el tipo de archivo para empezar a diseñar las piezas.



Inmediatamente después, se ingresa al espacio de trabajo y se selecciona la opción Crear boceto 2D, donde se escoge el plano apropiado para iniciar el trazo de la pieza; en este caso, se utiliza el plano ZX, según se ilustra en la figura 19.

Figura 19.

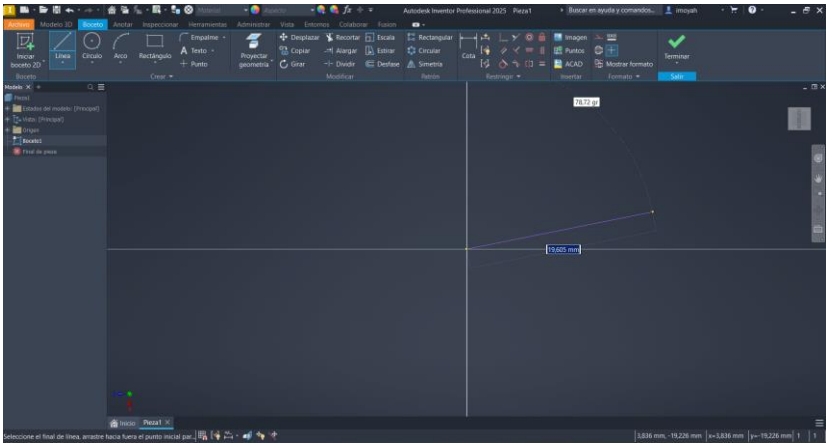
Espacio de trabajo donde se puede seleccionar en que parte del plano diseñar las piezas.



Con el boceto 2D iniciado, se procede a dibujar las geometrías deseadas para, finalmente, aplicar la herramienta de extrusión, lo que permite convertir las figuras en modelos tridimensionales como se puede observar en la figura 20. Este proceso se repite para cada pieza que forme parte de la estructura final del robot.

Figura 20.

Espacio de trabajo en el plano seleccionado para comenzar el diseño.



5.2.1 Diseño Pieza Body

Para el desarrollo de la pieza Body, se inició del modelo original y se realizaron ajustes con el fin de adaptarlo a los requerimientos del proyecto. El cambio principal consistió en agrandar la estructura, ya que el diseño original se encontraba optimizado para un shield de servos especial para ESP32 que incluía una fuente de alimentación externa con medidas exactas, lo que eliminaba la necesidad de un regulador adicional. En este nuevo proyecto, se decidió ampliar la pieza de manera que pudiese alojar tanto la tarjeta shield del ESP32 como los componentes asociados. Tal como se observa en la figura 21, las dimensiones iniciales de la pieza original eran 7.6 cm de largo, 7.3 cm de ancho y un grosor de 1 cm.

Figura 21.

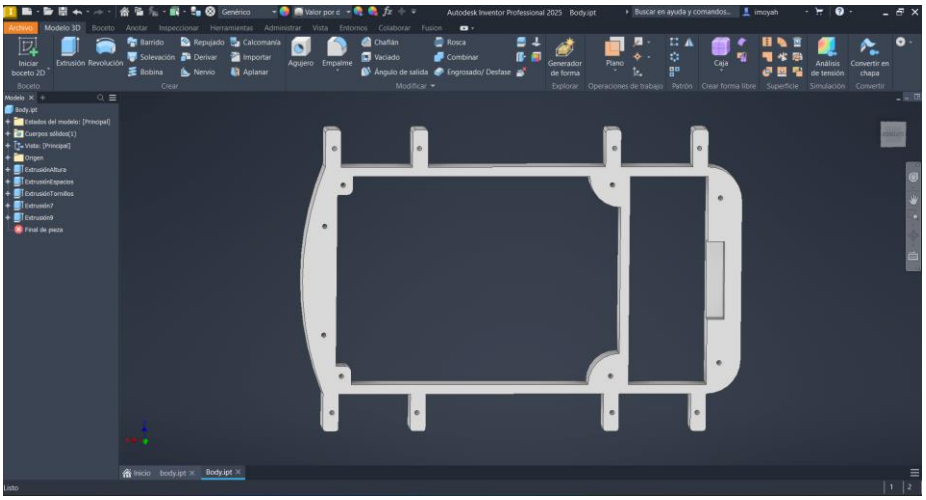
Pieza Body del modelo original.

Para el diseño propio, se incrementaron dichas medidas a 14.1 cm de largo, 9.7 cm de ancho y se mantuvo el grosor de 1 cm, tal como se muestra en la figura 22. Se dejó un espacio adicional en la parte frontal para facilitar la conexión del cable de la cámara y otro espacio para alojar la batería, cuyas dimensiones son 6.7 cm de largo por 2.5 cm de ancho. Finalmente, también se habilitó un área para la tarjeta electrónica principal del robot. A los costados de la pieza, se

diseñaron espacios destinados a la instalación de los cuatro servomotores que se ubicaron próximos al cuerpo, garantizando una integración sólida y estable de todos los componentes.

Figura 22.

Diseño propio pieza Body

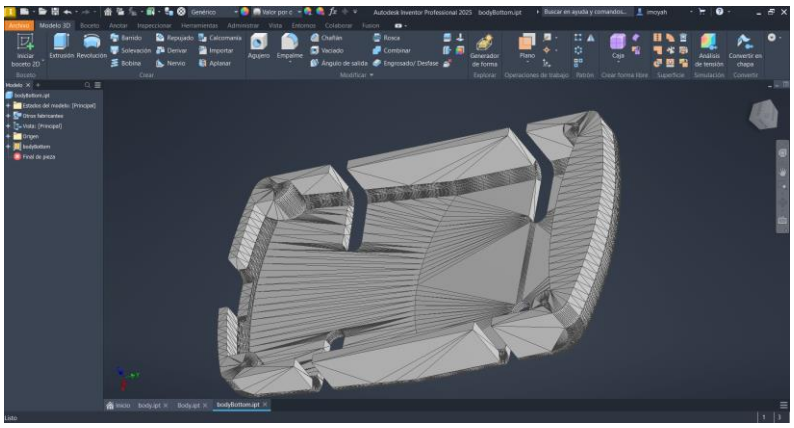


5.2.2 Diseño de la Cubierta Inferior

En el diseño de la pieza bodyBottom se llevó a cabo una modificación completa respecto al modelo original, debido a que la versión inicial presentaba un área elevada que abarcaba de la parte frontal a la trasera. Dicho elemento ocupaba un espacio que resultaba necesario para el proyecto, por lo que se optó por removerlo y, de esta manera, aprovechar mejor el interior del robot. Tal como se muestra en la figura 23, el modelo original tenía unas dimensiones de 7.03 cm de largo, 4.9 cm de ancho y un grosor de 0.2 cm en la mayor parte de la estructura, además de una altura aproximada de 1 cm.

Figura 23.

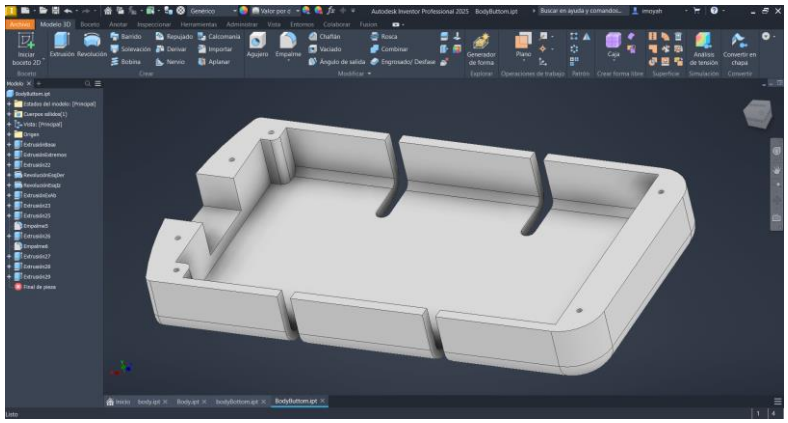
Pieza bodyButtom del modelo original



Para el nuevo diseño, se incrementaron las medidas a 14.1 cm de largo, 7.3 cm de ancho, un grosor de 0.3 cm y una altura total de 1.8 cm, tal como se observa en la figura 24. A pesar de las modificaciones, se procuró mantener una estructura similar para garantizar la compatibilidad con el cuerpo principal del robot. Los cambios realizados permiten, disponer de un espacio interior suficiente para integrar los componentes electrónicos y optimizar la distribución de los servomotores, sensores y baterías en el interior de la estructura.

Figura 24.

Diseño inferior bodyButtom del proyecto

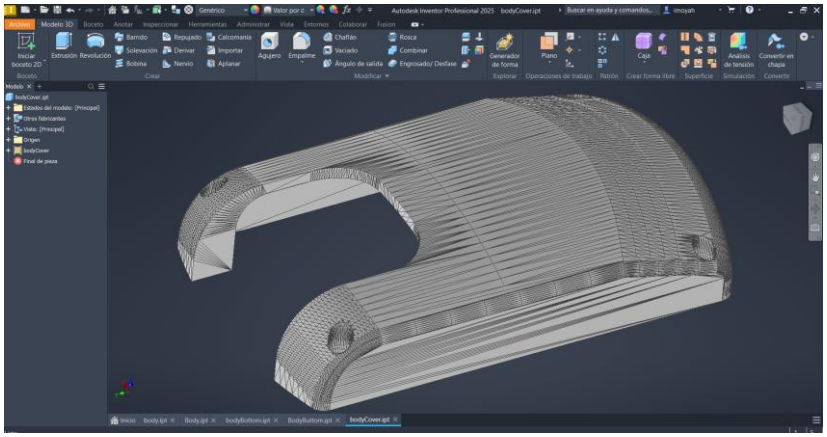


5.2.3 Cubierta Superior

Para el diseño de la pieza bodyTop, se realizó una modificación completa respecto al modelo original, dado que la versión inicial presentaba un arco que se extendía desde la parte frontal hasta la posterior. Este elemento limitaba el espacio interior necesario para alojar los componentes electrónicos. El diseño original medía 7.01 cm de largo, 4.9 cm de ancho y tenía un grosor de aproximadamente 0.2 cm en la mayor parte de su estructura, además de una altura cercana a 1.16 cm, tal como se observa en la figura 25.

Figura 25.

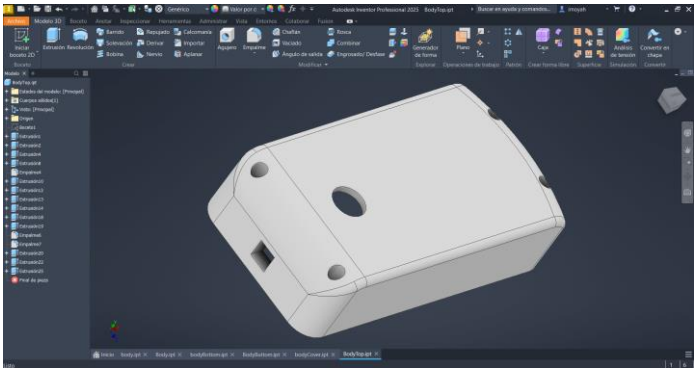
Pieza bodyTop del modelo original



En el nuevo diseño como se observa en la figura 26, las dimensiones se incrementaron a 14.1 cm de largo, 7.3 cm de ancho y 0.3 cm de grosor, con una altura total de 3 cm. Producto de estos cambios, fue posible optimizar el espacio disponible para la distribución de los componentes. Por ejemplo, en la parte frontal de la cubierta se incorporó una abertura de forma cuadrada destinada a la cámara del robot, mientras que en la sección superior se añadió una abertura circular para ubicar el switch de encendido y apagado.

Figura 26.

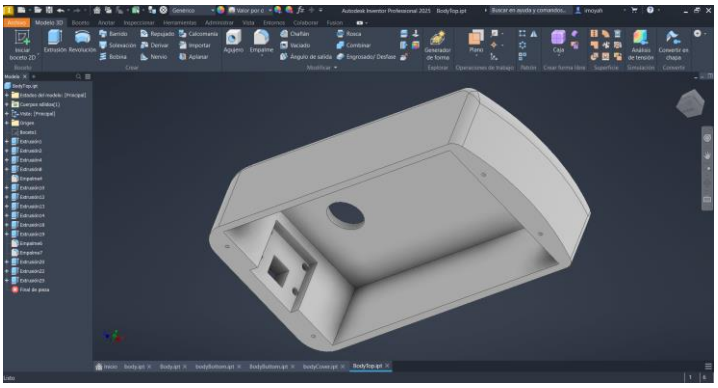
Diseño propio pieza bodyTop



Además, en la zona interior, detrás de la abertura destinada a la cámara, se habilitó un compartimento específico para acomodar el módulo, tal como se ilustra en la figura 27. Estas modificaciones permiten no solo una mejor organización de los componentes electrónicos, sino también un fácil acceso al sistema de visión y al switch de alimentación, garantizando así una integración más eficiente y funcional de la cubierta superior con el resto de la estructura del robot.

Figura 27.

Parte interna del diseño propio pieza bodyTop



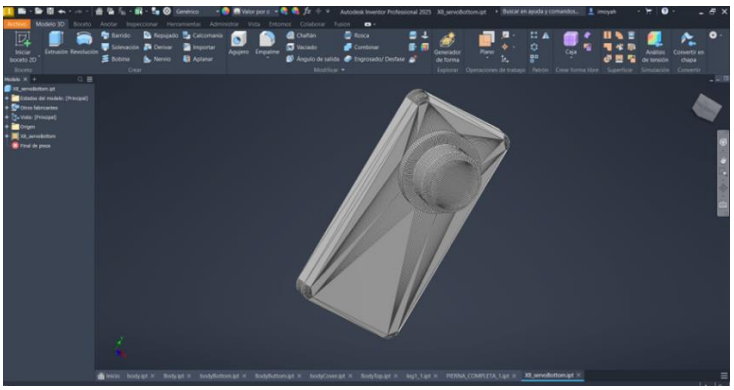
5.2.4 Diseño Piezas Debajo del Servo

En el caso de la pieza que se ubica debajo del servomotor, no fue necesario realizar modificaciones, puesto que sus dimensiones se ajustan adecuadamente a las medidas del servo, tal

como se observa en la figura 28, esta pieza cumple la función de sostener el servomotor y mantenerlo alineado con la estructura de las patas, lo que garantiza un movimiento de giro estable y preciso. Asimismo, al situarse debajo del servo, contribuye a optimizar la posición y el balance del robot durante su desplazamiento.

Figura 28.

Pieza servoButtom del modelo original.



5.2.5 Diseño del Eje de Sujeción de las Patas

Para el diseño del eje de sujeción de las patas del robot, se optó por realizar una modificación completa en comparación con el modelo original, el cual contemplaba tres archivos independientes que se ensamblaban mediante varios tornillos. En las figuras 29 y 30 se muestran estos los ejes individuales.

Figura 29.

Pieza leg1 del modelo original

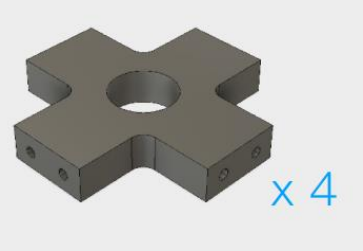
Figura 30.

Pieza leg2 del modelo original

En la figura 31 se muestra el eje individual donde se unen las patas mediante tornillos.

Figura 31.

Pieza leg3 del modelo original



En la figura 32 puede apreciarse la unión final de dichas piezas en el diseño base.

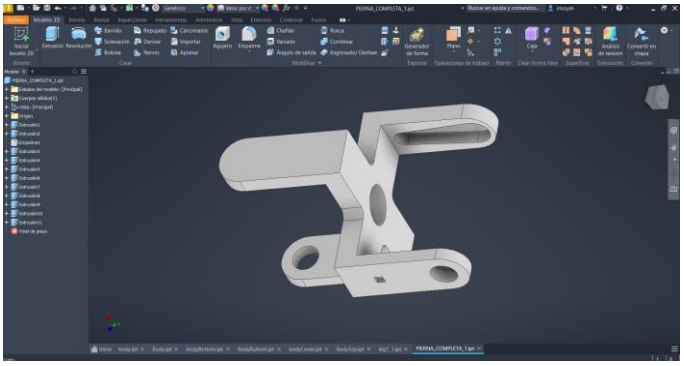
Figura 32.

Modelo de ensamblado de las piezas leg1, leg2, leg3 en el modelo original

El objetivo principal de la modificación fue simplificar la estructura al unificar los tres archivos en una sola pieza como se muestra en la figura 33. Esto redujo la complejidad del ensamblaje, mejoró la solidez del eje de la pata y amplió el espacio para la palanca del servomotor, ya que el tamaño original era insuficiente. Además, se añadió un orificio en la zona de acoplamiento para permitir el ingreso de la pieza "servoBottom", garantizando un giro adecuado del mecanismo y una correcta alineación del eje.

Figura 33.

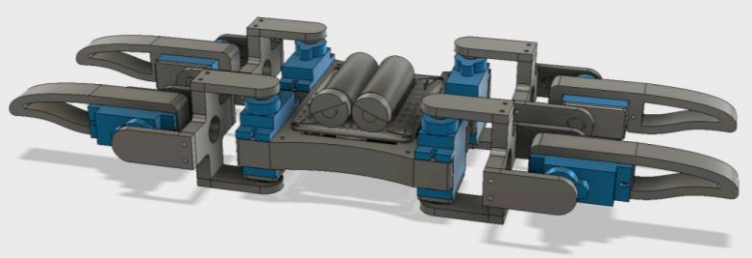
Diseño del eje donde se unen las piezas leg1, leg2, leg3 del modelo original.



En la figura 34 se muestra cómo se conectaban las piezas en el modelo original, lo que sirvió de referencia inicial para desarrollar esta nueva versión mejorada. Con estas modificaciones, se obtuvo una pata más robusta, funcional y fácil de ensamblar, contribuyendo así a la eficiencia general del robot.

Figura 34.

Modelo original del robot

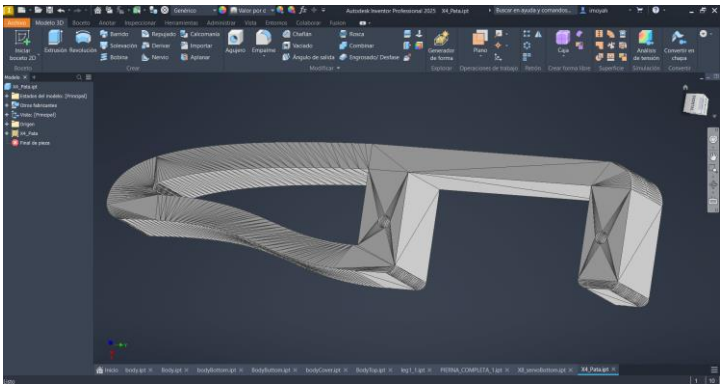


5.2.6 Diseño de las Patas

En el caso de la pieza destinada a las patas del robot cuadrúpedo, no fue necesario realizar modificaciones, ya que sus dimensiones se ajustan adecuadamente al servomotor utilizado. Como se observa en la figura 35, esta pieza permite el acoplamiento directo con el servo, posibilitando el movimiento vertical de la extremidad con eficacia. De este modo, se mantiene la funcionalidad original de la estructura, garantizando la correcta articulación de la pata durante la locomoción del robot.

Figura 35.

Diseño original de las patas



5.2.7 Impresión de Piezas y Resultados

En esta etapa se llevó a cabo la impresión de las piezas diseñadas en el software de modelado 3D, utilizando tecnología de impresión 3D con filamento PLA. Como se observa en la figura 36, el proceso se realizó en la impresora 3D del Club de Robótica de la Universidad Politécnica Salesiana de Guayaquil, lo que permitió fabricar con precisión la estructura principal del cuerpo del robot, los soportes para los servomotores y diversos elementos personalizados para la integración del módulo Grove AI Vision V2 y otros componentes electrónicos.

Figura 36.

Impresora 3D del club de robótica

Posteriormente, las piezas impresas como se muestra en la figura 37 fueron ensambladas y sometidas a pruebas de ajuste y funcionalidad. Estas pruebas incluyeron la verificación de la compatibilidad con el diseño original y la evaluación de la capacidad de las piezas para resistir las cargas y movimientos generados durante la operación normal del robot. Los resultados obtenidos confirmaron la precisión y robustez del diseño, asegurando así una base sólida para el montaje final del robot y contribuyendo de manera significativa a un desempeño óptimo del sistema.

Figura 37.

Piezas impresas del robot

5.3 Selección y Diseño del Circuito Electrónico

La selección y diseño del circuito electrónico constituye una etapa clave en el desarrollo del robot cuadrúpedo, ya que asegura la integración eficiente de los componentes electrónicos necesarios para su funcionamiento. En esta fase, se determinaron los módulos y sensores más adecuados, como el microcontrolador Esp32, el Grove AI Vision V2 y los reguladores de voltaje, garantizando compatibilidad y eficiencia energética. Además, se diseñó un esquema de distribución de energía para alimentar tanto los servomotores como los módulos de control y visión artificial, asegurando un rendimiento estable. Este circuito también integra los sistemas de comunicación necesarios para la interacción con la aplicación móvil y el procesamiento de datos en tiempo real, permitiendo un control preciso y autónomo del robot.

5.3.1 Referencias Para el Diseño de la Tarjeta

El diseño de la tarjeta electrónica del robot cuadrúpedo se fundamentó en el análisis y la adaptación de la estructura de un shield para Arduino Nano, orientado al control simultáneo de servomotores. Tal como se observa en la figura 38, este shield se caracteriza por la organización eficiente de sus pines y por la capacidad de alimentación a múltiples servos de manera estable. A partir de este modelo, se llevaron a cabo modificaciones clave para integrar el microcontrolador ESP32, optimizar la distribución de energía a través de reguladores dedicados y acomodar componentes adicionales, entre los que destaca el módulo Grove AI Vision V2. Estas adaptaciones permitieron crear una tarjeta personalizada que satisface los requerimientos específicos del proyecto, garantizando una correcta conexión y el funcionamiento adecuado de todos los módulos electrónicos en un espacio compacto y bien organizado.

Figura 38.

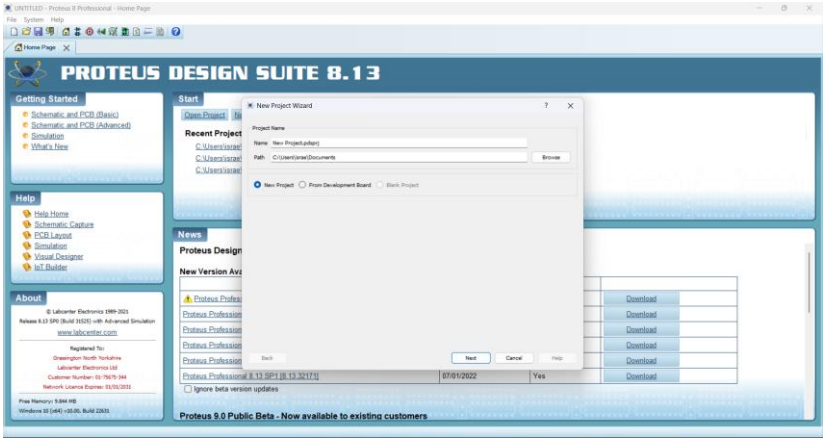
Shield para Arduino Nano

5.3.2 Creación del Archivo en Proteus

Para el diseño de las tarjetas electrónicas, se empleó Proteus en su versión 8.13, la cual ofrece mayor versatilidad y funcionalidades mejoradas. En un primer paso, se planteó la creación de dos tarjetas, una tarjeta base que integra los reguladores de voltaje, el ESP32 y los pines de conexión para los servomotores, y una segunda tarjeta, que se monta sobre la anterior, destinada a alojar el módulo Grove AI Vision V2. El proceso inicia con la opción New Project del menú principal de Proteus, tal como se aprecia en la figura 39.

Figura 39.

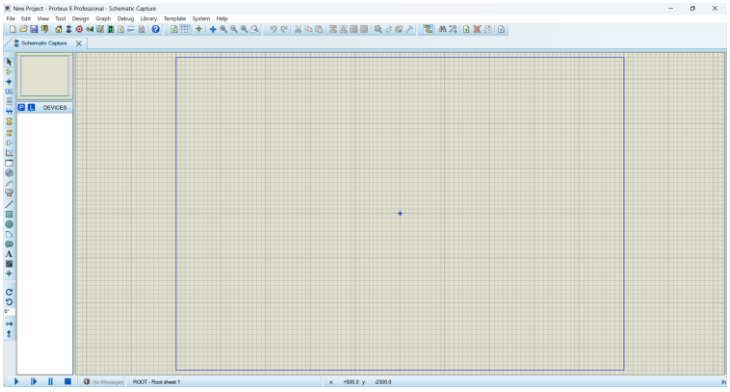
Creación del archivo en Proteus



Una vez asignado el nombre del proyecto; para este caso, se optó por “tarjetaEsp32” y “TarjetaGrove” a fin de distinguir cada una de las placas, se genera el espacio de trabajo en blanco como se muestra en la figura 40, donde se comienza el diseño esquemático de las dos tarjetas.

Figura 40.

Hoja de trabajo en Proteus

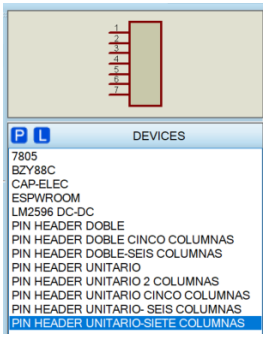


5.3.3 Diseño Esquemático de la Tarjeta ESP32

Para el diseño esquemático de la tarjeta Esp32 se utilizarán los siguientes materiales: reguladores de voltaje LM7805 y LM2596 DC-DC, el microcontrolador ESP32, capacitores, diodos y pin headers para la conexión de los pines de los servomotores y el espacio destinado a la fuente externa. Tal como se observa en la figura 41, los materiales mencionados fueron seleccionados y organizados en el entorno de Proteus.

Figura 41.

