UNIVERSIDAD NACIONAL DE CHIMBORAZO
FACULTAD DE INGENIERÍA
CARRERA DE TELECOMUNICACIONES

Sistema de navegación en interiores para asistir en la movilidad de personas con discapacidad que utilizan sillas de ruedas mediante tecnología LIFI y Machine Learning.

Trabajo de Titulación para optar al título de:
INGENIERA EN TELECOMUNICACIONES

Autor:
Vasco Viteri Dagmar Mishelle.

Tutor:

PhD. Leonardo Fabián Rentería Bustamante

Riobamba, Ecuador. 2025

DECLARATORIA DE AUTORÍA

Yo, **Dagmar Mishelle Vasco Viteri**, con cédula de ciudadanía **1600955593**, autora del trabajo de investigación titulado: **SISTEMA DE NAVEGACIÓN EN INTERIORES PARA ASISTIR EN LA MOVILIDAD DE PERSONAS CON DISCAPACIDAD QUE UTILIZAN SILLAS DE RUEDAS MEDIANTE TECNOLOGÍA LIFI Y MACHINE LEARNING**, certifico que la producción, ideas, opiniones, criterios, contenidos y conclusiones expuestas son de mí exclusiva responsabilidad.

Asimismo, cedo a la Universidad Nacional de Chimborazo, en forma no exclusiva, los derechos para su uso, comunicación pública, distribución, divulgación y/o reproducción total o parcial, por medio físico o digital; en esta cesión se entiende que el cesionario no podrá obtener beneficios económicos. La posible reclamación de terceros respecto de los derechos de autor (a) de la obra referida, será de mi entera responsabilidad; librando a la Universidad Nacional de Chimborazo de posibles obligaciones.

En Riobamba, 18 de Junio de 2025

Dagmar Mishelle Vasco Viteri

C.I:1600955593

DICTAMEN FAVORABLE DEL PROFESOR TUTOR

Quien suscribe, **Leonardo Fabián Rentería Bustamante** catedrático adscrito a la Facultad de Ingeniería, por medio del presente documento certifico haber asesorado y revisado el desarrollo del trabajo de investigación "**Sistema de navegación en interiores para asistir en la movilidad de personas con discapacidad que utilizan sillas de ruedas mediante tecnología LIFI y Machine Learning.**", bajo la autoría de **Dagmar Mishelle Vasco Viteri**; por lo que se autoriza ejecutar los trámites legales para su sustentación.

Es todo cuanto informar en honor a la verdad; en Riobamba, a los 15 días del mes de mayo de 2025

Phd. Leonardo Fabián Rentería Bustamante.
C.I:1104064132

CERTIFICADO DE LOS MIEMBROS DEL TRIBUNAL

Quienes suscribimos, catedráticos designados Miembros del Tribunal de Grado para la evaluación del trabajo de investigación **SISTEMA DE NAVEGACIÓN EN INTERIORES PARA ASISTIR EN LA MOVILIDAD DE PERSONAS CON DISCAPACIDAD QUE UTILIZAN SILLAS DE RUEDAS MEDIANTE TECNOLOGÍA LIFI Y MACHINE LEARNING,** presentado por **VASCO VITERI DAGMAR MISHELLE,** con cédula de identidad número **1600955593**, bajo la tutoría de PhD. Leonardo Fabián Rentería Bustamante; certificamos que recomendamos la APROBACIÓN de este con fines de titulación. Previamente se ha evaluado el trabajo de investigación y escuchada la sustentación por parte de su autor; no teniendo más nada que observar.

De conformidad a la normativa aplicable firmamos, en Riobamba 18 de Junio de 2025.

PhD. Luis Tello
PRESIDENTE DEL TRIBUNAL DE GRADO

PhD. Fernando Escudero
MIEMBRO DEL TRIBUNAL DE GRADO

Dr. Klever Torres
MIEMBRO DEL TRIBUNAL DE GRADO

C E R T I F I C A C I Ó N

Que, **VASCO VITERI DAGMAR MISHELLE** con CC:**1600955593**, estudiante de la Carrera **TELECOMUNICACIONES**, **VIGENTE,** Facultad de **Ingeniería**; ha trabajado bajo mi tutoría el trabajo de investigación titulado " **SISTEMA DE NAVEGACIÓN EN INTERIORES PARA ASISTIR EN LA MOVILIDAD DE PERSONAS CON DISCAPACIDAD QUE UTILIZAN SILLAS DE RUEDAS MEDIANTE TECNOLOGÍA LIFI Y MACHINE LEARNING**", cumple con el **7 %**, de acuerdo al reporte del sistema Anti plagio **COMPILATION**, porcentaje aceptado de acuerdo a la reglamentación institucional, por consiguiente autorizo continuar con el proceso.

Riobamba, 10 de junio de 2025

Firmado electrónicamente por:
LEONARDO FABIAN
RENTERIA BUSTAMANTE
Validar únicamente con FirmaEC

_____
PhD. Leonardo Fabian Rentería Bustamante
TUTOR(A) TRABAJO DE INVESTIGACIÓN

DEDICATORIA

A mi madre, por ser el alma que me sostiene, por su amor infinito, sus palabras de aliento en los momentos más duros y por enseñarme, con su ejemplo, que la fuerza verdadera nace del corazón.

A mi padre, por su sabiduría, su esfuerzo incansable y su confianza en mí incluso cuando yo dudaba. Gracias por ser guía, apoyo y ejemplo de perseverancia.

A mi hermana y mi hermano, compañeros de vida, por su cariño incondicional, por hacerme reír cuando más lo necesitaba y por estar siempre, sin condiciones ni preguntas, a mi lado.

A mi mejor amigo, por ser refugio, por sus consejos honestos, por su lealtad sin medida y por no soltarme nunca, incluso cuando el camino se volvió cuesta arriba.

Este trabajo, más que un logro académico, es un reflejo del amor, el apoyo y la fuerza que cada uno de ustedes me dio. A ustedes les dedico este esfuerzo, con todo mi corazón.

Mishelle Vasco

AGRADECIMIENTO

Agradezco primeramente a Dios, por darme la vida, la fuerza y la claridad necesarias para llegar hasta aquí. Su presencia me acompañó en cada paso, dándome paz en los momentos de incertidumbre y empuje cuando más lo necesité.

Agradezco profundamente a mi madre Flor y a mi padre Germán, pilares fundamentales en mi vida, por su amor incondicional, su esfuerzo constante y por enseñarme el valor de la perseverancia.
A mi hermana Samantha y a mi hermano Klever, por su compañía, comprensión y apoyo en cada etapa de este proceso.

A mi mejor amigo Kevin, por estar siempre presente, por sus palabras de aliento y por ser un sostén emocional en los momentos de mayor presión. A Miguel, por su apoyo constante, por creer en mí incluso en los momentos más difíciles, y por brindarme ánimo y tranquilidad cuando más lo necesitaba.

A mis tíos, quienes, con sus consejos, gestos de cariño y confianza, también han sido parte importante de este camino.

Y a todos mis profesores, por su guía, su paciencia y su compromiso con mi formación. Gracias por compartir su conocimiento y motivarme a superar cada desafío académico.
A todos ustedes, mi gratitud sincera.

Mishelle Vasco

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

En el presente trabajo de investigación se aborda la problemática de la movilidad de personas con discapacidad que utilizan sillas de ruedas como medio de transporte en entornos interiores. Se diseñó e implementó un sistema de navegación para interiores aplicado a una silla de ruedas en un ambiente controlado, utilizando algoritmos de aprendizaje automático. Además, se desarrolló un dispositivo LiFi con el propósito de asistir en la localización de la silla. El proyecto se llevó a cabo en tres fases. En la primera, se seleccionaron los algoritmos, arquitecturas y dispositivos más adecuados. Además, se hizo un análisis de los algoritmos y las condiciones que los afectarían cuando se integren al dispositivo final. En la segunda fase, se ensambló el dispositivo con una Raspberry Pi Zero W, dos placas basadas en el microcontrolador Atmega328p y la electrónica necesaria para el funcionamiento del sistema de navegación con visión artificial, redes neuronales convolucionales, Q-Learning y tecnología LiFi. En la tercera fase, se realizaron pruebas funcionales del sistema en un entorno controlado bajo dos configuraciones: una con desplazamiento aleatorio y otra con Machine Learning. Con los resultados obtenidos se pudo demostrar que la combinación LiFi con Machine Learning mejora el tiempo de traslado de la silla de un lugar a otro reduciendo de 242,76 segundos a 78,26 segundos respecto a la combinación LiFi con el algoritmo aleatorio.

Palabras claves: Redes neuronales convolucionales, visión artificial, LiFi, aprendizaje automático, Q-Learning, silla de ruedas.

ABSTRACT

This research project addresses the issue of mobility for people with disabilities who use wheelchairs as a means of transportation in indoor environments. An indoor navigation system was designed and implemented for a wheelchair in a controlled setting, using machine learning algorithms. Additionally, a LiFi device was developed to assist in locating the wheelchair. The project was carried out in three phases. In the first phase, the most suitable algorithms, architectures, and devices were selected. An analysis was also conducted on the algorithms and the conditions that could affect them when integrated into the final system. In the second phase, the device was assembled using a Raspberry Pi Zero W, two boards based on the Atmega328p microcontroller, and the necessary electronics for operating the navigation system based on computer vision, convolutional neural networks, Q-Learning, and LiFi technology. In the third phase, functional tests were conducted in a controlled environment under two configurations: one with random movement and the other using machine learning. The results demonstrated that combining LiFi with machine learning improves the wheelchair's travel time from one location to another, reducing it from 242.76 seconds to 78.26 seconds compared to the LiFi and random algorithm combination.

Keywords: Convolutional Neural Networks, Computer Vision, LiFi, Machine Learning, Q-Learning, wheelchair.

Reviewed by: Alison Tamara Varela Puente

ID: 0606093904

CAPÍTULO I.

1.1 Introducción

La libertad de movimiento constituye un pilar fundamental en la calidad de vida, ya que permite a las personas desenvolverse con autonomía e independencia en su entorno. Este aspecto cobra aún mayor relevancia en el caso de quienes presentan dificultades de movilidad, pues su capacidad para desplazarse de manera segura y eficiente influye directamente en su bienestar físico, emocional y social. Facilitar el acceso a tecnologías de asistencia y entornos accesibles es, por tanto, esencial para garantizar una vida digna e inclusiva para este grupo de la población.

En las últimas décadas, se han desarrollado diversos dispositivos de asistencia orientados a mejorar la movilidad, entre ellos las sillas de ruedas motorizadas y los sistemas de localización basados en sensores [1]. Sin embargo, la mayoría de los sistemas actuales presentan limitaciones significativas: no son rentables ni convenientes para los usuarios, y en algunos casos requieren monitoreo constante, lo que restringe su usabilidad y limita la posibilidad de una operación autónoma satisfactoria [2].

En este contexto, la presente investigación se orienta a explorar nuevas posibilidades tecnológicas que permitan la movilidad en silla de ruedas en entornos interiores [3]. Hasta donde alcanza nuestro conocimiento, las tecnologías de Aprendizaje Automático (Machine Learning) y LiFi (Light Fidelity) han sido exploradas de forma separada en estudios generales, pero no existen suficientes investigaciones que analicen su aplicación conjunta en escenarios cotidianos para la navegación en sillas de ruedas motorizadas [4], particularmente en entornos cerrados y desde una perspectiva centrada en el usuario con discapacidad [5].

Con el objetivo de mitigar esta brecha, esta tesis propone el desarrollo de un sistema de navegación que integra un modelo de Machine Learning con módulos de comunicación basados en tecnología LiFi. Para ello, se llevó a cabo un proceso metodológico en tres etapas: la primera etapa: se realizó la arquitectura de dispositivo, en la segunda etapa se implementó un sistema de navegación utilizando Machine Learning y LiFi y la última se realizaron las evaluaciones de rendimiento del sistema en un entorno controlado que simula condiciones reales de uso.

1.2 Planteamiento del problema

Según el Consejo Nacional para la Igualdad de Discapacidades (CONADIS), el 45,66 % de las personas con discapacidad en el país equivalente a 215.156 individuos presentan discapacidad física, mientras que 108957 tienen discapacidades cognitivas. En total, se contabilizan 324.113 con algún tipo de discapacidad. Adicionalmente, 69.959 presentan un grado de discapacidad entre el 50 % y el 74 %, 25.853 entre el 75 % y el 84 %, y 13.396 tienen un grado de discapacidad entre el 85 % y el 100 % [6].

Estas cifras evidencian una realidad crítica: una parte considerable de esta población depende de terceros para realizar actividades cotidianas, como desplazarse de un lugar a otro. Esta situación limita su autonomía y afecta directamente su calidad de vida [7]. La necesidad de soluciones tecnológicas que favorezcan su movilidad y asistencia personal es, por tanto, urgente y prioritaria.

En este contexto, tecnologías como LiFi **y** Machine Learning se presentan como alternativas. LiFi permite la transferencia de datos a velocidades superiores a 100 veces las del WiFi, y ofrece redes inalámbricas más seguras y estables [8]. Sin embargo, su efectividad se ve limitada por factores como su corto alcance, la exposición a perturbaciones ambientales y restricciones en espacios abiertos.

De manera complementaria, el Aprendizaje Automático (Machine Learning) permite el desarrollo de modelos capaces de tomar decisiones o realizar predicciones sin programar reglas explícitas. Cuanta más información recibe, mejor es su desempeño. No obstante, esta tecnología enfrenta desafíos como el alto consumo de recursos computacionales, la dificultad para adaptarse a entornos dinámicos y la necesidad de grandes volúmenes de datos para su entrenamiento.

Frente a este escenario, se plantea la necesidad de diseñar un sistema de navegación autónoma para sillas de ruedas, basado en el uso combinado de tecnologías LiFi y Machine Learning. Esta integración permitirá explorar nuevas formas de asistencia en los procesos de movilidad de personas con discapacidad, considerando criterios de eficiencia, economía y accesibilidad tecnológica.

1.3 Justificación

En ambientes cerrados como centros comerciales, hospitales y edificios, la precisión en la localización de sistemas de movilidad humana suele verse afectada por las limitaciones inherentes de ciertos tipos de señales de comunicación. En este contexto, la combinación de tecnología LiFi que utiliza luz para transmitir datos y algoritmos de Machine Learning se plantea como una posibilidad para desarrollar sistemas de navegación en interiores aplicados a sillas de ruedas, dado que ambas tecnologías podrían aportar características complementarias frente a las limitaciones de métodos convencionales como Bluetooth, WiFi o GPS, los cuales presentan restricciones en cuanto a precisión e interferencias electromagnéticas.

Actualmente, la autonomía en la movilidad representa uno de los principales retos para las personas con discapacidad, especialmente en espacios cerrados no adaptados. Las soluciones convencionales suelen ser insuficientes o no escalables, ya que no contemplan la interacción entre múltiples variables del entorno, ni permiten una navegación autónoma sin asistencia constante. Por ello, resulta necesario investigar y evaluar nuevas alternativas que permitan avanzar hacia sistemas más precisos y adaptables.

El objetivo primordial de esta investigación es implementar un sistema de navegación en interiores para personas con discapacidad que utilizan sillas de ruedas, con el fin de facilitar sus desplazamientos, aumentar su autonomía, y promover condiciones inclusivas que contribuyan a una mejora en su calidad de vida. Este trabajo busca aportar evidencia experimental sobre la viabilidad técnica de integrar tecnologías emergentes como LiFi y Machine Learning en el desarrollo de soluciones aplicadas a la movilidad asistida en entornos controlados.

1.4 OBJETIVOS

1.4.1 General

Implementar un sistema de navegación en interiores para asistir en la movilidad de personas con discapacidad que utilizan sillas de ruedas mediante tecnología LiFi y Machine Learning.

1.4.2 Específicos

- Diseñar el sistema de navegación usando LiFi y Machine Learning.
- Implementar el sistema de navegación en una silla de ruedas y realizar pruebas de funcionamiento.
- Evaluar el funcionamiento del sistema en un ambiente interior controlado a través de pruebas de campo con personas con multidiscapacidad física.

CAPÍTULO II.

2.1 Estado del arte

En los últimos años, particularmente desde 2018, el desarrollo de sistemas de asistencia para personas con movilidad reducida ha cobrado una creciente importancia. Este avance ha sido impulsado por el progreso en tecnologías de localización, sensores inteligentes y computación ubicua. En este contexto, la localización precisa de usuarios en sillas de ruedas se ha convertido en un factor clave para mejorar la autonomía, seguridad y calidad de vida de las personas, especialmente en entornos complejos como hospitales, centros comerciales, aeropuertos y espacios urbanos inteligentes.

Uno de los enfoques más destacados en la investigación sobre movilidad asistida es la implementación de sistemas de localización GPS combinados con control electrónico de sillas de ruedas. En la Politécnica Salesian**a**, Olivo y Gallegos desarrollaron un sistema de control electrónico para sillas de ruedas eléctricas dirigido a personas con discapacidad motriz en miembros inferiores. Este proyecto incorporó un módulo GPS para la ubicación en exteriores y un sistema de control remoto a través de una aplicación móvil desarrollada en Android. Como resultado, se logró una mejora en la regulación de niveles de velocidad, mayor suavidad en los movimientos y un mejor rendimiento en los frenos, reduciendo tiempos de respuesta y movimientos bruscos [9].

Por otro lado, Triviño y Alonso, en la Universidad de Valladolid, diseñaron un software para el control de sillas de ruedas motorizadas y crearon una interfaz de control con los pies utilizando Arduino. También optimizaron los componentes necesarios para el control mecánico indirecto del joystick. Las pruebas realizadas con personas discapacitadas y sujetos de control para evaluar la viabilidad y eficacia del sistema demostraron que el sistema es funcional, mejorando la facilidad de uso para personas con limitaciones en las manos, lo que contribuye a un control más preciso y a un aumento de la autonomía en la movilidad de los usuarios [10].

Además de las tecnologías convencionales como GPS, WiFi o Bluetooth, se están explorando soluciones alternativas, como LiFi (Light Fidelity), para tareas de localización en entornos cerrados. En la Universidad Nacional de Chimborazo**,** Quille presentó una experimentación con esta tecnología aplicada a dispositivos móviles y robots seguidores de línea. El sistema desarrollado utiliza un LED transmisor modulado por un Arduino Pro Mini, con alimentación

de 5V a 500mA, para enviar datos codificados en binario a través de la luz visible. Estos datos son captados mediante un sensor LDR, decodificados y luego transmitidos por un módulo ESP32 a una red WiFi convencional, lo que permite su visualización mediante una interfaz basada en WebSocket. Aunque la implementación logró establecer un canal de comunicación funcional, se identificaron márgenes de error en la detección de señales, lo que sugiere la necesidad de optimizaciones en la precisión y fiabilidad del sistema. No obstante, este trabajo demuestra el potencial de LiFi como una alternativa de bajo costo y libre de interferencias electromagnéticas para la localización en interiores, especialmente en escenarios donde otras señales podrían estar degradadas o no disponibles [11].

Complementariamente, En la Universidad Nacional de Chimborazo Balarezo realizó otro estudio se centró en el diseño e implementación de una red de comunicación basada en LiFi, comparando su rendimiento frente a una red WiFi convencional en entornos cerrados. Los resultados demostraron que el tiempo promedio de envío de archivos con LiFi fue significativamente menor que con WiFi, destacando su eficiencia para la transmisión de datos bajo ciertas condiciones [12].

Otra línea de investigación relevante en el ámbito de la movilidad asistida es la incorporación de inteligencia artificial (IA). En este contexto, ha surgido un creciente interés por aplicar técnicas de aprendizaje por refuerzo. En la Universidad Juan del Rey de San Carlos, Villalba desarrolló un Trabajo de Fin de Grado (TFG) en el que implementó un sistema de navegación aérea autónoma para drones, enfocado en el seguimiento de carriles en carreteras. El sistema combina técnicas de Deep Learning para la segmentación y detección de carriles con el algoritmo Q-Learning, que permite al dron aprender acciones óptimas a partir de la información sensorial del entorno. Este enfoque demuestra el potencial de los algoritmos de refuerzo en escenarios de navegación complejos y de difícil acceso para robots terrestres o humanos [13].

De manera complementaria, Olivares y Savendra, en la Universidad Señor de Sipán, realizaron un análisis sobre el impacto del Machine Learning (ML) y la Inteligencia Artificial (IA) en la optimización de rutas, un área crítica en logística y transporte. El estudio destaca el uso creciente de algoritmos de aprendizaje por refuerzo, redes neuronales LSTM con mecanismos de atención y enfoques híbridos que combinan técnicas tradicionales con aprendizaje profundo. Los resultados mostraron que, frente a los métodos convencionales, las soluciones basadas en Machine Learning ofrecen una mayor eficiencia computacional y adaptabilidad a entornos

dinámicos, aunque los algoritmos clásicos siguen teniendo ventajas en ciertos problemas específicos [9].

2.2 MARCO TEÓRICO

2.2.1 Movilidad humana en silla de ruedas.

Según las necesidades de cada persona las sillas pueden ser manuales, eléctricas y una combinación híbrida de ambas. Aunque existen otras clasificaciones, esta investigación se enfoca exclusivamente en estas tres categorías [14]. En la Tabla 1 se puede observar una comparación entre los enfoques manual y eléctrico en el diseño de sillas de ruedas.

Tabla 1:Comparación de sillas de ruedas

Tipo de silla	Ventajas	Desventajas
Silla manual	Control total	Esfuerzo físico
	Simplicidad y confianza	Limitaciones de distancia
	Portabilidad	
Silla eléctrica	Menor esfuerzo físico	Dependencia de batería
	Mayor autonomía	Peso y tamaño
	Funciones eléctricas de asiento	Costo

2.2.2 Principales desafíos en la movilidad autónoma para personas con discapacidad.

La movilidad autónoma representa diversos desafíos para las personas con discapacidad. La falta de ajustes a los requerimientos del usuario en los algoritmos de navegación y planificación de rutas es uno de los principales desafíos, lo que perjudica la eficiencia y confianza del sistema. Por otro lado, las sillas de ruedas robóticas recientes no toman en cuenta los espacios sociales de las personas, lo que puede originar un malestar en entornos públicos.

La limitación de movimiento independiente de extremo a extremo incide en la calidad de vida de las personas con discapacidad, dificultando la atención médica, acceso a la educación y el

empleo. Asimismo, puede aumentar el estrés mental y físico, que se pueden derivar en otras condiciones crónicas. Alrededor de 2,7 millones de personas en EE. UU. dependen de sillas de ruedas para sus tareas cotidianas, y se estima un crecimiento anual del 7%, por el aumento de la longevidad y la supervivencia post-trauma.

Frente a esta creciente demanda, se vuelve esencial fomentar el desarrollo de tecnologías asistidas innovadoras que incorporen sistemas avanzados de navegación y prevención de colisiones. En este contexto, los robots móviles asistidos han demostrado ser una alternativa viable para mejorar la calidad de vida y la autonomía de las personas con discapacidad en sus entornos cotidianos [15].

2.2.3 Tecnología LiFi

LiFi o (Light Fidelity) es una tecnología que permite la transmisión de datos mediante una fuente de luz LED cuya intensidad varía a alta velocidad. Este sistema es ideal para proporcionar conectividad inalámbrica en áreas confinadas con alta densidad de dispositivos. Además, mitigar las interferencias electromagnéticas propias de las tecnologías basadas en radiofrecuencia [16]. El principio de funcionamiento de LiFi se basa en la modulación de la luz visible para transmitir información. Un emisor LED parpadea a velocidades imperceptibles para el ojo humano, pero detectables por un fotodiodo conectado a un sistema de cómputo. Este receptor convierte los pulsos luminosos en señales digitales, codificadas mediante un sistema binario: el encendido de la luz representa un 1, y el apagado un 0. Al igual que una antena WiFi se conecta a una computadora, el receptor LiFi debe ser acoplado a un equipo de procesamiento para decodificar los datos transmitidos por los emisores LED [17]. En las

Tabla 2, Tabla 3, Tabla 4 se presentan las principales características y diferencias entre esta tecnología y otras alternativas de comunicación inalámbrica.

Tabla 2:Comparación de velocidades entre tecnologías

Tecnología	**Velocidad**

LiFi	1Gbps
WiFi	150Mbps
IrDA	4 Mbps
Bluetooth	3Mbps
NFC	424 kbps

Tabla 3:Comparación de transmisión entre LiFi y WiFi

Parámetro	LiFi	Wifi
Espectro utilizado	Luz visible	Radiofrecuencia
Estándar	IEEE 802.15.7	IEEE 802.11
Alcance	< 10m	< 300m
Tasa de transferencia de datos	Muy alta	Baja
Consumo energético	Bajo	Alto
Costo	Bajo	Alto
Ancho de Banda	Ilimitado	Limitado

Tabla 4:Limitaciones de LiFi [18]

Limitación	Descripción
Dependencia de la luz	No se puede acceder a internet sin una fuente de luz, lo que limita las ubicaciones y situaciones de uso.
Requiere Línea de visión cercana o perfecta	Es necesario que exista una línea de visión sin obstáculos entre el transmisor y receptor para que los datos sean transmitidos correctamente.
Sensibilidad a obstáculos opacos	Los obstáculos opacos en el camino pueden interrumpir la transmisión de datos.
Afectación por luz ambiental	La luz natural, la luz solar y la luz eléctrica puede influir negativamente en la velocidad de transmisión de datos.
Limitación por no atravesar paredes	Las ondas de luz no atraviesan paredes, lo que interfiere en la transmisión de datos.
Alto costo de instalación inicial	La instalación puede ser costosa en un inicio.