Materiales utilizados para el esquemático de la tarjeta ESP32



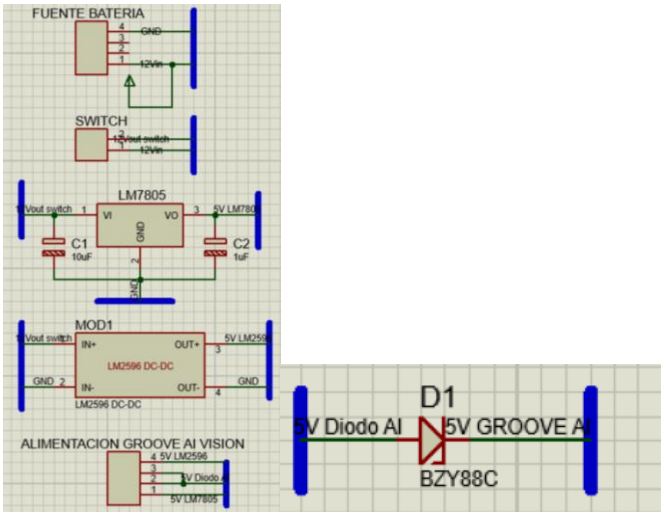
El primer paso consiste en diseñar la distribución del voltaje desde la fuente externa, utilizando un pin header de cuatro pines que encaje con el conector de la batería LiPo. Desde este conector, el voltaje se dirige hacia un interruptor (“switch”) que permitirá encender y apagar el robot. Posteriormente, el circuito se divide en dos rutas hacia los reguladores de voltaje.

La primera ruta lleva el voltaje al regulador LM7805, que proporciona 5V para alimentar directamente al ESP32. A ambos lados del regulador, se colocaron capacitores para mitigar interferencias: uno de 10µF en la entrada y otro de 1µF en la salida. La segunda ruta se dirige al módulo LM2596, que regula a 5V, pero para alimentar a los ocho servomotores. Esta configuración, tal como se muestra en la figura 42, permite evitar que el ruido generado por los servomotores interfiera con el funcionamiento del microcontrolador.

Para alimentar el módulo de visión artificial Grove AI Vision V2, se utilizó un pin header de cuatro pines. Este diseño permite cambiar entre fuentes de alimentación mediante un puente. Además, cualquier salida seleccionada para el Grove AI Vision pasa por un diodo para proteger el módulo de posibles daños.

Figura 42.

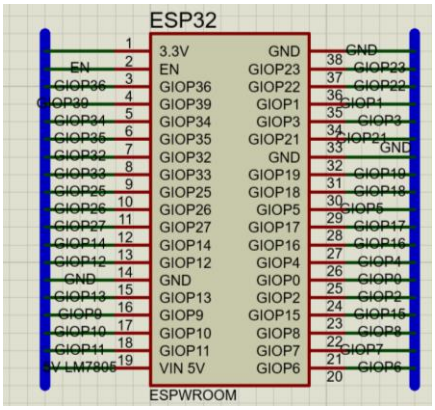
Distribución de voltaje de la tarjeta ESP32



En cuanto a la conexión del ESP32, se diseñó conforme al datasheet proporcionado por el fabricante, tal como se observa en la imagen 43. Un cambio importante en este diseño es omitir uso del pin de 3.3V.

Figura 43.

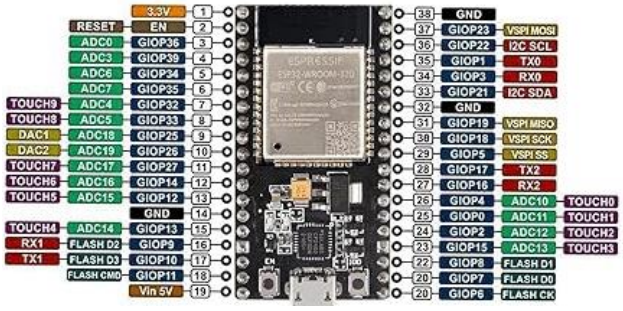
Diseño de la tarjeta ESP32



La figura 44 muestra la descripción y distribución detallada de los pines del ESP32 proporcionada por el fabricante, indicando los pines GIOP que se pueden usar, como los pines de la fuente de alimentación.

Figura 44.

Detalle de los pines de la tarjeta ESP32 según el fabricante

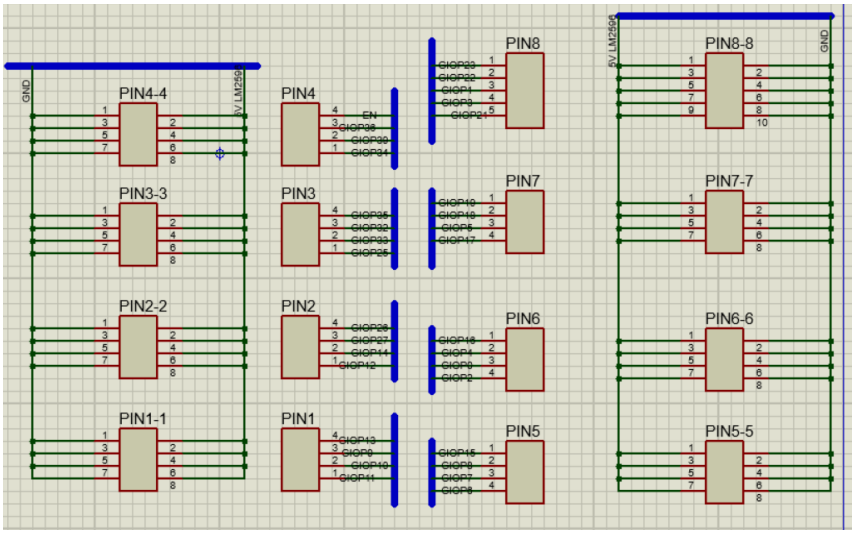


Para el diseño de los pines que conectarán los servomotores, se optó por un esquema similar al shield de servos del Arduino Nano. El orden establecido es GND, VCC y señal, asegurando que

la señal esté más cerca del ESP32. En el diseño se utilizaron pin headers que conectan los pines PIN4, PIN3, PIN2, PIN1, PIN5, PIN6, PIN7 y PIN8. Para cumplir con el orden mencionado, se usó un pin header de dos columnas y cuatro filas para conectar GND y la fuente regulada del LM2596, tal como se aprecia en la figura 45.

Figura 45.

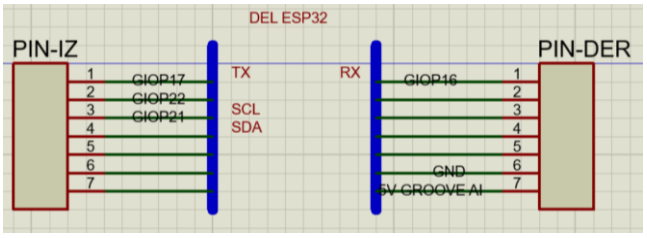
Conexiones de los pines a los servomotores



Por último, se incluyó un pin header de siete filas para transmitir la señal y la alimentación hacia la tarjeta que contiene el Grove AI Vision V2, que se montará sobre la placa principal. Este módulo de visión artificial utiliza comunicación I2C o UART, y se evaluó el uso de ambos protocolos para determinar cuál funciona mejor, como se ilustra en la figura 46.

Figura 46.

Pin header de siete filas

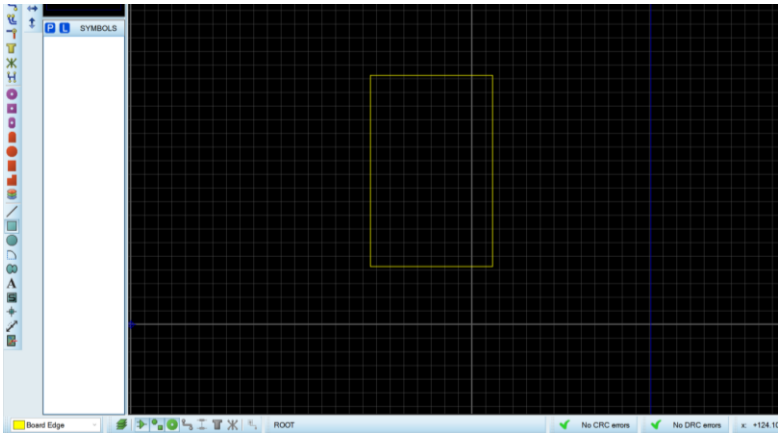


5.3.4 Diseño PCB de Tarjeta Para ESP32

Para el diseño en PCB de la tarjeta ESP32, primero se determina el tamaño de la tarjeta a realizar. En este caso, las dimensiones son de 9 cm de alto por 6.5 cm de ancho. El proceso comienza seleccionando la herramienta "2D Graphic Box" y configurando la capa correspondiente a la tarjeta para definir su forma, como se observa en la imagen 47.

Figura 47.

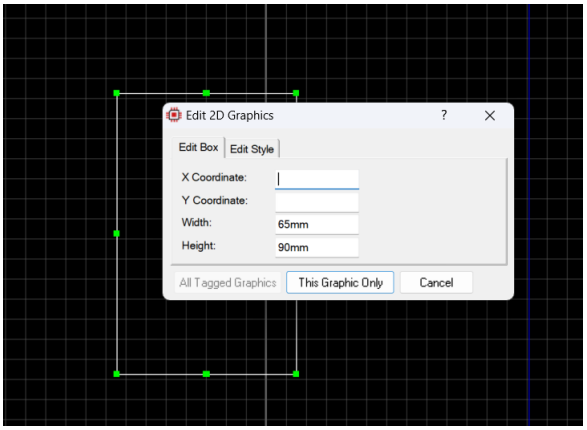
Configuración de la capa de la tarjeta Esp32



Una vez creada la forma, se hace clic derecho en sus propiedades para ingresar las medidas mencionadas previamente. Este paso se ilustra en las imágenes 48.

Figura 48.

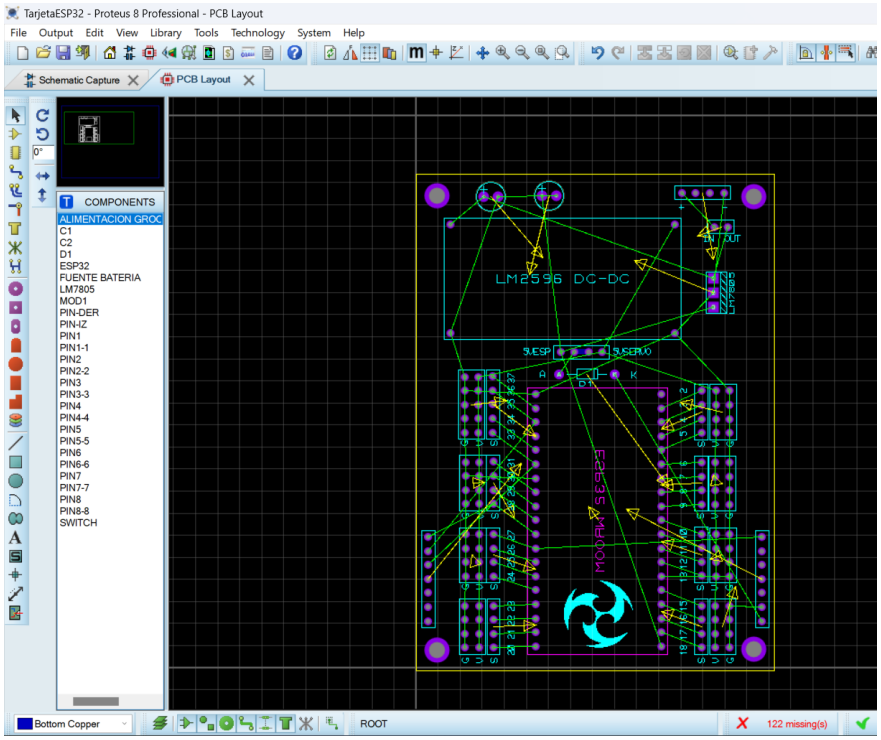
Dimensiones de la tarjeta Esp32 para el diseño en PCB



Con la forma definida, se procede a distribuir los componentes en la tarjeta. Es importante mencionar que la posición del ESP32 se orientó hacia la parte inferior para dejar espacio a la tarjeta que se montará encima, como se aprecia en la figura 49.

Figura 49.

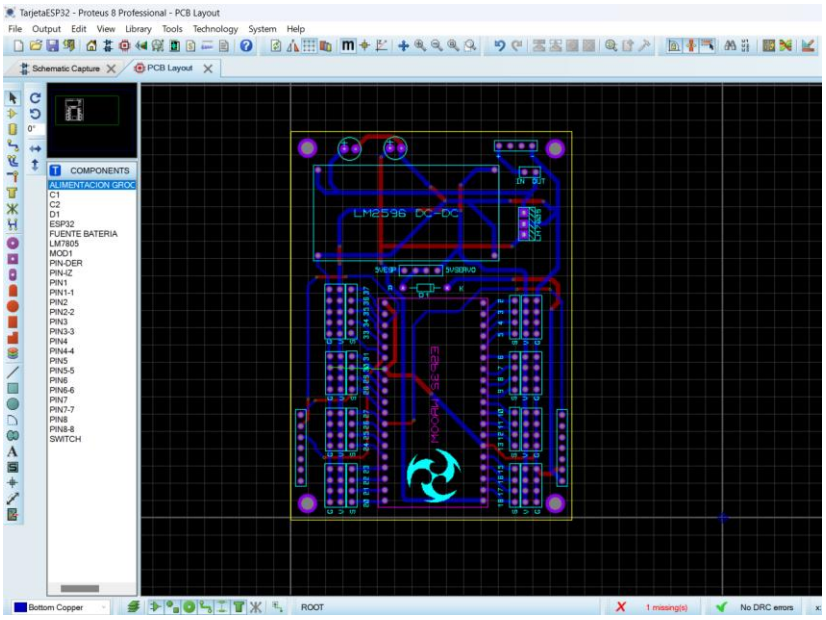
Distribución de los componentes en la tarjeta ESP32



Para las conexiones de la tarjeta, no se utilizó el auto-ruteo debido a que el diseño generado automáticamente presentaba errores. En su lugar, se realizaron las conexiones de manera manual, utilizando las capas de rutas tanto en la parte superior como en la inferior de la tarjeta, como se muestra en la figura 50.

Figura 50.

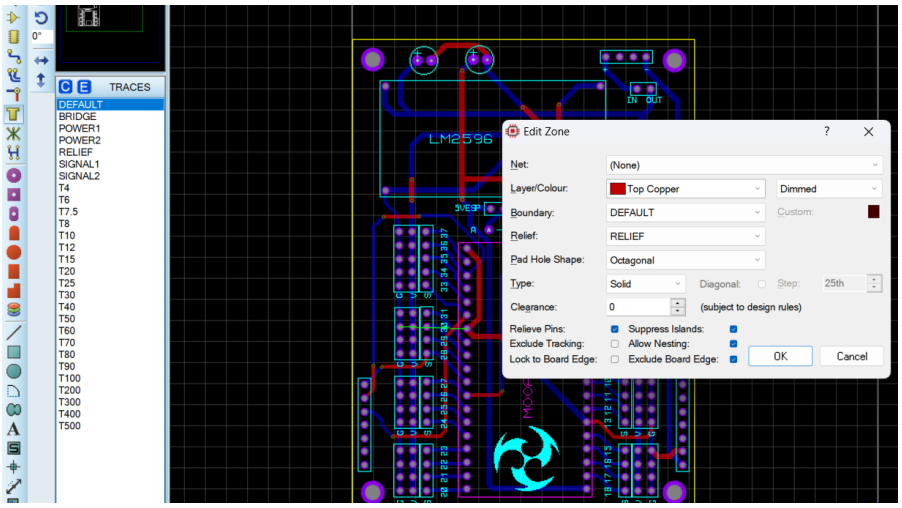
Conexiones realizadas de manera manual



Posteriormente, se agregaron las máscaras superiores e inferiores a la tarjeta utilizando la herramienta "Zone Mode" para cubrir toda la superficie de la PCB, tal como se observa en la figura 51.

Figura 51.

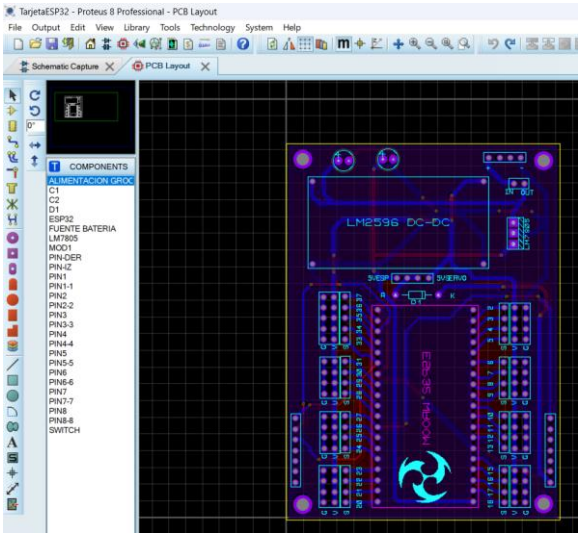
Máscaras superiores e inferiores de la tarjeta



El resultado final del diseño de la tarjeta se muestra en la figura 52, donde se puede apreciar la previsualización de la máscara que tendrá la tarjeta electrónica.

Figura 52.

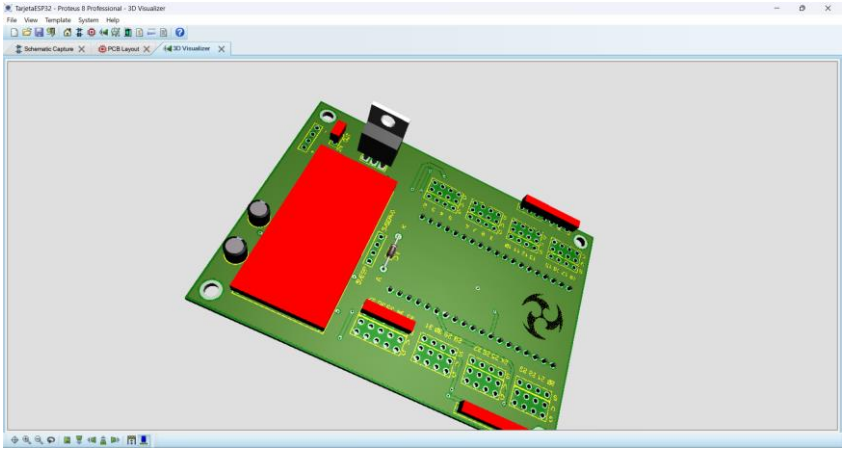
Diseño final de la tarjeta ESP32



Una vez completado este proceso, se puede obtener una vista previa en 3D del diseño, como se ilustra en la figura 53.

Figura 53.

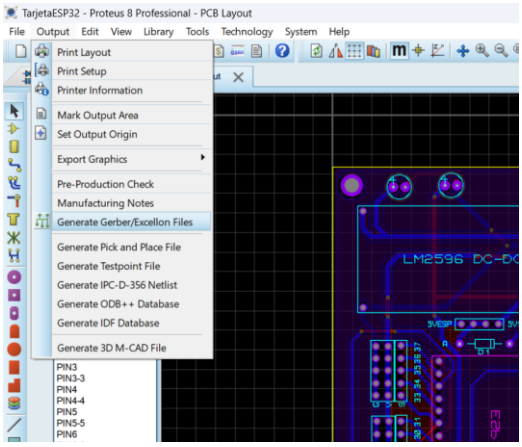
Vista 3D del diseño de la tarjeta ESP32



Para finalizar, se generan los archivos Gerber necesarios para la fabricación de la tarjeta. Este proceso comienza seleccionando la opción "Output" y luego "Generate Gerber", como se observa en la figura 54.

Figura 54.

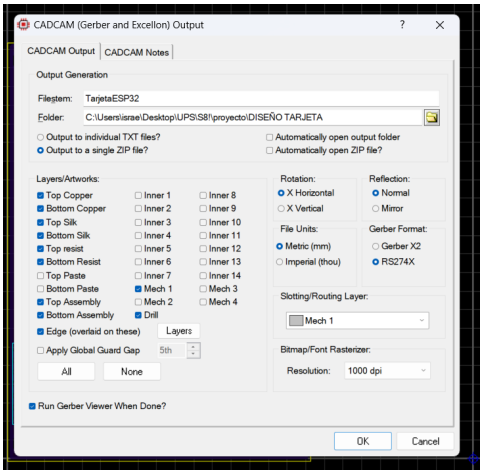
Archivos Gerber generados para la fabricación de la tarjeta ESP32



En la configuración de los archivos Gerber, se recomienda cambiar el formato a "RS-274X", ya que es el formato más adecuado para la fabricación y se ajusta la resolución a 1000 dpi para obtener una mejor definición, tal como se muestra en la figura 55.

Figura 55.

Formato RS-274X de los archivos Gerber para la fabricación de la tarjeta ESP32



5.3.5 Diseño Esquemático de la Tarjeta Grove

Para el diseño del esquemático de la tarjeta Grove, se utilizaron los siguientes materiales: un dipswitch que permite cambiar entre los modos de comunicación UART e I2C, el módulo Grove AI Vision V2 y pin headers que conectan la tarjeta con la placa inferior, como se observa en la figura 56.

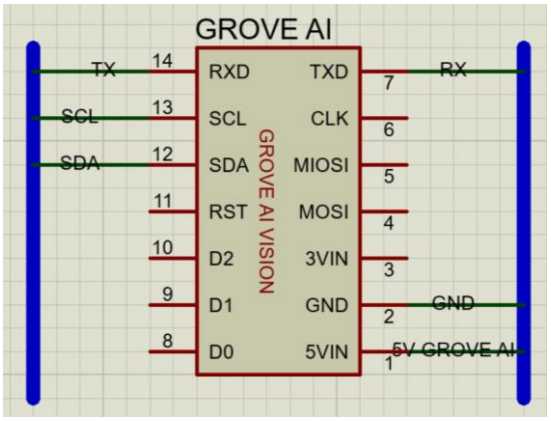
Figura 56.

Materiales utilizados para el diseño esquemático de la tarjeta Grove

El proceso comienza conectando los pines del Grove AI Vision V2, siguiendo el esquema de conexiones mostrado en su tarjeta, tal como se aprecia en la figura 57.

Figura 57.

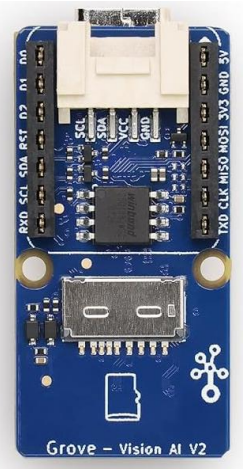
Conexiones del Grove AI Vision V2



La figura 58 proporciona una vista del módulo Grove AI Vision V2 en físico para una referencia adicional.

Figura 58.

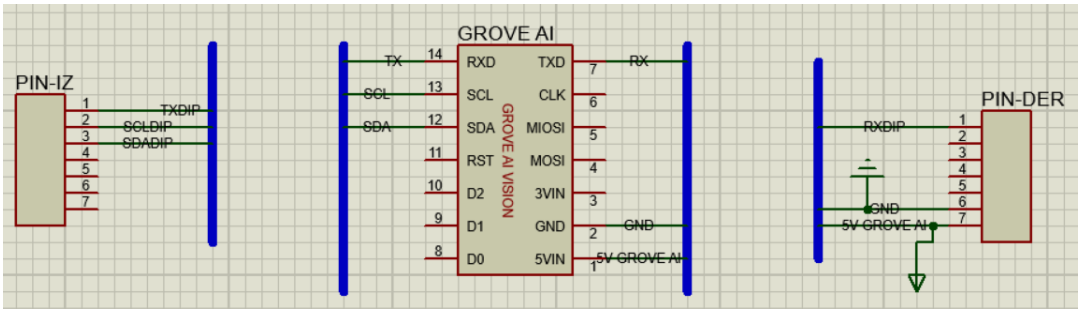
Módulo Grove AI Vision V2



En ambos lados de la tarjeta, se colocaron pin headers que transportan la señal y la fuente de alimentación desde la tarjeta inferior. Esta configuración asegura una distribución eficiente de las conexiones, como se ilustra en la figura 59.

Figura 59.

Distribución de las conexiones en la tarjeta Grove

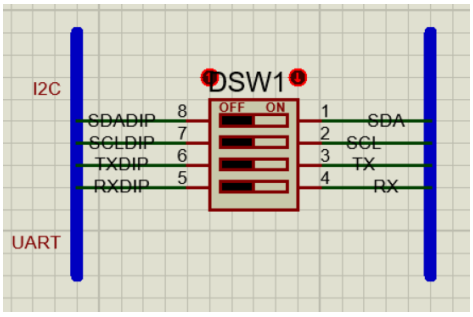


Finalmente, se integró el dipswitch, que recibe las señales provenientes de la tarjeta inferior y permite seleccionar el modo de comunicación deseado. En este diseño, los pines 1, 2, 8 y 7 están

asignados a SDA y SCL para la comunicación I2C, mientras que los pines 3, 4, 5 y 6 están configurados para TX y RX en la comunicación UART. La salida del dipswitch se dirige directamente al módulo Grove AI Vision V2, como se muestra en la figura 60.

Figura 60.

Distribución de los pines en el dipswitch

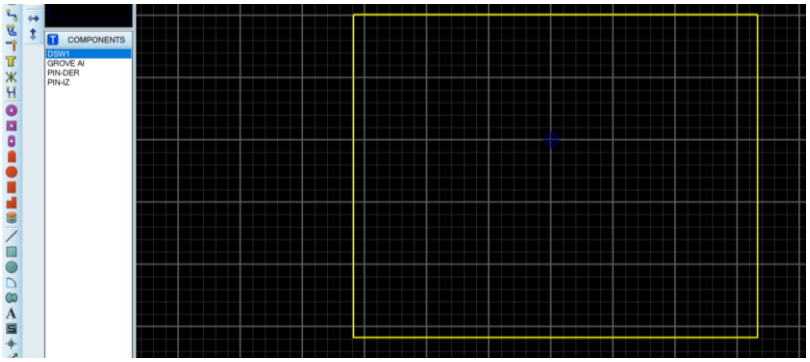


5.3.6 Diseño PCB Tarjeta Grove

El diseño en PCB de la tarjeta Grove comienza con la definición de las dimensiones de la tarjeta, establecidas en 5.2 cm de alto por 6.5 cm de ancho. Para ello, se utiliza la herramienta "2D Graphic Box" y se selecciona la capa correspondiente a la tarjeta, lo que permite crear su forma inicial. Una vez definida, se accede a las propiedades de la forma mediante clic derecho y se ingresan las medidas mencionadas. Este paso se ilustra en la figura 61.