Falta de desarrollo para adopción masiva	Aún está en fase de desarrollo, no ha sido adoptada ampliamente

2.2.4 Diseño de un sistema LiFi

El diseño de un sistema LiFi se fundamenta en el estándar VLC (Visible Light Communication) y emplea pulsos rápidos de luz para transmitir información de forma inalámbrica.
Este sistema está compuesto por:

- Un LED blanco de alta luminosidad que actúa como fuente emisora.
- Un fotodiodo de silicio con alta sensibilidad a la luz, que cumple la función de receptor.

El módulo emisor del sistema LiFi incluye cuatro subconjuntos principales, tal como se describen en [16] y se ilustra en la Figura 1.

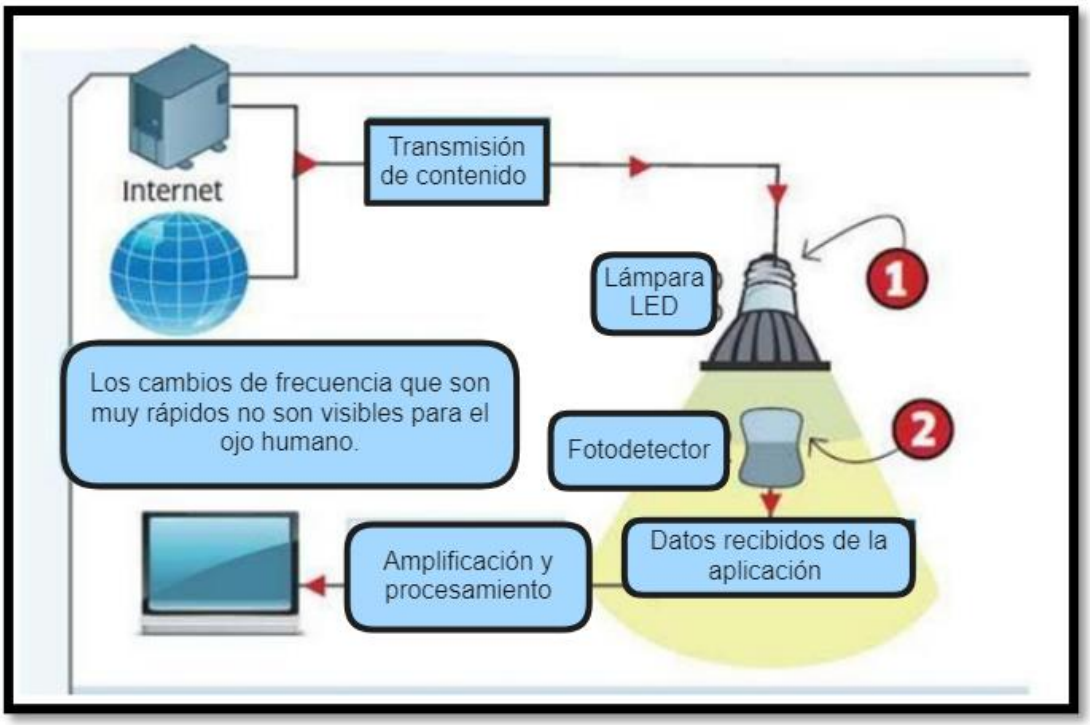

Figura 1:Diagrama de bloques del sistema LiFi [11].

2.2.5 Diodos led

En la comunicación LiFi, la transmisión de datos se realiza mediante LED, cuya luz visible se genera en función del material semiconductor utilizado en su fabricación. La Tabla 5 muestra cómo diferentes compuestos semiconductores, (como GaAs, GaN o InGaN), emiten luz de distintos colores. Esto se debe a que cada material posee una banda prohibida particular, que determina la frecuencia y longitud de onda de la luz emitida.

Tabla 5:Tabla comparativa entre distintos materiales de diodos LED [18]

Composición del chip	**Color de la luz emitida**	**Tensión de trabajo en volt (V)**	**Frecuencia en Hertz (Hz)**	**Longitud de onda en nm**
GaAs	Infrarrojo	< 1,9	$< 4{,}0 \times 10^{14}$	> 760
GaAlAs	Infrarrojo	< 1,9	$< 4{,}0 \times 10^{14}$	> 760
GaP	Rojo	± 1,8	$4{,}8 - 4{,}0 \times 10^{14}$	610 – 760
GaAlAs	Rojo	± 1,8	$4{,}8 - 4{,}0 \times 10^{14}$	610 – 760
AlInGaP/GaP	Rojo	± 1,8	$4{,}8 \times 10^{14}$	610 – 750
GaAsP/GaP	Naranja	± 2,0	$5{,}0 \times 10^{14}$	600 – 640
AlInGaP	Naranja	± 2,0	$5{,}1 - 4{,}8 \times 10^{14}$	590 – 610
GaAsP/GaP	Amarillo	± 2,0	$5{,}1 - 4{,}8 \times 10^{14}$	590 – 610
AlInGaP	Amarillo	± 2,0	$5{,}2 \times 10^{14}$	570 – 590
GaP	Verde	± 3,0	$5{,}3 \times 10^{14}$	550 – 570
InGaN	Verde	± 3,0	$5{,}8 \times 10^{14}$	510 – 550
InGaN / Zafiro	Verde	± 3,0	$5{,}9 \times 10^{14}$	500 – 550
SiC	Azul	± 3,3	$6{,}0 \times 10^{14}$	490 – 510
GaN	Azul	± 3,3	$6{,}3 \times 10^{14}$	480 – 500
InGaN / Zafiro	Azul	± 3,3	$6{,}7 \times 10^{14}$	450 – 480
InGaN	Azul	± 3,4	$7{,}1 \times 10^{14}$	450 – 470

InGaN	Ultravioleta	± 3,7	$7,5 \times 10^{14}$	430 – 450
GaN	Ultravioleta	± 3,7	$7,9 \times 10^{14}$	< 400
Ce:YAG	Espectro completo	± 3,4	Espectro completo	Espectro completo

2.2.6 Algoritmos de Machine Learning

Muchos de los trabajos de hoy en día utilizan el aprendizaje automático como técnica para implementar sistemas con algoritmos de predicción. Sin un sistema no tiene la capacidad de aprender no puede ser considerado inteligente [19]. La elección del algoritmo de *Machine Learning* adecuado, especialmente para la movilidad de sistemas robóticos depende de la detección de características relevantes del entorno. Incluso cuando estas características se encuentran disponibles normalmente suelen presentarse con una complejidad asociada. Es importante tomar especial cuidado en calidad de los datos y los requerimientos específicos del sistema a implementar con un modelo de aprendizaje automático. Donde se debe analizar su aplicabilidad en la implementación en un sistema de predicción. En la Tabla 6 se puede encontrar de algunos algoritmos muy utilizados en la práctica.

Tabla 6:Descripción de algoritmos de Machine Learning utilizados comúnmente en la práctica

Algoritmo	Tipo y Características
Árbol de decisiones	Algoritmo supervisado que utiliza una estructura tipo árbol para dividir los datos en subconjuntos según sus características, facilitando la toma de decisiones.
Bosque aleatorio	Conjunto de árboles de decisión entrenados con diferentes subconjuntos de datos y características mejorando la precisión y reduce el sobreajuste.
XGBoost	Algoritmo de boosting basado en árboles de decisión, optimizado para velocidad y rendimiento. Se construyen modelos de manera secuencial corrigiendo errores previos.
Máquinas de vectores de soporte (SVM)	Algoritmo supervisado que encuentra el hiperplano óptimo para separar datos en distintas

	clases, útil en problemas de clasificación y regresión.
SEFR	Clasificador lineal rápido diseñado para ejecutarse eficientemente en dispositivos embebidos y de bajo consumo energético.
PCA (Análisis de Componentes Principales)	Técnica no supervisada de reducción de dimensionalidad que transforma variables correlacionadas en un conjunto reducido de componentes independientes.
Redes Neuronales	Modelos computacionales inspirados en el cerebro humano; aprenden a partir de grandes volúmenes de datos y son capaces de reconocer patrones complejos.
Q-Learning	Algoritmo de aprendizaje por refuerzo que aprende una política óptima mediante la exploración del entorno y la maximización de recompensas acumuladas.

2.2.7 Aprendizaje supervisado y no supervisado

El aprendizaje supervisado implica entrenar un modelo a partir de datos etiquetados. La forma de ajuste paramétrico con este enfoque depende mucho de la calidad del etiquetado. En esta forma de aprendizaje se implica obligatoriamente a un elemento experto que determina la correspondencia de salida ante una determinada entrada [20]. Por otro lado, en el aprendizaje no supervisado se utiliza de manera exclusiva datos sin etiquetar y los patrones para clasificación o detección se los encuentra de manera autónoma. También existe un enfoque de aprendizaje semi supervisado donde se combina estas dos formas de aprendizaje. Una etapa de entrenamiento consiste en realizar aprendizaje supervisado para luego terminar el entrenamiento con un proceso de entrenamiento no supervisado. Este método es especialmente útil cuando la cantidad de datos a etiquetar se dispone en grandes cantidades. Aplicar esta metodología puede evitar el costo de tiempo asociado a etiquetar dichos datos de manera manual.

2.2.8 Redes neuronales

Una red neuronal es un modelo de procesamiento e inferencia de información que se inspira en el funcionamiento del cerebro humano [21]. El objetivo de una red neuronal es adquirir una función que vincule una o varias entradas con una o varias salidas. La vinculación se hace mediante el ajuste de parámetros internos del modelo mediante entrenamiento con datos. El modelo entrenado luego debe ser evaluado y hacer predicciones o clasificaciones con entradas que no han formado parte del entrenamiento [22]. En la Figura 2 se puede observar la estructura de una red neuronal donde se evidencia las capas que la componen y su estructura para crear un algoritmo.

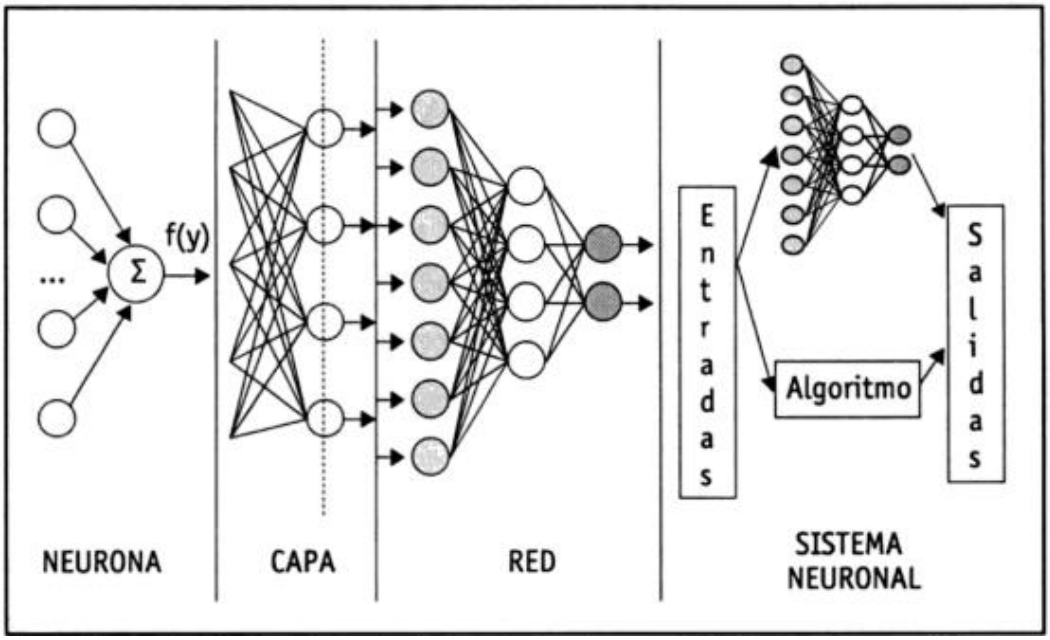

Figura 2: Estructura de una red neuronal [23]

2.2.9 Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN, por sus siglas en inglés) son un tipo de arquitectura de red neuronal diseñada específicamente para el procesamiento de datos con una estructura de tipo rejilla, como las imágenes. A diferencia de las redes neuronales tradicionales, las CNN están formadas por capas especializadas que permiten la extracción automática y jerárquica de características espaciales de los datos de entrada.

Estas redes se componen principalmente de tres tipos de capas: capas convolucionales, capas de reducción (pooling) y capas completamente conectadas. Las capas convolucionales aplican filtros (kernels) que recorren la imagen para detectar características locales como bordes, texturas o formas. Luego, las capas de reducción disminuyen la resolución espacial de los mapas de características, reduciendo así la cantidad de parámetros y el costo computacional, además de ayudar a evitar el sobreajuste. Finalmente, las capas completamente conectadas se encargan de la clasificación, basándose en las características extraídas por las capas anteriores.

La Figura 3 ilustra la estructura típica de una red neuronal convolucional. Se observa cómo la imagen de entrada pasa por una primera capa de convolución, que genera varios mapas de características. Posteriormente, una capa de reducción simplifica esa representación. Esta secuencia de convolución y reducción se repite en una segunda etapa para obtener características más profundas. Finalmente, las salidas de las capas convolucionales se aplanan y se envían a una capa clasificadora, compuesta por una red neuronal densa que determina la clase final de la imagen. Este diseño modular permite que las CNN sean efectivas en tareas de clasificación, segmentación y detección de objetos en imágenes [24].

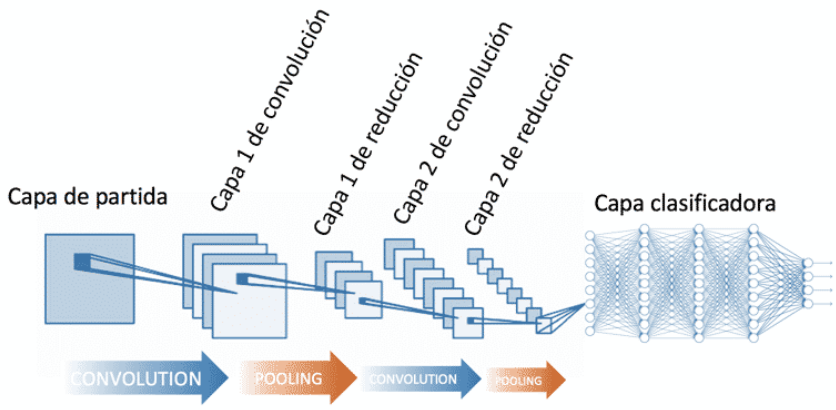

Figura 3:Estructura de una red neuronal convolucional [25]

2.2.10 Aprendizaje por refuerzo y algoritmo Q-Learning

El aprendizaje por refuerzo (Reinforcement Learning, RL) es una rama del aprendizaje automático donde un agente aprende a tomar decisiones óptimas mediante la interacción con un entorno dinámico. A diferencia de otros métodos supervisados o no supervisados, en RL no

se proporciona un conjunto de datos con respuestas correctas. En su lugar, el agente experimenta diferentes acciones y aprende a partir de recompensas o castigos recibidos como retroalimentación del entorno. [26] . El proceso de aprendizaje se da en base a prueba y error, y el objetivo del agente es maximizar la recompensa acumulada a lo largo del tiempo. Para ello, el agente evalúa las consecuencias de sus acciones y ajusta su comportamiento con base en los resultados obtenidos.

- **Q-Learning**

Q-Learning es uno de los algoritmos más populares dentro del aprendizaje por refuerzo. Se basa en el uso de una función de valor Q, que estima la utilidad de tomar una determinada acción en un estado específico, siguiendo una política óptima. Esta utilidad se representa como la Ecuación 1 .

Ecuación 1:Ecuación de Bellman para Q-Learning

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

Donde:

$\boldsymbol{s_t}$ es el estado actual

$\boldsymbol{a_t}$ es la acción tomada

$\boldsymbol{Q(s_t, a_t)}$: es el valor estimado de esa acción en ese estado.

La descripción de los términos es la siguiente:

$\boldsymbol{r(s_t, a_t)}$**:** recompensa inmediata obtenida al ejecutar la acción a_t en el estado s_t.

$\boldsymbol{\gamma}$**:** factor de descuento $(0 < \gamma \leq 1)$ que determina la importancia de las recompensas futuras.

es el valor máximo estimado para el siguiente estado s_{t+1}. Considerando todas las acciones posibles.

Funcionamiento del algoritmo

1. El agente observa su estado actual s_t.
2. Toma una acción a_t según una política (por ejemplo, explorando o explotando).
3. Recibe una recompensa $r(s_t, a_t)$ y observa el nuevo estado s_{t+1}..

4. Actualiza el valor Q de la acción realizada usando la ecuación anterior.
5. Repite el proceso hasta que converja a una política óptima.

2.2.11 Visión artificial

La visión artificial, también conocida como visión por computador, es una tecnología que permite a los sistemas informáticos analizar, interpretar y comprender imágenes o secuencias de video, con el fin de tomar decisiones o ejecutar acciones de forma automática. Este proceso simula la capacidad visual humana, permitiendo que una cámara conectada a un sistema pueda detectar objetos, reconocer formas, seguir trayectorias, o incluso identificar patrones [27].

El funcionamiento general de la visión artificial se basa en las siguientes etapas:

1. Captura de imagen: mediante una cámara.
2. Procesamiento: donde se mejora la imagen, se eliminan ruidos y se extraen características relevantes.
3. Análisis o interpretación: donde un algoritmo o modelo de Machine Learning identifica lo que hay en la imagen.
4. Acción o decisión: el sistema responde según lo que ve ya sea moviéndose, deteniéndose o ejecutando una orden.

2.2.12 ArUcos

Un marcador ArUco es un cuadrado negro con una cuadrícula binaria en su interior, la cual codifica un número de identificación único para cada marcador, esto permite determinar su orientación. En una imagen, el sistema de detección busca objetos de forma cuadrada, la posición y orientación del marcador ArUco se estima con respecto al sistema de coordenadas de la cámara. La posición en píxeles de las esquinas del marcador se toma de la imagen y se refina de manera preliminar mediante un cálculo de precisión subpíxel, utilizando la información de gradiente a lo largo de los bordes del marcador. Los marcadores ArUco están optimizados para estimar rápidamente la posición 3D de la cámara con respecto al marcador con una buena resistencia a detecciones incorrectas. En caso de que más de uno de los marcadores se capture en un único campo de visión (FOV) de la cámara, reconocer el código

interno permite particularizar cada uno de ellos. La forma en la que se presentan los ArUcos se puede observar en la Figura 4.

Figura 4:Ejemplo de modelos de un ArUco [28]

CAPÍTULO III.

3.1 Metodología

El enfoque de esta metodología es diseñar, implementar y evaluar un sistema de navegación autónomo en interiores para asistir a personas con discapacidad que utilizan sillas de ruedas. Este sistema estará basado en tecnologías como: LiFi (comunicación inalámbrica mediante luz visible) y Machine Learning (aprendizaje automático). El objetivo es ofrecer una solución que permita independencia y movilidad a los usuarios en la navegación en espacios interiores.

El diseño del sistema se centrará en la integración de algoritmos de visión por computadora para detectar el entorno y generar comandos de movimiento adecuados. En la implementación, se combinarán plataformas de bajo consumo energético, como Raspberry Pi y Arduino con la tecnología de comunicación LiFi y que por sus características permitirá una conectividad en interiores.

3.2 Tipo de investigación

La presente investigación es aplicada y cuantitativa. Es aplicada porque busca resolver un problema concreto que es facilitar la movilidad autónoma en interiores para personas con discapacidad mediante el uso de tecnologías emergentes. Es cuantitativa porque se basa en la recolección y análisis de datos numéricos (tiempos de desplazamiento) para comparar el rendimiento de diferentes enfoques tecnológicos. Además, se enmarca en un diseño experimental, ya que se realizaron pruebas controladas con distintas configuraciones tecnológicas para observar su impacto sobre una variable dependiente (el tiempo de desplazamiento).

3.3 Población

La población está compuesta por todos los datos obtenidos del tiempo de desplazamiento de la silla de ruedas autónoma tanto con Machine Learning como con el algoritmo aleatorio, con un total de 180 pruebas con cada tecnología bajo condiciones controladas.

3.4 Muestra

En esta tesis no se trabajó con muestra y se ha considerado la población total de datos la cual estuvo conformada por 360 registros obtenidos durante las pruebas.

3.5 Operacionalización de variables

El tipo de variable los indicadores y las técnicas de instrumentación utilizadas en este proyecto se puede observar en la Tabla 7.

Tabla 7:Operacionalización de variables

TIPO DE VARIABLE	VARIABLE	CONCEPTO	INDICADORES	TÉCNICAS E INSTRUMENTACIÓN
Dependiente	Tiempo de desplazamiento Tiempo de llegar de un lado a otro	Se define como el lapso necesario para llevar a cabo un traslado desde un punto hasta otro	Tiempo en segundos	• Observación • Cronómetro
Independiente	Tipo de tecnología	Se hace alusión a la manera en que el usuario controla la silla de ruedas, ya sea usando el sistema de navegación y sin el	Nominal: tipo No asistido (silla de ruedas convencional) Asistido (silla de ruedas equipada con la tecnología LiFi y Machine Learning)	• Observación

3.6 Procedimiento/desarrollo

La Figura 5 muestra un diagrama de tres fases dividido que representan las etapas del desarrollo de un sistema de navegación en este trabajo de investigación. La primera fase abarca el diseño del sistema, incluyendo la filtración y elección de algoritmos útiles, así como el diseño de la arquitectura. La segunda fase se enfoca en la implementación, comenzando con el montaje del hardware, seguido por la integración de tecnología LiFi, el desarrollo de modelos de Machine

Learning y la realización de pruebas de laboratorio. La tercera fase corresponde a la evaluación del sistema, donde se ajustan configuraciones, se evalúan trayectorias, se analizan los resultados obtenidos y finalmente se documentan los resultados.

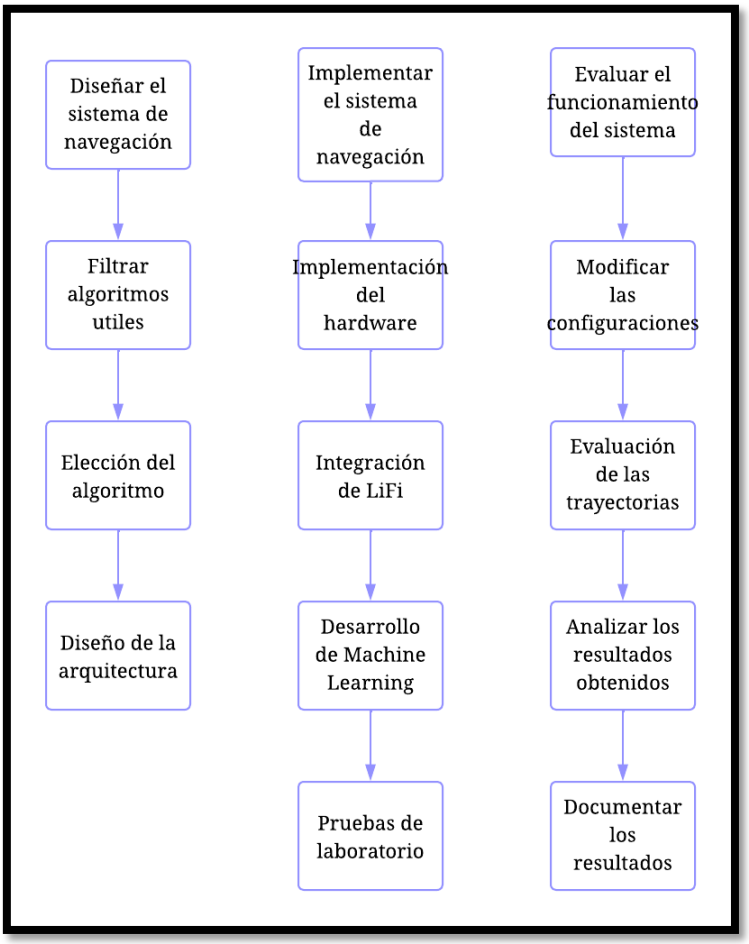

Figura 5:Fases del procedimiento de la investigación

3.7 Fase 1: Diseño del sistema de navegación

En esta fase se determina la arquitectura del dispositivo que se necesita para implementar el proyecto. Se establece un esquema de bloques necesarios para el desarrollo del dispositivo. Se hace la selección en base a los requerimientos del proyecto para poder implementar el dispositivo de navegación. Además, se detalla una serie de pruebas preliminares para escoger el algoritmo y entorno de pruebas. Además, se detalla ciertos aspectos que afectan a la implementación de dispositivo de navegación.

3.7.1 Arquitectura del dispositivo

Se desea una arquitectura para el dispositivo, como se muestra en la Figura 6. El sistema debe constar de tres módulos claramente definidos para su funcionamiento. El primer módulo es el de comunicación LiFi, el cual debe ser diseñado. Este módulo debe incluir un sistema emisor y receptor, junto con sus interfaces de conexión necesarias. El siguiente módulo es el sistema de procesamiento, el cual gestiona la comunicación con el resto de los dispositivos del proyecto. Además, una de sus características es que debe ejecutar los algoritmos de inferencia para el control del sistema de silla de ruedas. El tercer módulo necesario es integrar un sistema mecánico para el control de la silla de ruedas este mecanismo. Junto a un sistema que debe interpretar señales de control a movimientos para realizar un desplazamiento deseado.

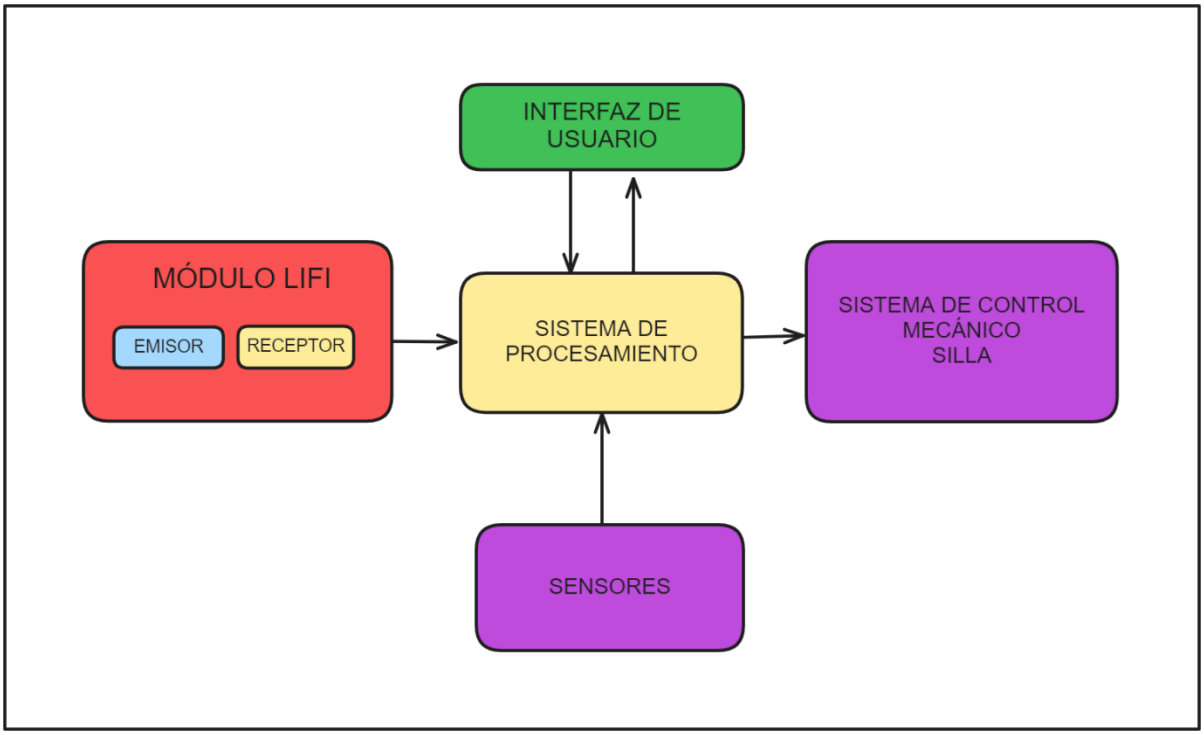

Figura 6: Esquema de diseño del dispositivo deseado

3.8 Fase 2: Implementación del sistema de navegación

En esta fase se detalla el proceso de selección de los elementos del sistema de navegación. Se considera la selección del hardware hasta el desarrollo del modelo de Machine Learning, los módulos de comunicación para el LiFi y el diseño de la interfaz de usuario.

3.8.1 Selección del dispositivo de cómputo

Debido a los requerimientos del proyecto, el sistema de cómputo debe ser compacto, de bajo consumo energético, bajo costo y contar con la capacidad de procesamiento suficiente para ejecutar algoritmos de Machine Learning. Asimismo, debe ofrecer compatibilidad con las principales librerías necesarias para el desarrollo, a fin de facilitar la implementación. Otro criterio esencial es la capacidad de gestionar periféricos de comunicación, permitiendo su integración con los demás dispositivos del sistema de navegación. En la Tabla 8 se presentan 4 tarjetas de desarrollo basadas en distintos procesadores, seleccionados por su disponibilidad en el mercado y de relativa facilidad de adquisición. Además, constan de un amplio uso en la comunidad de desarrollo lo que garantiza el acceso a documentación necesaria.

Tabla 8:Comparativa de tarjetas de desarrollo basadas en procesador

Característica	**Jetson Nano**	**Raspberry Pi 3 Modelo B**	**Raspberry Pi Zero W**	**Raspberry Pi 4 Modelo B**
CPU (modelo,arq, nucleos,freq.)	ARM CortexA57 (4 núcleos @ 1,43 GHz)	Broadcom BCM2837 (4× Cortex-A53 @ 1,2 GHz)	Broadcom BCM2835 (1× ARM11 @ 1,0 GHz)	Broadcom BCM2711 (4× Cortex-A72 @ 1,8 GHz)
GPU(modelo, soporte)	NVIDIA Maxwell (128 núcleos CUDA)	VideoCore IV (dual-core, OpenGL ES 2.0)	VideoCore IV (integrado, OpenGL ES 2.0)	VideoCore VI (OpenGL ES 3.0)
RAM(tipo, cantidad)	4 GB LPDDR4 (64-bit)	1 GB LPDDR2	512 MB SDRAM	1/2/4/8 GB LPDDR4
Almacenamiento	16 GB eMMC 5.1 (kit: ranura microSD)	Ranura microSD (sistema operativo)	Ranura microSD	Ranura microSD
Conectividad	Ethernet Gigabit ; sin Wi-Fi/BT (slot M.2 Key E)	Wi-Fi 802.11 b/g/ n; Bluetooth 4.1 ; Ethernet 10/100 Mbps	Wi-Fi 802.11 b/ g/n; Bluetooth 4.1/ BLE ; sin Ethernet	Wi-Fi 802.11 b/g/ n/ac (doble banda); Bluetooth 5.0 ; Ethernet Gigabit
Puertos	4×USB 3.0 + 1×USB 2.0 (micro-B);	4×USB 2.0; 1×HDMI + 1×RCA	1×mini-HDMI; 1×USB micro OTG; 1×USB	2×USB 3.0 + 2×USB 2.0; 2×micro-HDMI;

	1×HDMI 2.0 + 1×DisplayPort; CSI (cámara); GPIO 40 pines	compuesto; audio 3,5 mm; CSI; DSI; GPIO 40 pines	micro(alimentación) ; CSI; GPIO 40 pines	audio 3,5 mm; CSI; DSI; GPIO 40 pines
Consumo energético (alimentación)	Entrada 5V/4A (barrel) o 5V/ 2A micro-USB	Entrada 5V/2,5A micro-USB (≈12,5 W)	Entrada 5V/1A micro-USB (≈5 W máx.)	Entrada 5V/3A USB-C (≈15 W máx.)
Dimensiones físicas (mm)	100×80×29 (Kit)	85×56×17	65×30×5	85×56×17
Sistema operativo compatible	Linux (Ubuntu con JetPack)	Linux (Raspbian, opcional Windows 10 IoT)	Linux (Raspbian, etc.)	Linux (Raspberry Pi OS, Ubuntu, etc.)
Precio aproximado	USD 269.00	USD 89.99	USD 39.00	USD 160.00

3.8.2 Raspberry Pi Zero W

Es la unidad principal del proyecto y se selecciona en base a las características establecidas en la Tabla 8 dado por cada uno de los fabricantes. Esta unidad se encarga del procesamiento de datos provenientes de la interfaz de usuario, del módulo LiFi y de otros dispositivos para la comunicación con el sistema de control de la silla de ruedas.

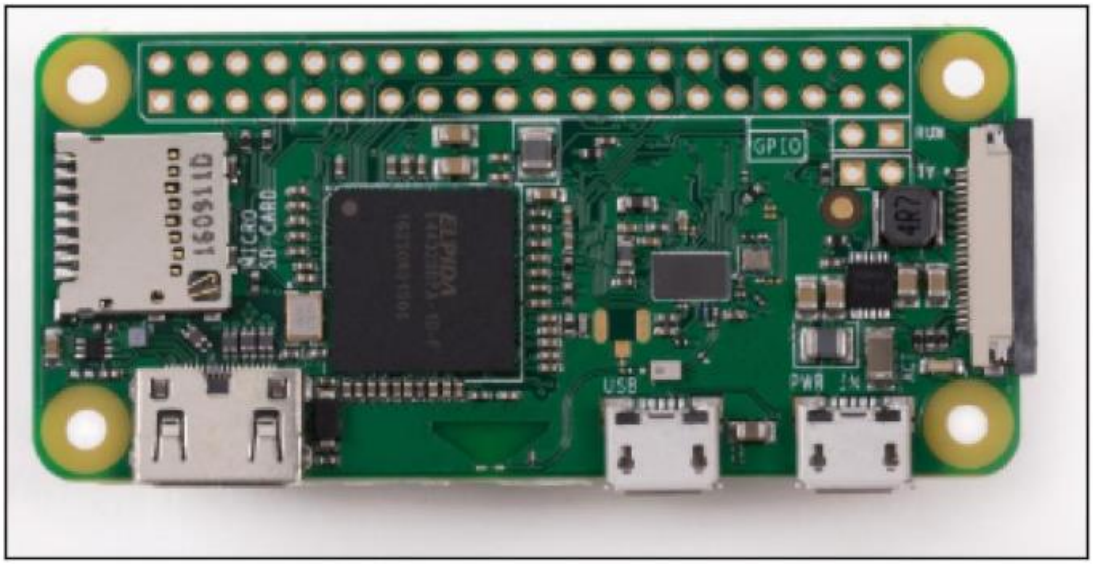

Figura 7:Raspberry Pi Zero w [29]

La tarjeta de desarrollo Raspberry Pi Zero W fue seleccionada por ofrecer capacidades de procesamiento superiores en comparación con sistemas basados en microcontroladores. Una característica que dispone es la posibilidad de ejecutar un sistema operativo Linux. lo cual permite una gestión eficiente de los recursos de hardware y software. La tarjeta en comparación con otras ofrece un tamaño reducido y menor consumo energético a un bajo costo. La Raspberry Pi incorpora interfaces de comunicación como UART, I2C y salidas PWM fundamentales para la integración con sensores, módulos de comunicación y dispositivos de control. En la Figura 7 se muestra la placa utilizada en el diseño del sistema.

Una vez adquirida, se procedió a instalar el sistema operativo Raspberry pi OS. La característica de la instalación de este sistema operativo es que se implementó sin interfaz gráfica para aprovechar mejor los recursos de procesamiento. La configuración del dispositivo debe hacerse mediante SSH una vez habilitado el periférico WIFI de la tarjeta. Además, se debe instalar las librerías de Python y los aplicativos necesarios para habilitar la ejecución del software a implementar. Cabe mencionar que además es necesario habilitar ciertos periféricos de manera manual. Tal es el caso de los módulos UART y Bluetooth los cuales no vienen configurados por defecto.

3.8.3 Selección del microcontrolador

La Tabla 9 muestra la comparativa de tres placas de desarrollo ampliamente utilizadas en proyectos de electrónica y sistemas embebidos: Arduino Nano, ESP32 y STM32 Blue Pill. Presenta sus principales características técnicas, incluyendo el tipo de microcontrolador, velocidad de reloj, memoria, voltaje de operación, número de pines de entrada/salida, interfaces de comunicación, conectividad inalámbrica, tipo de conector USB y entornos de programación compatibles. Esta comparación permite identificar capacidades, ventajas y limitaciones de cada placa para seleccionar la más adecuada según el tipo de aplicación o proyecto a desarrollar.

Tabla 9:Comparativa de tarjetas de desarrollo basados en microcontroladores

Característica	Arduino Nano	ESP32 (ESP-WROOM-32)	STM32 Blue Pill (STM32F103C8T6)
Microcontrolador	ATmega328P (8 bits)	Tensilica Xtensa LX6 (32 bits, doble núcleo)	ARM Cortex-M3 (32 bits)
Frecuencia de reloj	16 MHz	Hasta 240 MHz	72 MHz
Memoria Flash	32 KB (2 KB usados por el bootloader)	4 MB (externa)	64 KB
SRAM	2 KB	520 KB	20 KB
EEPROM	1 KB	No	No
Voltaje de operación	5V	3.3V	3.3V (soporta señales de 5V en algunos pines)
Pines digitales I/O	14 (6 PWM)	Hasta 34 (incluyendo funciones especiales)	37 (12 PWM)
Entradas analógicas	8	Hasta 18 canales ADC de 12 bits	10 canales ADC de 12 bits
Interfaces de comunicación	UART, SPI, I2C	3 UART, 4 SPI, 2 I2C, CAN, SDIO, Ethernet MAC, I²S, PWM, sensores táctiles, DAC, ADC	3 UART, 2 SPI, 2 I2C, CAN, USB
Conectividad inalámbrica	No	Wi-Fi 802.11 b/g/n, Bluetooth v4.2 BR/EDR y BLE	No
Conector USB	Mini-USB tipo B	Micro-USB	Micro-USB (requiere ST-LINK o bootloader para programación)
Programación	Arduino IDE	Arduino IDE, PlatformIO, ESP-IDF	STM32CubeIDE, Arduino IDE (requiere configuración adicional)
Dimensiones (mm)	45 x 18	25.5 x 18	53 x 22
Precio aproximado	USD 9.99	USD 14.99	USD 17.50

3.8.4 Arduino Nano

Según los criterios establecidos en la Tabla 9 se seleccionó la placa Arduino Nano debido a sus características de bajo costo, facilidad de programación y versatilidad de diseño. Una de las unidades se encargará de recopilar información de los sensores y enviarla a la Raspberry Pi para su procesamiento. Además, esta unidad recibe y ejecuta los comandos de accionamiento destinados al controlador mecánico del joystick de la silla de ruedas. La segunda placa Arduino se utiliza para gestionar el control del módulo de transmisión LiFi. En la Figura 8 se muestra el Arduino nano utilizado en este proyecto.

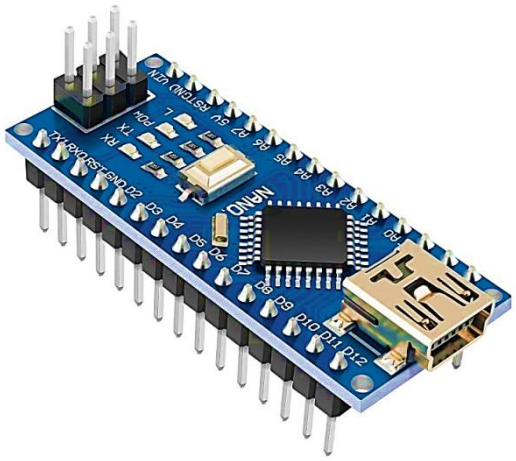

Figura 8:Arduino Nano [30]

3.8.5 Selección de servomotores

Tabla 10:Comparativa de servomotores

Características	**MG90S**	**MG996R**	**DS3218**
Tipo	Micro servo	Servo estándar de alto torque	Servo digital de alto torque
Voltaje de operación	4.8V – 6.6V	4.8V – 7.2V	4.8V – 6.8V
Torque máximo	1.8 kg·cm (4.8V), 2.2 kg·cm (6.6V)	9.4 kg·cm (4.8V), 11 kg·cm (6V)	20 kg·cm (6.8V)
Velocidad	0.10 s/60° (4.8V), 0.08 s/60° (6.0V)	0.19 s/60° (4.8V), 0.15 s/60° (6V)	0.16 s/60° (6.8V)
Ángulo de rotación	180°	180°	270°
Tipo de engranajes	Metálicos	Metálicos	Metálicos

Peso	13.4 g	55 g	60 g
Dimensiones (mm)	22.8 x 12.4 x 28.4	40.7 x 19.7 x 42.9	40 x 20 x 40
Corriente en reposo	~10 mA	~10 mA	~20 mA
Corriente en operación	~200 mA	500 mA – 1 A	1.5 A
Corriente de arranque	~700 mA	Hasta 2.5 A	Hasta 2.5 A
Tipo de control	PWM digital	PWM digital	PWM digital

Los servomotores son componentes fundamentales para el mecanismo de control de la silla de ruedas. Su función es aplicar una fuerza necesaria sobre el joystick, permitiendo generar el movimiento deseado. De acuerdo con la hoja de datos que se muestra en la Figura 33, el joystick integrado en la silla requiere una fuerza mínima de 1 Newton para desplazarlo desde su posición de reposo. En base a estos requerimientos y conforme a los datos presentados en la Tabla 10, se eligieron los servomotores modelo DS3218 y MG996R. Es importante considerar las características técnicas del servomotor, en particular el torque máximo que puede proporcionar. Estos servomotores garantizan un accionamiento adecuado para el sistema de movimiento de la silla de ruedas.

3.8.6 Selección de silla de ruedas eléctrica.

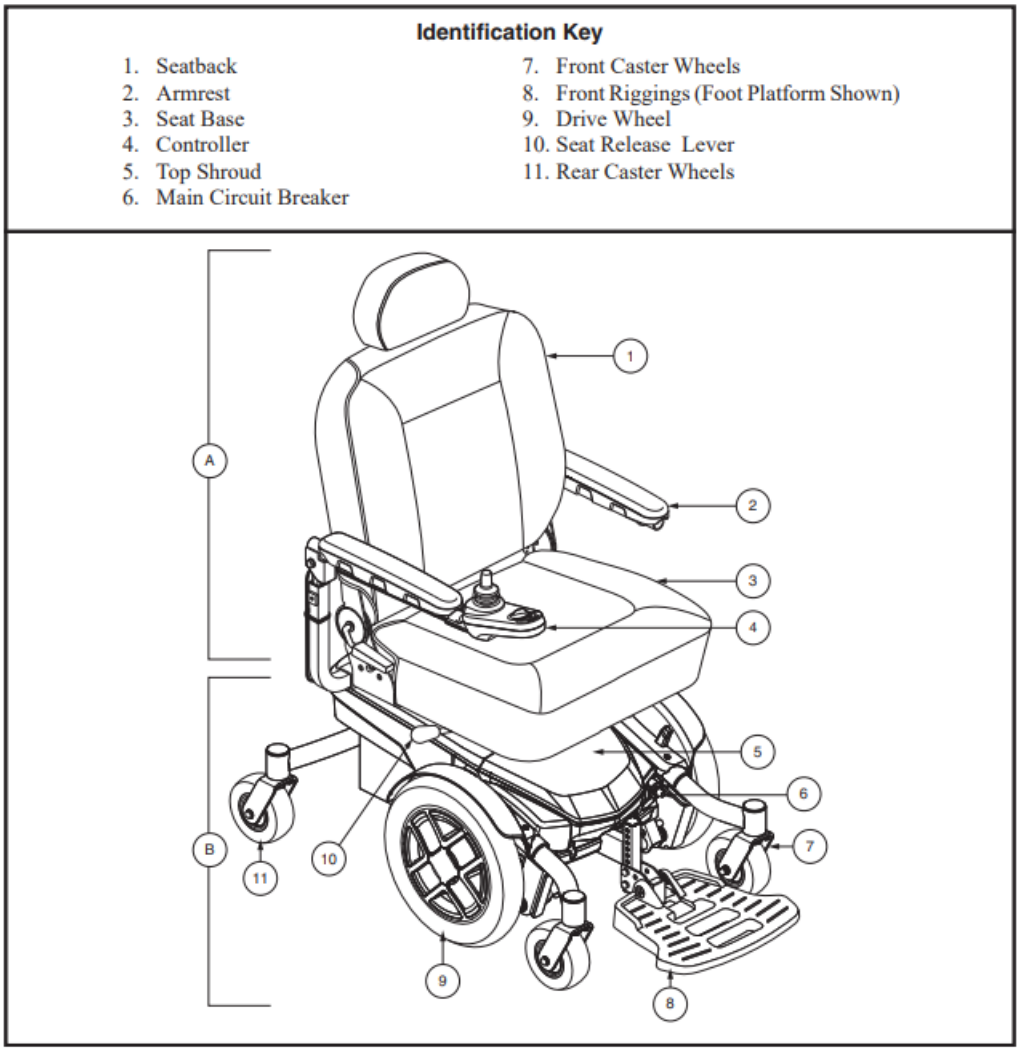

Figura 9:Estructura de la silla de ruedas [31]

La silla de ruedas eléctrica seleccionada se compone de los elementos mostrados en la Figura 9. Dicha silla fue adquirida por su disponibilidad en condición de préstamo dado que generalmente tienen un costo elevado en comparación con sillas sin mecanismos de movilidad electromecánicos. Para los fines de este proyecto resulta relevante resaltar que el controlador integrado debe permitir un control completo en el desplazamiento mediante el uso de un joystick. Las especificaciones eléctricas y mecánicas del sensor integrado en el joystick se pueden observar en el Anexo 2.

3.8.7 Mecanismo de control de la silla de ruedas

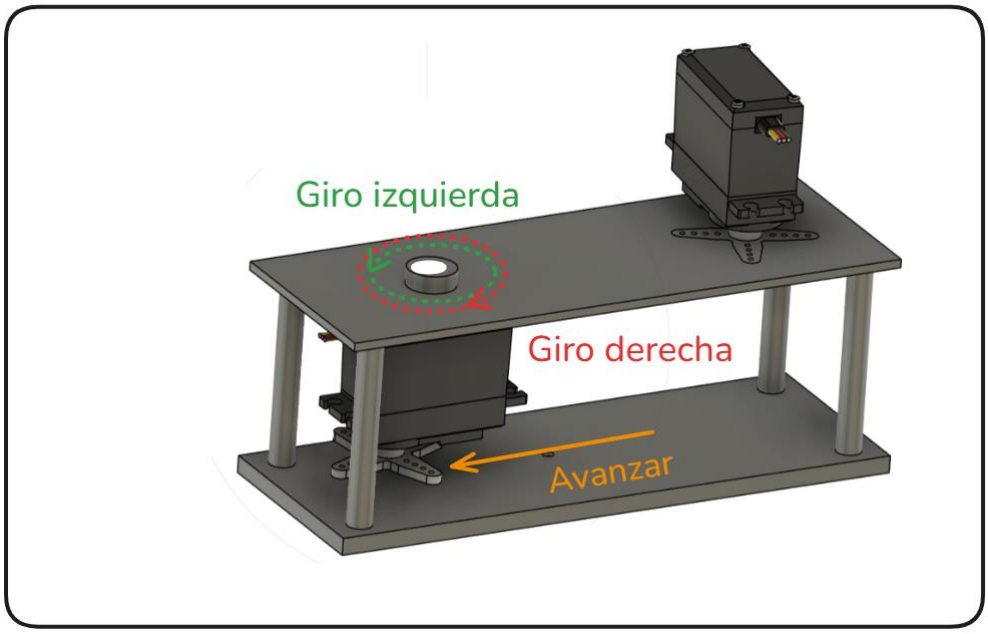

Figura 10:Mecanismo de control de la silla de ruedas

El mecanismo de control consiste en un acople directo sobre el mando de la silla de ruedas. Los movimientos están realizados por 2 servomotores dispuestos de tal manera que se pueda ejercer una fuerza controlada sobre el joystick. Debido a este diseño, existe una restricción en los movimientos a solo tres tipos de desplazamiento: avanzar, girar a la izquierda y girar a la derecha, un modelo del dicho mecanismo se puede observar en la Figura 10.

3.8.8 Desarrollo del modelo de Machine Learning

El sistema de detección con Machine Learning debe hacerse en varias etapas con el objetivo de seleccionar el modelo más adecuado para el control autónomo de la silla de ruedas. A continuación, se describe el proceso completo de selección del modelo.

3.8.9 Recopilación de datos para el modelo de red neuronal

Para entrenar el modelo se toman imágenes desde la silla de ruedas mientras se mueve en el entorno de prueba. Se realiza captura de imágenes de manera manual mediante la interfaz de usuario tratando de obtener información de características relevantes del entorno donde se moviliza la silla de ruedas. Esta fase permite al modelo reconocer características relevantes

durante el movimiento y reaccionar ante estas entradas. Durante el recorrido, se presiona manualmente los botones de avance, giro izquierdo y giro derecho, de modo que la silla registre la acción correspondiente a cada situación separándolas en 3 clases diferentes.

Otra posibilidad es capturar datos de entrenamiento, y hacer un etiquetado manual de las imágenes capturadas de la pista a partir de un video. Plataformas como Roboflow facilitan la organización, anotación y preprocesamiento de estos datos. De esta manera se crea la base de datos que servirá para entrenar el modelo. Una parte del diseño es tomar en cuenta el entorno donde se van a realizar pruebas y la captura de datos. Hay que asegurarse que el entorno no refleje luz y que se generen falsos positivos. Para cumplir este requerimiento se pintó el piso con pintura mate. Cada imagen se clasifica según la dirección que debería tomar el sistema en cada detección:

- Clase "Izquierda": imágenes donde se debía girar a la izquierda.
- Clase "Derecha": imágenes que indicaban un giro hacia la derecha.
- Clase "Avanzar": cuando la ruta estaba despejada hacia el frente.

3.8.10 Preprocesamiento y división de datos

Antes de entrenar el modelo, las imágenes se redimensionan para reducir la carga computacional además se transforman a escala de grises. Posteriormente, se dividen en tres conjuntos:

- Entrenamiento (70%)
- Validación (15%)
- Prueba (15%)

3.8.11 Entrenamiento del modelo de aprendizaje automático

Aquí existen varios enfoques y dependiendo del tipo de datos y el modelo puede variar la forma de entrenar el modelo.

- **Modelo de red neuronal convolucional**

En la Figura 11 se muestra el esquema general del proceso de entrenamiento llevado a cabo para un modelo de aprendizaje automático con redes neuronales. Este proceso abarca desde la recolección y etiquetado de los datos hasta la implementación final del modelo en el sistema computacional.