Figura 61.

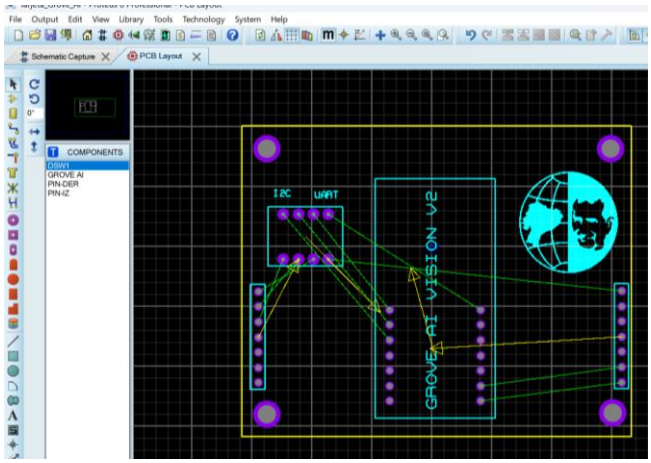
Diseño en PCB de la tarjeta Grove



A partir de la definición de la forma, se procede a distribuir los componentes a lo largo de la tarjeta de manera adecuada. La organización de los elementos tiene en cuenta las conexiones necesarias y el espacio disponible, como se observa en la figura 62.

Figura 62.

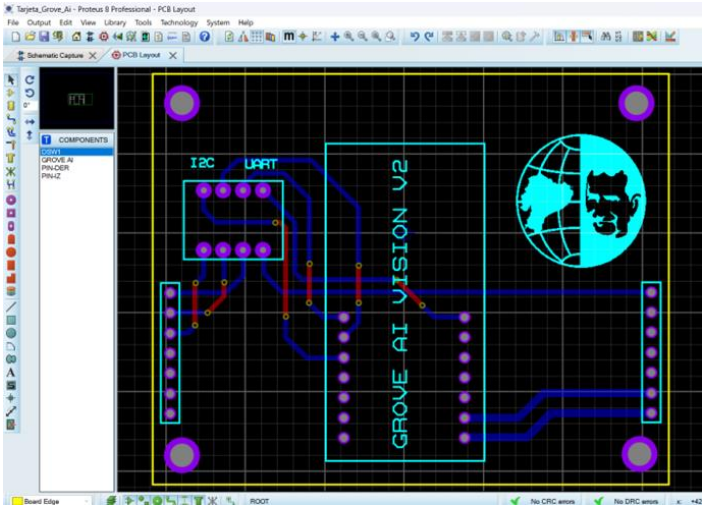
Distribución de los componentes en la tarjeta Grove



En el siguiente paso, se realizan manualmente las conexiones de los componentes, para evitar errores en las rutas de conexión como se muestra en la figura 63.

Figura 63.

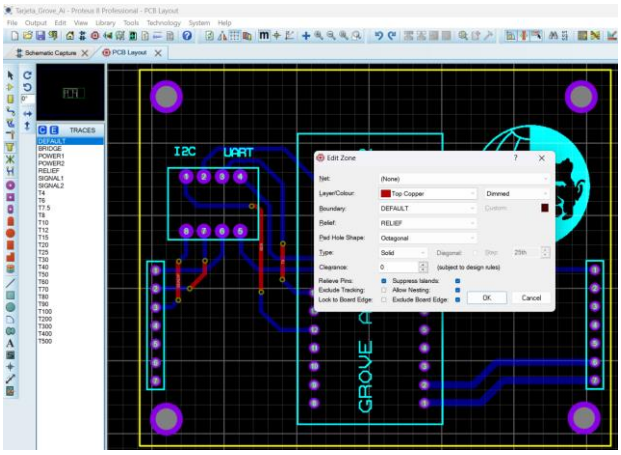
Conexión manual de los componentes de la tarjeta Grove



Posteriormente, se aplican las máscaras de cobre en las capas superior e inferior de la tarjeta utilizando la herramienta "Zone Mode" para cubrir toda la superficie de la PCB. Este procedimiento asegura una mejor distribución del flujo eléctrico y protección contra interferencias, tal como se muestra en la figura 64.

Figura 64.

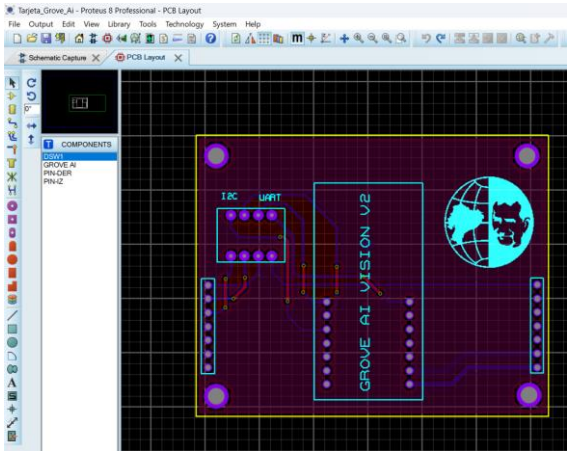
Máscaras de cobre en las capas superior e inferior de la tarjeta Grove



El diseño final de la tarjeta se presenta en la figura 65, donde se aprecia la configuración completa de los componentes y las conexiones.

Figura 65.

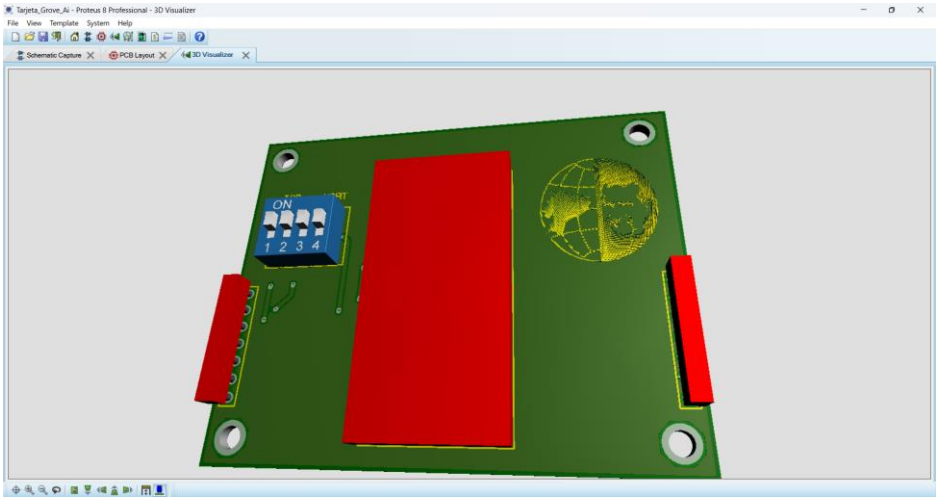
Diseño final de la tarjeta Grove



La vista previa en 3D permite visualizar cómo lucirá el diseño una vez fabricado, tal como se evidencia en la figura 66.

Figura 66.

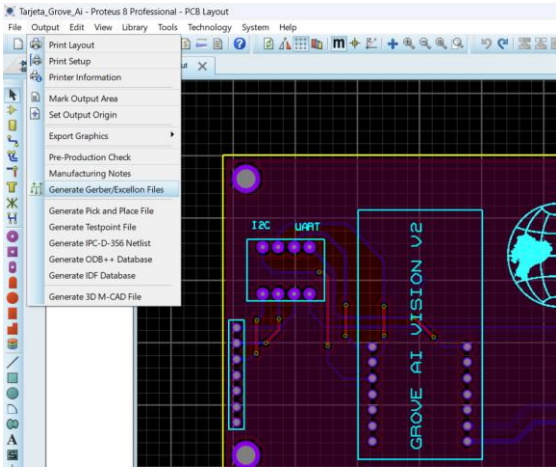
Vista 3D de la tarjeta Grove



Para concluir, se generan los archivos Gerber necesarios para la fabricación de la tarjeta. Este proceso inicia seleccionando la opción "Output" y posteriormente "Generate Gerber", como se ilustra en la figura 67.

Figura 67.

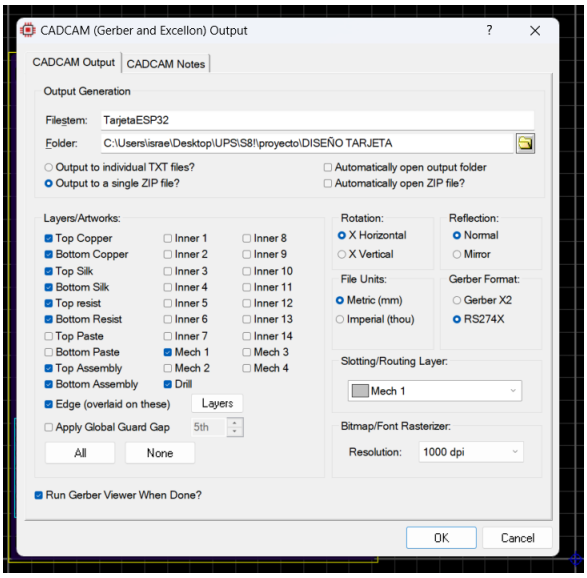
Archivos Gerber generados para la fabricación de la tarjeta Grove



Configuración de los archivos Gerber al formato a "RS-274X" para la fabricación de la tarjeta Grove como se muestra en la figura 68.

Figura 68.

Formato RS-274X de los archivos Gerber para la fabricación de la tarjeta Grove

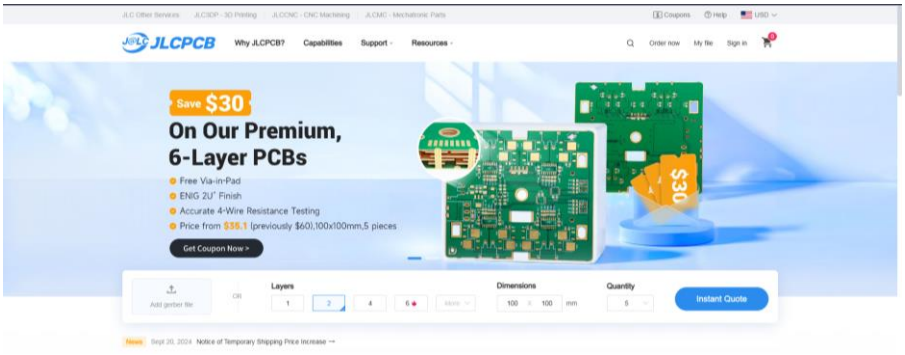


5.3.7 Fabricación de las Tarjetas

La fabricación de las tarjetas se realizó mediante la plataforma JLCPCB, que permite cargar fácilmente los archivos Gerber y enviarlos para producción, como se muestra en la figura 69.

Figura 69.

Carga de los archivos Gerber para la fabricación de las tarjetas ESP32 y Grove



Una vez que se han subido las tarjetas a la plataforma, se proporciona una vista preliminar que permite revisar el diseño antes de confirmar la fabricación. Las figuras 70 y 71 muestran estas vistas preliminares, donde se pueden apreciar los detalles de cada tarjeta diseñada.

Figura 70.

Vista preliminar de la tarjeta ESP32 en la plataforma JLCPCB

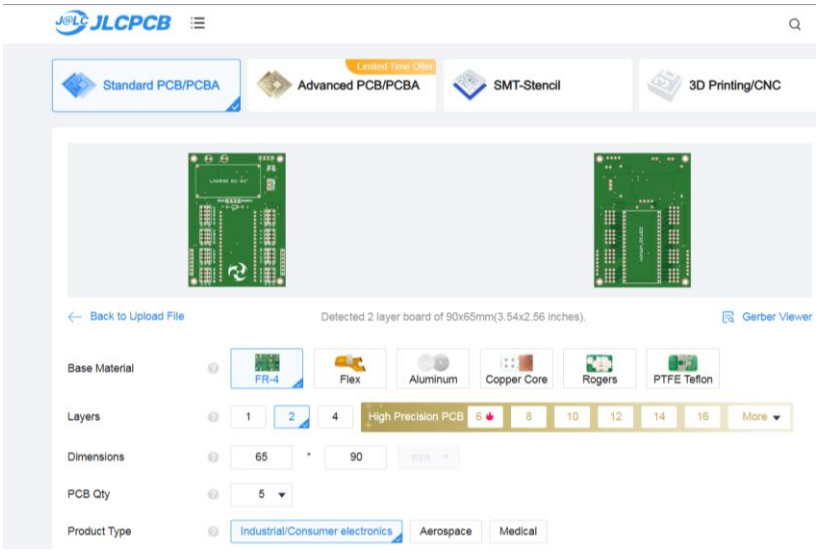


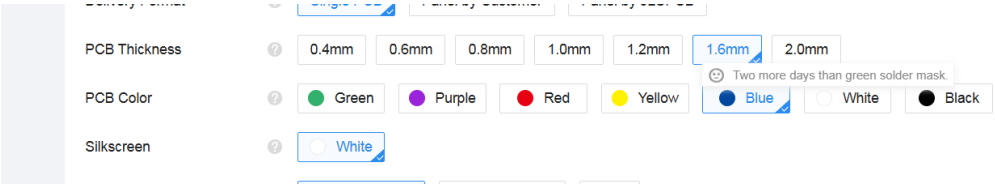
Figura 71.

Vista preliminar de la tarjeta Grove en la plataforma JLCPCB

En este caso, se realizó un ajuste estético en ambas tarjetas, cambiando el color del PCB de verde a azul para un acabado más personalizado. Este cambio está reflejado en la figura 72, donde se observa el color seleccionado.

Figura 72.

Selección de color de las tarjetas

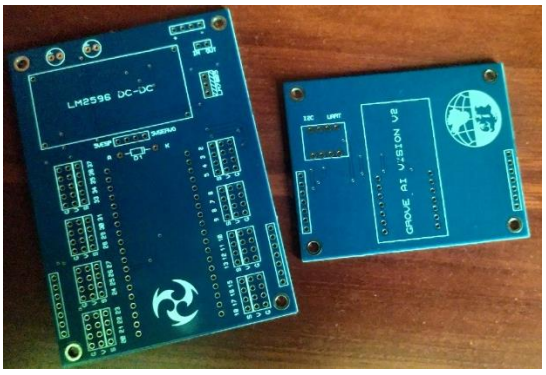


5.3.8 Resultado y Soldado de los Componentes

En esta etapa del proyecto, se obtuvo la tarjeta electrónica diseñada para integrar todos los componentes esenciales del robot cuadrúpedo. Entre estos se incluyen el microcontrolador ESP32, el módulo Grove AI Vision V2, los reguladores de voltaje LM2596 y LM7805, así como los conectores necesarios para los servomotores y sensores. Las tarjetas fabricadas, antes del ensamblaje, se pueden observar en la figura 73.

Figura 73.

Resultado de fabricación de las tarjetas ESP32 y Grove

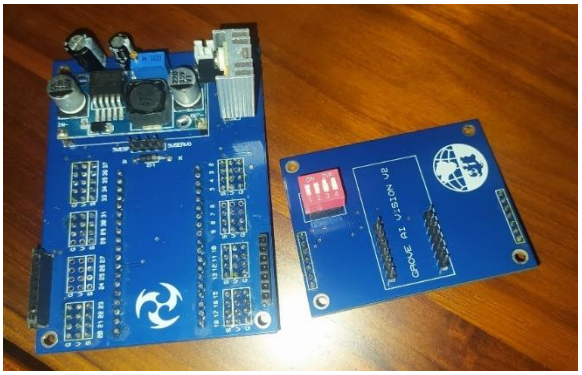


Con las tarjetas listas, se procedió al proceso de soldado de los componentes electrónicos. Este paso fue crucial para garantizar conexiones sólidas y eficientes entre las diferentes partes del circuito. La figura 74 muestra el resultado tras la soldadura, donde los componentes fueron fijados correctamente en sus respectivas posiciones en la tarjeta.

Una vez completado el soldado, se llevaron a cabo pruebas de continuidad y funcionamiento. Estas pruebas permitieron verificar que todas las conexiones fueran precisas y que los componentes electrónicos operaran según lo esperado, asegurando la funcionalidad integral del diseño. El resultado final fue una tarjeta electrónica compacta, robusta y funcional, capaz de satisfacer las exigencias del sistema.

Figura 74.

Soldadura de los componentes electrónicos en las tarjetas ESP32 y Grove



5.4 Desarrollo de la Aplicación Móvil

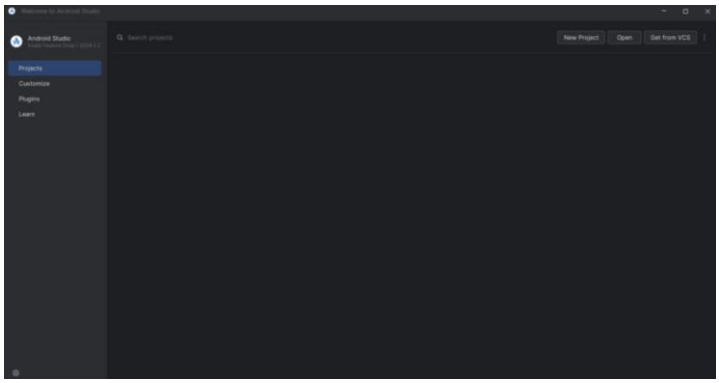
El desarrollo de la aplicación móvil representa una parte fundamental del proyecto, ya que proporciona una interfaz intuitiva para el control y monitoreo del robot cuadrúpedo. Diseñada en Android Studio, la aplicación permite gestionar los modos de operación manual y autónomo del robot, ofreciendo al usuario la capacidad de enviar comandos en tiempo real y recibir datos como la transmisión de video capturada por el módulo de visión artificial. En esta etapa, se definieron los requisitos funcionales y de diseño de la app, enfocándose en la usabilidad y la comunicación eficiente con el microcontrolador ESP32 a través de WiFi. Además, se incorporaron características que facilitan la interacción entre el usuario y el robot, garantizando un control preciso y la supervisión del desempeño del sistema.

5.4.1 Creación de la App

El diseño de la aplicación comienza con la creación de un nuevo proyecto en el entorno de desarrollo seleccionado como se puede observar en la figura 75.

Figura 75.

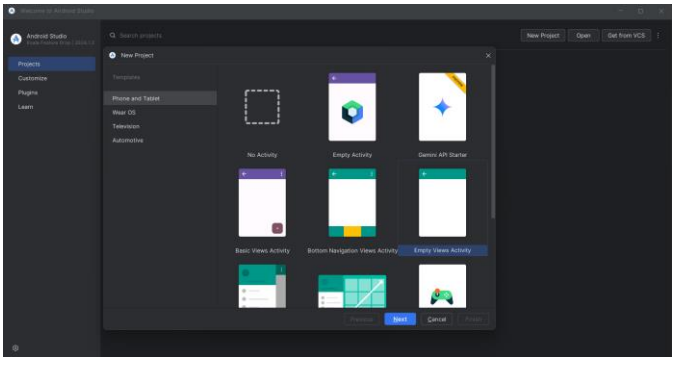
Creación de nuevo proyecto en Android Studio



Posteriormente, se elige una plantilla en blanco como punto de partida para el diseño. En este caso, se seleccionó la opción "Empty Views Activity", que proporciona una base sencilla y flexible para la estructura de la aplicación. Este paso está ilustrado en la figura 76.

Figura 76.

Selección de la plantilla para el diseño de la aplicación

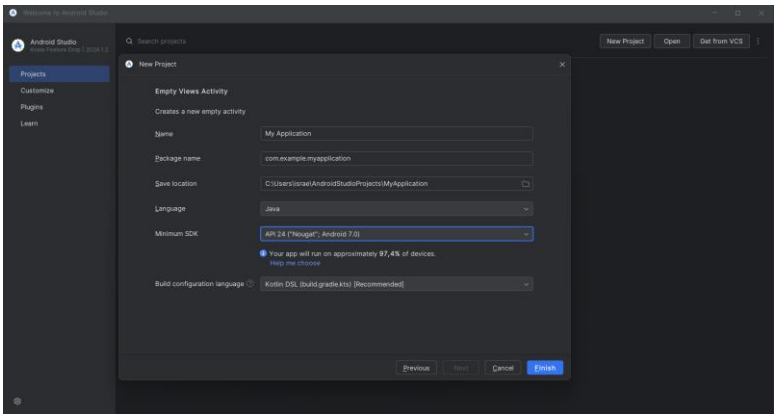


A continuación, se asigna un nombre al proyecto y se define que el lenguaje de programación utilizado será Java. Además, se establece Android 7.0 como versión mínima del

SDK, lo cual asegura que la aplicación será compatible con dispositivos que utilicen esta versión o superiores, como se detalla en la figura 77.

Figura 77.

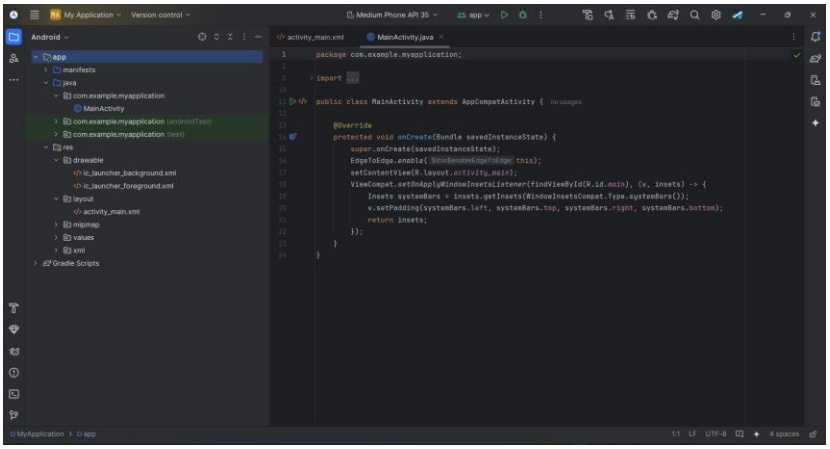
Definición del lenguaje de programación de la aplicación



Una vez completada la configuración inicial y tras la compilación, se obtiene el proyecto base como se muestra en la figura 78. En este punto, es importante destacar que el desarrollo de la aplicación se dividirá en dos componentes principales: el archivo XML, encargado del diseño de la interfaz de usuario, y las clases Java, que definirán la lógica y los comandos asociados a esas interfaces.

Figura 78.

Configuración inicial de la programación de la aplicación móvil

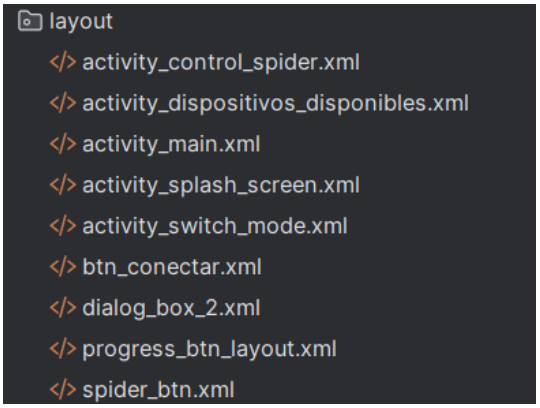


5.4.2 Interfaces de la App

El desarrollo de las interfaces de la aplicación móvil incluyó la creación de ocho diseños distintos como se muestra en la figura 79, cinco de ellos configurados como pestañas principales y tres diseños adicionales para complementar los botones utilizados en dichas pestañas. Las principales incluyen: activity_control_spider, activity_main, activity_dispositivos_disponibles, activity_splash_screen, activity_switch_mode, y dialog_box_2. Los botones complementarios son: btn_conectar, spider_btn y progress_btn_layout.

Figura 79.

Interfaces de la aplicación móvil

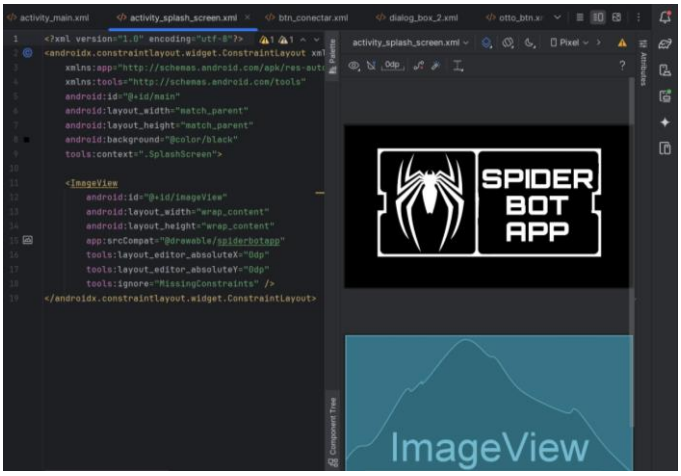


- **Activity_splash_screen.**

Esta interfaz funciona como un logotipo inicial que aparece al abrir la aplicación. Tras un periodo de cinco segundos, la App redirige automáticamente a la pantalla principal (activity_main). Este diseño está ilustrado en la figura 80.

Figura 80.

Interfaz Activity_splash_screen

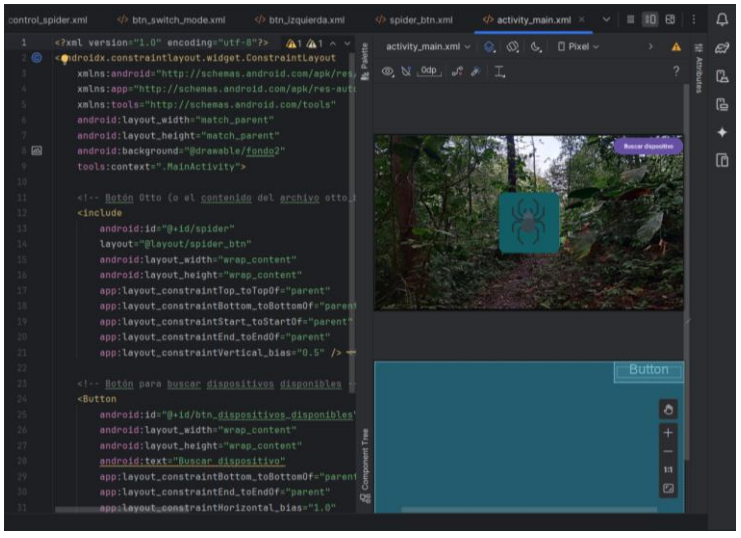


- **Main_activity.**

Tras el splash screen, la pantalla principal ofrece dos botones: uno central, que dirige a los controles del robot con el diseño spider_btn, y otro en la esquina superior derecha, que accede a la configuración de conexión WiFi. La interfaz incluye un fondo temático inspirado en la naturaleza y las arañas, como se muestra en la figura 81

Figura 81.