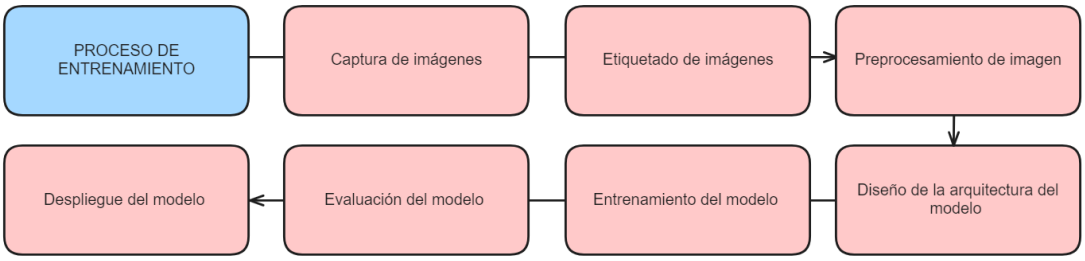

Figura 11:Esquema del entrenamiento

Con los datos organizados, se entrena la red neuronal convolucional (CNN) utilizando Keras y TensorFlow. Esta arquitectura se elige por su capacidad para extraer características relevantes de las imágenes mediante el entrenamiento.

- **Arquitectura de la red neuronal.**

Para plantear una red neuronal se toma en cuenta 3 aspectos importantes:

1. **Capa de entrada:** en este caso la capa de entrada sería las imágenes a escala de grises que se pasan al modelo.
2. **Capa de salida:** son las clases avanzar, derecha, izquierda
3. **Capa intermedia:** se extraen las características de la pista para hacer una predicción en la salida en base a los patrones de líneas que se pueden observar. Esta capa puede variar considerablemente los resultados que da un modelo.

Tabla 11:Arquitectura de red neuronal convolucional utilizada en el proyecto

Tipo de capa	Forma de salida	Parámetros
conv2d	(None, 198, 298, 16)	160
max_pooling2d	(None, 99, 149, 16)	0
conv2d_1	(None, 97, 147, 32)	4,640
max_pooling2d_1	(None, 48, 73, 32)	0

Flatten	(None, 112128)	0
Dense	(None, 64)	7,176,256
dense_1	(None, 3)	195

Una vez obtenidos los datos, se define un modelo de red neuronal, cuyo diseño se detalla en la Tabla 11. Para este proyecto se optó por una arquitectura de red neuronal convolucional (CNN), implementada mediante la librería Keras de Python. El entrenamiento del modelo se realizó en un entorno computacional compatible con dichas librerías.

Con el fin de optimizar recursos, se utilizó la plataforma Google Colab, que permite el uso de aceleradores de hardware como GPU de manera gratuita. El script correspondiente al entrenamiento del modelo se encuentra documentado en el Anexo 19.

- **Algoritmo de aprendizaje por refuerzo.**

El objetivo de implementar un algoritmo de aprendizaje por refuerzo es diseñar un agente capaz de tomar decisiones en función del entorno por el cual se desplaza la silla de ruedas. En este caso, uno de los desafíos principales es la presencia de regiones con poca iluminación, lo cual representa una restricción en el proyecto. Por esta razón, se plantea el desarrollo de un agente para el guiado de la silla de ruedas en este entorno. Se asume un entorno de navegación definido por los identificadores aruco en un espacio de 17 filas por 11 columnas como se puede observar en la Figura 12. El algoritmo determina que aruco forma parte de una trayectoria hacia un nodo objetivo. De esta manera un nodo válido es cualquier aruco que no pertenezca al borde es decir las filas 0,16 y las columnas 0,10 a los que denominamos nodos prohibidos ya que la silla de ruedas no está autorizada a salir de los límites establecidos. Para este propósito se implementa un algoritmo de aprendizaje por refuerzo basado en Q-Learning. El sistema de recompensas se define de la siguiente manera:

- Movimiento regular = -1
- Nodo prohibido = -100
- Llegar al objetivo = +100

Las acciones que puede seguir se determinan de la siguiente manera norte, sur, este, oeste y movimientos en diagonal sureste, suroeste, noreste, noroeste que son movimientos virtuales

internos para seleccionar el nodo en una trayectoria hacia un nodo objetivo. El modelo no está diseñado para enviar comandos de movimiento directamente a la silla. Lo que hace el modelo es seleccionar y proveer un mecanismo de selección para el nodo siguiente durante la detección. De esta manera se establece una trayectoria guiando la silla hacia su destino o nodo final.

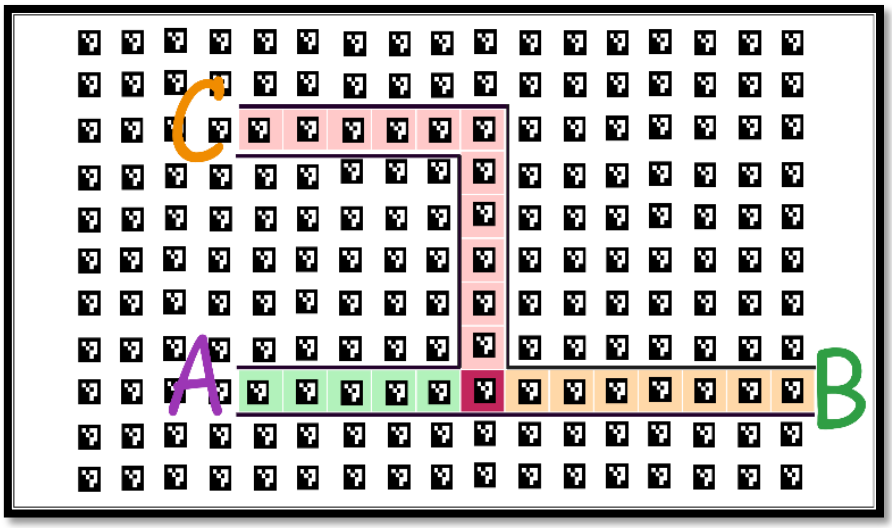

Figura 12:Planeamiento de trayectorias con ArUcos

En la Figura 13 se presenta una representación gráfica del proceso de aprendizaje de la tabla Q, mostrando como esta identifica una trayectoria hacia el nodo objetivo en distintos escenarios. La figura también permite visualizar las decisiones que el agente toma a partir de los valores almacenados en la tabla Q cuando se encuentra en cualquier nodo del entorno.

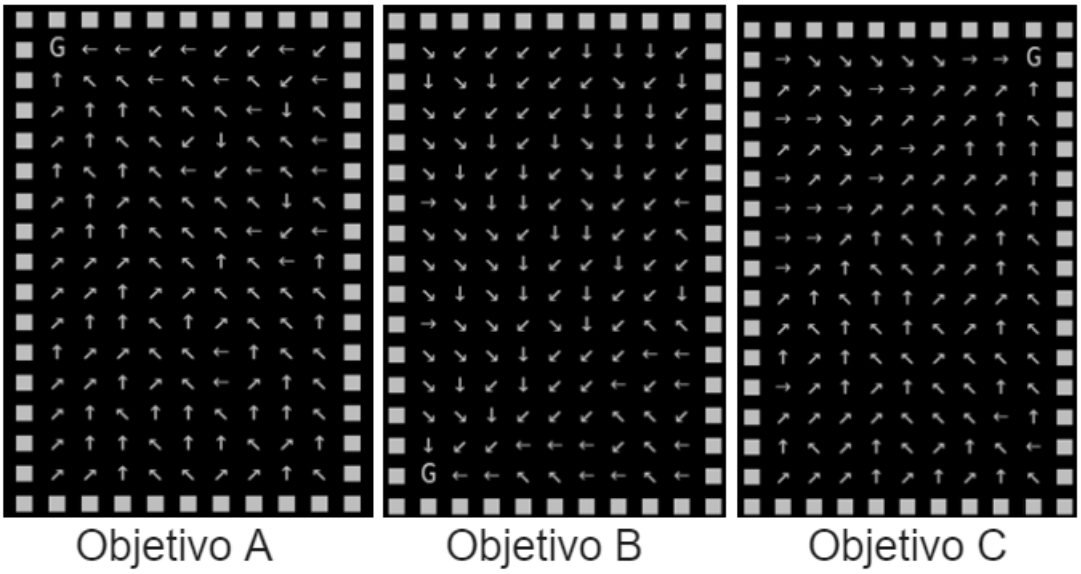

Figura 13:Trayectorias generadas para escoger el ArUco

Donde G es el nodo objetivo se puede observar la tendencia a encontrar el nodo siguiente en cada detección y por consiguiente una trayectoria es generada como se puede observar en la Figura 14.

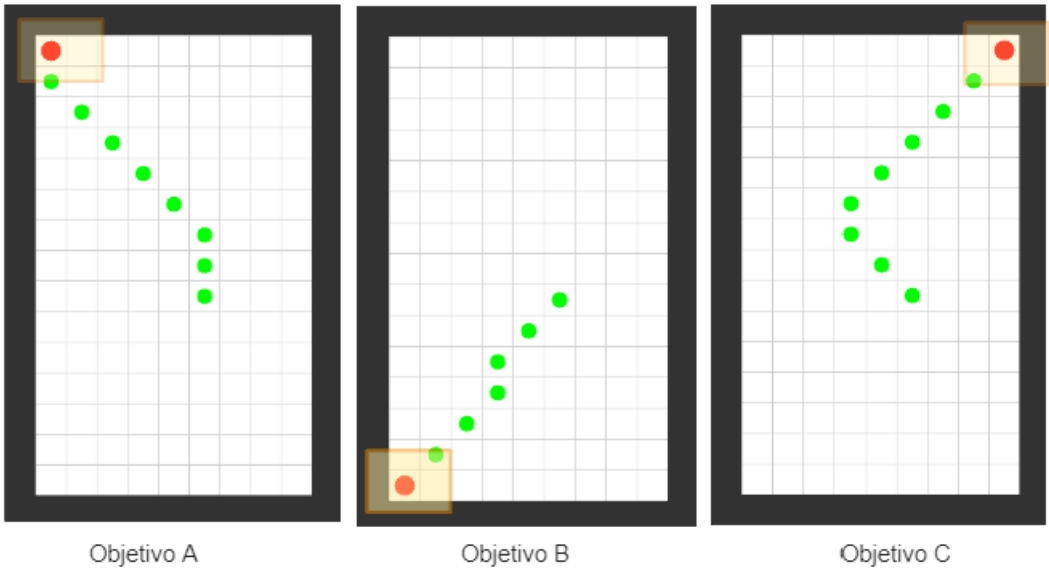

Figura 14:Trayectorias generadas por simulación utilizando una Tabla Q en cada caso.

De esta manera, se logra que, independientemente del nodo ArUco que el sistema detecte, siempre exista una trayectoria definida hacia un nodo específico. Esta estrategia permite que la silla de ruedas se desplace de forma autónoma y alcance una posición determinada, basada en ubicaciones relativas identificadas mediante los marcadores ArUco.

3.8.12 Integración del módulo LiFi

Para la comunicación LiFi se implementa módulos de transmisión y recepción para enviar datos de la habitación en la que se encuentra la lampara. La transmisión se realiza emulando las velocidades de transmisión de un protocolo UART a 9600 baudios con información codificada con pulsos eléctricos. Esto debido a la facilidad que supone en el diseño ya que solamente se implementaría el emisor con este protocolo. El sistema de transmisión se compone de un módulo emisor programado en un Arduino nano, el cual genera una señal eléctrica que envía impulsos lumínicos a un módulo receptor.

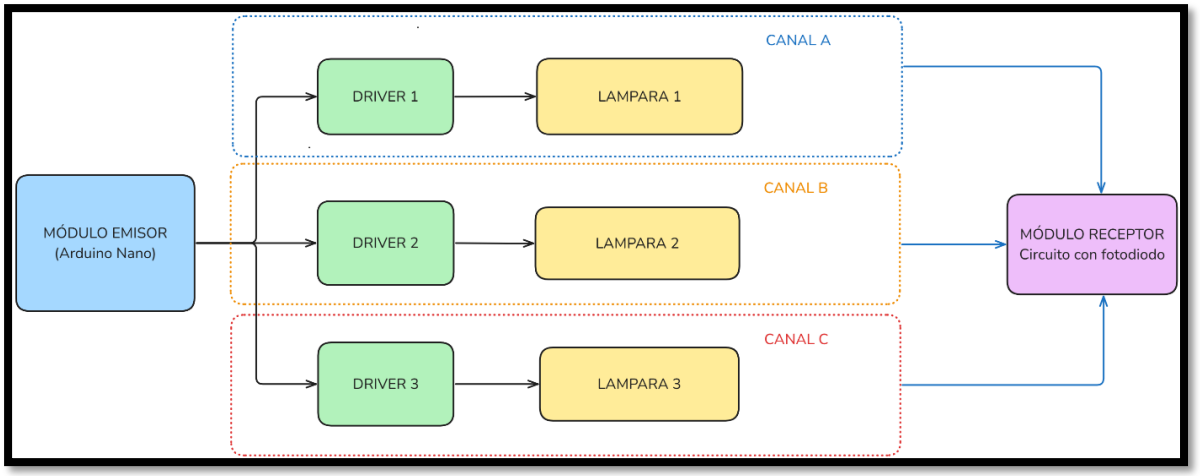

Figura 15:Diseño de transmisión de datos LiFi

El emisor es el encargado de generar y modular la señal de luz y realiza la transmisión de datos. La placa está alimentada por una fuente de 12 V y que con un regulador de voltaje se obtiene los 5 V necesarios para que el Arduino trabaje. El diseño consta de 8 canales que servirán para las salidas de los focos, cada canal maneja un dato o posición de transmisión dentro del Arduino, se tiene un controlador que gestiona el encendido y apagado de los diodos LED. El esquema de bloques de cómo se interconectarían estos elementos se puede observar en la Figura 16. El firmware del dispositivo se puede observar en el Anexo 12. El mismo que se implementa mediante el uso de temporización para el envío de datos codificados en protocolo UART.

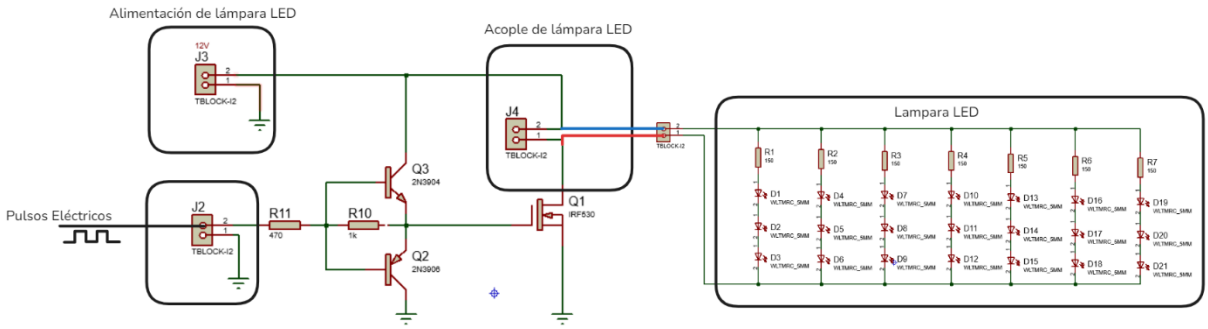

Figura 16:Esquema de conexiones del controlador de lámpara LiFi

El receptor está alimentado por 5V y recibe información por un fotodiodo a una tasa de transferencia aproximada de 9600 baudios que luego se interpreta y procesa con ayuda de un circuito de sincronismo con un 555 y un flip-flop tipo D como se puede observar en la Figura 17. Al final hay una etapa de inversión que depende de cómo se transmite la señal lumínica. En este caso no fue necesario invertir por lo que se puede descartar la compuerta not. Esta señal se conecta a un módulo bluetooth HC05 para ser transmitido a la Raspberry Pi, que cuenta con un

módulo bluetooth integrado. La elección del módulo HC05 es conveniente como esclavo en la recepción de información. Se escogió este módulo por compatibilidad por lo que puede utilizarse cualquier microcontrolador que reciba estas señales.

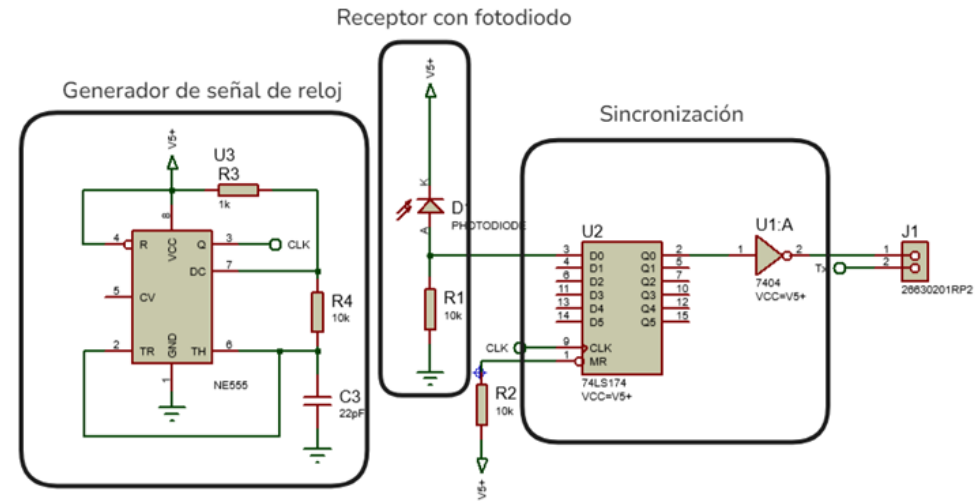

Figura 17:Esquema del receptor LiFi

Para el cálculo de la fuente necesaria se toma en cuenta que los leds blancos tienen una caída de voltaje de 3.3 volts y una corriente máxima de 25 mA. Si la fuente seleccionada es de 12 voltios entonces la resistencia debe ser mayor a R > (12-9.9)/0.025 > 84 Ohms. En este caso al ser una configuración en paralelo y 7 ramas por lámpara se escogió resistencias de 120 Ohms donde se espera que consuma 17.5 mA por rama dando un total de 122.5 mA por lámpara. Al ser 10 canales se espera que la fuente que se necesita debe proveer una corriente de al menos 1.225 A en esta configuración de luminarias.

3.8.13 Desarrollo de la interfaz de usuario

La interfaz gráfica permite controlar en tiempo real el funcionamiento del dispositivo, esta interfaz interactúa con un servidor http implementado en la Raspberry Pi utilizando App Inventor facilitando la interacción del usuario con el sistema y proporcionando información sobre su estado y modos de funcionamiento como se puede observar en la Figura 18.

Figura 18:Aplicativo para el usuario

3.8.14 Implementación y uso de los modelos

Una vez entrenados los algoritmos de detección se pueden utilizar en un flujo de código. En este caso al ser librerías de Python la implementación se realiza en dicho lenguaje de programación. Para el uso de los modelos de red neuronal se debe instalar Tensor Flow Lite. Y para la ejecución de la Tabla Q la librería de pickle. En la detección de los arucos y procesamiento de imágenes es necesario instalar opencv. La instalación y configuración de todas estas librerías suele ser una tarea de prueba y error. Depende mucho de la compatibilidad y de la arquitectura del procesador en este caso de la Raspberry Pi Zero W.

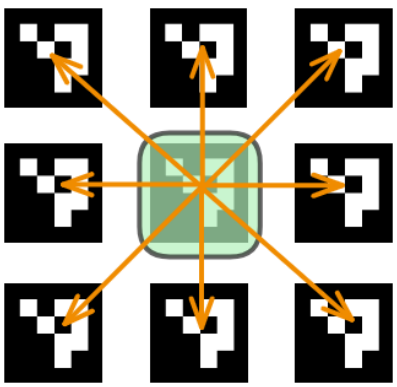

Figura 19:Búsqueda del aruco siguiente en la detección

En ambos casos, las predicciones generadas por el algoritmo se traducen en comandos que se envían a la silla de ruedas. Los ángulos están preprogramados en la tarjeta Arduino nano para los estados posibles avanzar, girar izquierda, girar derecha y detener. Los comandos que hacen posibles estos movimientos se codifican en la Raspberry Pi y se envían por puerto serial como D, C, B y A para avanzar, izquierda, derecha y alto respectivamente. Para el caso de las trayectorias determinadas por los arucos sigue el esquema de seguimiento mostrado en la Figura 19. El algoritmo de aprendizaje por refuerzo mediante la tabla Q determinará que ArUco debe ser el siguiente objetivo en la trayectoria hacia el objetivo.

3.9 Fase 3: Evaluar el funcionamiento del sistema

3.9.1 Pruebas preliminares de modelos en varias pistas

Esta es una etapa de pruebas preliminares, se hicieron en varias pistas y observó el comportamiento de un modelo y verificar problemas en su implementación final. Se analizaron por separado el rendimiento del LiFi y del modelo de Machine Learning, luego se hicieron pruebas combinadas para verificar el funcionamiento del sistema en conjunto en el Anexo 10 se puede observar las pistas donde se hicieron las pruebas preliminares.

Para evaluar el rendimiento del modelo, se realizaron pruebas en diferentes pistas con variaciones en el material y el contraste de la línea de seguimiento. Se probó configuraciones distintas:

1. Papel periódico blanco con cinta negra (cinta aislante): Se observó que el sensor respondía de manera inconsistente debido a reflejos en el papel.
2. Pintura blanca: La detección mejoró, pero aún se presentaban variaciones en los valores obtenidos por el sensor.
3. Pintura blanca con línea negra y bordes de cinta aislante negra: Se obtuvo un mejor contraste, pero el modelo aún tenía dificultades en algunas zonas.
4. Pintura blanca con bordes negros pintados: Esta configuración resultó ser la más adecuada, ya que proporcionaba una detección estable y uniforme.

Se compararon varios métodos de clasificación e inferencia tal como Random Forest, XGBoost, CNN y Q-Learning. Cada uno de estos métodos fue elegido por pruebas prácticas mediante experimentos en condiciones controladas. En las primeras pruebas con el modelo de Random

Forest, la precisión obtenida fue del 54%. Esta baja precisión se debía a que en el entrenamiento inicial no incluía todas las combinaciones posibles de movimiento (izquierda, derecha y adelante), lo que generaba inconsistencias en la toma de decisiones. Para mejorar el rendimiento, se optó por entrenar el modelo considerando todas las opciones posibles. Con este ajuste, la precisión aumentó al 60%.

Posteriormente, al repetir el entrenamiento con el sensor calibrado, la precisión se incrementó hasta 75%. Finalmente, al optimizar la pista eliminando la cinta aislante negra y dejando solo pintura negra en los bordes, se alcanzó un 85% de precisión utilizando un modelo basado en árboles de decisión. Estos datos se pueden observar en la Tabla 12.

Tabla 12:Pruebas experimentales sobre diferentes pistas

Condición de la pista	**Precisión inicial**	**Precisión con sensor calibrado**
Papel periódico con cinta aislante negra	54%	60%
Pintura blanca	54%	60%
Pintura blanca con línea negra y bordes de cinta aislante negra	75%	75%
Pintura blanca con bordes negros pintados	75%	85%

Estos resultados muestran la importancia de un entrenamiento completo del modelo para mejorar la precisión en el seguimiento de línea mientras el porcentaje de precisión influye mucho en la pista y en cómo se entrena dando bien las instrucciones, todas las instrucciones se dan desde la aplicación para movilizarla manualmente.

En una primera etapa, se utilizó un sensor QTR-8 que proporcionaba datos provenientes de lecturas de una pista en blanco y negro. Cada uno de los ocho canales del sensor en conjunto con el comando asociado fue registrado como entrada para entrenar un modelo Random Forest. No obstante, este enfoque no cumplió con los requerimientos del proyecto. Esto debido a la sensibilidad del sensor a variaciones de color en la superficie y errores provocados por imperfecciones o desgaste de la pintura de la pista por el movimiento de la silla. Esta situación generaba lecturas poco fiables y detecciones impredecibles durante las pruebas.

En una segunda etapa, se optó por integrar una cámara al sistema y adoptar un enfoque basado en visión por computadora. Este cambio permitió obtener información visual suficiente para el proceso de toma de decisiones del sistema de navegación. Se capturaron múltiples imágenes que representaban distintos escenarios como curvas, trayectorias rectas y bifurcaciones. La base de datos generada fue lo suficiente amplia y variada para asegurar una representación adecuada de los casos en el entorno donde se realizan las pruebas. Las detecciones con este método sirven para implementar el algoritmo de navegación que interpreta las condiciones del entorno para determinar los movimientos adecuados de la silla.

Se realizaron pruebas utilizando el modelo YOLOv8, un modelo pre entrenado especializado en detección de objetos. Se eligió por su versatilidad al momento de entrenar y facilidad de implementación, lo cual es útil en sistemas con recursos limitados como una Raspberry Pi. Sin embargo, al centrarse en la detección de objetos y no en detección de patrones de movimiento, los resultados no fueron del todo satisfactorios para las necesidades específicas del sistema.

Para mejorar el procesamiento de imágenes, se entrenó una CNN usando datos de imagen típicos del entorno real. Las CNN proporcionaron un mejor reconocimiento de patrones en la detección de obstáculos y trayectoria. Aunque las clasificaciones mostraron buenos resultados, el proceso de traducir estas detecciones en operaciones físicas dentro del entorno continuó siendo un problema debido a la pérdida de enfoque de la cámara.

Para mejorar el control se integró un modelo de aprendizaje por refuerzo basado en Q-Learning. Esta estrategia permitió a la silla de ruedas adquirir un mecanismo para encontrar trayectorias hacia un objetivo mejorando sus capacidades de seguimiento. La toma de decisiones se basa en una tabla de decisiones denominada Q-Table y que se entrenó en base a simulaciones. Este enfoque permitió que el algoritmo encuentre una manera de seguir la trayectoria hacia un objetivo en el espacio de pruebas en caso de perder el enfoque de la cámara.

Los resultados de las pruebas permitieron determinar el grado en que cada algoritmo implementado durante la fase de desarrollo contribuyó al rendimiento del sistema. La evaluación se basó en dos criterios principales: la precisión en la estimación de trayectorias, y el tiempo requerido para completar trayectorias específicas dentro del entorno experimental.

CAPÍTULO IV.

4.1 Resultados

En esta sección, se presentan los resultados obtenidos de la metodología y el proceso de validación del sistema de navegación en interiores para asistir en la movilidad de personas con discapacidad que utilizan sillas de ruedas mediante tecnología LiFi y Machine Learning.

Durante la etapa de evaluación la variable principal considerada es el tiempo de desplazamiento que se tarda la silla de ruedas en completar una trayectoria con diferentes configuraciones tecnológicas. Las pruebas se llevaron a cabo bajo en igualdad de condiciones, evaluando el comportamiento de la silla de ruedas al utilizar el sistema LiFi con un algoritmo de desplazamiento aleatorio. Posteriormente, al combinar el sistema LiFi con un modelo de Machine Learning. Esta evaluación permitió determinar cuál de las opciones resulta más adecuada para la movilidad y autonomía de personas con discapacidad en escenarios de interiores. Los resultados se presentan según las fases metodológicas identificadas en el desarrollo del proyecto:

Fase 1: Diseño
Fase 2: Implementación
Fase 3: Evaluación

Los datos iniciales se tomaron de un sensor QTR-8 de sus 8 canales infrarrojos para detectar una línea de color blanco y negro. Luego se utilizó el algoritmo Random Forest para implementar el modelo y predecir los comandos que deben aplicarse para controlar la silla. El modelo obtuvo un 54% de precisión en las pruebas de entrenamiento iniciales. Después de ajustar los datos y los parámetros del modelo, la precisión mejoró a un 65% en su ajuste.

Se utilizó imágenes capturadas en tiempo real, para usar métodos de visión computacional y buscar técnicas de detección más sólidas. Sin embargo, Random Forest y XGBoost tuvieron un mal desempeño. Ya que no fueron diseñados para extraer y manipular características visuales complejas. Esta restricción, a su vez, enfatizó la importancia de usar modelos diseñados para operar en contenido no estructurado, como la información visual.

En contraste, las redes neuronales convolucionales (CNN), entrenadas con datos visuales, alcanzaron niveles de precisión entre el 75% y el 89%, dependiendo de las condiciones de iluminación y del entorno. Estas redes mostraron una mayor capacidad para identificar elementos relevantes y tomar decisiones de navegación utilizando la información visual, representando una mejora significativa respecto a los enfoques anteriores.

El modelo de aprendizaje por refuerzo (ARL), en conjunto con el sistema de visión artificial, se integró con el objetivo de mejorar el control de movimiento. Aunque su desempeño no se midió en términos de precisión como los modelos clasificadores, su impacto fue evidente en la reducción del tiempo de desplazamiento.

Para la evaluación del sistema, se llevarán a cabo pruebas en un entorno controlado, comparando los tiempos de desplazamiento y la precisión de la navegación bajo distintas condiciones (utilizando solo LiFi, solo Machine Learning, y una combinación de ambas tecnologías).

4.1.1 Fase 1: Diseño

La Figura 20 muestra el esquema de los elementos que componen el sistema de navegación. El sistema inicia con un Arduino Nano el cual está alimentado por una fuente de 12 V. Esta tarjeta de desarrollo genera los pulsos eléctricos codificados que controlan el encendido y apagado de una lámpara LED. Las señales lumínicas se propagan en el entorno y son detectadas por un receptor LiFi. El receptor adapta esas señales para ser detectadas por un dispositivo que actúa de esclavo, en este caso un módulo bluetooth HC-05. Este módulo HC-05 se comunica con una Raspberry Pi que incorpora un módulo Bluetooth integrado.

La Raspberry Pi cumple múltiples funcionalidades y actúa como dispositivo maestro del sistema de navegación. En este sistema de cómputo se ejecutan los algoritmos de Machine Learning, utilizando los datos proporcionados por los sensores y la cámara conectados a la misma. Otro componente del sistema es un segundo Arduino Nano, que está conectado mediante comunicación UART con la Raspberry Pi. El Arduino recolecta información de los sensores y recibe los comandos enviados por la Raspberry Pi. Luego interpreta dichos comandos como señales PWM para controlar el mecanismo de servomotores que manipula el joystick de la silla de ruedas. Como elemento final, se incluye una interfaz de usuario que permite configurar las funcionalidades del dispositivo. Esta interfaz se ejecuta en un dispositivo Android, con el cual es necesario establecer conexión. Por último, el esquema muestra los requerimientos de alimentación eléctrica de cada componente, aspecto fundamental a considerar durante el diseño del sistema. El sistema implementado se puede observar dos placas Arduino Nano, una cámara y un mecanismo con 2 servomotores como se puede observar la Figura 21 y Figura 22.

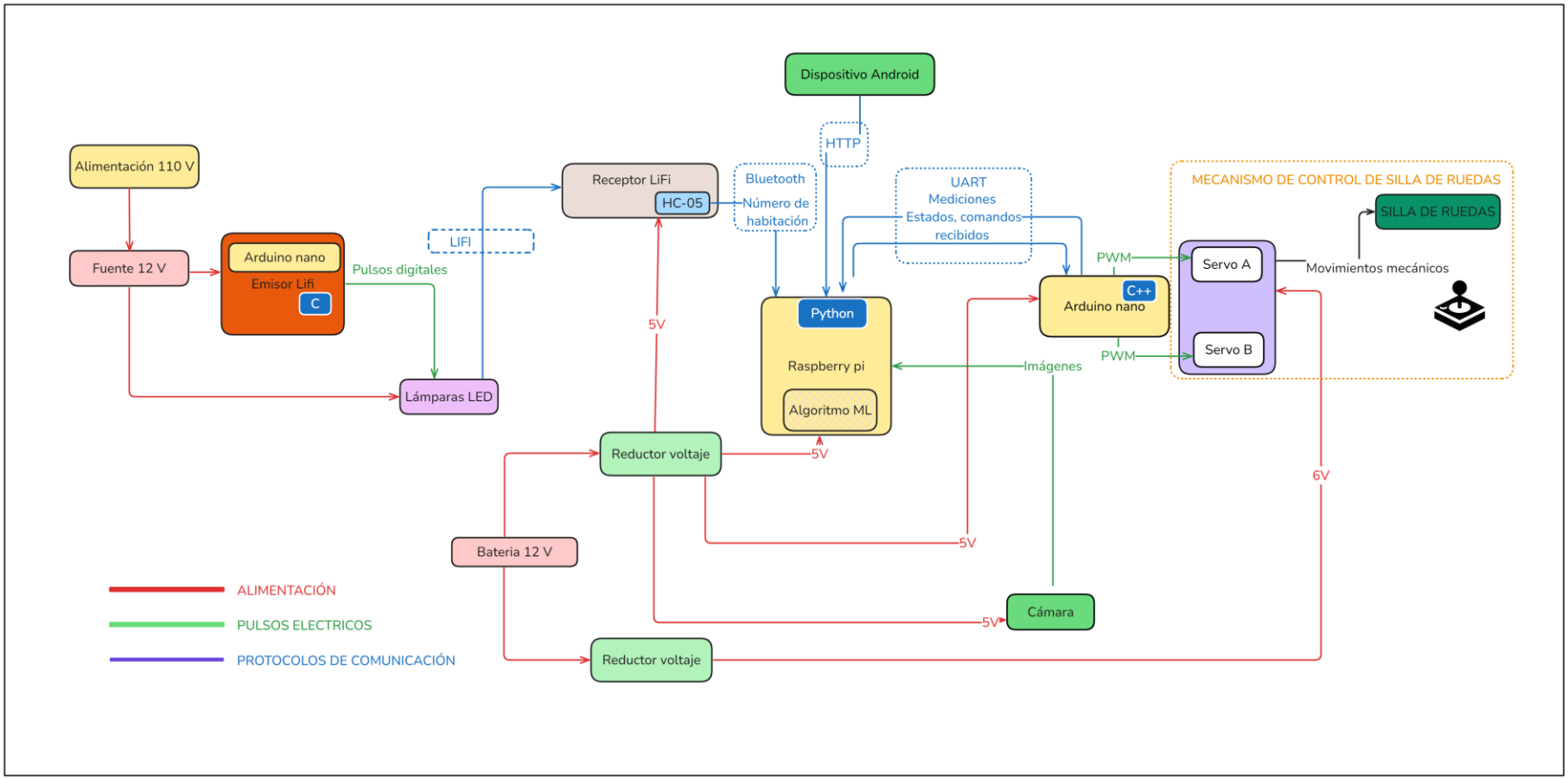

Figura 20:Esquema de bloques con los elementos integrados en el proyecto

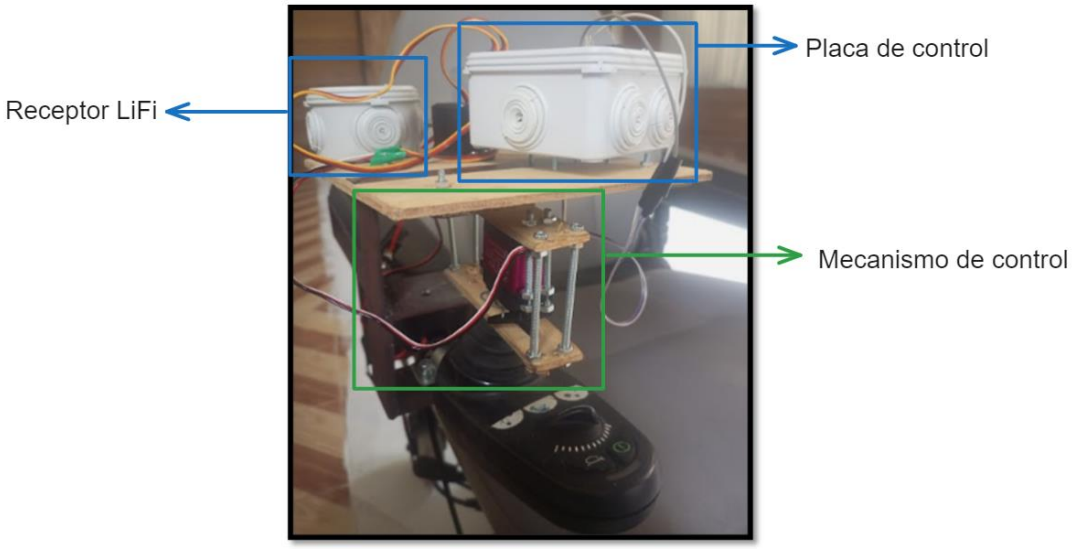

Figura 21:Mecanismo de control de joystick de la silla de ruedas

El software fue escrito con Python para el control general desde la Raspberry Pi, mientras que el código de los microcontroladores se desarrollará en C y C++. Para implementar los modelos de aprendizaje automático, se emplearon bibliotecas como Scikit-learn, TensorFlow y Keras, que son bibliotecas ampliamente utilizadas y probadas en este campo. Por último, se desarrollará una interfaz gráfica utilizando App Inventor, con el objetivo de que el sistema sea fácil de usar e intuitivo desde un dispositivo móvil.

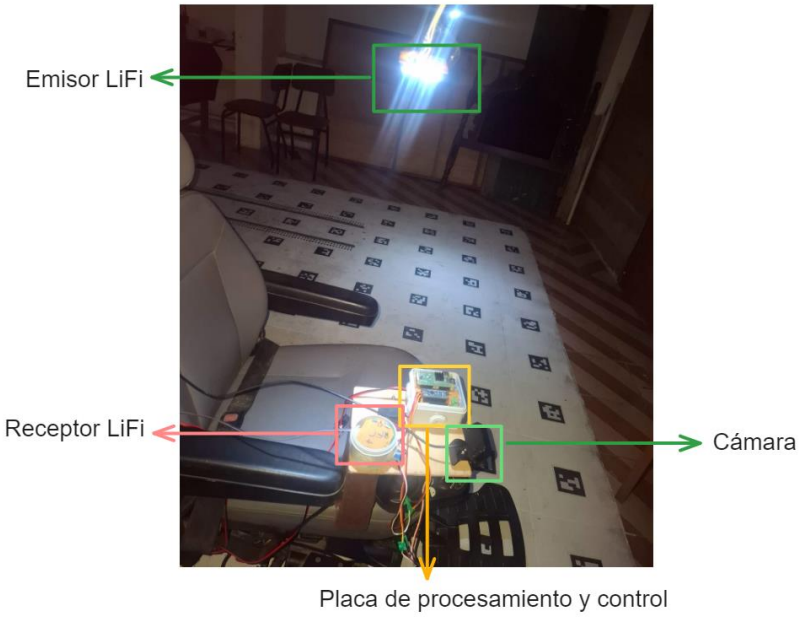

Figura 22:Sistema de navegación implementado sobre la silla de ruedas

4.1.2 Fase 2: Implementación

El modelo fue entrenado utilizando TensorFlow con las imágenes etiquetadas y separadas en clases diferentes. Los resultados del comportamiento durante el entrenamiento se pueden observar en la Figura 39. En las pruebas con el conjunto de datos se observó que el mejor modelo alcanzó un 98% en las validaciones. Como se puede observar en la Figura 23 hay algunas predicciones realizadas por el modelo.

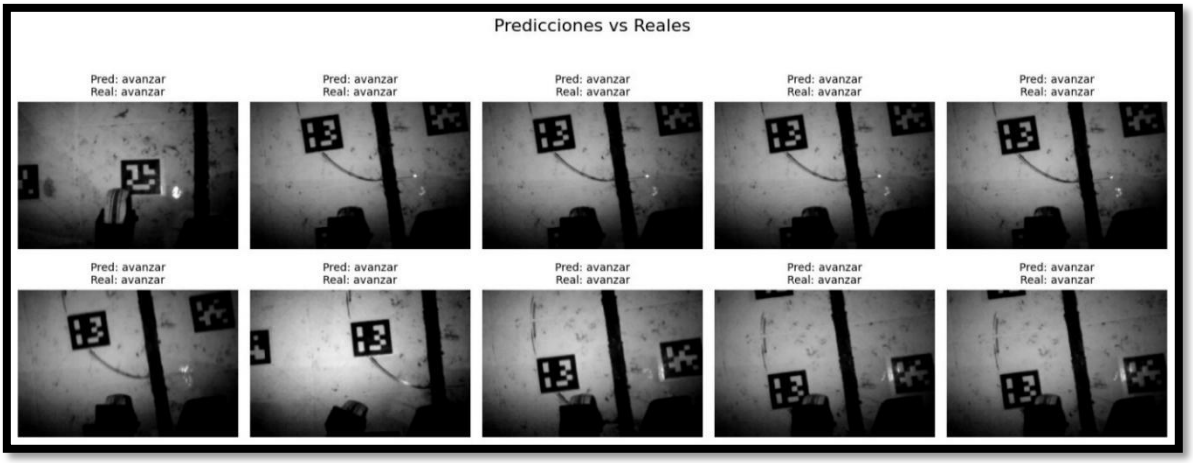

Figura 23:Predicciones vs reales

- **Preparación del entorno de evaluación**

Para poder realizar las pruebas se establecerá un entorno controlado garantizando así la repetitividad de los experimentos. Se diseñará un circuito con líneas guía que servirán como referencia para el sistema de navegación de la silla de ruedas y se colocarán marcadores ArUcos en el piso a una distancia considerable para que la cámara los pueda detectar y vaya generando trayectorias aleatorias.

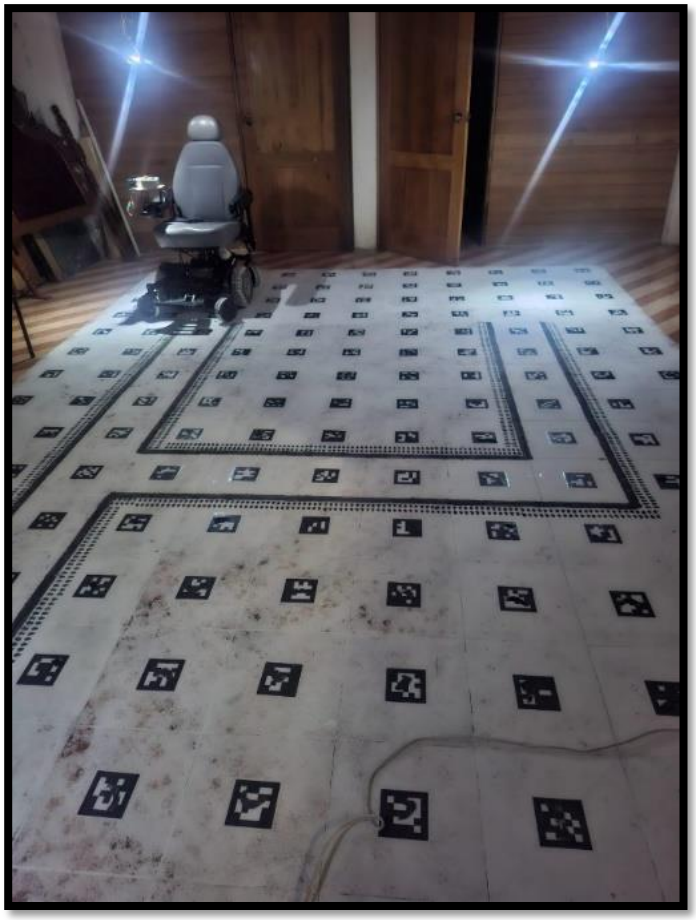

Figura 24:Pista completa

- **Pruebas con LiFi:**

Como se puede observar en las Figura 25, en la aplicación se muestra en qué habitación se encuentra la silla si el receptor capta la luz del foco LiFi la silla procede a parar en la habitación. La lampara LED implementada y su respuesta como señal detectada se puede observar en la Figura 26 y la Figura 27 respectivamente.

Figura 25:Ubicación de los emisores LiFi en el entorno de pruebas

Figura 26:Foco LiFi implementado

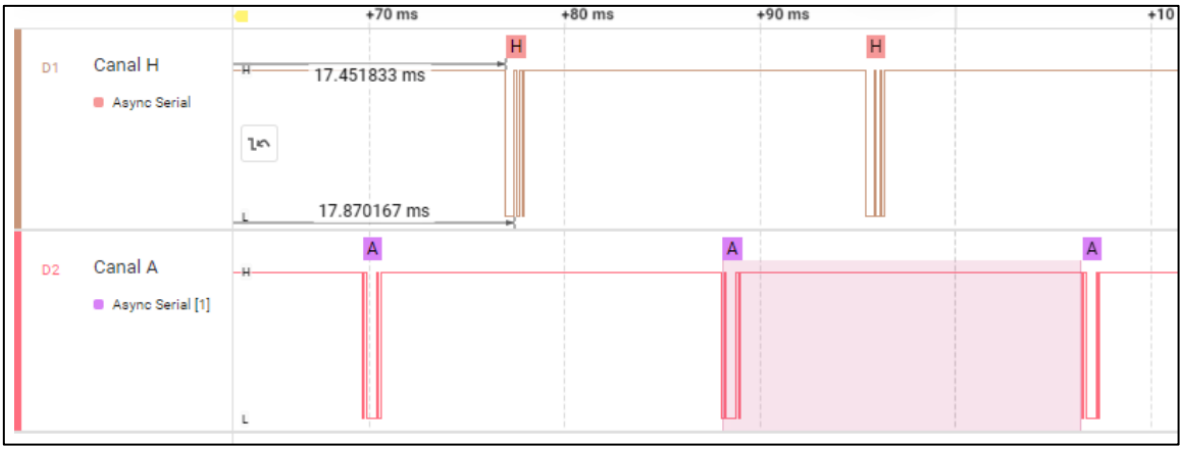

Figura 27:Señal enviada de cada habitación en la que se encuentra en el computador

El dispositivo LiFi implementado se muestra en el Anexo 5. Cuando se hace una detección de la habitación la misma es mostrada en la interfaz gráfica como se muestra en la Figura 28.

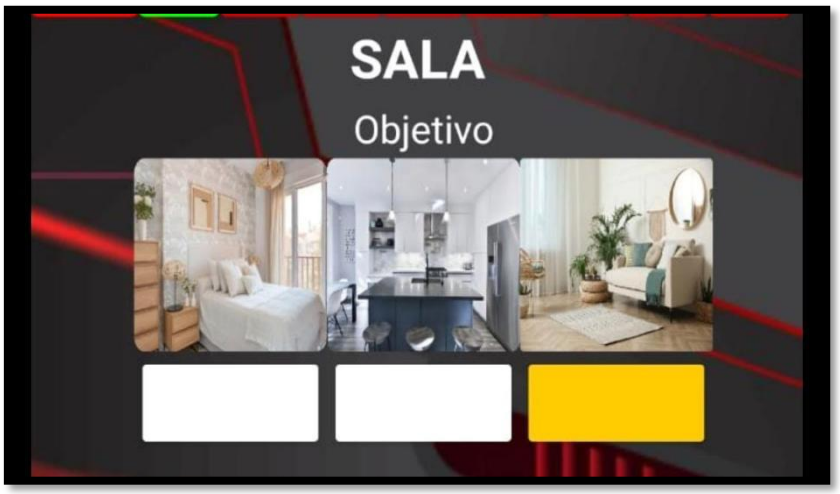

Figura 28:Habitación en la que se encuentra la silla mostrada en el aplicativo

- **Pruebas con el algoritmo aleatorio:**

Para poder realizar una comparación se procedió a implementar un algoritmo aleatorio en un entorno determinado por marcadores Aruco. La cámara de esta manera tiene la capacidad de seguir el marcador y lo selecciona de manera aleatoria de los que son visibles en el entorno como se puede observar en la Figura 29. La detección de los arucos vistos por la cámara se puede observar gráficamente en Figura 30.

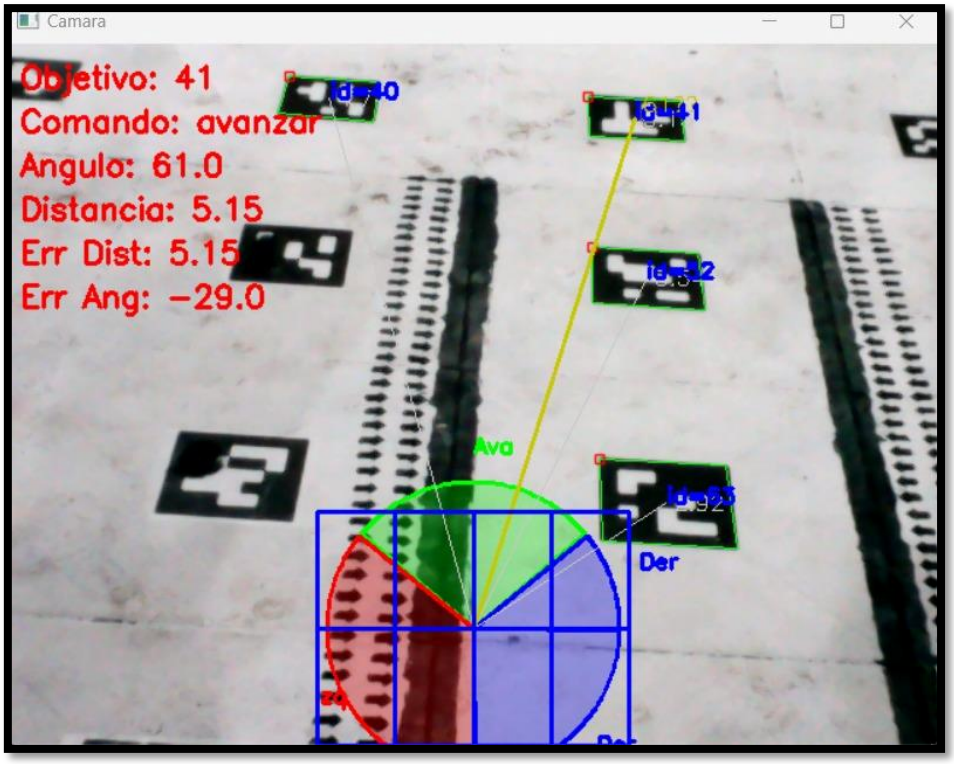

Figura 29:Vista de arucos desde la cámara

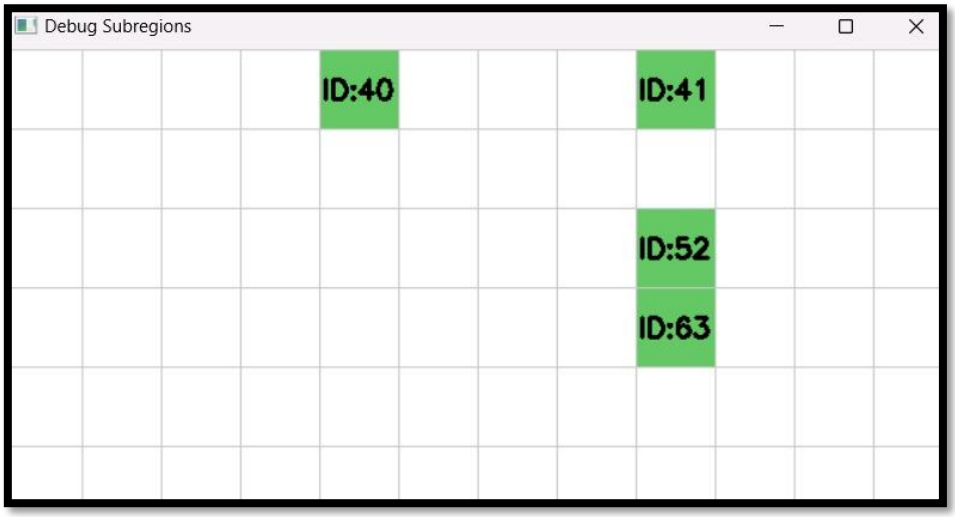

Figura 30:Marcadores detectados en un instante de captura.