Interfaz Main_activity

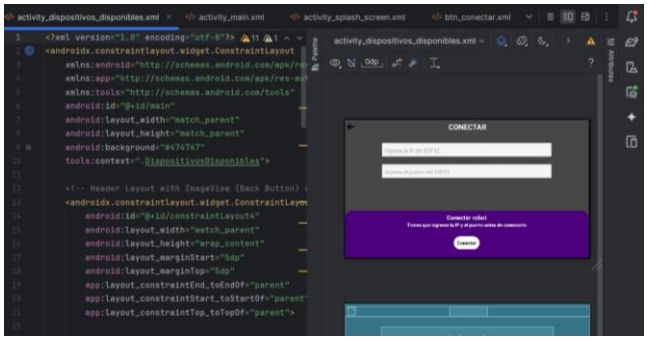


- **Activity_dispositivos_disponibles.**

Desde el botón de búsqueda de dispositivos en la pantalla principal, se configura la comunicación WiFi del robot como se muestra en la figura 82, incluyendo campos para ingresar la dirección IP, el puerto de conexión y un botón para conectar. En esta pestaña se utiliza el diseño btn_conectar, que mejora la apariencia del botón de conexión.

Figura 82.

Interfaz Activity_dispositivos_disponibles

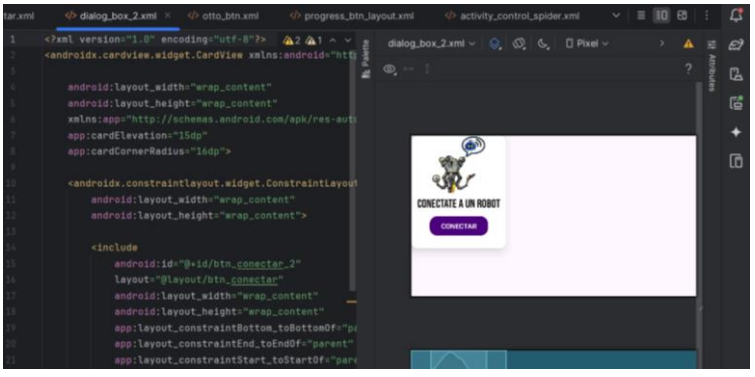


- **Dialog_box_2.**

Esta interfaz aparece como un cuadro de diálogo superpuesto, ocupando una parte de la pantalla. Su propósito es alertar al usuario de que primero debe establecer la conexión con el robot antes de acceder a los controles. Esta interfaz se puede observar en la figura 83.

Figura 83.

Interfaz Dialog_box_2

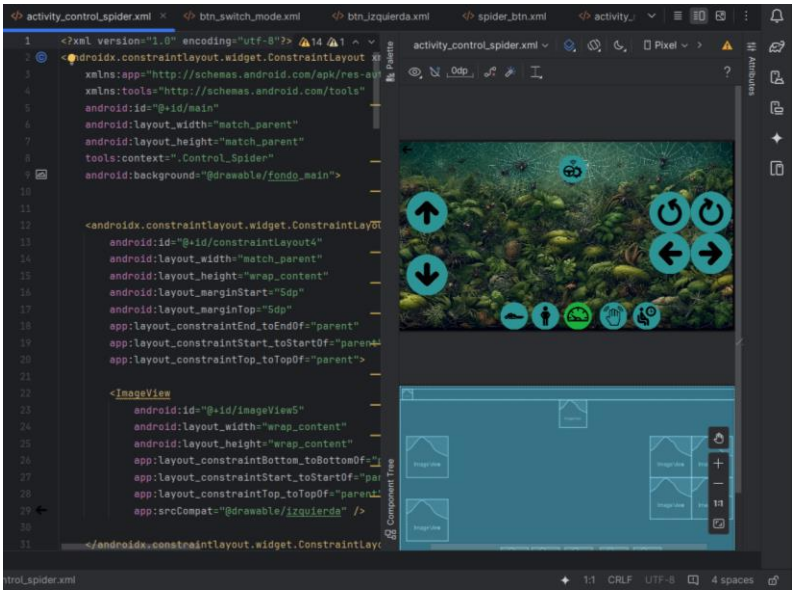


- **Activity_control_spider.**

Esta interfaz está dedicada al control del robot como se aprecia en la figura 84. Incluye un botón en la esquina superior izquierda para regresar a la pantalla principal y una serie de botones funcionales: avanzar, retroceder, girar a la izquierda, girar a la derecha y moverse lateralmente. Además, incorpora botones específicos para acciones adicionales: "acostarse" (robot en reposo total), "levantarse" (robot de pie), un botón con el símbolo de un velocímetro para alternar entre tres velocidades (lenta, normal y rápida), un botón para saludar y otro de "stand by" para que el robot realice un pequeño paso. Por último, se agregó un botón que me llevará a la pestaña switchMode para cambiar de modo controlado a autónomo y viceversa.

Figura 84.

Interfaz Activity_control_spider

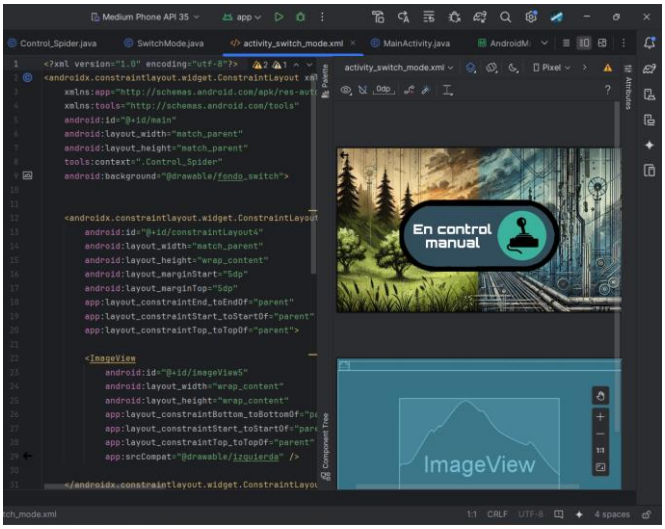


- **Activity_switch_mode.**

En esta interfaz está destinada para que se pueda cambiar el modo del robot de manual a autónomo y viceversa, donde el botón central envía el comando al microcontrolador.

Figura 85.

Interfaz Activity_switch_mode

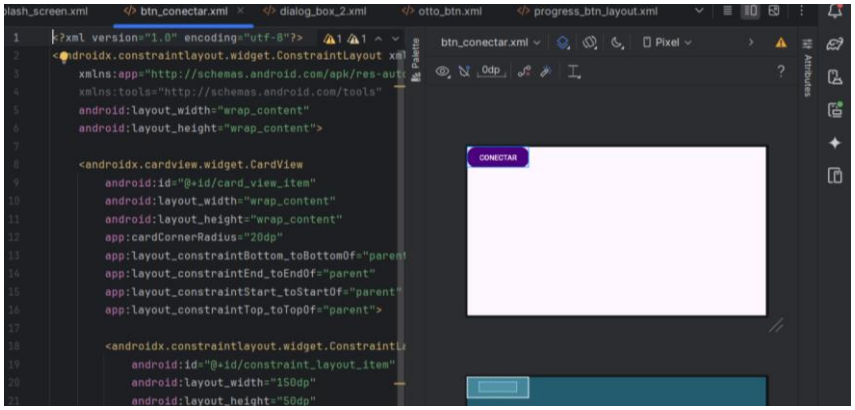


- **Btn_conectar, spider_btn, y progress_btn_layout.**

Finalmente, estos tres diseños de botones fueron utilizados en las diferentes pestañas descritas anteriormente, aportando un estilo mejorado y uniforme a la aplicación. El diseño del botón btn_conectar se puede visualizar en la figura 86.

Figura 86.

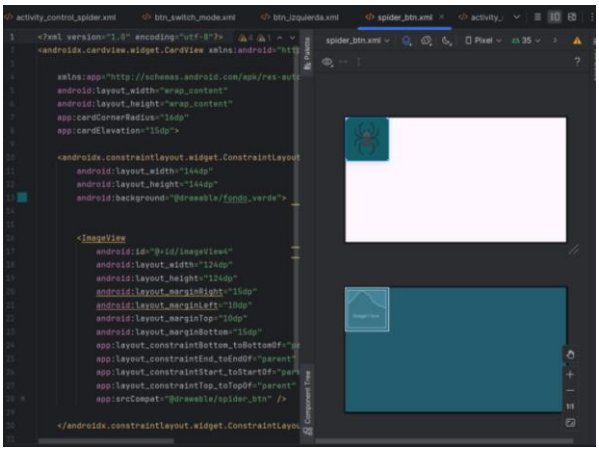
Diseño del botón btn_conectar



El diseño del botón spider_btn se muestra en la figura 87, está diseñado para facilitar la interacción del usuario con la interfaz gráfica, permitiendo ejecutar acciones específicas al ser presionado.

Figura 87.

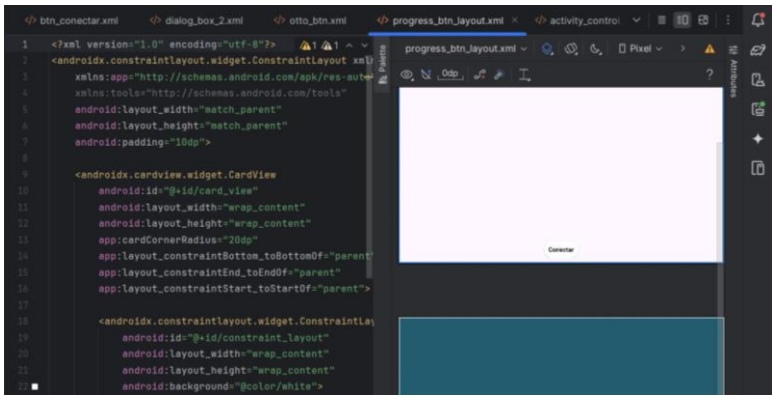
Diseño del botón spider_btn



El diseño del botón progress_btn_layout se muestra en la figura 88, combina diseño visual con funcionalidad para mostrar progreso dinámico al ser activado, asegurando interacción eficiente.

Figura 88.

Diseño del botón progress_btn_layout

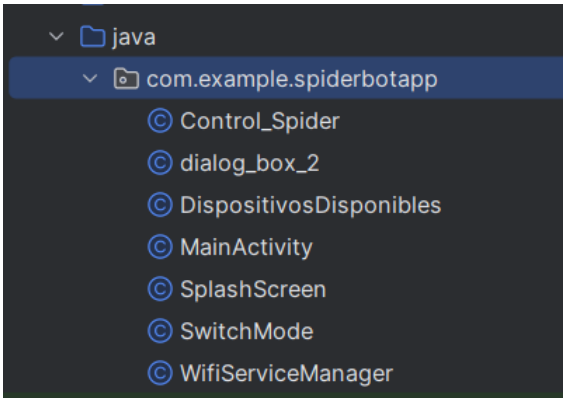


5.4.3 Clases de la App

El desarrollo de la aplicación involucró la implementación de seis clases principales: ControlSpider, MainActivity, DispositivosDisponibles, SplashScreen, DialogBox2, SwitchMode y WifiServiceManager. Estas clases estructuran la lógica y funcionalidad de la App, como se aprecia en la figura 89.

Figura 89.

Clases de la aplicación móvil

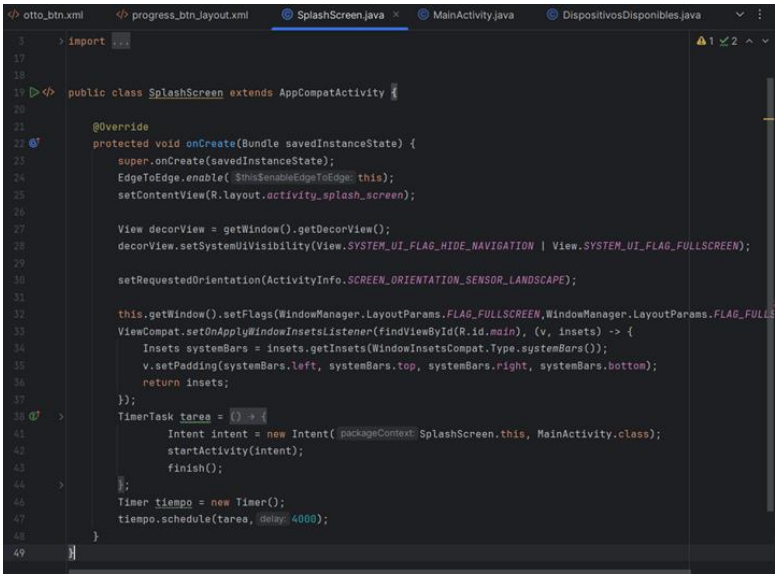


- **SplashScreen.**

Esta clase configura la pantalla de inicio para mostrarse a pantalla completa y en orientación horizontal fija. Además, define un tiempo de visualización de cinco segundos antes de redirigir al usuario a la pantalla principal (MainActivity). Este flujo inicial se ilustra en la figura 90.

Figura 90.

Clase SplashScreen



- **WifiServiceManager.**

La clase WifiServiceManager gestiona la conexión a un servidor WebSocket, permitiendo la comunicación entre la aplicación Android y el ESP32. Incluye funcionalidades para establecer, mantener y cerrar conexiones WebSocket, así como para enviar y recibir mensajes. Entre sus elementos clave se encuentran variables como webSocketClient para manejar la conexión y tempIp y tempPort para almacenar temporalmente la dirección IP y el puerto del servidor, como se describe en la figura 91.

Figura 91.

Clase WifiServiceManager

```
public class WifiServiceManager {  9 usages
    private static final String TAG = "WifiServiceManager";  6 usages
    private WebSocketClient webSocketClient;  9 usages
    private static WifiServiceManager instance;  3 usages
    private Context context;  3 usages

    // Variables temporales para IP y puerto
    private String tempIp;  2 usages
    private int tempPort;  2 usages
```

Luego, el constructor privado obliga a usar el método getInstance para crear o acceder a la única instancia de esta clase como se muestra en la figura 92.

Figura 92.

Método getInstance

```
private WifiServiceManager(Context context) {  1 usage
    this.context = context.getApplicationContext();
}
```

Posteriormente el patrón Singleton, como se muestras en la figura 93, asegura que haya una única instancia de la clase en toda la aplicación. Esto es útil cuando se necesita un único punto de conexión para manejar el WebSocket. Si la instancia ya existe la devuelve, si no existe crea una nueva, esto garantiza que no se creen múltiples conexiones que puedan interferir entre sí.

Figura 93.

Patrón Singleton

```
public static synchronized WifiServiceManager getInstance(Context context) {  3
    if (instance == null) {
        instance = new WifiServiceManager(context);
    }
    return instance;
}
```

El método connectToWebSocket como se ve en la figura 94 establece una conexión con el servidor WebSocket en la dirección IP y puerto especificados.

Figura 94.

Conexión al WebSocket

```
// Conectar al WebSocket
public void connectToWebSocket(String ipAddress, int port) { 2 usages
    setTempIpAndPort(ipAddress, port);  // Guardar IP y puerto temporalmente

    URI uri;
    try {
        uri = new URI( str: "ws://" + ipAddress + ":" + port);
    } catch (URISyntaxException e) {
        e.printStackTrace();
        return;
    }

    webSocketClient = new WebSocketClient(uri, new Draft_6455()) {
        @Override 1 usage
        public void onOpen(ServerHandshake handshakedata) {
            Log.d(TAG, msg: "Conexión WebSocket abierta");
            new Handler(Looper.getMainLooper()).post(() ->
                    Toast.makeText(context, text: "Conectado al ESP32", Toast.LENGTH_SHORT).show()
            );
        }

        @Override 3 usages
        public void onMessage(String message) {
            Log.d(TAG, msg: "Mensaje recibido: " + message);
        }

        @Override 2 usages
        public void onClose(int code, String reason, boolean remote) {
            Log.d(TAG, msg: "Conexión WebSocket cerrada: " + reason);
        }
```

Luego, en la clase almacena la IP y el puerto proporcionado por el usuario de manera temporal como se observa en la figura 95.

Figura 95.

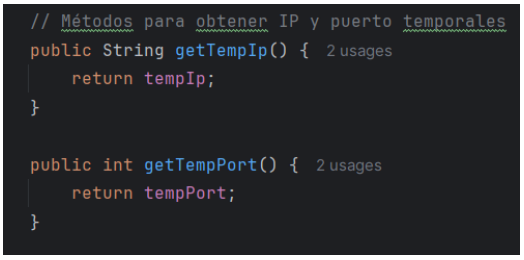
Guardar IP y puerto temporal

```
// Guardar IP y puerto temporalmente
public void setTempIpAndPort(String ip, int port) {  2 usages
    this.tempIp = ip;
    this.tempPort = port;
```

Estos valores se pueden recuperar más adelante mediante gettempip y gettempport, para manejar reconexiones u otras operaciones sin necesidad de pedir nuevamente estos datos al usuario como se observa en la figura 96.

Figura 96.

Métodos para obtener IP y puerto temporales



```
// Métodos para obtener IP y puerto temporales
public String getTempIp() {  2 usages
    return tempIp;
}

public int getTempPort() {  2 usages
    return tempPort;
}
```

El método sendData permite enviar un mensaje al servidor WebSocket, es decir que permitirá enviar el comando de la acción seleccionada por los botones del control_spider como se detalla en la figura 97.

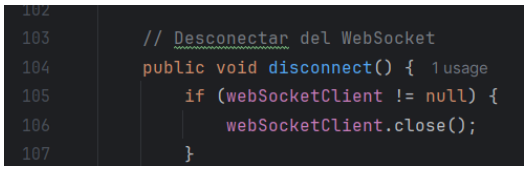
Figura 97.

Utilización del método senData para enviar datos al ESP32

Por último, el método disconnect cierra la conexión WebSocket si está activa como se observa en la figura 98.

Figura 98.

Método disconnect que cierra la conexión del WebSocket

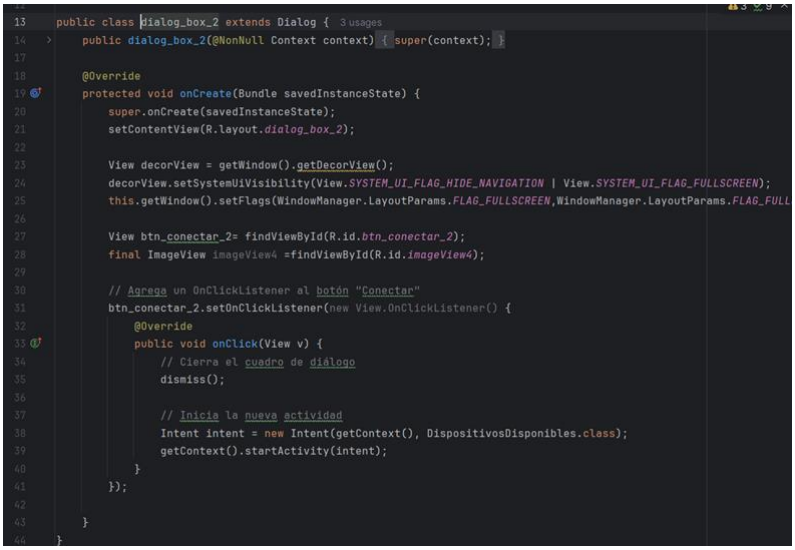


- **Dialog_box_2.**

Este código, al igual que el splash screen, se configura para mostrarse en pantalla completa y solo en horizontal. Define una clase personalizada, dialog_box_2 como se observa en la figura 99, que extiende Dialog para mostrar un cuadro de diálogo a pantalla completa con un diseño específico. Incluye el botón *btn_conectar_2*, que cierra el diálogo y lanza la actividad DispositivosDisponibles mediante un intent. También oculta la barra de navegación y habilita el modo de pantalla completa.

Figura 99.

Clase Dialog_box_2



```
public class dialog_box_2 extends Dialog {  3 usages
    public dialog_box_2(@NonNull Context context) { super(context); }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.dialog_box_2);

        View decorView = getWindow().getDecorView();
        decorView.setSystemUiVisibility(View.SYSTEM_UI_FLAG_HIDE_NAVIGATION | View.SYSTEM_UI_FLAG_FULLSCREEN);
        this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,WindowManager.LayoutParams.FLAG_FULL

        View btn_conectar_2= findViewById(R.id.btn_conectar_2);
        final ImageView imageView4 =findViewById(R.id.imageView4);

        // Agrega un OnClickListener al botón "Conectar"
        btn_conectar_2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Cierra el cuadro de diálogo
                dismiss();

                // Inicia la nueva actividad
                Intent intent = new Intent(getContext(), DispositivosDisponibles.class);
                getContext().startActivity(intent);
            }
        });

    }
}
```

- **DispositivosDisponibles.**

Este código define la clase DispositivosDisponibles que permite al usuario configurar y conectarse a un dispositivo ESP32 utilizando una dirección IP y un puerto ingresado manualmente como se muestra en la figura 100. La actividad utiliza un diseño asociado (activity_dispositivos_disponibles) y opera en modo de pantalla completa con orientación horizontal bloqueada. En el método onCreate, se inicializan las vistas (campos de texto para IP y puerto, y un botón de conexión) y se configuran los comportamientos interactivos de los elementos.

Figura 100.

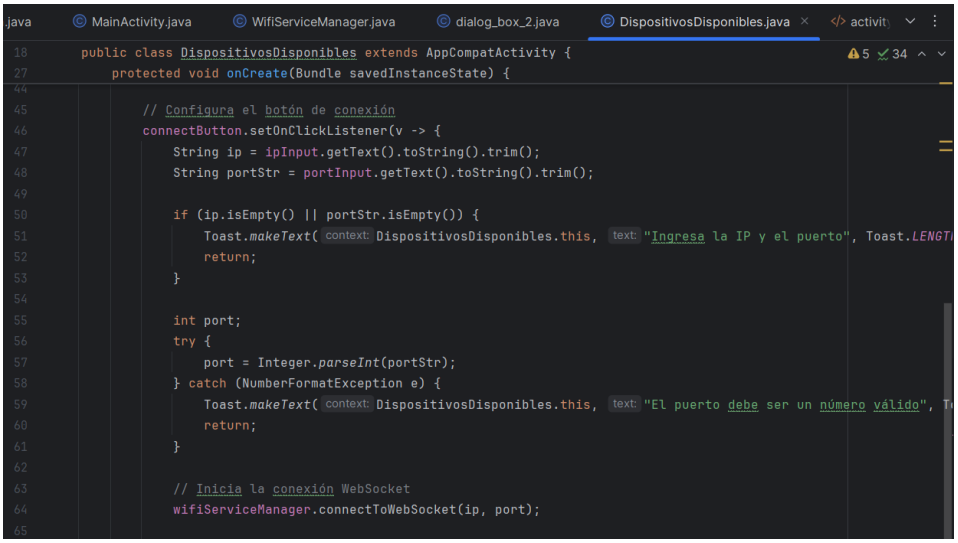
Clase DispositivosDisponibles

```java
public class DispositivosDisponibles extends AppCompatActivity {

    private WifiServiceManager wifiServiceManager;  4 usages
    private EditText ipInput;  2 usages
    private EditText portInput;  2 usages
    private View connectButton;  2 usages
    private static final String TAG = "DispositivosDisponibles";  no usages

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_dispositivos_disponibles);
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        View decorView = getWindow().getDecorView();
        decorView.setSystemUiVisibility(View.SYSTEM_UI_FLAG_HIDE_NAVIGATION | View.SYSTEM_UI_FLAG_FULLSCREEN);

        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_SENSOR_LANDSCAPE);
        this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FUL

        // Inicializa el WifiServiceManager
        wifiServiceManager = WifiServiceManager.getInstance( context: this);

        // Obtiene las referencias de las vistas
        ipInput = findViewById(R.id.ipInput);
        portInput = findViewById(R.id.portInput);
        connectButton = findViewById(R.id.myProgessButton);
```

Cuando el usuario presiona el botón de conexión, el código primero valida que los campos de IP y puerto no estén vacíos y que el puerto sea un número válido. Si la validación falla, se muestra un mensaje Toast notificando al usuario del error. Si los datos son válidos, se utiliza el objeto WifiServiceManager para iniciar una conexión WebSocket con los valores ingresados.

Figura 101.