4.1.3 Fase 3: Evaluación

Se evaluó el funcionamiento del sistema final mediante pruebas de desplazamiento en un entorno controlado. El objetivo fue medir el tiempo que tarda la silla de ruedas en completar trayectorias predefinidas utilizando distintas configuraciones.

En la primera configuración, el sistema utilizó tecnología LiFi junto con un algoritmo de navegación aleatorio. El tiempo promedio registrado para completar el trayecto fue de 242,76 segundos**.** En la segunda configuración, se integró visión artificial con aprendizaje por refuerzo. Con esta combinación, el sistema logró completar el mismo trayecto en 78,26 segundos**.**

La reducción en el tiempo de desplazamiento refleja una mejora significativa en la eficiencia, capacidad de adaptación y autonomía del sistema. Estos resultados confirman que la integración de visión artificial y aprendizaje por refuerzo permite una navegación más precisa y rápida en interiores, lo que favorece la movilidad asistida de personas con discapacidad.

- **Prueba de normalidad en los diferentes entornos.**

Se realizó la prueba de normalidad para determinar si los datos obtenidos por parte de las tecnologías Machinen Learning y de forma aleatoria, tienen una distribución normal.

Ho: Los datos de tiempo obtenidos tienen una distribución normal
Hi: los datos de tiempo obtenidos no tienen una distribución normal

En la Tabla 13, se muestra el p-valor como resultado de la prueba de normalidad, en la tecnología Machine Learning es mayor a 0.05 y en la aleatoria es menor a 0.05 por lo tanto los datos corresponden a una distribución normal y a una distribución no normal respectivamente. En este sentido se aplicó una prueba no paramétrica de Wilcoxon para muestras relacionadas.

Tabla 13:Prueba de normalidad en Machine Learning y Aleatorio

	P-Valor	
Variables	**Machine Learning**	**Aleatorio**
Tiempo de desplazamiento	0.053	<0.001

- **Prueba de Wilcoxon para muestras relacionadas entre Machine Learning y aleatorio**

Ho: La mediana de los tiempos de desplazamiento con Machine Learning y el algoritmo de desplazamiento aleatorio son iguales.
Hi: La mediana de los tiempos de desplazamiento con Machine Learning y el algoritmo de desplazamiento aleatorio son diferentes.

Luego de realizar la prueba de Wilcoxon para muestras relacionadas, como se puede observar en la Tabla 14 se rechazó la hipótesis nula, lo cual indica que existen diferencias estadísticas significativas entre los tiempos de desplazamiento de la silla de ruedas registrados con la tecnología de 'Machine Learning' y el método 'Aleatorio' al trasladarla de un punto A un punto B.

Tabla 14:Pruebas de Wilcoxon

Hipótesis nula	**Prueba**	**P-valor**	**Decisión**
La mediana de diferencias entre Machine Learning y aleatorio es igual a 0	Prueba de rangos con signo de Wilcoxon para pruebas relacionadas	0,000	Se rechaza la hipótesis nula

Al analizar estas diferencias mostradas en la Tabla 15:Descriptivo estadístico de las tecnologías, se observa de manera consistente que la tecnología basada en 'Machine Learning' produce tiempos más bajos y estables en comparación con el enfoque aleatorio. Esto sugiere que el uso de algoritmos de aprendizaje automático ofrece un mejor rendimiento en la tarea evaluada, al optimizar la trayectoria o decisiones de movimiento, haciendo el proceso no solo más rápido sino también más predecible y eficiente.

Tabla 15:Descriptivo estadístico de las tecnologías

Variable	**Media**	**Mediana**
Machine Learning	78,26	77,80
Aleatorio	242,76	223,45

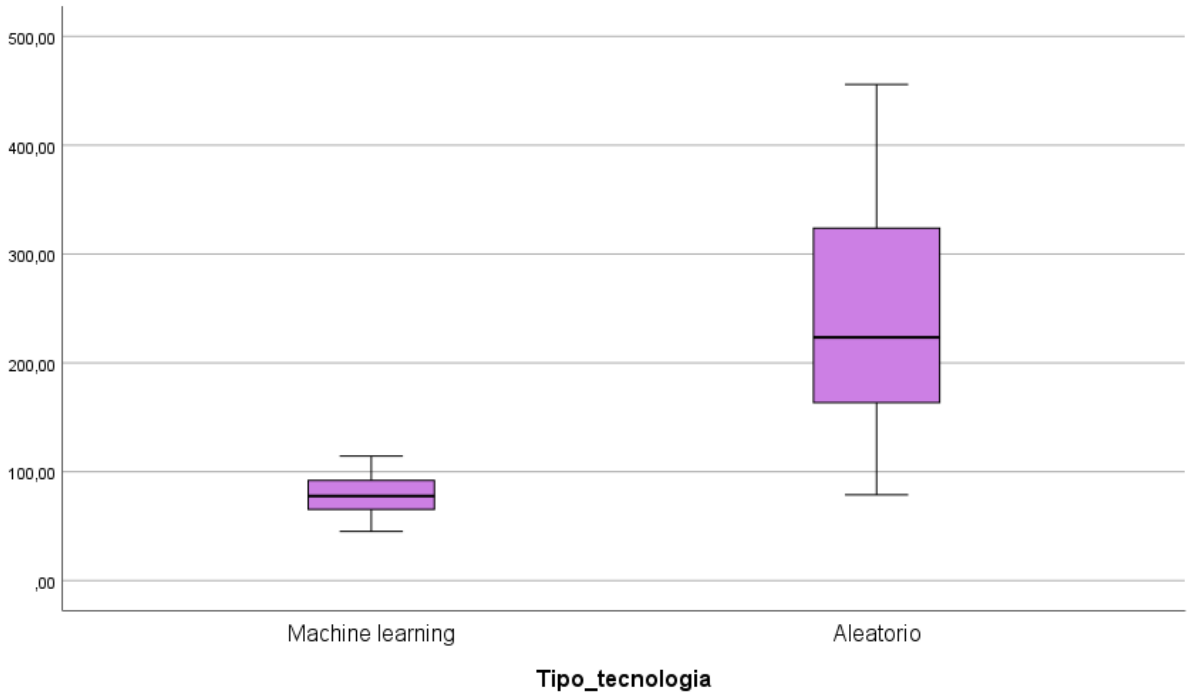

Figura 31:Diagrama de caja entre Machine Learning y Aleatorio

En la Figura 31 se muestran los diagramas de cajas donde se ve las diferencias en la dispersión y tendencia central entre las tecnologías Machine Learning y Aleatorio. Se observa que en el aleatorio existe una mayor dispersión de los datos y los valores de los extremos más alejados. Por el otro lado, Machine Learning tiene los datos con una menor variabilidad y no presenta valores atípicos. Esta brecha que existe entre ambas tecnologías sugiere una ventaja a favor de la tecnología Machine Learning.

CAPÍTULO V

5.1 CONCLUSIONES

- Con el desarrollo de este trabajo de investigación se implementó un sistema de navegación en interiores para asistir en la movilidad de personas con discapacidad que utilizan sillas de ruedas mediante tecnología LiFi y Machine Learning.

- Se diseñó un sistema de navegación integrado con LiFi capaz de ejecutar algoritmos de aprendizaje automático como redes neuronales convolucionales y Q-Learning. La arquitectura del dispositivo se diseñó con tarjetas de desarrollo como Raspberry Pi, Arduino Nano junto a la electrónica necesaria. Además, se utilizó un mecanismo basado en servomotores para el control de la movilidad de la silla de ruedas.

- Se implementó el sistema de navegación en una silla de ruedas eléctrica, integrando tanto el módulo de control con Machine Learning y el sistema de localización en interiores mediante LiFi. Durante las pruebas de funcionamiento, el sistema respondió adecuadamente a los comandos generados por los algoritmos. Además, el sistema de detección con LiFi funcionó de la manera esperada para detectar la silla de ruedas en el entorno de prueba.

- El funcionamiento del sistema implementado fue evaluado en un entorno interior controlado mediante múltiples pruebas, lo que permitió verificar su desempeño en condiciones similares a las reales. Los resultados demostraron que la integración de Machine Learning y LiFi mejora los tiempos de desplazamiento y la precisión en la navegación de 242.76 a 78.26 segundos, confirmando la viabilidad de esta solución como apoyo a la movilidad autónoma de personas con discapacidad en escenarios interiores.

5.2 RECOMENDACIONES

- Se recomienda investigar otras alternativas o combinaciones de comunicación que se puedan complementar con el LiFi en situación de baja luminosidad o grandes distancias

- Dado que las ruedas pequeñas presentan dificultades de direccionamiento, se sugiere trabajar en un diseño de ruedas omnidireccionales para facilitar la maniobrabilidad, especialmente en espacios reducidos.

- Se recomienda emplear servomotores de mayor torque y menor tiempo de respuesta. Estos deben estar diseñados para aplicaciones de movimiento rápido y preciso, con el fin de reducir el retardo percibido en la manipulación del joystick.

- Se recomienda evaluar en futuros trabajos la posibilidad de combinar el control mecánico externo con una intervención mínima en la electrónica interna del joystick, como la lectura directa de señales analógicas, lo cual permitiría un control más rápido sin perder la garantía del dispositivo.

- Se recomienda el uso de otro tipo de cámara, como la cámara de lente tipo ojo de pez. Esto debido a la posibilidad de abarca el entorno de mejor manera previniendo la pérdida del objetivo de seguimiento por el movimiento de la silla de ruedas.

- El sistema de control del joystick representa un área de estudio que podría abordarse como un proyecto de tesis independiente. Para futuros trabajos, se recomienda optimizar el control del joystick mediante técnicas de retroalimentación en tiempo real, mejorar la respuesta de los servomotores, y rediseñar el sistema de tracción de las ruedas para facilitar la movilidad y reducir retardos.

BIBLIOGRAFÍA

[1] P. Ferreira, «Autonomous Navigation of Wheelchairs in IndoorEnvironments using Deep Reinforcement Learning andComputer Vision,» Universidade Federal de Pelotas -UFPEL, Brasil, 2023.

[2] A. P. Vaishanth R, «Development of a Modular Real-time Shared-control System for a Smart Wheelchair,» USA, 2022.

[3] C. Sevastopoulos, «An RGB-D Fusion System for Indoor Wheelchair Navigation,» Grecia , 2023.

[4] T.-H. N. Ba-Viet N, «Positioning an electric wheelchair in 2D grid map based on natural landmarks for navigation using Q-learning,» *Indonesian Journal of Electrical Engineering and Computer Science,* vol. 31, pp. 115-125, 2023.

[5] V. Gallo, «Design and Characterization of a Powered Wheelchair Autonomous Guidance System,» Italia, 2024.

[6] CNID, «Estadisticas de discapacidad,» Diciembre 2024. [En línea]. Available: https://www.consejodiscapacidades.gob.ec/estadisticas-de-discapacidad/. [Último acceso: 27 Mayo 2025].

[7] M. Arroyo, «LiFi: qué es, ventajas, limitaciones y casos de uso de la tecnología para tener conexión a Internet con luz,» Guayaquil, 2018.

[8] Xataca, «LiFi: qué es, ventajas, limitaciones y casos de uso de la tecnología para tener conexión a Internet con luz,» 16 Jluio 2019. [En línea]. Available: https://www.xataka.com/investigacion/lifi-que-ventajas-limitaciones-casos-uso-tecnologia-para-tener-conexion-a-internet-luz. [Último acceso: 27 05 2025].

[9] M. M. O. Arroyo, «Diseño e implementación de un sistema de control electrónico de una silla de ruedas eléctrica con ubicación GPS y mando local,» Universidad Salesiana, Guayaquil, 2018.

[10] T. A, «Adaptación y Diseño de un Nuevo Sistema de Interfaz para una Silla de Ruedas,» Universidad de Valladolid, Valladolid, 2015.

[11] V. A, «Experimentación de la tecnología LIFI en plicaciones de localización en interiores para aparatos moviles,» Universidad Nacional de Chimborazo, Riobamba, 2022.

[12] F. A. G. Balarezo, «Diseño e implementación de una red de comunicación a través de lifi para comparar el rendimiento con la red wifi para entronos cerrados,» Universidad Nacional de Chimborazo, Riobamba, 2017.

[13] V. B, «Seguimiento Autonomo de Carril con Drones basado en Aprendizaje Profundo y Aprendizaje por Refuerzo,» Universidad Rey Juan Carlos, Madrid, 2024.

[14] S. Medical, «Sillas de Ruedas Eléctricas vs. Manuales: ¿Cuál es la elección correcta para ti?,» 01 Abril 2024. [En línea]. Available: https://www.sunrisemedical.es/blog/sillas-electricas-vs-manuales. [Último acceso: 27 Mayo 2025].

[15] Q. W. Yifan Xu, Socially-Aware Shared control navigation for assistive mobile robots in the built enviroment, Chicago: arXiv preprint arXiv:2405.17279, 2024.

[16] R. R. Sharma, «Li-fi Technology Transmission of data through light,» *IJCTA,* vol. 5, nº 2229-6093, p. 1, 2014.

[17] B. S. A. López, Redes LiFi y su implementación en la nueva generación de redes moviles 5g, Colombia, 2020.

[18] LiFi (Light Fidelity) & its Applications, FN Division, TEC .

[19] E. Alpaydin, Aprendizaje automático:La nueva IA, London: Prensa del MIT, 2016.

[20] J. P., « Fundamentals of Machine Learning an Introduction to Neural Network,» HiTeX Press, USA, 2024.

[21] M. D. C. J. Escobar R, «Análisis de suelos utilizando redes neuronales en las florícolas de Rosas del Sector Norte de la Provincia de Cotopaxi,» *RECIMUNDO,* vol. 5, nº 2588-073X, pp. 316-330, 2021.

[22] C. C. Aggarwal, «Neural Networks and Deep Learning,» Springer, NY, USA, 2023.

[23] L. R. Flórez, Las redes neuronales artificiales, España, 2008.

[24] C. B. Natalia, «Redes neuronales convolucionales y aplicaciones.,» Universidad Complutense Madrid, Madrid, 2022.

[25] C. D, «Clasificación de redes neuronales artificiales,» 13 Julio 2017. [En línea]. Available: https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/#:~:text=Red%20neuronal%20recurrente%20(RNN),que%20la%20red%20tenga%20memoria. [Último acceso: 28 Mayo 2025].

[26] R. H. Sandra, «Aprendizaje por refuerzo en sistemas robóticos.,» Universidad Politécnica de Madrid, Madrid, 2019.

[27] F. y. A. Internet de las Cosas y Visión Artificial, «Alvear V.,» Enfoque UTE, Ibarra, 2017.

[28] S.-M. Luna, «Event-based Detector of ArUco Markers,» UMH, Alicante, 2023.

[29] T. Vasilis, «Raspberry Pi Zero W Wireless Projects,» Packt Publishing Ltd, 2017.

[30] E. Steren, «Steren,» 2022. [En línea]. Available: https://www.steren.com.ec/placa-de-desarrollo-nano.html. [Último acceso: 24 Mayo 2025].

[31] P. Mobility, «Jazzy Select series,» 22 Noviembre 2022. [En línea]. Available: www.pridemobility.com. [Último acceso: 28 Mayo 2025].

[32] J. Wang, «"LiFi-based indoor navigation system for people with disabilities,» 2022.

[33] Y. Zhang, «A machine learning-based indoor navigation system for wheelchair users,» *IEEE Transactions on Neural Networks and Learning Systems,* 2021.

[34] J. Alcubierre, «Silla de Ruedas Inteligente Controlada por Voz,» Fundación ONCE, España, 2005.

[35] E. T. Hernandez, Predicciones de criptomonedas utilizando un modelo de prediccion basado en el algoritmo de bosque aleatorio, Colombia, 2024.

[36] V. Diego, «Diseño de un prototipo de una silla de ruedas para personas minusválidas no videntes,» *XIX Jornadas en Ingeniería Eléctrica y Electrónica,* vol. 19, nº JIEE, p. 85, 2005.

[37] E. M. Pérez, Microcontroladores PIC. Sistema integrado para el autoaprendizaje, España: Marcombo, 2007.

ANEXOS

Anexo 1. Diagrama de conexiones del dispositivo emisor LiFi. Se puede observar el módulo emisor junto a los canales con la conexión para cada lámpara del módulo.

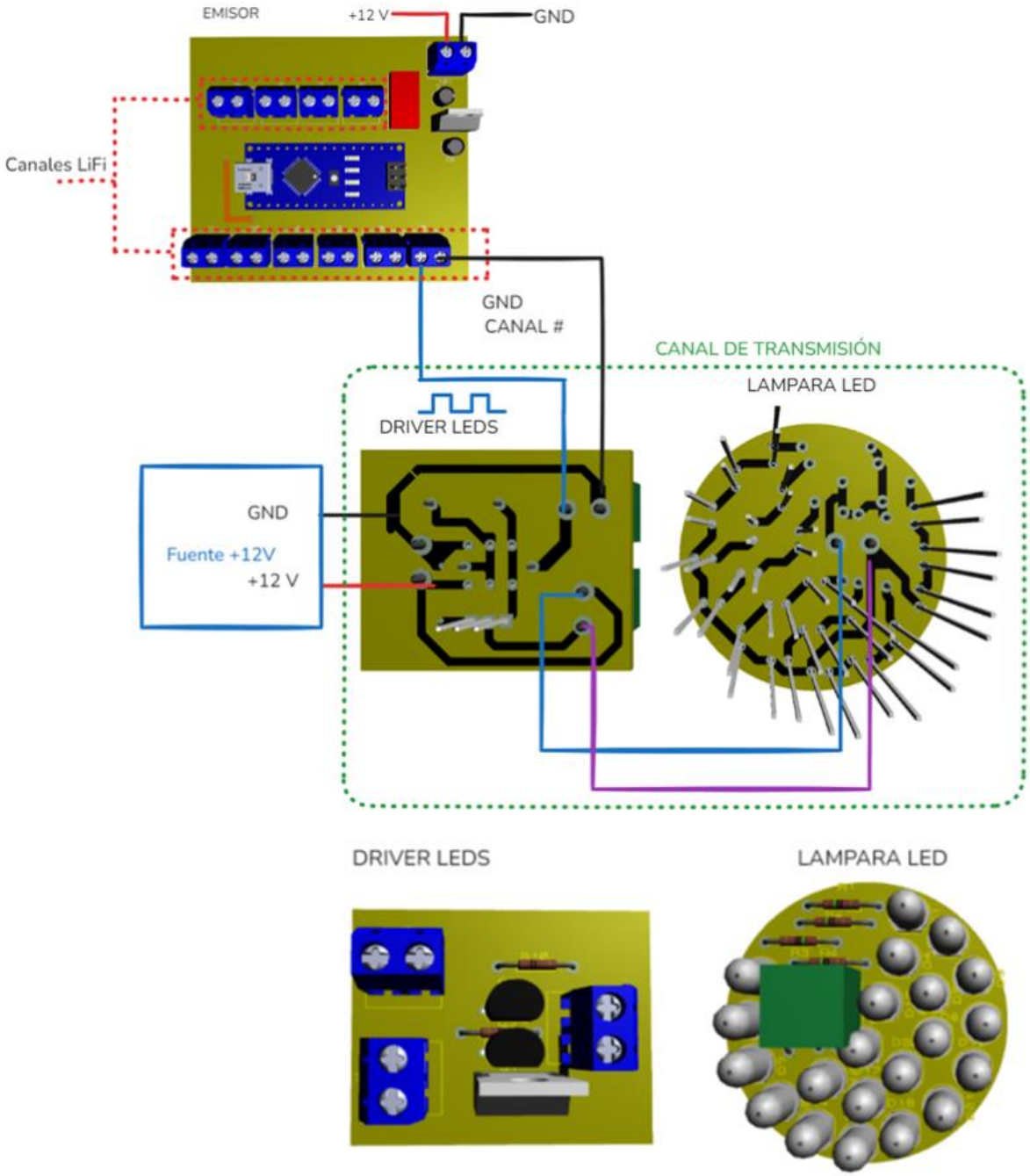

Figura 32: Modelo en 3D del dispositivo emisor y las conexiones de cada canal

Anexo 2. Características técnicas del joystick integrado en la silla de ruedas. El parámetro para tomar en cuenta para el desarrollo del proyecto es la fuerza necesaria para operar dicho dispositivo es de 1 Newton.

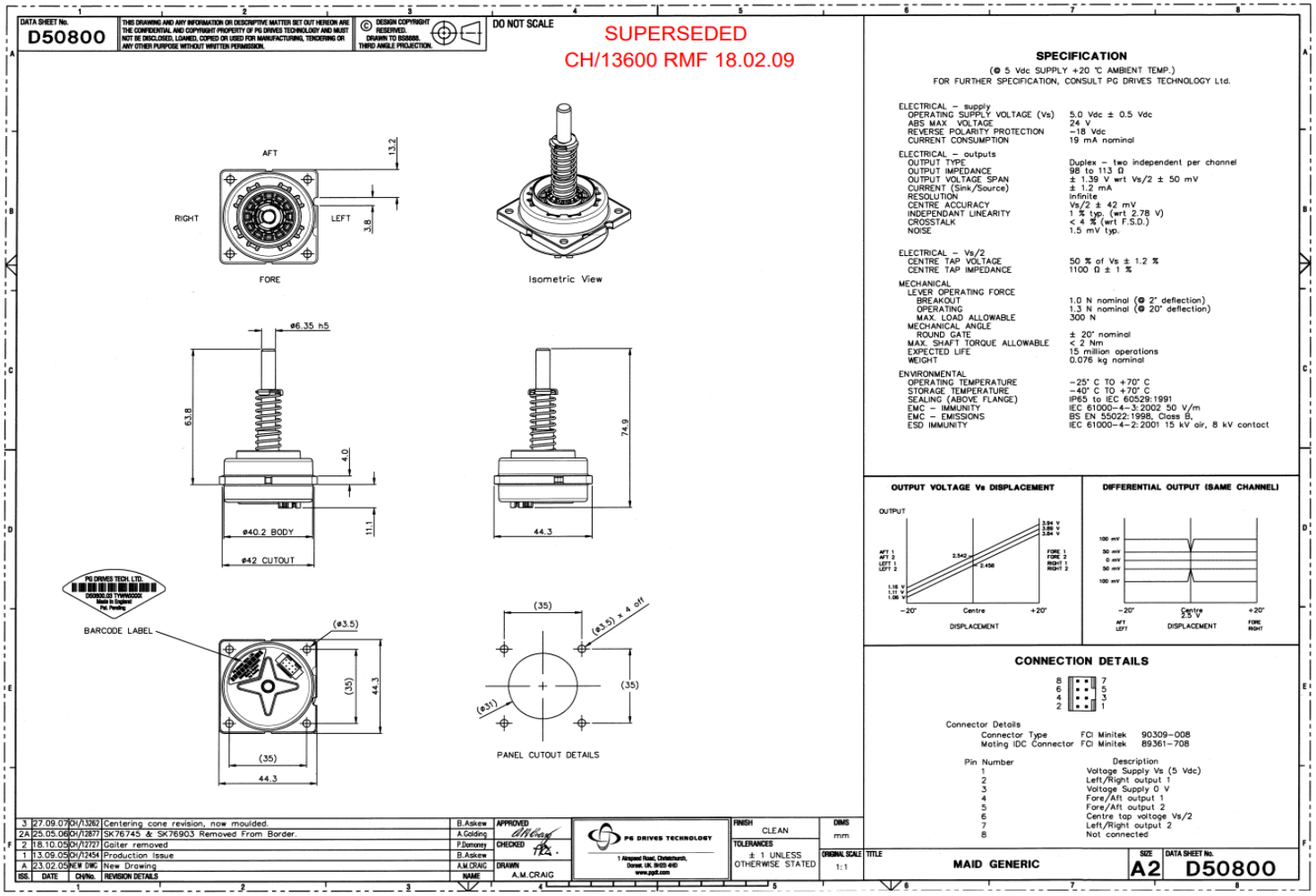

Figura 33 Especificaciones técnicas del sensor del joystick silla de ruedas

Anexo 3. Esquema de conexiones de la placa de procesamiento y control. Se puede observar los módulos principales conformados por una tarjeta Raspberry pi Zero W y un Arduino Nano. La comunicación entre los 2 dispositivos se hace mediante UART.

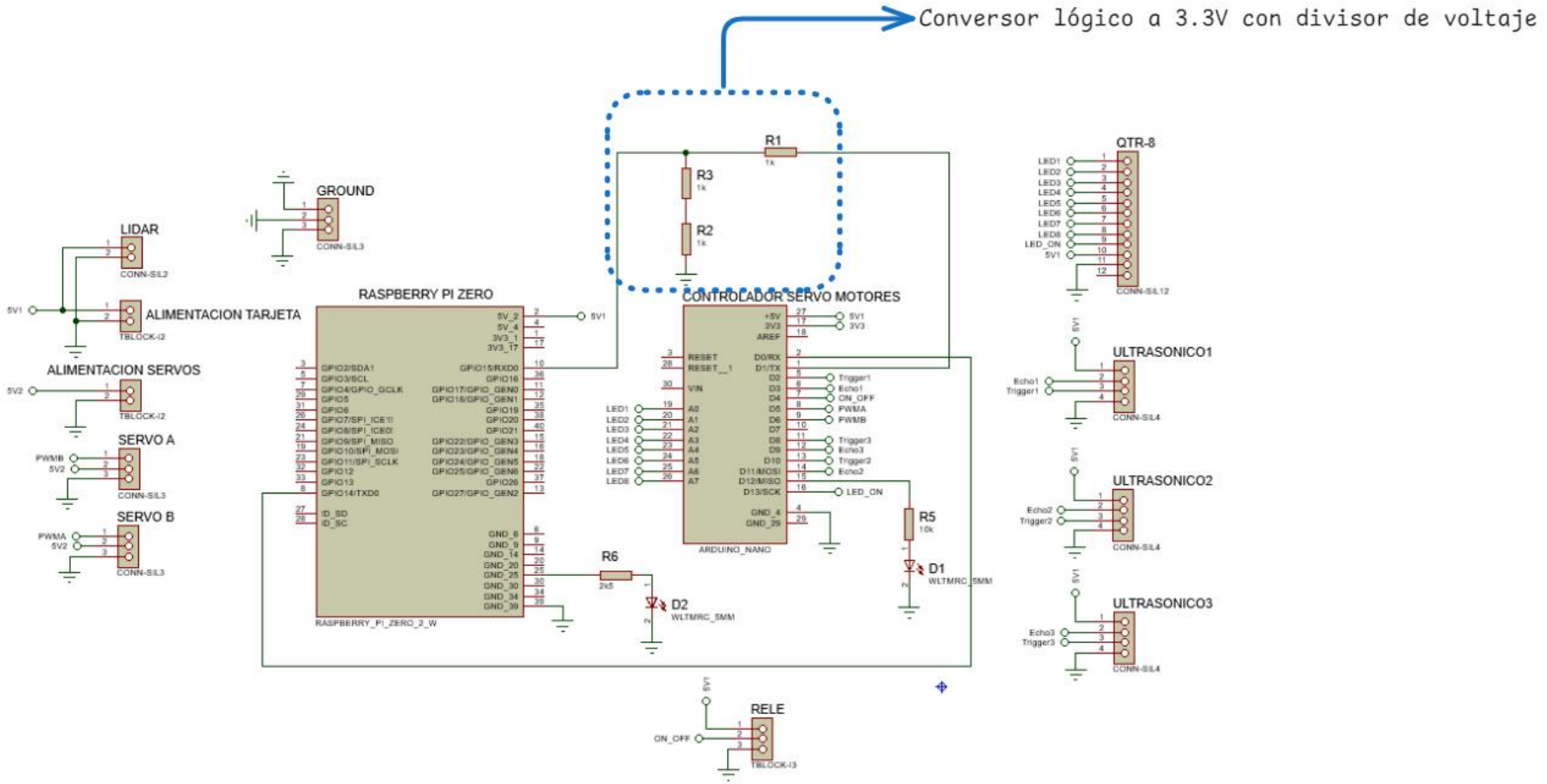

Figura 34: Esquema de diseño en Proteus 8.13 de la placa de control y navegación.

Anexo 4. Esquema de conexiones de la placa de control y navegación. Se puede observar las interfaces y su conexión los sensores y actuadores.

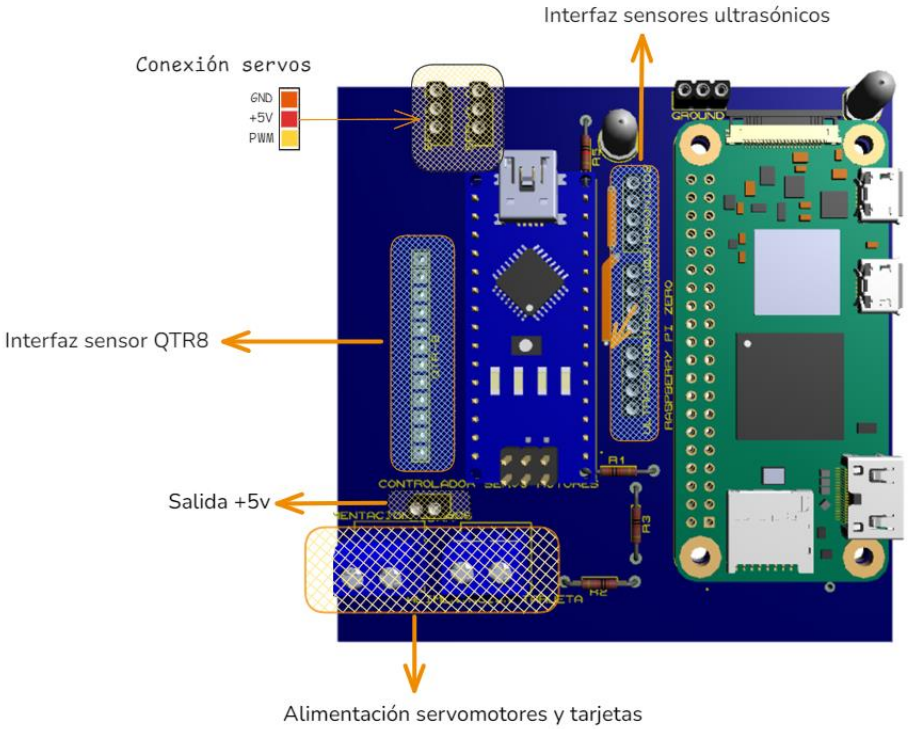

Figura 35: Diagrama de conexiones de la placa de control y navegación

Anexo 5. El diseño implementado del dispositivo LiFi. En la imagen A se puede observar la placa de transmisión. En la imagen B se puede observar el dispositivo receptor.

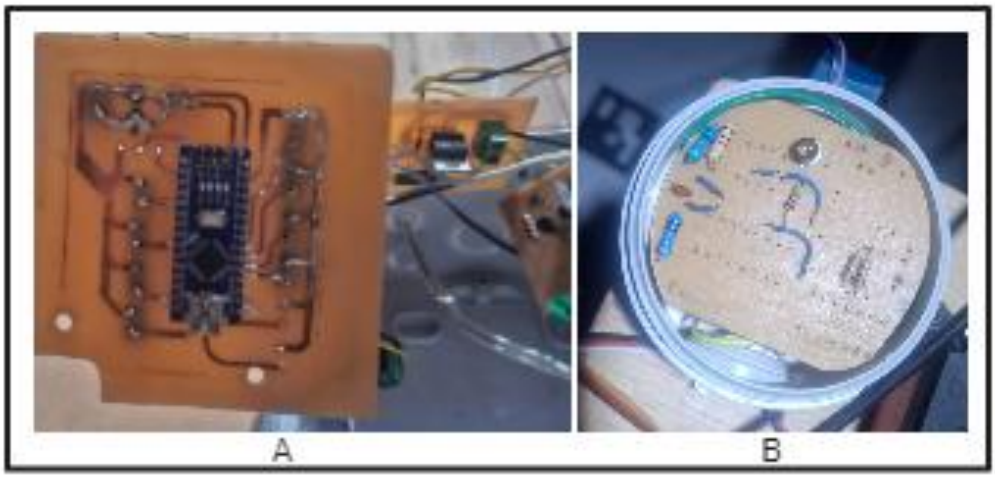

Figura 36:Dispositivo LiFi

Anexo 6. El esquema de conexiones de la placa y los pines con los periféricos integrados en la placa de desarrollo.

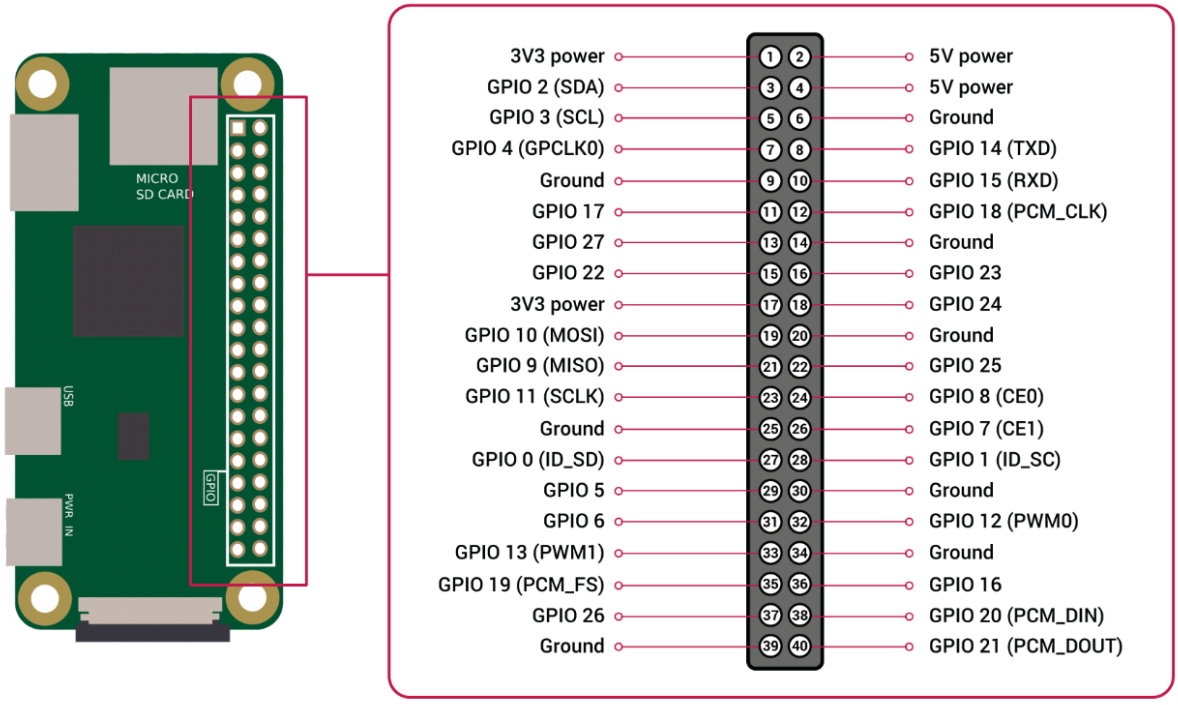

Figura 37: Pinout de periféricos de la Raspberry Pi Zero W

Anexo 7. Esquema de conexiones de los puertos del microcontrolador Atmega328p con las etiquetas de la placa Arduino Nano.

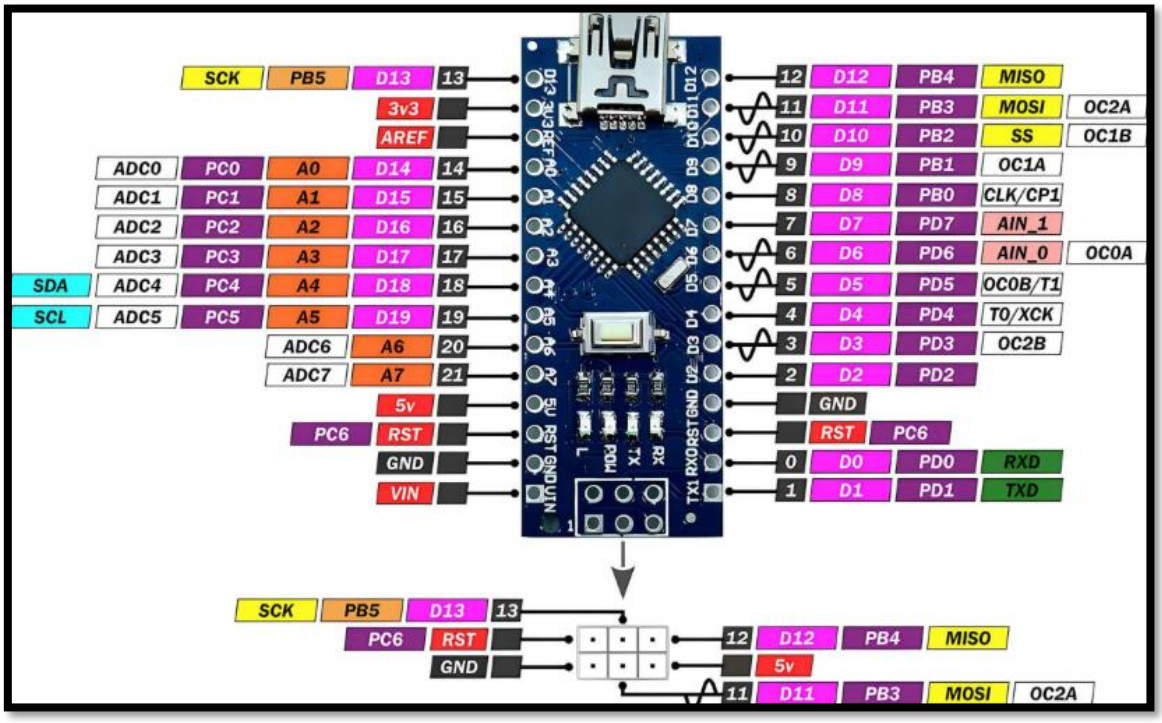

Figura 38: Pinout de la tarjeta Arduino Nano

Anexo 8. El entrenamiento muestra la precisión en el modelo durante el entrenamiento y la validación con los datos de la prueba.

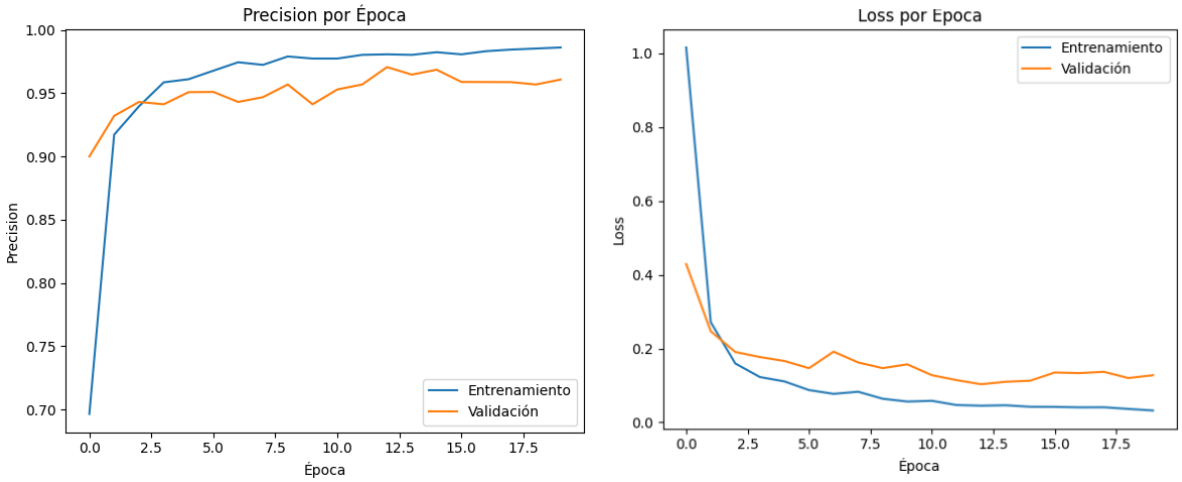

Figura 39: Gráficas del entrenamiento

Anexo 9. Diseño de las placas de los módulos para el proyecto. En la fotografía se puede observar el diseño de las placas de lámparas LED del módulo LiFi.

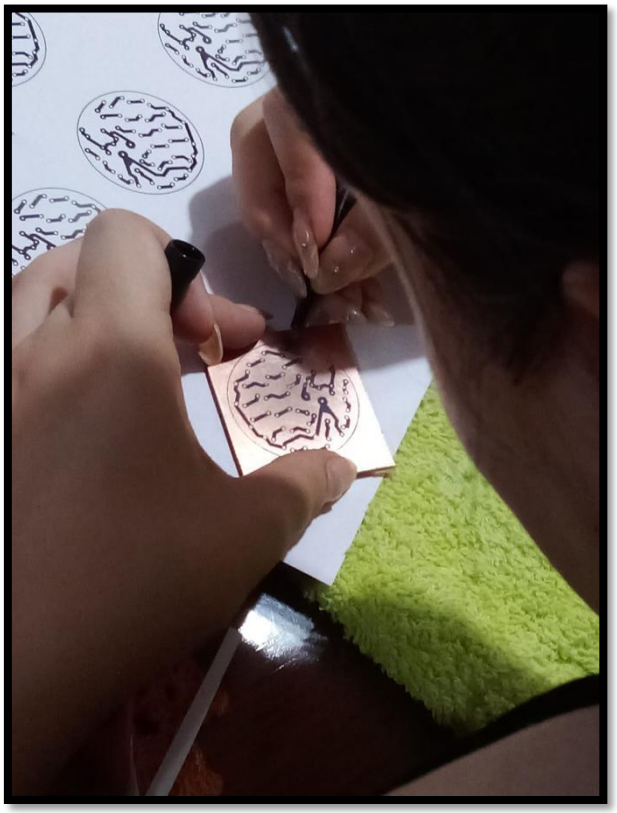

Figura 40: Diseño de las placas LiFi

Anexo 10. Pistas de prueba en las pruebas preliminares. Todas las pistas se pintaron con pintura mate para evitar reflectancias debido a las luminarias.

Figura 41: Pistas diseñadas para pruebas preliminares

Anexo 11. Esquema de bloques de la programación gráfica en App Inventor se puede observar los comandos enviados mediante http hacia la Raspberry Pi.

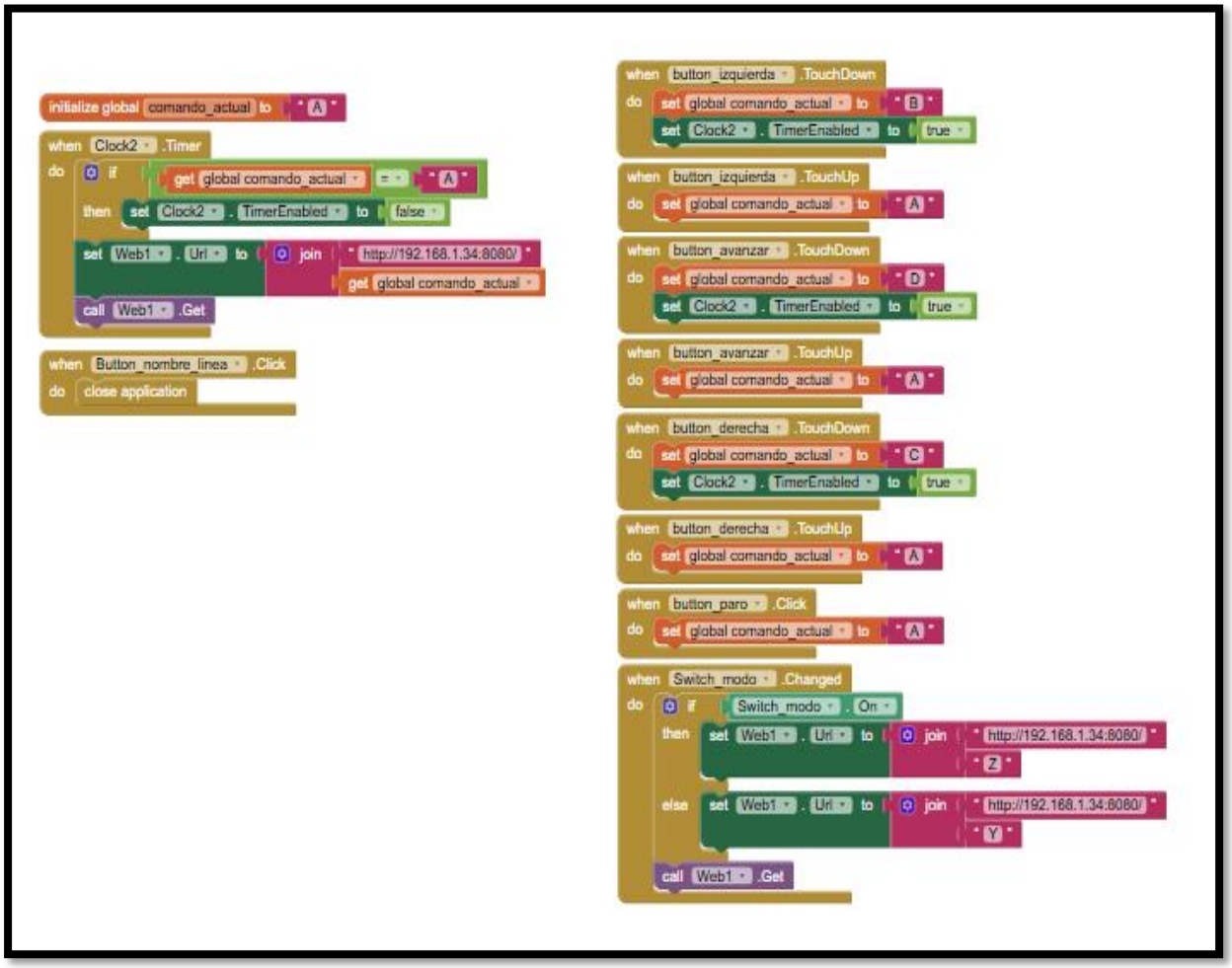

Figura 42: Esquema de bloques de programación de la interfaz de usuario en App Inventor