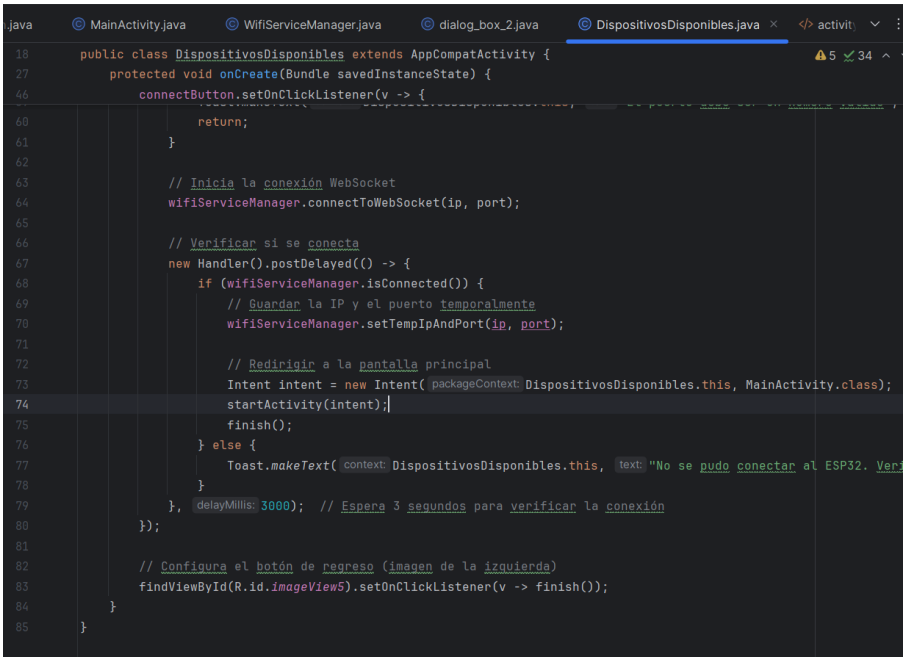
Validación de los campos del IP y del puerto



Tras intentar la conexión como se muestra en la figura 102, se espera tres segundos para verificar si fue exitosa. Si la conexión es exitosa, se guardan temporalmente la IP y el puerto mediante el WifiServiceManager, y se redirige al usuario a la pantalla principal (MainActivity). Si la conexión falla, se muestra un mensaje Toast indicando que no se pudo conectar al ESP32 y sugiere verificar los datos ingresados. Finalmente, la actividad incluye una funcionalidad para regresar a la pantalla anterior con el botón identificado como imageView5. Este botón simplemente finaliza la actividad actual cuando se presiona.

Figura 102.

Verificación de conexión al ESP32

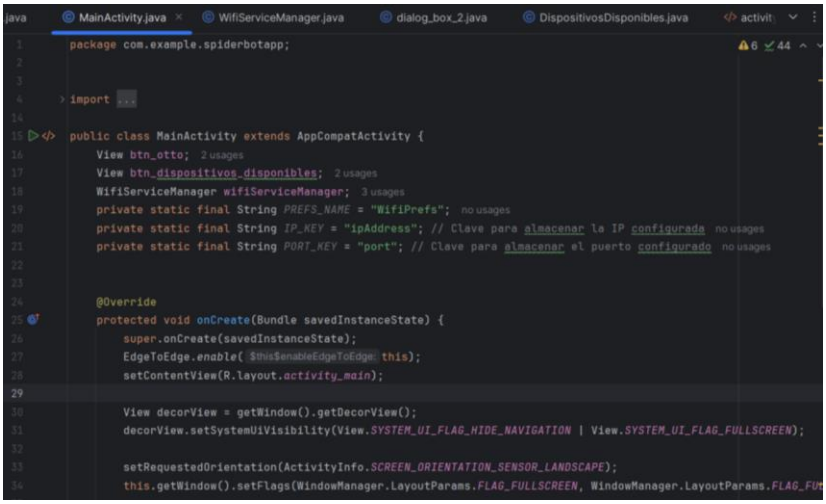


- **MainActivity.**

En la clase mainActivity como se muestra en la figura 103 su función principal es ofrecer dos opciones al usuario: acceder a los controles del robot (Control_Spider) o gestionar dispositivos disponibles (DispositivosDisponibles). La actividad se configura en modo de pantalla completa, con orientación horizontal fija, y utiliza un diseño XML asociado llamado activity_main.

Figura 103.

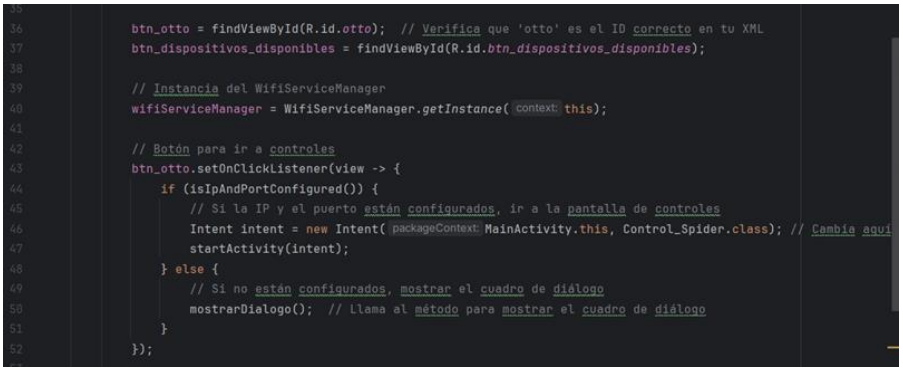
Clase MainActivity



El botón identificado como btn_spider como se observa en la figura 104, está diseñado para llevar al usuario a la pantalla de controles del robot (Control_Spider). Antes de realizar esta acción, la aplicación verifica si una dirección IP y un puerto han sido configurados temporalmente en el WifiServiceManager. Si los datos están disponibles, la actividad de controles se inicia mediante un Intent. Si no están configurados, se muestra un cuadro de diálogo (dialog_box_2) que permite al usuario ingresar estos datos.

Figura 104.

Configuración del botón btn_spider



El botón identificado como btn_dispositivos_disponibles redirige al usuario a la pantalla de gestión de dispositivos disponibles (DispositivosDisponibles) como indica la figura 105. Esta funcionalidad no requiere verificar si hay IP y puerto configurados, ya que su propósito es permitir al usuario configurar o seleccionar un dispositivo.

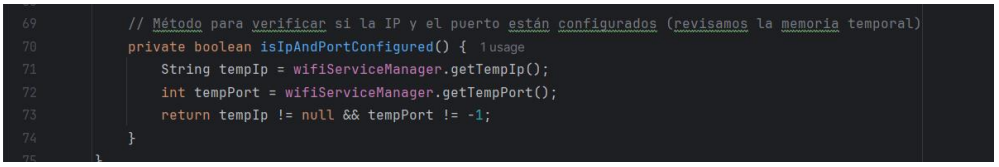
Figura 105.

Botón para ir a la pantalla de dispositivos disponibles

El método isIpAndPortConfigured() revisa si los valores de IP y puerto están almacenados temporalmente en el objeto WifiServiceManager como se muestra en la figura 106. Si ambos datos están presentes (IP no es null y el puerto es diferente de -1), el método devuelve true. Esto asegura que la conexión pueda establecerse sin mostrar el cuadro de diálogo de configuración.

Figura 106.

Método isIpAndPortConfigured()



El método mostrarDialogo() se utiliza para crear y mostrar una instancia de dialog_box_2. Este cuadro de diálogo permite al usuario ingresar y guardar la IP y el puerto del dispositivo ESP32 como se muestra en la figura 107. La ventana del diálogo tiene un fondo transparente para mejorar la estética visual.

Figura 107.

Método mostrardialogo()

```
60
61      // Método para mostrar el cuadro de diálogo para configurar IP y puerto
62      private void mostrarDialogo() {  1 usage
63          dialog_box_2 dialogBox_2 = new dialog_box_2( context: MainActivity.this);
64          dialogBox_2.getWindow().setBackgroundDrawable(new ColorDrawable(getResources().getColor(android.R.color.tr
65          dialogBox_2.setCancelable(true);
66          dialogBox_2.show();
67      }
```

- **ControlSpider.**

El código de la clase Control_Spider como se muestra en la figura 108 representa la pantalla que permite controlar al robot cuadrúpedo desde la aplicación Android. Al inicio, en el método onCreate como se observa en la figura 109, se configura la actividad para ejecutarse en pantalla completa con orientación horizontal. Se utiliza un diseño llamado activity_control_spider para definir los elementos visuales. Además, se obtiene una instancia del WifiServiceManager para gestionar la conexión al ESP32 mediante WebSocket. Si ya se tiene configurada una dirección IP y un puerto temporal, se intenta establecer la conexión. En caso contrario, se muestra un mensaje al usuario indicando que los datos no están configurados, y la actividad se cierra.

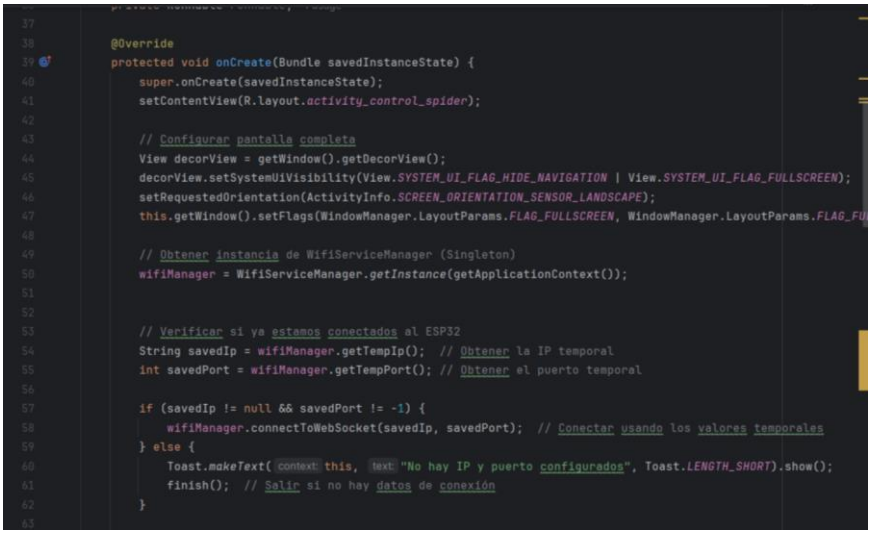
Figura 108.

Clase Control_Spider

```
17  public class Control_Spider extends AppCompatActivity {
18      private ImageView btnVelocidad;  5 usages
19      ImageView btn_arriba;  1 usage
20      ImageView btn_abajo;  1 usage
21      ImageView btn_TurnLeft;  1 usage
22      ImageView btn_TurnRight;  1 usage
23      ImageView btn_hi;  1 usage
24      ImageView btn_ToRight;  1 usage
25      ImageView btn_ToLeft;  1 usage
26      ImageView btn_Zero;  1 usage
27      ImageView btn_Land;  1 usage
28      ImageView btn_Standby;  1 usage
29
30      private ImageView cameraPreview;  no usages
31
32      private int estado = 1; // 0: verde, 1: amarillo, 2: rojo  4 usages
33
34      private WifiServiceManager wifiManager;  7 usages
35      private Handler handler = new Handler();  1 usage
36      private Runnable runnable;  1 usage
37
```

Figura 109.

Método onCreate



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_control_spider);

    // Configurar pantalla completa
    View decorView = getWindow().getDecorView();
    decorView.setSystemUiVisibility(View.SYSTEM_UI_FLAG_HIDE_NAVIGATION | View.SYSTEM_UI_FLAG_FULLSCREEN);
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_SENSOR_LANDSCAPE);
    this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FU

    // Obtener instancia de WifiServiceManager (Singleton)
    wifiManager = WifiServiceManager.getInstance(getApplicationContext());


    // Verificar si ya estamos conectados al ESP32
    String savedIp = wifiManager.getTempIp();  // Obtener la IP temporal
    int savedPort = wifiManager.getTempPort(); // Obtener el puerto temporal

    if (savedIp != null && savedPort != -1) {
        wifiManager.connectToWebSocket(savedIp, savedPort);  // Conectar usando los valores temporales
    } else {
        Toast.makeText( context: this,  text: "No hay IP y puerto configurados", Toast.LENGTH_SHORT).show();
        finish();  // Salir si no hay datos de conexión
    }
```

La actividad incluye una serie de botones que permiten enviar comandos específicos al ESP32 para controlar las acciones del robot. Estos botones están configurados para manejar eventos de toque (TouchListener), permitiendo que el robot responda mientras el botón se mantiene presionado (ACTION_DOWN) y detenga el movimiento al soltar el botón (ACTION_UP).

Los botones configurados incluyen Movimiento Arriba, abajo, giro a la izquierda, giro a la derecha, entre otros como se muestra en la figura 110. Acciones específicas como Hi (HI), aterrizar (LA), estar en modo de espera (SB), entre otros. Cada botón envía un comando predefinido al ESP32 a través del método sendCommand. El botón de velocidad (btnVelocidad) permite alternar entre tres niveles de velocidad: baja, normal y alta. Cada cambio de velocidad también envía un comando correspondiente al ESP32 (1, 2, o 0). El cambio de estado se realiza en un ciclo, donde cada vez que se presiona el botón, la velocidad avanza al siguiente nivel y se actualiza la imagen asociada al botón para reflejar el cambio.

Figura 110.

Botones para configurar las acciones del robot

```
        // Configurar el botón de retroceso pagina
        ImageView imageViewBack = findViewById(R.id.imageView5);
        imageViewBack.setOnClickListener(v -> finish());


        // Configurar botón de velocidad
        btnVelocidad = findViewById(R.id.imgVelocidadVerde);
        btnVelocidad.setOnClickListener(v -> cambiarEstado());



        // Configurar controles de dirección
        configurarBotonDireccion(btn_arriba, R.id.imgArriba,  comando: "UP");
        configurarBotonDireccion(btn_abajo, R.id.imgAbajo,  comando: "D");
        configurarBotonDireccion(btn_ToLeft, R.id.imgToLeft,  comando: "TL");
        configurarBotonDireccion(btn_ToRight, R.id.imgToRight,  comando: "TR");

        configurarBotonDireccion(btn_TurnRight, R.id.imgTurnRight,  comando: "R");
        configurarBotonDireccion(btn_TurnLeft, R.id.imgTurnLeft,  comando: "L");
        configurarBotonDireccion(btn_Zero, R.id.imgZero,  comando: "Z");
        configurarBotonDireccion(btn_hi, R.id.imgHi,  comando: "HI");
        configurarBotonDireccion(btn_Land, R.id.imgLand,  comando: "LA");
        configurarBotonDireccion(btn_Standby, R.id.imgStandby,  comando: "SB");
    }
```

El método configurarbotondireccion como se muestra en la figura 111, permite que ya sea el comando seleccionado de un botón, para evitar enviar repetidas veces un comando, se realizó la lógica, para que, si se mantiene aplastado el botón solo envíe una vez el comando y una vez que se deje de presionar el botón envíe el comando S que es para detener el movimiento.

Figura 111.

Método para configurar botones de dirección

```
    // Método para configurar los botones de dirección
    private void configurarBotonDireccion(ImageView boton, int botonId, String comando) {  10 usages
        boton = findViewById(botonId);
        boton.setOnTouchListener((v, event) -> {
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    // Mostrar un mensaje Toast
                    Toast.makeText( context: Control_Spider.this,  text: "Boton " + comando, Toast.LENGTH_SHORT).show
                    // Comienza a enviar el comando de movimiento
                    sendCommand(comando);
                    return true;
                case MotionEvent.ACTION_UP:
                    // Envia el comando de parada
                    stopSendingCommands();
                    sendCommand( data: "S");
                    return true;
            }
            return false;
        });
    }
```

En el método cambiar estado permite enviar el código de la velocidad que se seleccione, y cada vez que se aplaste vuelve a cambiar, y se repite en ciclo como se observa en la figura 112.

Figura 112.

Método para cambio de velocidades

El método sendCommand utiliza el WifiServiceManager para enviar un comando al ESP32. Antes de enviar el comando, verifica que el WebSocket esté conectado como se puede observar en la figura 113. Si no lo está, se muestra un mensaje de error al usuario.

Figura 113.

Método sendCommand

Cuando la actividad se destruye, se desconecta del WebSocket para liberar recursos y garantizar que no queden conexiones abiertas, esto se puede observar en la figura 114.

Figura 114.

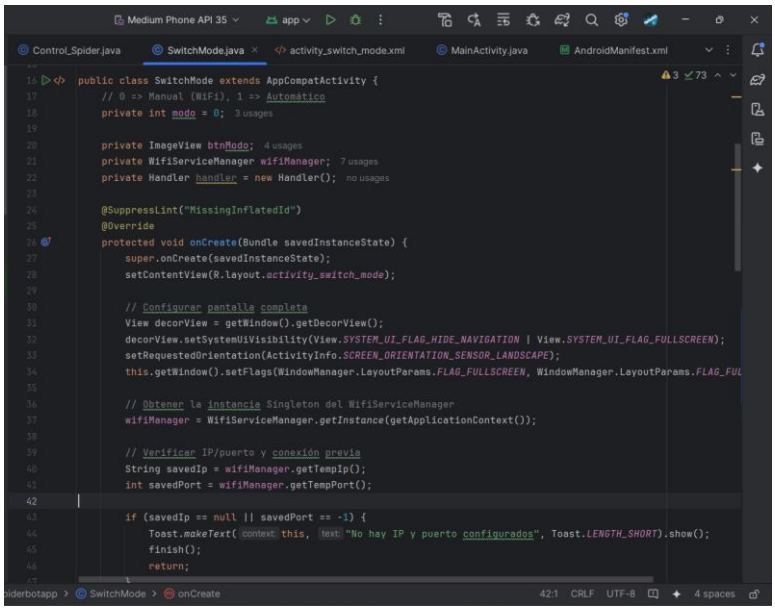
Método onDestroy

- **SwitchMode.**

El código de la clase SwitchMode como se muestra en la figura 115, define una pantalla de configuración de modo (manual o automático) para controlar al robot a través de WebSocket. Al iniciarse, la clase SwitchMode verifica si hay una conexión establecida con el ESP32 (vía WifiServiceManager), y en caso contrario, la inicia. El modo de control se representa por la variable modo (0 para manual, 1 para automático). Al presionar el botón (la imagen btnModo), el método cambiarEstado() alterna entre estos dos modos, actualiza el ícono correspondiente, muestra un Toast con el modo actual y envía el comando "AUTO" o "WIFI" al ESP32. Si se regresa a modo manual, además cierra esta Activity (para retornar a la pantalla anterior). Se evita desconectar el WebSocket en onDestroy() para mantener la sesión activa y permitir que el control continúe funcionando al cambiar de pantallas.

Figura 115.

Código de la clase SwitchMode

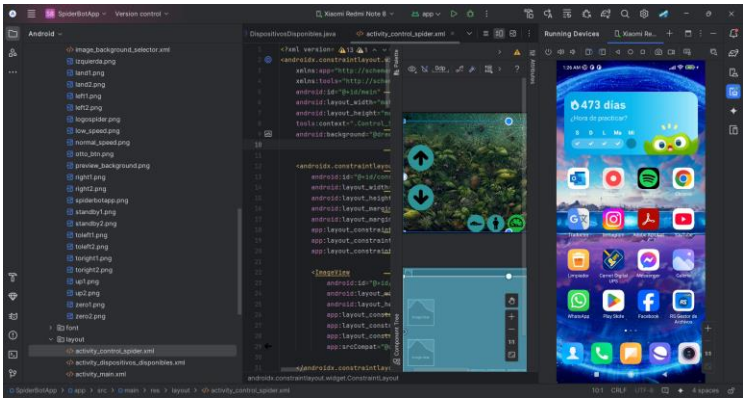


5.4.4 Resultado de la Aplicación Conectado a un Dispositivo Móvil

Para probar la funcionalidad de la aplicación, se utilizó un teléfono que se conectó a la computadora mediante un cable USB. Antes de proceder, se aseguró que en el teléfono estuviera habilitada la opción de depuración USB, una configuración esencial para instalar la aplicación directamente desde el entorno de desarrollo. Este paso inicial está representado en la figura 116.

Figura 116.

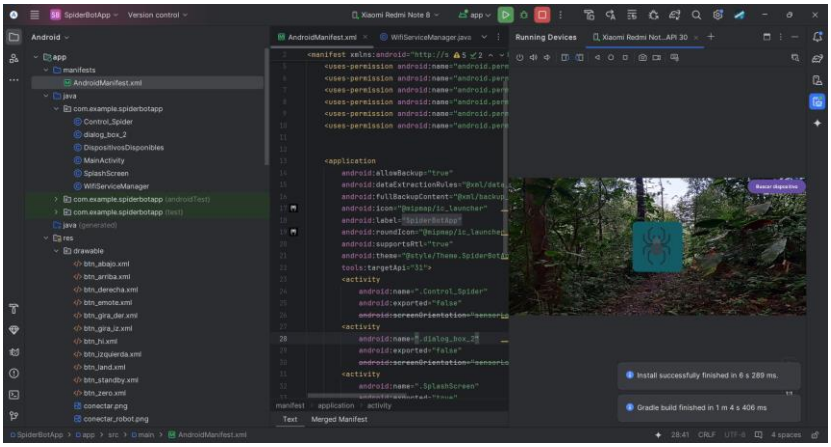
Instalación de la aplicación en el teléfono



Con la aplicación instalada en el dispositivo, se llevó a cabo una verificación de su funcionamiento. Esto se realizó visualizando la pantalla del teléfono desde la computadora, lo que permitió observar la interfaz y las interacciones en tiempo real. Esta etapa de pruebas y validación se ilustra en la figura 117, destacando el correcto despliegue y operatividad de la aplicación.

Figura 117.

Pruebas y validación de la aplicación en el teléfono



5.5 Implementación del Sistema de Visión Artificial

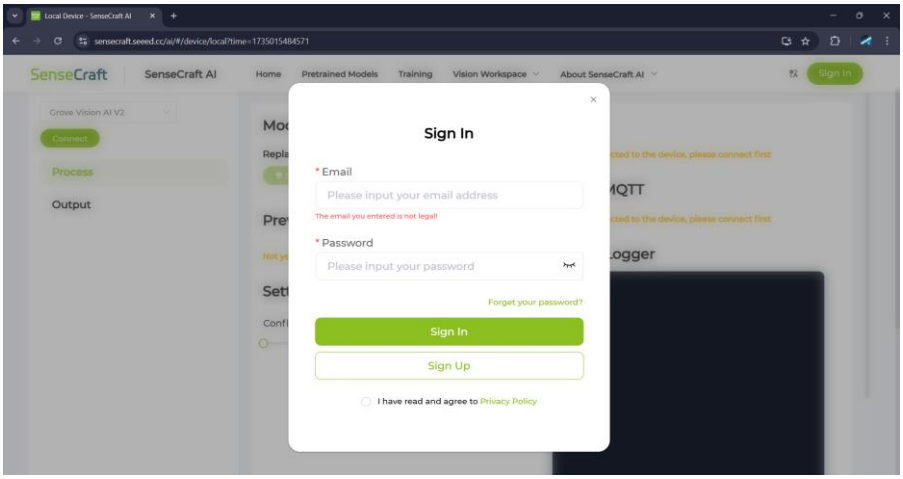
La implementación del sistema de visión artificial es una etapa crucial en el desarrollo del robot cuadrúpedo, ya que dota al sistema de la capacidad de detectar y evitar obstáculos de manera autónoma. Para ello, se utilizó el módulo Grove AI Vision V2, que permite capturar imágenes en tiempo real y procesarlas mediante algoritmos entrenados previamente. En esta fase, se entrenó al modelo para que pueda detectar las piedras, por lo que en la competencia de carrera de insecto el camino suele ser de piedras. Para poder entrenarlo se usó la página de sensecraft.

5.5.1 Crear el Modelo

Para iniciar el proceso de entrenamiento de un modelo, el primer paso consiste en crear una cuenta en la plataforma SenseCraft, como se observa en la figura 118.

Figura 118.

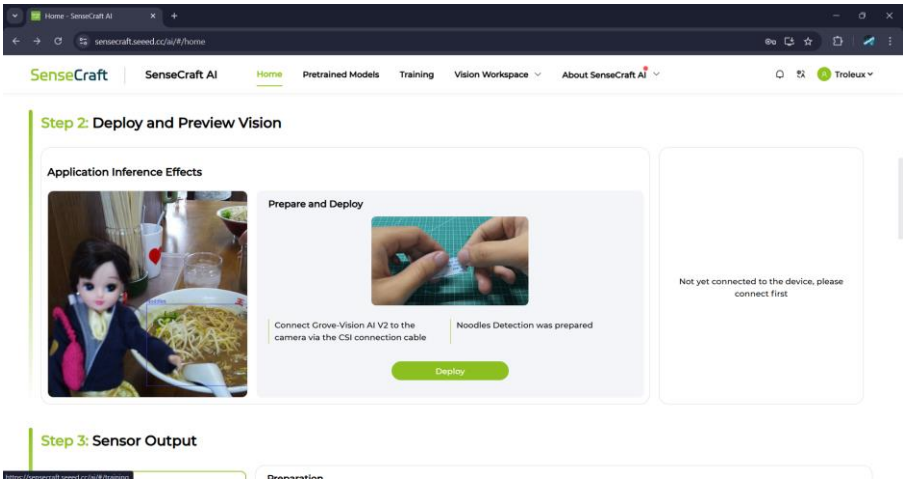
Cuenta en la plataforma SenseCraft



Una vez creada la cuenta, se debe acceder a la opción "Training" para comenzar a configurar y entrenar el modelo. Este paso está ilustrado en la figura 119.

Figura 119.

Opción Training para configurar el modelo



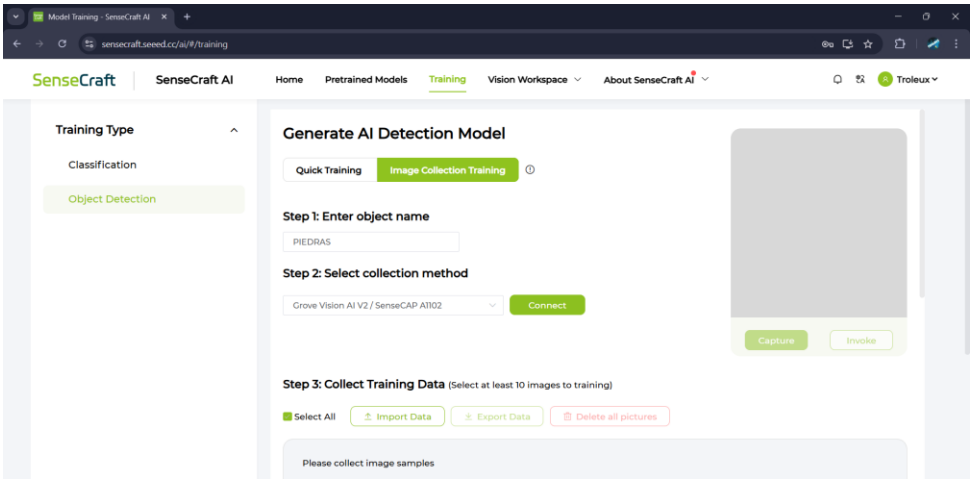
Dentro de la sección de entrenamiento, se selecciona la opción "Object Detection" seguida de "Image Collection Training". Esta función permite recopilar y etiquetar imágenes propias,

proporcionando datos más ricos y variados para el modelo. Usar imágenes reales en diferentes entornos e iluminaciones mejora la precisión del modelo, ya que aprende a identificar obstáculos específicos y reduce falsos positivos o negativos. Por el contrario, la opción "Quick Training" genera modelos rápidamente con información textual, pero carece del nivel de detalle que aportan las imágenes reales, lo que afecta su precisión.

En este proyecto, se nombró al objeto como "Piedras" y se conectó el módulo Grove AI Vision V2 junto con su cámara para capturar las imágenes. Es importante resaltar que las muestras deben capturarse con la misma cámara que se utilizará en el modelo final, ya que una cámara de mayor resolución podría causar errores en la detección debido a diferencias en las características visuales, como se explica en la figura 120.

Figura 120.

Opción "Object Detection" para comenzar a entrenar el modelo



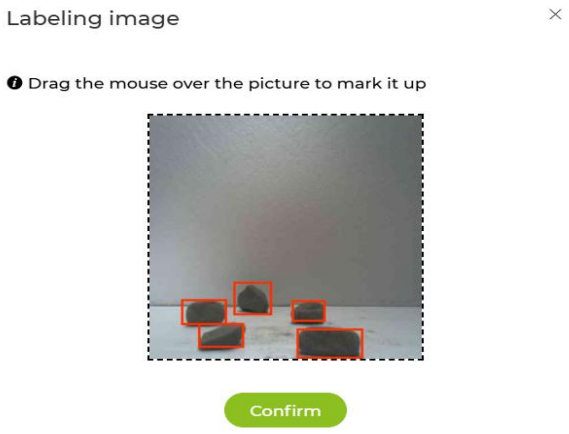
Con la cámara activa, se procedió a capturar muestras iniciales. Dado que la pista en la que operará el robot es blanca, las primeras 100 muestras incluyeron piedras sobre un fondo blanco. En este proceso, es posible seleccionar más de un objeto a detectar en cada imagen, como se muestra en las figuras 121 y 122.

Figura 121.

Creación de escenario para las capturas de imágenes

Figura 122.

Captura de varias imágenes



Posteriormente, se recolectaron otras 200 imágenes con fondos variados, como hojas y ramas, elementos comúnmente presentes en la pista de competencias. Para obtener estas muestras, se utilizó directamente la pista del club de robótica, lo que aseguró un contexto más realista para el modelo. Este enfoque se ilustra en las figuras 123 y 124.

Figura 123.

Toma de imágenes en la pista de competencias del club de robótica

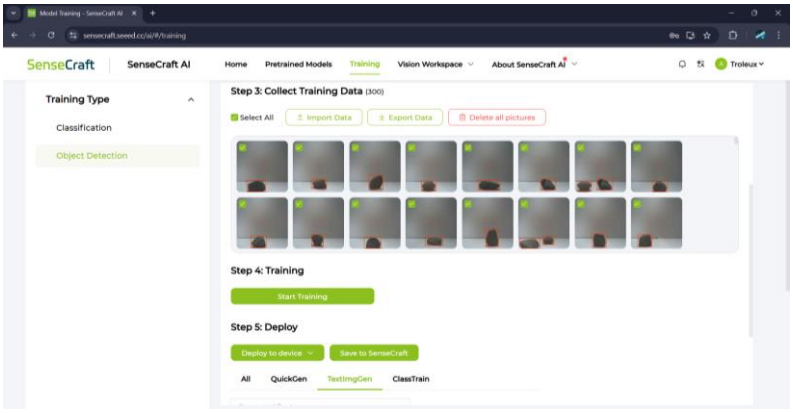
Figura 124.

Uso de la plataforma SenseCraft para captura de imágenes

Una vez que se completó la recolección de muestras, se seleccionó la opción "Start Training" para iniciar el entrenamiento del modelo. Al finalizar, el modelo fue cargado en el módulo Grove AI Vision V2, como se muestra en la figura 125.

Figura 125.

Modelo cargado en el módulo Grove AI Vision V2



5.6 Programación del Microcontrolador

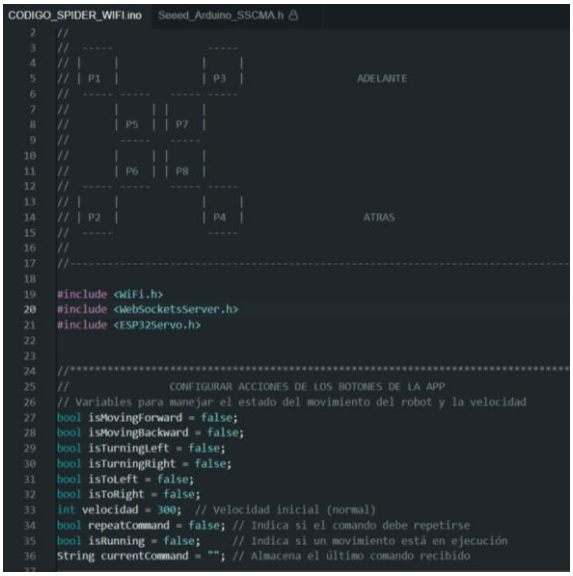
La programación del microcontrolador ESP32 es de gran importancia para el desarrollo del robot cuadrúpedo, ya que coordina todas las funcionalidades y asegura una operación eficiente y sinérgica de sus componentes. En esta etapa, se usó el programa de Arduino IDE para la programación del microcontrolador, donde se implementó la comunicación WiFi, permitiendo que la aplicación móvil controle el robot de manera remota y reciba datos en tiempo real, facilitando una interacción fluida y responsiva. Además, se desarrollaron los algoritmos necesarios para el movimiento de los servomotores, asegurando una locomoción precisa y adaptable a diferentes terrenos mediante comandos de dirección y velocidad. Paralelamente, se estableció la comunicación UART entre el ESP32 y el módulo Grove AI Vision V2, lo que permite la transmisión de datos visuales en tiempo real para la detección y evasión de obstáculos. Esta integración de WiFi, control de servos y comunicación UART garantiza que el microcontrolador gestione de manera eficiente las tareas de procesamiento de datos, toma de decisiones autónomas y recepción de comandos externos, contribuyendo significativamente al rendimiento y autonomía del robot en entornos dinámicos.

5.6.1 Código Robot Controlado Por Wifi

En este código se muestra la programación de un robot cuadrúpedo con servomotores controlados por un ESP32 a través de una conexión Wi-Fi como se muestra en la figura 126. El robot puede ejecutar movimientos como avanzar, retroceder, girar, desplazarse lateralmente, adoptar posturas especiales (por ejemplo, posición inicial, standby o saludo) y responder a diferentes comandos enviados desde una aplicación o interfaz web. Para ello, se integran varias librerías, entre ellas las que habilitan la comunicación Wi-Fi, y el control de servomotores. El código comienza con la inclusión de librerías esenciales para el control inalámbrico (WiFi.h, WebSocketsServer.h) y para el control de servomotores (ESP32Servo.h).

Figura 126.

Código robot controlado por wifi



Además, define un arreglo con 8 servos que representan las “piernas” (o articulaciones) del cuadrúpedo, asignándolos a pines específicos del ESP32. Se establecen variables para almacenar el ángulo actual de cada servo, la calibración de estos y la velocidad global de movimiento. Se crea una variable numberOfAce con número nueve porque será usado más adelante para recorrer

las columnas en una fila, del cual los ocho primeros son movimientos de los servos y el ultimo es un valor en microsegundos que se lo puede observar en la figura 127.

Figura 127.

Creación de variable numberOfAce

```
41  //*************************************************************************************************************
42  //                          Definiciones de los servos
43  const int numberOfServos = 8;
44  Servo servos[numberOfServos];
45  // ---------------P3, P7,P8,P4,P1,P5,P6,P2
46  int servoPins[] = {32,33,26,12,19,18,5,15}; // Pines conectados a los servos
47  int servoCal[numberOfServos] = {0, 0, 0, 0, 0, 0, 0, 0}; // Calibración de servos
48  int currentAngles[numberOfServos] = {90, 90, 90, 90, 90, 90, 90, 90}; // Posiciones iniciales de los servos
49
50  const int numberOfACE = 9; // Programas de movimiento
51
52  // Factor de velocidad (lo cambiarás desde tu app)
53  // 0.5 => la mitad de rápido (más lento)
54  // 1.0 => velocidad normal
55  // 2.0 => el doble de rápido, etc.
56  float globalSpeed = 1.0;
57
58  //*************************************************************************************************************
```

Luego para los movimientos que va a realizar el robot son los arreglos multidimensionales que definen las rutinas de movimiento. Cada rutina se almacena en una matriz (por ejemplo, servoPrg02 para avanzar o servoPrg03 para retroceder), cada fila representa un paso dentro de la secuencia: los primeros ocho valores corresponden a los ángulos deseados para cada servo y el último valor representa el tiempo (en milisegundos) que se mantendrá ese paso. Dichas rutinas permiten que el robot realice distintos “pasos” o “poses” de manera coordinada, logrando un movimiento más o menos fluido según se indique, para que los servos se muevan simultáneamente se los asignó en arrays como se muestra en la figura 128, en donde la variable const int servoXXStep significa el número de arrays que contiene el movimiento, y el const int servoPgrXX es el arreglo del movimiento en el que toma la variable servoXXStep para el numero de filas, y numberOfAce para el numero de columnas. Para esta sección se realizaron once movimientos, como son adelante, atrás, izquierda, derecha, girar izquierda, girar derecha, velocidad, land (tirarse al piso), zero (posición en la que se encuentra de pie), hi (saludo), y standby (pequeño salto).

Figura 128.

Arrays de los movimientos del robot

```
//*************************************************************************************************************
//                                ARRAYS DE LOS MOVIMIENTOS DEL CUADRUPEDO

// Posición  INICIAL (Land) en el piso
const int servoPrg00step = 1;
const int servoPrg00[servoPrg00step][numberOfACE]= {
 //P3, P7, P8, P4, P1, P5, P6,  P2, ms
  {90,90, 90, 90, 90, 90, 90, 90, 400} // Posición inicial para todos los servos
};

// Posición  saludo
const int servoPrg01step = 10;
const int servoPrg01[servoPrg01step][numberOfACE]= {
 //P3, P7, P8, P4, P1, P5, P6,  P2, ms
  {0,135,45, 180, 180, 45, 135, 0, 200},// Posición inicial para todos los servos
  {0,135,45, 180, 180, 45, 115, 0, 200},// Patas traseras casi al piso y p6 ajustar
  {20,135,45, 180, 180, 45, 115, 0, 200},//expandir un poco patas delanteras
  {20,135,45, 180, 90, 20, 115, 0, 200},//alzar una de las patas delanteras hacia alfrente y mover esa pierna al frente
  {20,135,45, 180, 20, 45, 115, 0, 200},//levantar mas la pata y regresar la pierna
  {20,135,45, 180, 20, 20, 115, 0, 200},//llevar la pierna al frente
  {20,135,45, 180, 20, 45, 115, 0, 200},//regresar la pierna
  {20,135,45, 180, 20, 20, 115, 0, 200},//llevar la pierna al frente
  {20,135,45, 180, 180, 45, 115, 0, 200},//regresar la pierna y bajar la pata
  {0,135,45, 180, 180, 45, 135, 0, 200}//levantar patas traseras y quedar en posicion zero
};

// Posición  (ZERO) posicion parado
const int servoPrg04step = 1;
const int servoPrg04[servoPrg04step][numberOfACE]= {
 //P3, P7, P8, P4, P1, P5, P6,  P2, ms
  {0,135,45, 180, 180, 45, 135, 0, 500}, // Posición inicial para todos los servos
};

```

Para ejecutar las rutinas, el código emplea la función runSequenceSmooth() como se muestra en la figura 129. Esta función se encarga de tomar cada uno de los pasos de la matriz (es decir, los ángulos y el tiempo), calcular la diferencia con el ángulo actual y generar pequeños incrementos graduales (interpolaciones) antes de llegar al ángulo deseado como se observa en la figura 130. De esta forma, los servomotores no saltan bruscamente de una posición a otra, sino que se mueven suavemente. Además, runSequenceSmooth() verifica si ya existe un movimiento en ejecución (con la variable isRunning) para evitar choques de secuencias y maneja la repetición de comandos si se desea un desplazamiento continuo (por ejemplo, para caminar hacia adelante indefinidamente).

Figura 129.

Función para ejecutar movimientos suaves + velocidad

```
CODIGO_SPIDER_WIFI.ino   Seeed_Arduino_SSCMA.h
188 //**********************************************************************************************************
189 //                          FUNCION PARA EJECUTAR MOVIMIENTOS suaves + velocidad
190
191 void runSequenceSmooth(const int sequence[][numberOfACE], int steps, int subSteps = 10) {
192   if (isRunning) return; // Salir si ya se está ejecutando un movimiento
193   isRunning = true;      // Marcar que un movimiento está en ejecución
194
195   for (int step = 0; step < steps; step++) {
196     // Lee la posición final deseada para cada servo
197     int finalAngles[numberOfServos];
198     for (int i = 0; i < numberOfServos; i++) {
199       finalAngles[i] = sequence[step][i] + servoCal[i];
200       // Limitar a 0..180
201       if (finalAngles[i] < 0)   finalAngles[i] = 0;
202       if (finalAngles[i] > 180) finalAngles[i] = 180;
203     }
204
205     // Tiempo base (columna 8)
206     int totalTime = sequence[step][8];
207
208     // Ajustamos el tiempo según la velocidad global
209     int adjustedTime = (int)((float)totalTime / globalSpeed);
210
211     // Interpolamos en 'subSteps' partes
212     for (int sStep = 1; sStep <= subSteps; sStep++) {
213       for (int s = 0; s < numberOfServos; s++) {
214         int interpolated = currentAngles[s] +
215           ((finalAngles[s] - currentAngles[s]) * sStep) / subSteps;
216         servos[s].write(interpolated);
217       }
218       // Retraso mínimo de 1 ms
219       int stepDelay = adjustedTime / subSteps;
220       if (stepDelay < 1) stepDelay = 1;
221       delay(stepDelay);
222     }
```

Figura 130.

Código para actualizar el ángulo actual de cada servo

```
224
225       // Actualiza el ángulo actual de cada servo
226       for (int i = 0; i < numberOfServos; i++) {
227         currentAngles[i] = finalAngles[i];
228       }
229     }
230
231     isRunning = false; // Marcar que el movimiento ha terminado
232
233     // Solo reiniciar estados si no se debe repetir el comando
234     if (!repeatCommand) {
235       isMovingForward = false;
236       isMovingBackward = false;
237       isTurningLeft = false;
238       isTurningRight = false;
239       isToLeft = false;
240       isToRight = false;
241     }
242   }
```

El ESP32 se configura como punto de acceso (Access Point), creando una red Wi-Fi propia con un SSID y contraseña definidos como se muestra en la figura 131. Además, se asigna una dirección IP estática, se inicia un servidor WebSocket en el puerto 81.

Figura 131.

Configuración de wifi del ESP32

```
253  //*****************************************************************************************
254  //              CONFIGURACION WIFI ESP32
255  // Definir el SSID y la contraseña de la red Wi-Fi que creará el ESP32
256  const char* ssid = "ESP32-Network";  // Nombre de la red Wi-Fi
257  const char* password = "passwordpassword";  // Contraseña de la red
258
259  // Asignar una dirección IP estática para el ESP32
260  IPAddress local_IP(192, 168, 4, 1);       // Dirección IP del ESP32
261  IPAddress gateway(192, 168, 4, 1);        // Puerta de enlace (igual que la IP en modo AP)
262  IPAddress subnet(255, 255, 255, 0);       // Máscara de subred
263
264  // Crear un servidor WebSocket en el puerto 81
265  WebSocketsServer webSocket = WebSocketsServer(81);
266
267
268  //*****************************************************************************************
```

Luego se definen las funciones de manejo de eventos (webSocketEvent(...)) para recibir y procesar las órdenes enviadas por el usuario o la aplicación como se muestra en la figura 132.

Figura 132.

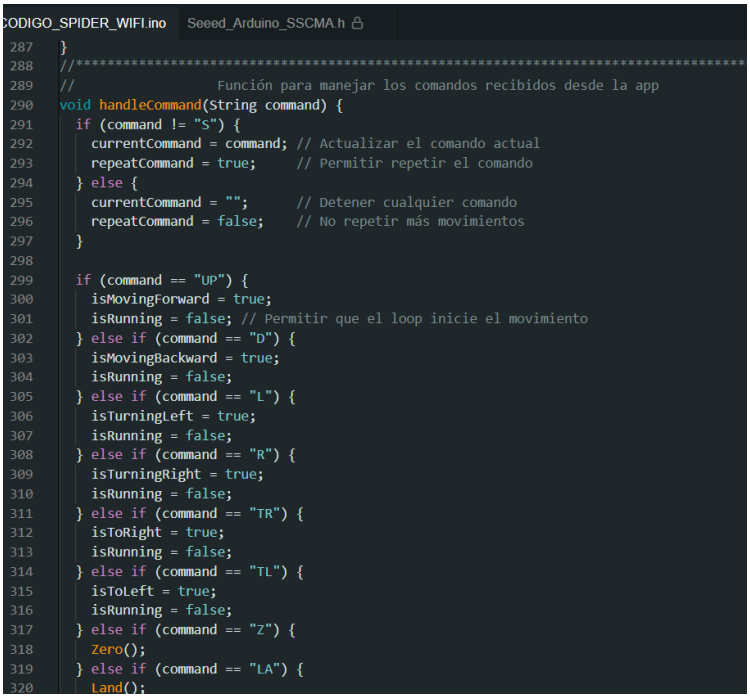
Código manejo de eventos del webSocket

```
//*****************************************************************************
//                                  Manejador de eventos del WebSocket
void webSocketEvent(uint8_t num, WStype_t type, uint8_t *payload, size_t length) {
  switch (type) {
    case WStype_DISCONNECTED:
      Serial.printf("[%u] Desconectado\n", num);
      stopServos();  // Detener los servomotores cuando se desconecta
      break;
    case WStype_CONNECTED:
      {
        IPAddress ip = webSocket.remoteIP(num);
        Serial.printf("[%u] Conectado desde %s\n", num, ip.toString().c_str());
      }
      break;
    case WStype_TEXT:
      Serial.printf("Recibido comando: %s\n", payload);
      handleCommand((char*)payload);  // Manejar el comando recibido
      break;
  }
}
//*****************************************************************************
```

Cuando el ESP32 recibe un mensaje de texto con un comando (por ejemplo, “UP”, “D”, “L”, “R”), se llama a la función handleCommand(...), que activa las banderas de movimiento correspondientes (avanzar, retroceder, girar, etc.) y puede configurar diferentes velocidades como se puede observar en las figuras 133 y 134.

Figura 133.

Código para manejo de comandos recibidos desde la App




```
}
//*****************************************************************************
//                Función para manejar los comandos recibidos desde la app
void handleCommand(String command) {
  if (command != "S") {
    currentCommand = command; // Actualizar el comando actual
    repeatCommand = true;     // Permitir repetir el comando
  } else {
    currentCommand = "";      // Detener cualquier comando
    repeatCommand = false;    // No repetir más movimientos
  }

  if (command == "UP") {
    isMovingForward = true;
    isRunning = false; // Permitir que el loop inicie el movimiento
  } else if (command == "D") {
    isMovingBackward = true;
    isRunning = false;
  } else if (command == "L") {
    isTurningLeft = true;
    isRunning = false;
  } else if (command == "R") {
    isTurningRight = true;
    isRunning = false;
  } else if (command == "TR") {
    isToRight = true;
    isRunning = false;
  } else if (command == "TL") {
    isToLeft = true;
    isRunning = false;
  } else if (command == "Z") {
    Zero();
  } else if (command == "LA") {
    Land();
```

Figura 134.

Código para ajustar la velocidad según el comando recibido

```
//*****************************************************************************
//                Función para ajustar la velocidad según el comando recibido
void setSpeed(String command) {
  if (command == "1") {
    globalSpeed = 0.6;  // Velocidad baja
    Serial.println("Velocidad Baja activada.");
  } else if (command == "2") {
    globalSpeed = 1.0;  // Velocidad normal
    Serial.println("Velocidad Normal activada.");
  } else if (command == "0") {
    globalSpeed = 2.0;   // Velocidad rápida
    Serial.println("Velocidad Rápida activada.");
  }
}
//*****************************************************************************
```

Cada movimiento específico (caminar hacia adelante, girar a la izquierda, saludar, etc.) está encapsulado en una función como se detalla en la figura 135. Por ejemplo, moveForward() llama a runSequenceSmooth(servoPrg02, servoPrg02step, 15); para ejecutar la secuencia de avance. De igual manera, moveBackward(), turnLeft(), turnRight(), Hi(), entre otras, usan matrices diferentes para lograr la pose adecuada. Estas funciones se activan de forma condicional en el

bucle principal (loop()) cuando las banderas de estado (isMovingForward, isTurningLeft, etc.) están en true y no hay ningún otro movimiento ejecutándose.

Figura 135.

Código para movimientos del robot según orden de la App

```
//*****************************************************************************
//                    FUNCIONES MOVIMIENTOS DEL ROBOT SEGUN ORDEN DE LA APP
  void moveForward() {
    Serial.print("Moviendo adelante a velocidad: ");
    Serial.println(velocidad);
    runSequenceSmooth(servoPrg02, servoPrg02step,15);
  }

  void moveBackward() {
    Serial.print("Moviendo hacia atrás a velocidad: ");
    Serial.println(velocidad);
    runSequenceSmooth(servoPrg03, servoPrg03step,15);
  }

  void ToLeft() {
    Serial.print("Ir a la izquierda a velocidad: ");
    Serial.println(velocidad);
    runSequenceSmooth(servoPrg08, servoPrg08step,15);
  }

  void ToRight() {
    Serial.print("Ir a la derecha a velocidad: ");
    Serial.println(velocidad);
    runSequenceSmooth(servoPrg09, servoPrg09step,15);
  }

  void turnLeft() {
    Serial.print("Girando a la izquierda a velocidad: ");
    Serial.println(velocidad);
    runSequenceSmooth(servoPrg07, servoPrg07step,15);
  }
```

La función setup() se ejecuta una sola vez al iniciar el programa en el ESP32 y se encarga principalmente de configurar los periféricos y módulos que el robot utilizará como se detalla en la figura 136. En primer lugar, inicializa la comunicación serial (Serial.begin(115200)) para depuración y luego, realiza la configuración de todos los servomotores (definidos en el arreglo servos[]), estableciendo parámetros de funcionamiento (rango de pulsos, pines a los que están conectados) y fijándolos temporalmente en una posición inicial (por ejemplo, 90 grados).

En esta misma función, se configura el ESP32 como un punto de acceso Wi-Fi (Access Point) utilizando WiFi.softAP(...) y asignándole una IP estática para que los dispositivos puedan conectarse directamente. Después de esto, se inicia el servidor WebSocket en el puerto 81 (webSocket.begin()) y se registra la función de manejo de eventos

webSocket.onEvent(webSocketEvent) para procesar las solicitudes entrantes y desconexiones de los clientes que se conecten al robot. Al finalizar el setup(), el robot ya está listo para recibir y ejecutar los comandos que se le envíen.

Figura 136.

Función setup configura los periféricos y módulos que el robot utilizará

```
void setup() {
  Serial.begin(115200);
  delay(200);

  // Inicializa los servos
  // Adjuntamos cada servo a su pin (para MG90S, 500..2400 es usual)
  for (int i = 0; i < numberOfServos; i++) {
    servos[i].attach(servoPins[i], 500, 2400);
    // Inicialmente en 90° (o lo que gustes)
    servos[i].write(currentAngles[i]);
    delay(100);
  }

  // Configurar el ESP32 en modo Access Point (AP) con IP estática
  Serial.println("Configurando Access Point...");

  // Asignar la IP estática al ESP32
  if (!WiFi.softAPConfig(local_IP, gateway, subnet)) {
    Serial.println("Error al configurar la IP estática");
  }

  // Iniciar el modo AP con el nombre de red y la contraseña
  WiFi.softAP(ssid, password);

  // Mostrar la dirección IP del ESP32
  IPAddress IP = WiFi.softAPIP();
  Serial.print("Dirección IP del ESP32: ");
  Serial.println(IP);

  // Iniciar el servidor WebSocket
  webSocket.begin();
  webSocket.onEvent(webSocketEvent);

  Serial.println("Servidor WebSocket iniciado.");
}
```

La función loop(), por su parte, se ejecuta de manera continua mientras el microcontrolador esté encendido. Dentro de ella, se llama en primer lugar a webSocket.loop(), lo que permite mantener actualizada la comunicación WebSocket y procesar cualquier comando nuevo que llegue desde la aplicación o interfaz de control. A continuación, se evalúa si el robot no está actualmente en medio de una secuencia de movimiento (controlado por la variable isRunning).

Si isRunning es false, significa que el robot está disponible para realizar un nuevo movimiento, así que se revisa cada bandera de estado (como isMovingForward, isTurningLeft, etc.). En caso de que alguna esté en true, se llama a la función correspondiente (por ejemplo, moveForward() o turnLeft()) que, a su vez, ejecutará la secuencia de ángulos almacenada en el

arreglo apropiado (por ejemplo, servoPrg02 para avanzar). Cuando la rutina del movimiento termina, la variable isRunning regresa a false, quedando el robot de nuevo listo para atender más comandos. De este modo, el loop() garantiza un flujo continuo de recepción de órdenes y la ejecución secuencial de las diferentes rutinas de movimiento sin que se solapen ni interfieran entre sí como se detalla en la figura 137.

Figura 137.

Código de la función loop

```
463 void loop() {
464   // Atender conexiones WebSocket
465   webSocket.loop();
466
467   // Procesar movimiento solo si no está en ejecución
468   if (!isRunning) {
469     if (isMovingForward) {
470       moveForward();
471     } else if (isMovingBackward) {
472       moveBackward();
473     } else if (isTurningLeft) {
474       turnLeft();
475     } else if (isTurningRight) {
476       turnRight();
477     } else if (isToRight) {
478       ToRight();
479     } else if (isToLeft) {
480       ToLeft();
481     }
482   }
483 }
```

5.6.2 Código Robot Con Funcionamiento Autónomo

Este código permite que un robot cuadrúpedo funcione de manera autónoma en la competencia de robótica “carrera de insectos” como se muestra en la figura 138. Se basa en un ESP32 que controla ocho servomotores para el desplazamiento y utiliza el módulo Grove AI Vision para detectar obstáculos, específicamente piedras. El programa organiza diferentes rutinas de movimiento (avanzar, retroceder, girar, etc.) y combina esa lógica con la detección de piedras, de modo que el robot pueda esquivarlas automáticamente mientras sigue avanzando en la carrera.

Para este código es similar al código en el anterior punto, con la diferencia que se eliminó todo lo relacionado con el uso de la comunicación wifi del esp32, en cambio se agregó variables relacionadas con el uso del módulo “grove AI visión v2”. Primero se agrega la librería

Seed_Arduino_SSCMA que es la encargada de interactuar con el módulo Grove a través de distintos protocolos de comunicación, en este caso la comunicación UART. En términos generales, se ocupa de Inicializar y configurar el módulo de IA en el modo de comunicación UART, Enviar y recibir comandos (por ejemplo, invoke(), reset(), etc.) que permiten al dispositivo llevar a cabo la inferencia de modelos de IA, extraer datos de detección de objetos, coordenadas de "bounding boxes", clases detectadas y otros resultados, Proveer métodos de alto nivel para que el usuario reciba la información de detección (por ejemplo, boxes(), classes(), points(), etc.) y así procesar eventos como la presencia de un objeto, coordenadas de bounding boxes, puntuaciones de confianza, etc.

Por otro lado, se agregaron dos variables booleanas como son isRunning y repeatCommand que será usada para la lógica del robot para avanzar y evaluar su entorno mientras camina.

Figura 138.

Código para manejar el estado del movimiento y la velocidad el robot

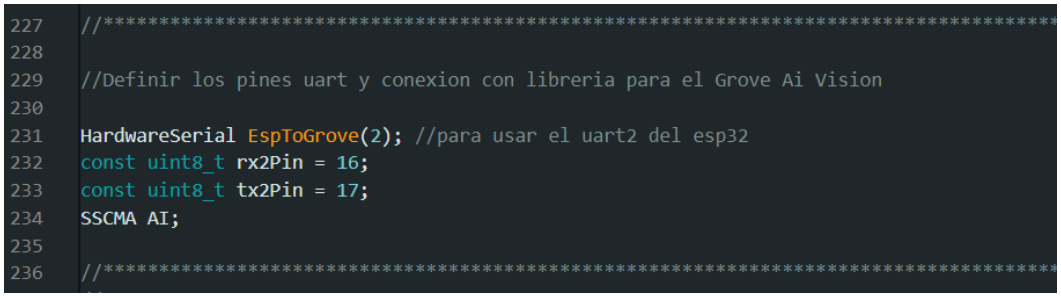
```
#include <HardwareSerial.h>
#include <Seeed_Arduino_SSCMA.h>
#include <ESP32Servo.h>



//*************************************************************************
//                CONFIGURAR ACCIONES DE LOS BOTONES DE LA APP
// Variables para manejar el estado del movimiento del robot y la velocidad
bool isMovingForward = false;
bool isMovingBackward = false;
bool isTurningLeft = false;
bool isTurningRight = false;
bool isToLeft = false;
bool isToRight = false;
int velocidad = 300;  // Velocidad inicial (normal)
bool isRunning        = false; // Indica si runSequenceSmooth está en ejecución
bool repeatCommand    = false; // Para no resetear flags si quisieras repetir

```

Como se mencionó anteriormente, se usó la comunicación UART entre el módulo grove AI visión v2 y el ESP32 principalmente por la velocidad de transferencia de datos como se observa en la figura 139. El Grove AI Vision V2 puede transmitir resultados de detección (coordenadas de

bounding boxes, clases detectadas, etc.) de manera eficiente a través de UART, especialmente si se configura un baud rate alto (por ejemplo, 921600 bps). Esto permite una tasa suficiente para enviar datos de inferencia sin provocar cuellos de botella importantes. Para ello se usan los pines 16 como Rx y el 17 para Tx. También se define la variable AI para usar la librería Seed_Arduino_SSCMA.

Figura 139.

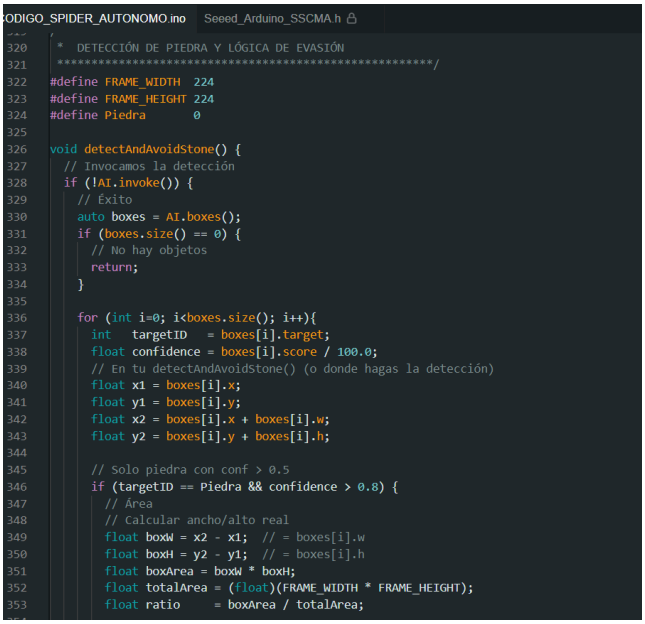
Comunicación uart entre el módulo grove Ai visión v2 y el esp32



En la lógica de evasión de obstáculos, después de iniciar este módulo con AI.begin(...), la función detectAndAvoidStone() invoca rutinariamente a AI.invoke() para obtener información sobre objetos detectados como se muestra en la figura 140. Si se reconoce una “piedra” con el área ocupada en la imagen es significativa (ratio ≥ 0.4), el robot ejecuta maniobras de evasión (girar a la izquierda o derecha, retroceder, etc.) dependiendo de la ubicación del obstáculo en la imagen. Además, cuando el módulo realiza una inferencia, para que pueda realizar una acción primero se puso un filtro para que el objeto si tiene un porcentaje de confiabilidad de más del 80% pueda ser evaluado.

Figura 140.

Código para detección de obstáculos y lógica de evasión



En la función setup() como se muestra en la figura 141, se ejecuta una sola vez al encender o reiniciar el ESP32, primero se configura la comunicación UART entre el ESP32 y el módulo Grove AI Vision. Para ello, se llama a EspToGrove.begin(921600, SERIAL_8N1, rx2Pin, tx2Pin), indicando un baud rate alto (921600), el formato de datos (8 bits, sin paridad, 1 bit de parada) y los pines asignados para la recepción y transmisión (rx2Pin, tx2Pin). A continuación, se invoca AI.begin(&EspToGrove), que inicia el módulo de inteligencia artificial a través del puerto serial recién configurado, dejándolo preparado para procesar inferencias y enviar resultados.

Seguidamente, se establece la comunicación serial principal (Serial.begin(115200)) a 115200 bps, para mostrar información de depuración en el monitor serie. Esto permite que el programador reciba mensajes acerca de los procesos internos y el estado del robot durante la ejecución.

La inicialización de los servomotores se realiza mediante un bucle que itera sobre cada uno de los ocho servos. Dentro de él, se usa attach(...) para vincular cada servo con su respectivo pin, definiendo además el rango mínimo y máximo de pulso (generalmente entre 500 µs y 2400 µs para los servos MG90S). Luego, con servos[i].write(currentAngles[i]), cada servo se mueve a la posición inicial configurada en el arreglo currentAngles (por defecto, 90°). Se agrega un retraso corto (delay(100)) para dar tiempo a los servos a adoptar su posición sin sobresaltos.

Finalmente, se imprime un mensaje en el monitor serie y se llama a la función Zero(), que lleva al robot a una posición conocida (posición "Zero"). Tras un breve retraso (delay(500)), se activa la variable isMovingForward = true, lo que indica que el robot empezará a caminar automáticamente hacia adelante tan pronto como el programa entre en el bucle principal.

Figura 141.

Código de la función setup sirve para preparar la comunicación del ESP32 con el Uart2

```
419  void setup() {
420
421    //preparar esp32 para comunnicacion uart2
422    EspToGrove.begin(921600, SERIAL_8N1, rx2Pin, tx2Pin);
423    delay(200);
424
425    //inicializar el grove
426    AI.begin(&EspToGrove);
427    delay(200);
428
429    Serial.begin(115200);
430    delay(200);
431
432    // Inicializa los servos
433    // Adjuntamos cada servo a su pin (para MG90S, 500..2400 es usual)
434    for (int i = 0; i < numberOfServos; i++) {
435      servos[i].attach(servoPins[i], 500, 2400);
436      // Inicialmente en 90° (o lo que gustes)
437      servos[i].write(currentAngles[i]);
438      delay(100);
439    }
440
441    Serial.println("ESP32 iniciado. Posición Zero...");
442    Zero();  // Llevar al robot a la posición 'Zero'
443    delay(500);
444
445    // **Iniciar la carrera**: el robot comienza a caminar hacia adelante
446    isMovingForward = true;
447    Serial.println("Robot en modo Carrera: moviendo adelante por defecto");
448  }
```

La función loop(), que se ejecuta de manera cíclica e indefinida, inicia llamando a detectAndAvoidStone(). Con ello, se verifica si el módulo Grove AI Vision ha encontrado alguna

piedra en la trayectoria del robot. Si es así, se activa la lógica de evasión, deteniendo o redirigiendo el robot según la ubicación y tamaño de la piedra como se muestra en la figura 142.

Una vez procesada la detección de obstáculos, se revisa si isRunning es falso (es decir, si el robot no está en medio de un movimiento). En ese caso, el programa verifica cuál de las banderas de dirección está activa (isMovingForward, isMovingBackward, etc.) y, según corresponda, invoca la rutina de movimiento adecuada (moveForward(), moveBackward(), turnLeft(), etc.). Esto hace que el robot avance, retroceda o gire en la dirección especificada.

Si no hay ninguna bandera de movimiento activa, el robot permanece en la pose en la que se encuentra, sin ejecutar acción adicional. Finalmente, un delay(100) introduce una ligera pausa antes de repetir el ciclo, evitando que el código se ejecute en bucles demasiado rápidos que podrían saturar el microcontrolador o dificultar la lectura de los datos.

Figura 142.

Código para detectar obstáculos en cada ciclo

```
void loop() {

 // 1) Detectar piedras en cada ciclo (o cada cierto delay).
  detectAndAvoidStone();

  // 2) Si no estamos ejecutando un movimiento (isRunning = false),
  //    podemos activar la acción actual (si la hay).
  if (!isRunning) {
    if (isMovingForward) {
      moveForward();
    }
    else if (isMovingBackward) {
      moveBackward();
    }
    else if (isTurningLeft) {
      turnLeft();
    }
    else if (isTurningRight) {
      turnRight();
    }
    else if (isToLeft) {
      ToLeft();
    }
    else if (isToRight) {
      ToRight();
    }
    // Si no hay flags activos, no hace nada especial
  }

  // Pequeña pausa
  delay(100);
}
```

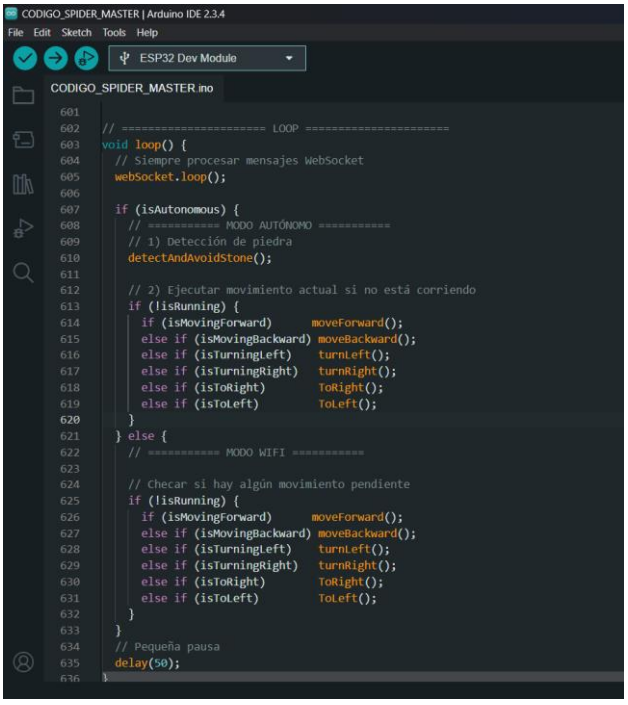
5.6.3 Código Funcionamiento Manual y Autónomo

En esta etapa del proyecto, se desarrolló un código integral que combina las funcionalidades de los modos de funcionamiento manual y autónomo del robot cuadrúpedo en un

solo programa. Este enfoque se adoptó con el propósito de centralizar el control, optimizar los recursos del microcontrolador ESP32 y simplificar tanto la programación como la interacción con el robot. La estructura del código fue diseñada de manera modular, lo que permitió organizar las funciones de cada modo de manera independiente y garantizar su correcta integración. Para lograr esta combinación, se implementó una variable global que actúa como selector dinámico, permitiendo al usuario alternar entre el control manual y autónomo a través de la aplicación móvil de manera eficiente y sin interrupciones como en la figura 143. Este diseño no solo mejora la versatilidad del robot, sino que también reduce la complejidad de mantener dos programas separados, lo cual podría incrementar la posibilidad de errores. Con esta integración, el robot es capaz de adaptarse de forma rápida y sencilla a diferentes escenarios, como el manejo preciso en competencias o el desplazamiento autónomo en terrenos dinámicos, maximizando su funcionalidad y rendimiento.

Figura 143.

Código funcionamiento Manual y Autónomo



VI Análisis de Resultados

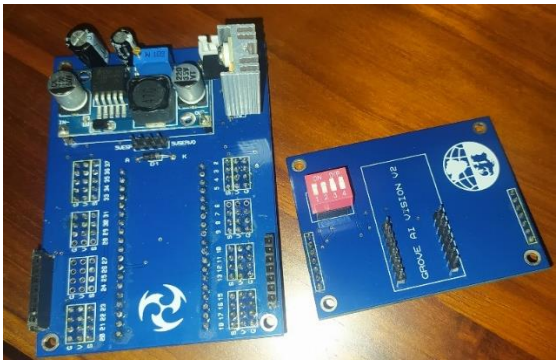
Esta sección aborda la etapa de pruebas y análisis de la simulación del sistema se enfoca en evaluar el desempeño del robot cuadrúpedo en relación con los objetivos planteados durante su diseño y desarrollo. A través de las pruebas realizadas, tanto en el modo de control manual como en el modo autónomo, se obtuvieron datos clave sobre la efectividad del sistema de comunicación wifi, la precisión del movimiento de los servomotores y la capacidad del módulo de visión artificial para detectar y evitar obstáculos. Además, se analizaron los ajustes implementados en la estructura física, el circuito electrónico y la programación, con el fin de optimizar la estabilidad y funcionalidad del robot en diferentes escenarios. Este análisis permite identificar las fortalezas y limitaciones del sistema, destacando las áreas en las que el proyecto cumplió sus objetivos y aquellas que ofrecen oportunidades para futuras mejoras.

6.1 Resultados del Sistema Electrónico

Una vez completado el soldado, se llevaron a cabo pruebas de continuidad y funcionamiento. Estas pruebas permitieron verificar que todas las conexiones fueran precisas y que los componentes electrónicos operaran según lo esperado, asegurando la funcionalidad integral del diseño. El resultado final fue una tarjeta electrónica compacta, robusta y funcional, capaz de satisfacer las exigencias del sistema como se puede observar en la figura 144.

Figura 144.

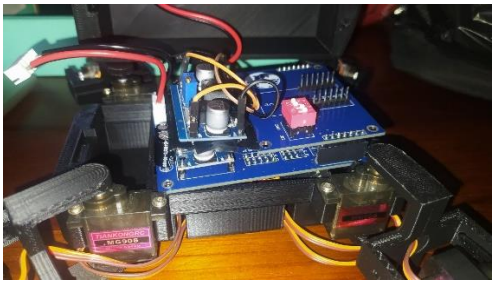
Tarjeta electrónica con los elementos soldados



Sin embargo, se tuvo que realizar un cambio de regulador que estaba destinado a alimentar el ESP32 y al grove AI visión v2, ya que el L7805CV a pesar de tener un disipador de calor, el microcontrolador al usar la función de wifi consume más miliamperios lo que provoca que tenga que disipar como calor la energía excedente que resulta de la diferencia entre el voltaje de entrada y el voltaje de salida, lo que pudo provocar daños en los componentes ya que la tarjeta estará en un espacio cerrado con poca ventilación, como solución se lo reemplazo con otro modulo regulador LM2596 en la que se solucionó el problema del calor como se observa en la figura 145.

Figura 145.

Tarjeta electrónica con regulador LM2596



6.2 Resultados de las Piezas en 3D

Con las piezas impresas listas se procedió a ensamblar las partes y unirlas con la tarjeta electrónica previamente diseñada y fabricada. Durante el ensamblaje, se verificó la alineación

precisa de las piezas para garantizar la estabilidad estructural del robot y la correcta integración de los componentes electrónicos. Se utilizaron tornillos y otros elementos de fijación para unir las piezas de manera segura, asegurando la rigidez necesaria para soportar los movimientos del robot durante las pruebas. Además, se integraron los servomotores en sus respectivas posiciones, conectándolos al circuito principal. Finalmente, se realizó una revisión general del ensamblaje para corroborar que no existieran piezas mal ajustadas o cables expuestos que pudieran comprometer el funcionamiento del sistema. El resultado fue un robot completamente ensamblado como se observa en la figura 146, listo para las etapas de pruebas y validación.

Figura 146.

Robot cuadrúpedo completamente ensamblado

Por otro lado, la dimensión final del cuadrúpedo está al borde del límite máximo permitido que es de 20 cm de largo x 20 cm de ancho, teniendo todas las patas recogidas, como se muestra en la figura 147 donde se usó una caja con las medidas mencionadas para su comprobación.

Figura 147.

Caja de 20x20 cm

6.3 Prueba Control del Robot Cuadrúpedo Desde la Aplicación

En esta prueba se evaluó el desempeño del robot cuadrúpedo bajo el modo de control manual, utilizando la aplicación móvil desarrollada en Android Studio como se muestra en la figura 148. La comunicación entre la aplicación móvil y el ESP32 fue fluida, permitiendo un control eficiente de los movimientos básicos como avanzar, retroceder, girar y detenerse. Los resultados confirmaron una buena comunicación entre la App y el robot, además de un control estable y eficiente en diversos tipos de terreno, validando el correcto funcionamiento del sistema en este modo de operación para detectar con facilidad los obstáculos entrenados como lo es para este robot las piedras.

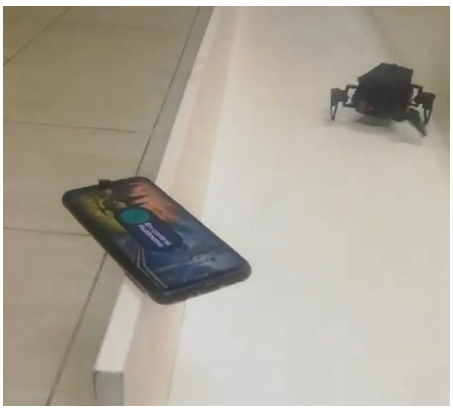
Figura 148.

Prueba control del robot cuadrúpedo desde la aplicación

También se comprobó que funcionamiento cuando se realiza el cambio de modo manual a autónomo y viceversa, como se muestra en la figura 149. Durante la prueba, el robot funcionó aproximadamente 30 minutos, y después de ese tiempo, la batería (LiPo) aún tenía un 60% de carga. Esto sugiere que el robot podría operar alrededor de 1 hora y 15 minutos (aproximadamente) antes de agotar por completo la batería.

Figura 149.

Prueba modo autónomo en la pista sin obstáculos.



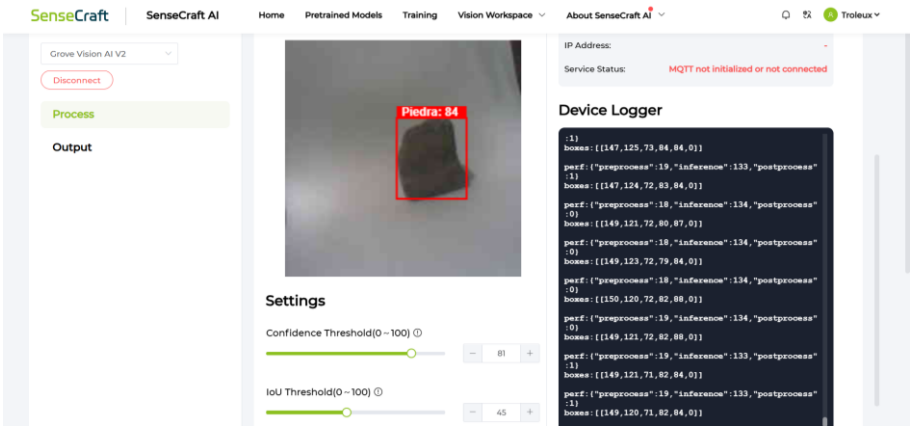
6.4 Resultados del Modelo de Visión Artificial

El modelo de visión artificial entrenado e implementado en el módulo Grove AI Vision V2 fue diseñado para detectar de manera eficiente objetos como piedras de diversos tamaños y formas, replicando las condiciones reales de las competencias de robótica. Durante las pruebas realizadas, el sistema demostró un alto nivel de precisión, incluso en escenarios desafiantes con variaciones de iluminación y fondos complejos, como se aprecia en la figura 150.

Una de las principales capacidades del modelo es su habilidad para asignar una etiqueta específica a los objetos detectados. En este caso, las piedras detectadas son clasificadas como "número 0" en la información de salida generada por el sistema. Este dato se convierte en un punto clave para que el ESP32 pueda procesar la información y ejecutar la lógica correspondiente para tomar decisiones basadas en los resultados obtenidos. La robustez de esta configuración asegura que el robot pueda responder de manera precisa y eficiente a los obstáculos detectados, integrando las capacidades del modelo con la funcionalidad general del sistema.

Figura 150.

Resultados del modelo de visión artificial

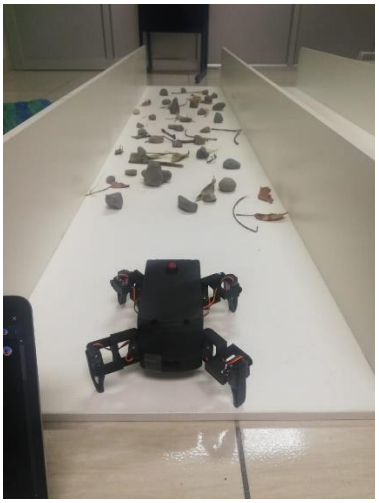


6.5 Resultados en Entorno Simulado de Competencia

En esta prueba se evaluó la capacidad del robot cuadrúpedo para operar de forma autónoma mediante el sistema de visión artificial implementado con el módulo Grove AI Vision V2. El objetivo fue comprobar la efectividad del algoritmo entrenado para detectar obstáculos, calcular trayectorias y realizar maniobras evasivas en tiempo real. Durante las pruebas, se colocaron diferentes objetos simulando obstáculos en el camino del robot como se muestra en la figura 151.

Figura 151.

Simulación de obstáculos en el camino del robot



Durante las pruebas realizadas, en el caso de la pista con los obstáculos de piedras, ramas, y hojas, el robot en su modo autónomo usando su visión artificial fue capaz de diferenciar las piedras de los otros objetos durante su trayecto por el camino. Por otro lado, se pudo comprobar que las piedras pequeñas las ignora ya que no resulta un gran obstáculo en su camino siendo capaz de pasarlas por encima de ellas, sin embargo, ciertas piedras pequeñas o medianas que a simple vista podía caminar encima de ellas, debido a la forma de la piedra hubo veces que se quedaba atorado incapaz de continuar su camino.

Además, se realizó pruebas en su modo autónomo con velocidad normal y rápida, se pudo comprobar que en la velocidad normal en un inicio el robot caminaba hacia adelante pero cada vez se iba a la izquierda, con la corrección realizada el robot pudo caminar hacia adelante sin que se pegara más a la izquierda o a la derecha, sin embargo, en su velocidad rápida, el robot al realizar movimientos rápidos provocaba que no podía mantenerse caminando hacia delante de forma recta, ya que se inclinaba más a la derecha y terminaba chocando con la pared a la mitad del camino o a un cuarto de ella como se muestra en la figura 152; como solución se modificó el código, es decir, que el robot tendrá dos movimientos diferentes dependiendo de la velocidad en la que esté, con velocidad normal se mantiene el movimiento inicial para adelante, pero con velocidad rápida usará otro movimiento en la que intercalará cada ciertos pasos dos funciones diferentes, una que al caminar se incline a la derecha y otra que se incline a la izquierda, de esta forma compensando su desvío en el camino, dando una ilusión como si fuera hacia adelante en zigzag.

Figura 152.

Modo autónomo en velocidad rápida

Por otro lado, en la acción que realizaba después de detectar un objeto mediano o grande en su camino, hubo problemas ya que no realizaba la acción a tiempo y se iba de largo sin

esquivarlo, además de que en los casos que si lograba realizar su acción después de detectar una piedra su movimiento era ineficiente, es decir, que al realizar su movimiento para esquivar tenía algunos movimientos que tomaba algo de tiempo en terminar de ejecutarla, aproximadamente unos 40 segundos ,o que la acción no lograba esquivar la piedra hasta el punto de quedar atascado; pero una solución al problema fue modificar el código que realizaba la acción después de detectar una piedra, el cambio realizado fue que ahora si detecta una piedra mediana o grande, realice unos cuantos pasos hacia adelante y dependiendo de qué lado de su visión esta más la piedra, el robot gira y se lanza al piso y realiza un barrido con su pata empujando las piedras al frente suyo y despejando su camino, luego se levanta y gira al sentido contrario que al principio en posición recta y continua hacia adelante, este cambio logró que el robot fuera capaz de mover los obstáculos cercanos frente suyo para evitar quedar atascado, y por otro lado la simplificación de la acción cuando detecta un obstáculo se redujo pasos y tiempo se redujo a 18 segundos.

Se registraron las decisiones tomadas por el sistema para evitar colisiones mientras mantenía un desplazamiento estable. Los resultados demostraron un alto grado de precisión en la detección de obstáculos y una respuesta eficiente en la ejecución de maniobras, validando la integración del sistema de visión artificial y su capacidad para adaptarse a entornos dinámicos. Las pruebas realizadas en torno al tiempo del robot con obstáculo y sin obstáculos están detalladas en la tabla 1.

Tabla 1.

Pruebas del robot con obstáculos y sin obstáculos

Prueba	Control	Velocidad	Espacio recorrido (metros)	Tiempo sin obstáculos	Tiempo con obstáculos
1	Manual	Normal	2.44	1:22	3:10
2	Manual	Rápido	2.44	0:59	2:10
3	Autónomo	Normal	2.44	1:19	2:46
4	Autónomo	Rápido	2.44	0:57	1:49

VII Presupuesto

Teniendo en cuenta que el proyecto se desarrollará como un prototipo dentro del presupuesto establecido, se han considerado tanto las horas de trabajo de ingeniería destinadas al diseño e implementación del equipo como los recursos necesarios para completar el proyecto. Estos valores, detallados en la tabla 2, ascienden a un estimado de 312.50.

Tabla 2.

Tabla del presupuesto del proyecto de Titulación

Cantidad	Materiales	Valor unitario	Valor total
1	ESP32 Wifi/Bluetooth	$9.00	$9.00
1	Regulador de voltaje Lm2596	$7.00	$7.00
8	Servomotor Tower Pro MG90S	$3.00	$24.00
1	Batería de Lipo	$15.71	$15.71
1	Cargador de batería De Lipo	$ 13.99	$13.99
1	Cable plano flexible	$ 6.49	$6.49
1	Switch On/Off	$ 0.50	$ 0.50
1	Módulo Vision AI V2	$ 25.00	$ 25.00
1	Módulo de cámara OV5647-62	$13.00	$ 13.00
20	Pin macho de doble fila	$0.45	$ 9.00

10	Pin hembra de una fila	$0.60	$ 6.00
1	Crimpadora	$16.89	$ 16.89
1	Kit de conectores JST-XH	$7.58	$ 7.58
1	Diseño de tarjeta ESP32	$2.00	$ 2.00
1	Diseño de tarjeta Grove	$4.00	$ 4.00
30	Horas de Ingeniería	$2.87	$86.1
1	Gastos de importación	$66.24	$66.24
	VALOR TOTAL		**$312.50**

VIII Cronograma

En la figura 153, se presenta el diagrama de flujo del desarrollo del proyecto, mientras que en la figura 154 se muestra el cronograma en el cual se dividen las actividades a desarrollar para la ejecución del proyecto.

Figura 153.

Cronograma de actividades para realizar el proyecto.

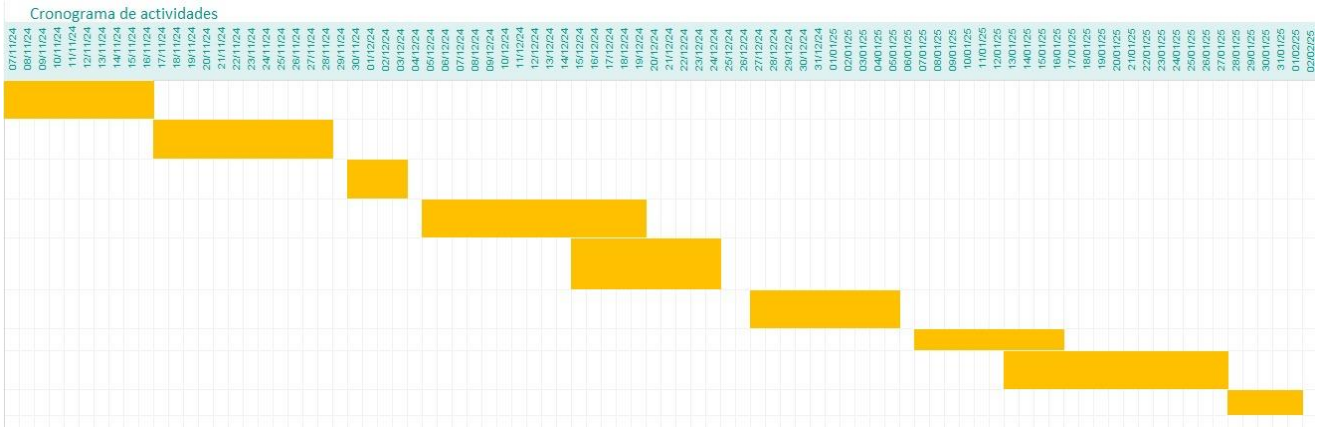


Figura 154.

Descripción de las actividades realizadas en el proyecto.

N°	Descripción de la etapa	Duración de la etapa (días)	Tarea dependiente	Tipo de Dependencia	Días de dependencia	Comienzo	Fin	Responsable	Estatus	Fecha de finalización	Días que efectivamente llevó la etapa
1	Propuesta del tema y redacción del anteproyecto del trabajo de titulación	10	No Aplica	No Aplica	+0	07/11/24	16/11/24	Moya / Zambran	Completado	16/11/24	10
2	Revisiones con el tutor y aprobación del tema de titulación.	12	1	CC	+10	17/11/24	28/11/24	Moya / Zambrano	Completado	28/11/24	12
3	Diseño de la tarjeta eléctronica	4	2	FC	+2	30/11/24	03/12/24	Moya / Zambrano	Completado	05/12/24	6
4	Diseño en 3D del robot y desarrollo de aplicación móvil.	15	3	FC	+2	05/12/24	19/12/24	Moya / Zambrano	Completado	14/12/24	10
5	Programación del microcontrolador y entrenamiento del módulo de la cámara.	10	4	CC	+10	15/12/24	24/12/24	Moya / Zambrano	Completado	23/12/24	9
6	Pruebas del funcionamiento de la aplicación móvil con el	10	5	FC	+3	27/12/24	05/01/25	Moya / Zambrano	Completado	06/01/25	11
7	Correción de errores	10	6	FC	+2	07/01/25	16/01/25	Moya / Zambrano	Completado	17/01/25	11
8	Entrega del avance del documento al tutor (revisón 1, 2, 3, 4).	15	7	CC	+6	13/01/25	27/01/25	Moya / Zambrano	Completado	27/01/25	15
9	Finalización del documento de titulación.	5	8	FC	+1	28/01/25	01/02/25	Moya / Zambrano	Completado	01/02/25	5

IX Conclusiones

El presente trabajo de titulación del robot cuadrúpedo para competencias demostró la viabilidad de integrar sistemas de control manual y autónomo a través de una aplicación móvil. A lo largo del proyecto, se logró diseñar e implementar un prototipo funcional que combina capacidades avanzadas de locomoción y estabilidad, destacando la eficiencia de los servomotores y la adaptabilidad del sistema en terrenos irregulares. Este trabajo contribuye significativamente al ámbito de la robótica educativa, brindando una plataforma práctica que fomenta la investigación y el aprendizaje en áreas como la inteligencia artificial y el diseño de sistemas autónomos.

El sistema de visión artificial utilizado en el proyecto estuvo basado en el módulo Grove AI Vision V2, el cual trabajó en conjunto con una cámara OV5647-62 y el microcontrolador ESP32, es aquí donde, este sistema permitió la detección y clasificación de obstáculos en tiempo real, contribuyendo a la navegación autónoma del robot. Durante las pruebas, se evaluó su precisión en la identificación de piedras en distintos entornos, logrando una detección efectiva en la mayoría de los casos. Sin embargo, se evidenció que el sistema tenía dificultades en la identificación de ciertos objetos con formas irregulares y que, en condiciones de iluminación adversa, su rendimiento se ve afectado.

En cuanto al desempeño del hardware y software, se verificó que la comunicación entre el microcontrolador ESP32 y la aplicación móvil fue estable, permitiendo un control fluido y preciso del robot. Las pruebas realizadas en entornos simulados y reales confirmaron la efectividad de los algoritmos de locomoción y visión artificial. Sin embargo, se observó que el robot presentaba dificultades en el mantenimiento de una trayectoria recta en velocidades altas, lo que llevó a

implementar un algoritmo de corrección basado en movimientos alternos de balanceo para compensar la desviación.

Finalmente, las pruebas realizadas permitieron validar que el robot es capaz de operar tanto en modo manual como autónomo con un rendimiento satisfactorio en términos de estabilidad, detección de obstáculos y navegación en terrenos diversos. No obstante, se identificaron áreas de mejora, como la optimización de la respuesta del sistema de visión artificial en tiempo real y el desempeño en competencias.

X Recomendaciones

En futuros desarrollos del proyecto, se recomienda mejorar la capacidad de adaptación del robot en entornos dinámicos, se sugiere incorporar un sistema de locomoción adaptativo basado en sensores como un giroscopio, que permita ajustar en tiempo real el equilibrio del robot y el movimiento de las patas según las condiciones del terreno. Este sistema ayudaría a evitar caídas, mejorar la estabilidad en superficies irregulares y aumentar la precisión de los movimientos, lo que otorgaría una ventaja competitiva en pruebas de velocidad y obstáculos.

Se sugiere también implementar algoritmos más avanzados de aprendizaje automático y visión artificial, como redes neuronales convolucionales, que permitan una detección más precisa de obstáculos y la ejecución de tareas autónomas más complejas, es así que, estas mejoras podrían hacer que el robot sea más adaptable a entornos dinámicos y desafiantes, fortaleciendo su desempeño en escenarios reales.

Adicionalmente, se recomienda la implementación de redes neuronales convolucionales (CNN) para mejorar la precisión en la detección de obstáculos y permitir la clasificación de múltiples tipos de objetos en el entorno del robot. La inclusión de técnicas avanzadas de aprendizaje automático podría hacer que el robot sea más adaptable a entornos dinámicos y con cambios imprevistos en la pista de competencia.

Para validar y optimizar el desempeño del robot en escenarios competitivos, se recomienda realizar pruebas en entornos similares a los de competencias internacionales. Esto permitirá detectar posibles limitaciones del sistema y realizar los ajustes necesarios para mejorar su rendimiento en condiciones reales. Documentar estos ensayos servirá como referencia para futuras mejoras y desarrollos dentro de la Universidad Politécnica Salesiana.

XI Bibliografía

Aritex Cading S.A.U. (2022). *ATX Robotics*. Obtenido de ATX Robotics: https://atx-robotics.com/noticias/desafios-vision-artificial-industria/

ABB. (2022). *new.abb.com.* Obtenido de https://search.abb.com/library/Download.aspx?DocumentID=3HAC030005-001&LanguageCode=en&DocumentPartId=&Action=Launch

ABB. (27 de 07 de 2022). *new.abb.com*. Obtenido de Manual del usuario de RobotStudio: https://new.abb.com/news/es/detail/93548/abb-lanza-el-robot-delta-de-cinco-ejes-mas-rapido-de-su-clase-para-la-recogida-el-embalaje-y-la-reorientacion-de-productos-ligeros

ABB Group. (s.f.). *Manual del operador*. Recuperado el 10 de Junio de 2024, de Manual del operador: https://library.e.abb.com/public/6aeb483836740e11c1257b4b0052375b/3HAC032104-005_revE_es.pdf

ABB. (s.f.). *new.abb.com*. Obtenido de https://new.abb.com/products/robotics/es/robotstudio

Aguirre aguirre, A. W. (2021). *dspace.ups.edu.ec.* Obtenido de https://dspace.ups.edu.ec/bitstream/123456789/21543/1/MSQ247.pdf

AMAZON. (7 de Agosto de 2017). *AMAZON*. Obtenido de AMAZON: https://www.amazon.com/-/es/850mAh-Bater%C3%ADa-Multirotor-Tama%C3%B1o-Quadcopter/dp/B074MG6YGS/ref=sr_1_15?__mk_es_US=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=3N4RM9NS81CWZ&dib=eyJ2IjoiMSJ9.jDcVUcpwbeYxBi6Ko-OFPhuys5_rd31YfZS2hnq0bcscR3yr6strSqEMrd5CUUpkyIeu

Anibalismo. (19 de Mayo de 2016). *WordPress*. Obtenido de WordPress: https://anibalismo.wordpress.com/2016/05/19/arduino-y-enc28j60-en-proteus-isis-con-foticos/

aprendiendoarduino. (11 de Diciembre de 2016). *WordPress*. Obtenido de WordPress: https://aprendiendoarduino.wordpress.com/2016/12/11/ide-arduino/

Arduino. (16 de Febrero de 2019). *Arduino*. Obtenido de Arduino: https://arduino.cl/programacion/

Autodesk. (24 de Mayo de 2024). *Autodesk*. Obtenido de Autodesk: https://www.autodesk.com/es/products/inventor/overview

Avtek. (2 de Noviembre de 2022). *Avtek*. Obtenido de Avtek: https://www.avtek.com/pag/reguladores-de-voltaje

Balseca, J., & Chusin, B. (2022). IMPLEMENTACIÓN DE UN ROBOT ANTROPOMÓRFICO PARA LA CLASIFICACIÓN DE OBJETOS MEDIANTE VISIÓN ARTIFICIAL. *(Tesis de Ingenieria)*. UNIVERSIDAD TÉCNICA DE COTOPAXI, Cotopaxi. Obtenido de https://repositorio.utc.edu.ec/bitstream/27000/10356/1/PI-002434.pdf

Be Tech! with Santander. (22 de mayo de 2023). *Medium*. Obtenido de Medium: https://medium.com/be-tech-with-santander/la-visi%C3%B3n-artificial-y-el-reconocimiento-de-im%C3%A1genes-procesamiento-automatizado-f94cf30ece54

Bonilla Yoza, M. M., Cevallos Pin, G. S., Zambrano Zambrano, S. M., & Marcillo Merino, M. J. (2022). Uso de la inteligencia artificial en los dispositivos móviles. *Revista UNESUM-Ciencias*.

Casero, A. (15 de MARZO de 2024). *KEEPCODING*. Obtenido de https://keepcoding.io/blog/procesamiento-de-imagenes-con-matrices/

Castañeda, C. A. (13 de Febrero de 2018). *El Tiempo*. Obtenido de El Tiempo: https://www.eltiempo.com/tecnosfera/novedades-tecnologia/boston-dynamics-presento-un-perro-robot-que-abre-puertas-182150

CITRIC. (12 de Diciembre de 2023). *SEEED STUDIO*. Obtenido de SEEED STUDIO: https://wiki.seeedstudio.com/grove_vision_ai_v2a/?utm_source=chatgpt.com

COGNEX. (18 de NOVIEMBRE de 2019). *COGNEX.COM*. Obtenido de https://support.cognex.com/docs/is_580/web/es/ezb/Content/EasyBuilder/EasyBuilderLite/Communication_RSLogixv20_is2k.htm?TocPath=Comunicaci%C3%B3n%7CComunicaciones%20con%20PLC%20de%20Rockwell%20ControlLogix%7CConfiguraci%C3%B3n%20de%20sistemas%20de%20visi%

COGNEX. (2024). *IN-SIGHT DOCUMENTACIÓN*. Obtenido de IN-SIGHT DOCUMENTACIÓN: https://support.cognex.com/es-es/documentation/in-sight

Cognex Corporation. (2023). *In-Sight Explorer User Manual* . Obtenido de https://www.cognex.com/

Copol. (1 de Julio de 2022). *Copol*. Obtenido de Copol: https://www.copol.edu.ec/pdf/steel-challenge/es/08_Carrera_Insectos.pdf

Definicion.de. (17 de Noviembre de 2016). *Definicion.de*. Obtenido de Definicion.de: https://definicion.de/robotica/

Developers. (s.f.). *Developer*. Obtenido de Developer: https://developer.android.com/studio?gad_source=1&gclid=Cj0KCQjwrKu2BhDkARIsAD7GBov0fUhbvFz3QHcOFtNtqPy64CHxstMomxAl_sAdwMcdmgV-R7Q8EMMaAhGXEALw_wcB&gclsrc=aw.ds&hl=es-419

Developers, E. d. (12 de Enero de 2024). *Android Developers*. Obtenido de Android Developers: https://developer.android.com/codelabs/basic-android-kotlin-compose-install-android-studio?hl=es-419

EdgeImpulse. (22 de Mayo de 2024). *EdgeImpulse*. Obtenido de EdgeImpulse: https://docs.edgeimpulse.com/docs/concepts/edge-ai/what-is-edge-impulse

EDS Robotics. (05 de 05 de 2020). *edsrobotics*. Obtenido de https://www.edsrobotics.com/blog/sistemas-de-vision-artificial-tipos-aplicaciones/

Eduardo. (28 de Febrero de 2019). *vermabaterias*. Obtenido de vermabaterias: https://vermabaterias.com/baterias-lipo-que-son/?srsltid=AfmBOook7ZCWe2LkUn18SLFiy31d46pWFggq5FCWTXVYrsDwUKw4L_wu

Egasen. (13 de Octubre de 2021). *Egasen*. Obtenido de Egasen: https://www.egasen.com/es/blog/noticias/que-es-servomotor-para-que-se-utiliza

E-Marmolejo, D. R. (11 de Julio de 2021). *hetpro-store*. Obtenido de hetpro-store: https://hetpro-store.com/TUTORIALES/microcontrolador/

Enerxia. (15 de Septiembre de 2014). *Enerxia*. Obtenido de Enerxia: https://www.enerxia.net/portal/index.php?option=com_content&view=article&id=406:electronica-proteus-simulador-digital-y-analogico&catid=61&Itemid=142

ESIC . (08 de 2018). *esic.edu*. Obtenido de https://www.esic.edu/rethink/tecnologia/automatizacion-industrial-que-es-presente-y-futuro

Gonzalez, R., & Woods, R. (2018). *Procesamiento digital de Imagenes.* New York: Pearson.

Grupo BCNVISION. (18 de 8 de 2023). *bcnvision.es*. Obtenido de https://bcnvision.es/blog-vision-artificial/la-revolucion-de-la-automatizacion-industrial-gracias-a-la-vision-artificial/

Hugging Face. (s.f.). *Hugging Face*. Recuperado el 15 de mayo de 2024, de Hugging Face: https://huggingface.co/learn/computer-vision-course/en/unit1/image_and_imaging/imaging

IBM. (s.f.). *¿Qué es la visión artificial?* Recuperado el 15 de maryo de 2024, de ¿Qué es la visión artificial?: https://www.ibm.com/es-es/topics/computer-vision#citation5

Instrumentación, A. e. (08 de Abril de 2022). *Automática e Instrumentación* . Obtenido de Automática e Instrumentación : https://www.automaticaeinstrumentacion.com/texto-diario/mostrar/3544151/duda-papel-robotica-dinamica-cuadrupeda-tiene-papel-muy-importante-industria-40-futuro

International Society of Automation. (s.f.). *isa.org*. Recuperado el 07 de 2024, de https://www.isa.org/about-isa/what-is-automation

Ismael, A. R. (12 de marzo de 2022). *Reconocimiento de patrones e inteligencia artificial*. Obtenido de Reconocimiento de patrones e inteligencia artificial: https://revistaecys.github.io/17Edicion/articulo12.html

jason-workshop. (12 de Agosto de 2018). *WikiFactory*. Obtenido de WikiFactory: https://wikifactory.com/@jason-workshop/q1-mini

Larzabal, I. A. (2015). *Análisis de las capacidades del software de simulación robótica RobotStudio.* Recuperado el 23 de 7 de 2024, de https://addi.ehu.es/handle/10810/16491

Lis data solutions. (s.f.). *Visión artificial en producción*. Recuperado el 15 de mayo de 2024, de Visión artificial en producción: https://www.lisdatasolutions.com/es/blog/vision-artificial-en-produccion/

London, U. C. (1 de Enero de 2022). *University College London*. Obtenido de https://www.ucl.ac.uk/robotics/research-projects/2022/oct/robohike-autonomous-quadrupedal-robot-navigation-and-hiking-challenging

López, A. (29 de 10 de 2015). *voltium*. Obtenido de https://www.voltimum.es/articulos-tecnicos/robotica-industrial-sector

Manosalvas, C. (2023). IMPLEMENTACIÓN DE UN PROTOTIPO PARA LA CLASIFICACIÓN AUTOMÁTICA DE TOMATES RIÑÓN BASADO EN LA NORMA INEN 1745, APLICANDO TÉCNICAS DE VISIÓN ARTIFICIAL. *(Tesis de Ingenieria).* UNIVERSIDAD NACIONAL DE CHIMBORAZO, Chimborazo.

MAPIR. (4 de Abril de 2014). *uma divulga*. Obtenido de uma divulga: https://www.umadivulga.uma.es/noticias/tecnologia/una-aplicacion-de-control-de-robots-que-supera-los-retardos-en-la-conexion-a-internet/

Marta. (4 de Diciembre de 2023). *3Dnatives*. Obtenido de 3Dnatives: https://www.3dnatives.com/es/que-es-autodesk-inventor-130620222/

Pardo, L. (23 de Mayo de 2018). *neoteo*. Obtenido de neoteo: https://www.neoteo.com/como-funcionan-los-servomotores/

Pascual, C. (29 de Enero de 2022). *programarfacil*. Obtenido de programarfacil: https://programarfacil.com/esp32/esp32-cam/

Piedra, M. (25 de Septiembre de 2020). *Youtube*. Obtenido de Youtube: https://www.youtube.com/watch?v=YstrnFudzT8

Repsol. (2024). *Repsol.com*. Obtenido de https://www.repsol.com/es/energia-futuro/tecnologia-innovacion/inteligencia-artificial/index.cshtml

Robotnik. (15 de Enero de 2024). *Robotnik*. Obtenido de Robotnik: https://robotnik.eu/es/tendencias-en-robotica-2024-para-que-se-usaran-los-robots/

Robotnik. (2024). *robotnik.eu*. Obtenido de https://robotnik.eu/

Rosario, E. D. (15 de Diciembre de 2018). *Blog Espol*. Obtenido de Blog Espol: http://blog.espol.edu.ec/girni/ide-arduino-con-esp32/

Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach.* Pearson.

Salesiana, R. I. (2024). *Universidad Politécnica Salesiana*. Obtenido de https://dspace.ups.edu.ec/simple-search?query=robot+cuadrupedo+

Sapia. (08 de 08 de 2021). *Sapia.com*. Obtenido de https://www.sapia.com.pe/ia/que-son-las-redes-neuronales/

STUDIO, S. (15 de MAYO de 2024). *AMAZON*. Obtenido de AMAZON : https://www.amazon.com/Grove-Cortex-M55-Ethos-U55-TensorFlow-Compatible/dp/B0D2LC7R4K?th=1

STUDIO, S. (28 de NOVIEMBRE de 2024). *SEEED STUDIO*. Obtenido de https://wiki.seeedstudio.com/sensecraft_ai_overview/?utm_source=chatgpt.com

Teleamazonas. (25 de Octubre de 2024). *Teleamazonas*. Obtenido de Teleamazonas: https://www.teleamazonas.com/funciona-optimus-robot-humanoide-tesla/

TL Dev Tech. (15 de agosto de 2021). *TL Dev Tech*. Obtenido de TL Dev Tech: https://www.tldevtech.com/what-is-image-acquisition-in-image-processing/

Todomicro. (23 de Diciembre de 2022). *Todomicro*. Obtenido de Todomicro: https://www.todomicro.com.ar/step-down/204-regulador-de-voltaje-step-down-lm2596s-13v-35v.html

UPS, N. (05 de JUNIO de 2024). *UNIVERSIDAD POLITECNICA SALESIANA* . Obtenido de https://www.ups.edu.ec/noticias?articleId=2088255&byid

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

SALESIANOS DON BOSCO

Guayaquil, 31 de enero del 2025

Ing. Orlando Barcia, Msc.

Director de Carrera de Electrónica y Automatización.

De mis consideraciones:

Yo, Rafael Christian Franco Reina, portador de la cédula de ciudadanía No. 0923328629 tutor de trabajo de titulación **"DISEÑO E IMPLEMENTACIÓN DE UN ROBOT CUADRÚPEDO PARA COMPETENCIAS CON CONTROL AUTÓNOMO O MANUAL MEDIANTE UNA APLICACIÓN MÓVIL"**, informo la calificación al Trabajo de Tituación de los estudiantes de la Malla Ajuste: ISRAEL MARCELO MOYA HERRERA, ARLENA OMAYRA ZAMBRANO CUADRO

Criterio	Descripción del criterio	Puntaje	Observaciones
Planteamiento e identificación del problema	Se muestra la importancia del problema y la contribución que se quiere alcanzar con el Proyecto técnico.	15	
Revisión del marco teórico y fuentes de información	Este criterio estable la relación entre la revisión literaria y el problema a abordar en el Proyecto técnico, así como el adecuado nivel de exhaustividad en la revisión de las fuentes de información.	15	
Contenido Metodológico	Se establecen con claridad y de manera estructurada las distintas fases, uso de métodos, herramientas, diseños, recursos, materiales, etc, para el desarrollo del Proyecto técnico y la propuesta de solución.	20	
Funcionalidad	Permite evaluar el nivel de funcionalidad del trabajo desarrollado, tomando en cuenta los objetivos del mismo.	30	
Presentación de Resultados	Se expresan o presentan los resultados alcanzados en el desarrollo del proyecto técnico y cómo se relacionan con el cumplimiento de los objetivos, el impacto y la innovación.	15	
Conclusiones Recomendaciones	Este criterio establece la claridad con que el autor expone su posición y sus ideas respecto a las conclusiones y recomendaciones expresadas.	5	
	PUNTAJE FINAL:	**100**	

Por la atención que se sirva dar a la presente, quedo de usted muy agradecido.

Atentamente,

Ing. Rafael Franco Reina, MSc.

Docente Tutor.