Anexo 12: Firmware del módulo emisor LiFi. La librería principal es *serial_lifi.c* aquí se implementa los canales mediante el uso de temporización para emitir una señal codificación UART a 9600 baudios para emitir 8 canales de transmisión.

```
-----------------------------------------------------------------
/*Archivo fuente main.c*/
-----------------------------------------------------------------
#include "main_task.h"
int main() {
  main_task();
  while (1) {}
  return 0;
}
-----------------------------------------------------------------
/*Archivo encabezado main_task.h*/
-----------------------------------------------------------------
#ifndef MAIN_TASK_H
#define MAIN_TASK_H
#include "avr_system.h"
void main_task();
#endif
-----------------------------------------------------------------
/*Archivo fuente main_task.c*/
-----------------------------------------------------------------
#include "main_task.h"
#include "avr_gpio.h"
#include "serial_lifi.h"
#include <stdint.h>
#include <util/delay.h>
void main_task() {
  /* Inicialización de periféricos */
  cli();
  avr_gpio_init();
  serial_init();
  sei();
  // Mensajes a enviar para cada canal
  char *messages[] = {"0", "1", "2", "3", "4", "5", "6", "7"};
  uint8_t num_channels = sizeof(messages) / sizeof(messages[0]);
  while (1) {
    for (uint8_t channel = 0; channel < num_channels; channel++) {
      uint8_t message_length = 0;
      // Calcula la longitud del mensaje
      while (messages[channel][message_length] != '\0') {
        message_length++;
      }
      // Envía el mensaje al canal actual
      serial_write(messages[channel], message_length, channel);
      // Espera 100 ms antes de pasar al siguiente canal
```

```
      _delay_ms(100);
    }
   }
}
-----------------------------------------------------------------
/*Archivo encabezado avr_gpio.h*/
-----------------------------------------------------------------
#ifndef AVR_GPIO_H
#define AVR_GPIO_H
#include "avr_system.h"
void avr_gpio_init();
#endif
-----------------------------------------------------------------
/*Archivo fuente avr_gpio.c*/
-----------------------------------------------------------------
#include "avr_gpio.h"
#include <avr/io.h>
void avr_gpio_init() {
  /*Puertos del uart
   * Tx -> PD1 OUT
   * Rx -> PD0 IN*/
  DDRD |= (1 << PD1);
  DDRD &= ~(1 << PD0);
}
-----------------------------------------------------------------
/*Encabezado Libreria LiFi*/
-----------------------------------------------------------------
#ifndef SERIAL_LIFI_H
#define SERIAL_LIFI_H
#include <stdint.h>
#define BAUDRATE 9600
#define ONE_BIT_CYCLES (uint8_t)(2000000 / BAUDRATE)
#define SAMPLING_DELAY_CYCLES (uint8_t)(ONE_BIT_CYCLES / 2)
#define SERIAL_RECV_BUF_SIZE 64
#define SERIAL_SEND_BUF_SIZE 32
#define FORCE_SEND 0
#define BITS_WAITING_AFTER_RECV 5
// Evita conflictos con <stdbool.h>
#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif
#define TX_PIN 4
// Definir  los canales de transmisión
typedef enum {  CHANNEL0,  CHANNEL1,  CHANNEL2,
  CHANNEL3,  CHANNEL4,  CHANNEL5,  CHANNEL6,  CHANNEL7
} channels_TypeDef;
```

```
// Enumeración de los estados del serial
typedef enum { IDLE_MODE = 0, RECV_MODE, SEND_MODE } serialStateTypeDef;
// Declaraciones de funciones
void serial_init();
void serial_listen(serialStateTypeDef listening_mode);
uint8_t serial_available();
void serial_read();
uint8_t serial_prepare_to_send(uint8_t channel); // Modificado para incluir el canal
void serial_write(char *data_array, uint8_t data_size, uint8_t channel);
void serial_init_channels();
void serial_init_timer();
#endif // SERIAL_LIFI_H
----------------------------------------------------------------
/*Archivo fuente serial_lifi.c*/
----------------------------------------------------------------
#include "serial_lifi.h"
#include "avr_system.h"
#include <avr/io.h>
#include <stdint.h>
// Buffers para recepción
volatile uint8_t serial_recv_buffer[SERIAL_RECV_BUF_SIZE];
volatile uint8_t serial_recv_buffer_last_log_item = 0;
volatile uint8_t serial_recv_buffer_last_pro_item = 0;
volatile uint8_t recv_bit_num = 0;
volatile uint8_t recv_byte_buffer;
// Buffers para envío
volatile uint8_t serial_send_buffer[SERIAL_SEND_BUF_SIZE];
volatile uint8_t serial_send_buffer_last_added = 0;
volatile uint8_t serial_send_buffer_to_send_now = 0;
volatile uint8_t send_bit_num = 0;
volatile uint8_t send_byte_buffer;
volatile serialStateTypeDef serial_mode = IDLE_MODE;
volatile uint8_t current_channel = 0;
void serial_init_channels() {
  DDRC |= (1 << PC0) | (1 << PC1) | (1 << PC2) | (1 << PC3);
  DDRB |= (1 << PB4) | (1 << PB5);
  DDRD |= (1 << PD2) | (1 << PD7);
  EICRA &= ~(_BV(ISC10));
  EICRA |= _BV(ISC11);
  EIFR |= _BV(INTF1);
  EIMSK |= _BV(INT1);
}
void serial_init_timer() {
  TCCR2A = _BV(WGM21);
  TCCR2B = 0;
  TCNT2 = 0;
  TIMSK2 = _BV(OCIE2A);
}
void serial_init() {
```

```
  serial_mode = IDLE_MODE;
  serial_init_channels();
  serial_init_timer();
}
uint8_t serial_available() {
  uint8_t temp_last_log_item = serial_recv_buffer_last_log_item;
  uint8_t temp_last_pro_item = serial_recv_buffer_last_pro_item;

  if (temp_last_pro_item == temp_last_log_item) {
    return (temp_last_log_item - temp_last_pro_item);
  } else {
    return (SERIAL_RECV_BUF_SIZE - temp_last_pro_item + temp_last_log_item);
  }
}

void serial_listen(serialStateTypeDef listening_mode) {
  if ((listening_mode != IDLE_MODE) && (listening_mode != RECV_MODE))
    listening_mode = IDLE_MODE;
  if (listening_mode == IDLE_MODE) {
    TCCR2B = 0;
    EIFR |= _BV(INTF1);
  }
  serial_mode = listening_mode;
  EIMSK |= _BV(INT1);
}

uint8_t serial_prepare_to_send(uint8_t channel) {
  if (serial_send_buffer_to_send_now == serial_send_buffer_last_added)
    return FALSE;
  EIMSK &= ~(_BV(INT1));
  serial_mode = SEND_MODE;
  send_byte_buffer = serial_send_buffer[serial_send_buffer_to_send_now];
  serial_send_buffer_to_send_now++;
  if (serial_send_buffer_to_send_now == SERIAL_SEND_BUF_SIZE)
  serial_send_buffer_to_send_now = 0;
  send_bit_num = 0;

  TCCR2B &= ~(_BV(CS22) | _BV(CS21) | _BV(CS20));
  TCNT2 = 0;
  OCR2A = ONE_BIT_CYCLES;
  TCCR2B = _BV(CS21);

  current_channel = channel;

  if (channel == 0)
    PORTC &= ~(1 << PC0);
  else if (channel == 1)
    PORTC &= ~(1 << PC1);
  else if (channel == 2)
```

```
    PORTC &= ~(1 << PC2);
  else if (channel == 3)
    PORTC &= ~(1 << PC3);
  else if (channel == 4)
    PORTB &= ~(1 << PB4);
  else if (channel == 5)
    PORTB &= ~(1 << PB5);
  else if (channel == 6)
    PORTD &= ~(1 << PD2);
  else if (channel == 7)
    PORTD &= ~(1 << PD7);
  return TRUE;
}

void serial_write(char *data_array, uint8_t data_size, uint8_t channel) {
  for (uint8_t i = 0; i < data_size; i++) {
    serial_send_buffer[serial_send_buffer_last_added] = data_array[i];
    serial_send_buffer_last_added++;
    if (serial_send_buffer_last_added == SERIAL_SEND_BUF_SIZE)
      serial_send_buffer_last_added = 0;
  }
  serial_prepare_to_send(channel);
}

ISR(TIMER2_COMPA_vect) {
  if (serial_mode == SEND_MODE) {
    if (send_bit_num < 8) {
      if (send_byte_buffer & (1 << send_bit_num)) {
        if (current_channel == 0)
          PORTC |= (1 << PC0);
        else if (current_channel == 1)
          PORTC |= (1 << PC1);
        else if (current_channel == 2)
          PORTC |= (1 << PC2);
        else if (current_channel == 3)
          PORTC |= (1 << PC3);
        else if (current_channel == 4)
          PORTB |= (1 << PB4);
        else if (current_channel == 5)
          PORTB |= (1 << PB5);
        else if (current_channel == 6)
          PORTD |= (1 << PD2);
        else if (current_channel == 7)
          PORTD |= (1 << PD7);
      } else {
        if (current_channel == 0)
          PORTC &= ~(1 << PC0);
        else if (current_channel == 1)
          PORTC &= ~(1 << PC1);
```

```
      else if (current_channel == 2)
        PORTC &= ~(1 << PC2);
      else if (current_channel == 3)
        PORTC &= ~(1 << PC3);
      else if (current_channel == 4)
        PORTB &= ~(1 << PB4);
      else if (current_channel == 5)
        PORTB &= ~(1 << PB5);
      else if (current_channel == 6)
        PORTD &= ~(1 << PD2);
      else if (current_channel == 7)
        PORTD &= ~(1 << PD7);
    }
   } else if (send_bit_num == 8) {
    if (current_channel == 0)
      PORTC |= (1 << PC0);
    else if (current_channel == 1)
      PORTC |= (1 << PC1);
    else if (current_channel == 2)
      PORTC |= (1 << PC2);
    else if (current_channel == 3)
      PORTC |= (1 << PC3);
    else if (current_channel == 4)
      PORTB |= (1 << PB4);
    else if (current_channel == 5)
      PORTB |= (1 << PB5);
    else if (current_channel == 6)
      PORTD |= (1 << PD2);
    else if (current_channel == 7)
      PORTD |= (1 << PD7);
   } else if (send_bit_num == 9) {
    serial_mode = IDLE_MODE;
    serial_prepare_to_send(current_channel);
    return;
   }
   send_bit_num++;
  }
}
```

Anexo 13: Firmware de control de control de servos y lectura de sensores. Los comandos se reciben por puerto serial y el dispositivo interpreta y ejecuta la acción asociada a estos.

```
#include <Servo.h>
#include <QTRSensors.h>
```

```
char comando_registro = 0;
bool sensorUltrasonicoActivo = false;

class ControlServo {
  private:
    Servo servo_id;
    Servo servo_avance;
    const int pin_id;
    const int pin_avance;
    const int ANGULO_PARADA = 100;
    const int ANGULO_AVANCE = 40;
    const int ANGULO_GIRO_IZQUIERDA = 80;
    const int ANGULO_GIRO_DERECHA = 120;
    const unsigned long TIEMPO_LIMITE = 2000;
    unsigned long ultima_actividad;
    int angulo_actual_id = 100;
    void moverServoConPendiente(int angulo_objetivo, int paso = 1, int retardo_ms = 10) {
       while (angulo_actual_id != angulo_objetivo) {
          if (angulo_actual_id < angulo_objetivo) {
             angulo_actual_id += paso;
             if (angulo_actual_id > angulo_objetivo) angulo_actual_id = angulo_objetivo;
          } else {
             angulo_actual_id -= paso;
             if (angulo_actual_id < angulo_objetivo) angulo_actual_id = angulo_objetivo;
          }
          servo_id.write(angulo_actual_id);
          delay(retardo_ms);
       }
    }
  public:
    ControlServo(int pin1, int pin2)
      : pin_id(pin1), pin_avance(pin2), ultima_actividad(0) {}
    void iniciar() {
       servo_id.attach(pin_id);
       servo_avance.attach(pin_avance);
       detener();
    }
    void procesarComando(char comando) {
       if (comando == 'A'||  comando =='B'||comando =='C' || comando == 'D') {
          switch (comando) {
             case 'A': detener(); break;
             case 'B': moverServoConPendiente(ANGULO_GIRO_IZQUIERDA); break;
             case 'C': moverServoConPendiente(ANGULO_GIRO_DERECHA); break;
             case 'D': servo_avance.write(ANGULO_AVANCE); break;
          }
          comando_registro = comando;
          ultima_actividad = millis();
       } else {
          comando_registro = 0;
```

```
        }
    }
    void verificarInactividad() {
        if (millis() - ultima_actividad > TIEMPO_LIMITE) {
            detener();
        }
    }

    void detener() {
        angulo_actual_id = ANGULO_PARADA;
        servo_id.write(ANGULO_PARADA);
        servo_avance.write(ANGULO_PARADA);
        ultima_actividad = millis();
    }
};

ControlServo controlador(5, 6);

#define NUM_SENSORES 8
#define MUESTRAS 4
#define PIN_EMISOR_QTR 13

QTRSensorsAnalog qtra((unsigned char[]){0, 1, 2, 3, 4, 5, 6, 7},
 NUM_SENSORES, MUESTRAS, PIN_EMISOR_QTR);
unsigned int sensorValues[NUM_SENSORES];

// Pines para sensores ultrasónicos
const int Trigger1 = 2;
const int Echo1    = 3;
const int Trigger2 = 8;
const int Echo2    = 9;
const int Trigger3 = 10;
const int Echo3    = 11;
const int testigo  = 7;

void setup() {
    Serial.begin(115200);
    controlador.iniciar();
    pinMode(PIN_EMISOR_QTR, OUTPUT);
    pinMode(testigo, OUTPUT);
    digitalWrite(testigo, LOW);
    pinMode(Trigger1, OUTPUT); pinMode(Echo1, INPUT); digitalWrite(Trigger1, LOW);
    pinMode(Trigger2, OUTPUT); pinMode(Echo2, INPUT); digitalWrite(Trigger2, LOW);
    pinMode(Trigger3, OUTPUT); pinMode(Echo3, INPUT); digitalWrite(Trigger3, LOW);
    for (int i = 0; i < 400; i++) {
        qtra.calibrate();
    }
}
```

```
void loop() {
  if (Serial.available() > 0) {
    char comando = Serial.read();
    if (comando == '0') {
      sensorUltrasonicoActivo = !sensorUltrasonicoActivo;
    } else {
      controlador.procesarComando(comando);
    }
    if (comando_registro != 0) {
      digitalWrite(testigo, LOW);
      qtra.readLine(sensorValues);
      digitalWrite(testigo, HIGH);
      long d1 = 0, d2 = 0, d3 = 0;
      if (sensorUltrasonicoActivo) {
        digitalWrite(Trigger1, HIGH);
        delayMicroseconds(10);
        digitalWrite(Trigger1, LOW);
        long t1 = pulseIn(Echo1, HIGH);
        d1 = t1 / 59;
        digitalWrite(Trigger2, HIGH);
        delayMicroseconds(10);
        digitalWrite(Trigger2, LOW);
        long t2 = pulseIn(Echo2, HIGH); d2 = t2 / 59;
        digitalWrite(Trigger3, HIGH);
        delayMicroseconds(10);
        digitalWrite(Trigger3, LOW);
        long t3 = pulseIn(Echo3, HIGH); d3 = t3 / 59;
      }

      String salida = "";

      for (int i = 0; i < NUM_SENSORES; i++) {
        salida += String(sensorValues[i]);
        if (i < NUM_SENSORES - 1) salida += ",";
      }

      salida += "," + String(d1) + ","
      + String(d2) + "," + String(d3) +
       "," + String(comando_registro);
      Serial.println(salida);
    }
  }
  controlador.verificarInactividad();
}
```

Anexo 14: Clases de control de puertos seriales de la Raspberry Pi. La primera clase maneja el puerto la conexión del puerto UART. La segunda clase maneja el periférico Bluetooth integrado en la tarjeta.

```
import serial
import time

class ArduinoSerial:
    def __init__(self, port='/dev/serial0', baudrate=115200, timeout=0.1):
        self.arduino = serial.Serial(port, baudrate, timeout=timeout)
        self.arduino.flush()
    def enviar_mensaje(self, mensaje):
        self.arduino.write(mensaje.encode('utf-8'))
    def leer_respuesta(self):
        return self.arduino.readline().decode('utf-8').rstrip()
    def close(self):
            self.arduino.close()

class LifiSerial:
    def __init__(self, port='/dev/rfcomm0', baudrate=9600, timeout=0.1):
        self.lifi = serial.Serial(port, baudrate, timeout=timeout)
        self.lifi.flush()
    def leer_respuesta(self):
        return self.lifi.readline().decode('utf-8').rstrip()
    def close(self):
            self.lifi.close()
```

Anexo 15: Script de que se ejecuta para gestionar la comunicación con la interfaz , interpretar comandos de comunicación desde la Raspberry con el resto de dispositivos del proyecto.

```
import os
import csv
import threading
import time
import requests

from http.server import BaseHTTPRequestHandler, ThreadingHTTPServer
from serial_arduino import ArduinoSerial, LifiSerial

# ============================
# Parámetros Globales
# ============================
SERVER_ADDRESS      = ('192.168.1.34', 8080)
IMAGE_SERVER_URL    = "http://192.168.1.5:8082/K"
CSV_FILENAME        = "datos_entrenamiento.csv"

# Manual / automático / aleatorio
AUTOMATIC_MODE_CHANGE = True

# Ráfagas  de comandos a enviar al Arduino
TIMES_IZQ    = 20
TIMES_DER    = 20
```

```
TIMES_AVANCE = 120

BURST_CONFIG = {
    "B": {"count": TIMES_IZQ,    "stop_count": 2},
    "C": {"count": TIMES_DER,    "stop_count": 2},
    "D": {"count": TIMES_AVANCE, "stop_count": 2},
    "A": {"count": 1,            "stop_count": 1},
}

# Estados globales
latest_lifi_data         = "NO_DATA"
modo_automatico          = False
modo_aleatorio           = False
modo_aleatorio_thread    = None
modo_aleatorio_stop_event = threading.Event()

# Interfaces serial y HTTP
arduino_serial = ArduinoSerial()
lifi_serial    = LifiSerial()
session        = requests.Session()

# ============================
# CSV Logging
# ============================
def initialize_csv():
    if not os.path.exists(CSV_FILENAME):
        with open(CSV_FILENAME, "w", newline="") as f:
            writer = csv.writer(f)
            writer.writerow(["Comando", "Respuesta_Arduino", "Datos_LiFi"])

def save_to_csv(cmd, ar_resp, li_data):
    with open(CSV_FILENAME, "a", newline="") as f:
        writer = csv.writer(f)
        writer.writerow([cmd, ar_resp, li_data])

# ============================
# Mapear comandos ArUco → Arduino
# (nunca mapea “detener” a A en automático/aleatorio)
# ============================
def map_aruco_command(aruco_cmd):
    m = {
        "derecha":   "C",
        "izquierda": "B",
        "avanzar":   "D",
    }
    # Por defecto (incluye "detener") → avanzar
    return m.get(aruco_cmd, "D")

# ============================
```

```
# Envío de ráfagas genérico
# ============================
def send_burst_commands(arduino_cmd, count_override=None):
    cfg = BURST_CONFIG.get(arduino_cmd, BURST_CONFIG["D"])
    count      = count_override if count_override is not None else cfg["count"]
    stop_count = cfg["stop_count"]

    for _ in range(count):
        arduino_serial.enviar_mensaje(arduino_cmd)
        resp = arduino_serial.leer_respuesta() or "Sin respuesta"
        save_to_csv(arduino_cmd, resp, latest_lifi_data)
    for _ in range(stop_count):
        arduino_serial.enviar_mensaje("A")
        resp = arduino_serial.leer_respuesta() or "Sin respuesta"
        save_to_csv("A", resp, latest_lifi_data)
    print(f"[BURST] {arduino_cmd} x{count} + stop x{stop_count}")
def modo_aleatorio_loop():
    global TIMES_IZQ, TIMES_DER
    while not modo_aleatorio_stop_event.is_set():
        try:
            r = session.get(IMAGE_SERVER_URL, timeout=2.0)
            if r.status_code == 200 and r.headers.get("Content-
Type","").startswith("application/json"):
                d   = r.json()
                ac  = d.get("comando", "avanzar")
                err = float(d.get("error_ang", 0.0) or 0.0)

                ar_cmd = map_aruco_command(ac)
                print(f"[ALEATORIO] ArUco='{ac}' err={err:.2f} → Arduino='{ar_cmd}'")

                if ac.startswith(("izquierda","derecha")):
                    corr = controlador.update(err)
                    count = max(10, int(abs(corr)))
                    # ráfaga adaptativa
                    for _ in range(count):
                        arduino_serial.enviar_mensaje(ar_cmd)
                        resp = arduino_serial.leer_respuesta() or "Sin respuesta"
                    # ráfaga de parada
                    for _ in range(BURST_CONFIG[ar_cmd]["stop_count"]):
                        arduino_serial.enviar_mensaje("A")
                        resp = arduino_serial.leer_respuesta() or "Sin respuesta"
                else:
                    # avanzar por defecto
                    send_burst_commands(ar_cmd)

            else:
                # sin detección → esperar un poco
                time.sleep(0.05)
```

```
        except Exception as e:
            print(f"[ERROR Aleatorio]: {e}")
            time.sleep(1)

# ============================
# Lectura continua LiFi
# ============================
def read_lifi_serial():
    global latest_lifi_data
    try:
        while True:
            d = lifi_serial.leer_respuesta()
            if d:
                latest_lifi_data = d
            time.sleep(0.05)
    except:
        pass

# ============================
# Servidor HTTP y RequestHandler
# ============================
class RequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        global modo_automatico, modo_aleatorio
        cmd = self.path.strip("/")

        if modo_aleatorio:
            self.handle_random_mode(cmd)
        elif modo_automatico:
            self.handle_automatic_mode(cmd)
        else:
            self.handle_manual_mode(cmd)

    def handle_manual_mode(self, command):
        global modo_automatico
        # Acepta stop/manual (A) y movimientos (B,C,D) + X,Y,M,O
        if command in ["A","B","C","D"]:
            arduino_serial.enviar_mensaje(command)
            resp = arduino_serial.leer_respuesta() or "Sin respuesta"
            print(f"[MANUAL] Enviado: {command} → Recibido: {resp}")
            self.respond("Manual")
        elif command == "Z":   # activa automático
            modo_automatico = True
            print("[MODO] Automático activado")
            self.respond("Modo automático activado")
        elif command == "R":   # activa aleatorio
            self.activate_random_mode()
        else:
            self.respond("Comando no reconocido")
```

```
    def handle_automatic_mode(self, command):
        global modo_automatico
        if command == "Y":     # desactiva automático
            modo_automatico = False
            print("[MODO] Manual activado")
            self.respond("Modo manual activado")
            return

        try:
            r = session.get(IMAGE_SERVER_URL, timeout=2.0)
            if r.status_code == 200 and r.headers.get("Content-
Type","").startswith("application/json"):
                d   = r.json()
                ac  = d.get("comando", "avanzar")
                ar_cmd = map_aruco_command(ac)

                arduino_serial.enviar_mensaje(ar_cmd)
                resp = arduino_serial.leer_respuesta() or "Sin respuesta"
                save_to_csv(ar_cmd, resp, latest_lifi_data)

                print(f"[AUTOMÁTICO] ArUco='{ac}' → Arduino='{ar_cmd}' | Recibido:
{resp}")
                self.respond(f"OK {ac}→{ar_cmd}")
            else:
                self.respond("No detectado")
        except Exception as e:
            print(f"[ERROR Automático]: {e}")
            self.respond("Error automático")

    def handle_random_mode(self, command):
        if command == "N":
            self.deactivate_random_mode()
            self.respond("Modo aleatorio desactivado")
        else:
            self.respond("Modo aleatorio activo")

    def activate_random_mode(self):
        global modo_aleatorio, modo_aleatorio_thread
        modo_aleatorio = True
        modo_aleatorio_stop_event.clear()
        modo_aleatorio_thread = threading.Thread(
            target=modo_aleatorio_loop, daemon=True)
        modo_aleatorio_thread.start()
        print("[MODO] Aleatorio activado")
        self.respond("Modo aleatorio activado")

    def deactivate_random_mode(self):
        global modo_aleatorio
```

```
        modo_aleatorio = False
        modo_aleatorio_stop_event.set()
        print("[MODO] Aleatorio desactivado")

    def respond(self, msg, status=200):
        b = msg.encode()
        self.send_response(status)
        self.send_header("Content-Type", "text/plain")
        self.send_header("Content-Length", str(len(b)))
        self.send_header("Connection", "keep-alive")
        self.end_headers()
        self.wfile.write(b)

# ============================
# Inicialización y main
# ============================
def start_server():
    print(f"Iniciando servidor en {SERVER_ADDRESS[0]}:{SERVER_ADDRESS[1]}")
    httpd = ThreadingHTTPServer(SERVER_ADDRESS, RequestHandler)
    httpd.serve_forever()

def main():
    initialize_csv()
    threading.Thread(target=read_lifi_serial, daemon=True).start()
    start_server()

if __name__ == "__main__":
    main()
```

Anexo 16: Script de inferencia de algoritmos implementados. Estos algoritmos se implementan junto a las librerías de procesamiento de imágenes de opencv.

```
-----------------------------------------------------------------------------------------------------------
/*Script inferencia de imágenes */
-----------------------------------------------------------------------------------------------------------
import os
import time
import cv2
import threading
import json
import numpy as np
import random
from collections import deque
import tensorflow as tf
from http.server import BaseHTTPRequestHandler, HTTPServer
```

```
from aruco_localizer_pure import ArucoLocalizerPure
from trayectory_planner import TrajectoryPlannerTo41
import signal
import sys

# ------------------------------
# Parámetros de calibración y marcador
# ------------------------------
CAMERA_MATRIX = np.array([[800,   0, 320],
                          [  0, 800, 240],
                          [  0,   0,   1]], dtype=np.float32)
DIST_COEFFS   = np.array([0.1, -0.05, 0, 0, 0], dtype=np.float32)
MARKER_LENGTH = 0.010  # metros

HABITACION_A = (1,1)
HABITACION_B = (1,9)
HABITACION_C = (15,1)

MODO = 2 #0 habitacion A, 1 habitacion B, 2 habitacion C

#Configuracion modelo Red Neuronal
MODELO_ML = False
model_path = 'mejor_modelo.keras'
img_height, img_width = 200, 300
clases = ['avanzar', 'derecha', 'izquierda']

planner_habitacion_a =
TrajectoryPlannerTo41(model_path='q_model_improved_goal_11.pkl',
step_delay=0.3,target=HABITACION_A)
planner_habitacion_b =
TrajectoryPlannerTo41(model_path='q_model_improved_goal_19.pkl',
step_delay=0.3,target=HABITACION_B)
planner_habitacion_c = TrajectoryPlannerTo41(model_path='q_model_straightest_path.pkl',
step_delay=0.3,target=HABITACION_C)

_half = MARKER_LENGTH / 2.0
_OBJ_POINTS = np.array([
    [-_half,  _half, 0],
    [ _half,  _half, 0],
    [ _half, -_half, 0],
    [-_half, -_half, 0],
], dtype=np.float32)
# ------------------------------
# Configuración global
# ------------------------------
SERVER_ADDRESS     = ('192.168.1.5', 8082)
DEBUG_MODE         = False
REACH_THRESHOLD    = 0.2    # celdas para considerar alcanzado
ANGLE_SMOOTH_ALPHA = 1.0    # [0..1] menor = más suave
```

```
ANGLE_REGIONS = [
    {"name": "avanzar",   "start":  40, "end": 140, "color": (0,255,0)},
    {"name": "izquierda", "start": 141, "end": 270, "color": (0,0,255)},
    {"name": "derecha",   "start": 271, "end": 359, "color": (255,0,0)},
    {"name": "derecha",   "start":   0, "end":  39, "color": (255,0,0)},
]

# Bandera para logging y carpeta de destino
DATA_LOG   = False
SAVE_PATH  = "data_logging"
if DATA_LOG:
    os.makedirs(SAVE_PATH, exist_ok=True)
# ------------------------------
# Espacio de IDs: matriz 11×17
# ------------------------------
# Ajusta esta matriz según tus IDs reales en el piso
id_space = np.arange(1, 17*11 + 1, dtype=np.int32).reshape((17,11))

class CommandGenerator:
    def __init__(self, subregion,
                 debug_rows, debug_cols,
                 angle_regions, reach_threshold, smooth_alpha):
        self.subregion          = subregion
        self.angle_regions      = angle_regions
        self.reach_threshold    = reach_threshold
        self.smooth_alpha       = smooth_alpha
        self.debug_rows         = debug_rows
        self.debug_cols         = debug_cols

        self.current_target_id  = None
        self.prev_cmd           = "avanzar"
        self.last_angle         = 0.0
        self.last_dist          = 0.0
        self.last_err_d         = 0.0
        self.last_err_a         = 0.0
        self.missed_target_count = 0

        rows = [r for (r, c) in subregion]
        cols = [c for (r, c) in subregion]
        self.centroid_cell = (sum(rows)/len(rows), sum(cols)/len(cols))

    def _compute_control(self, target_id, loc_mat):
        pos = np.argwhere(loc_mat == target_id)
        if not pos.size:
            return (self.prev_cmd,
                    self.last_angle,
                    self.last_dist,
                    self.last_err_d,
```

```
                self.last_err_a)

        r, c = tuple(pos[0])
        cr, cc = self.centroid_cell
        dx, dy = c - cc, cr - r
        raw_angle = (np.degrees(np.arctan2(dy, dx)) + 360) % 360
        dist = float(np.hypot(dx, dy))
        err_dist = dist
        angle_used = round(raw_angle) % 360

        fr = next(rg for rg in self.angle_regions if rg["name"] == "avanzar")
        fmid = (fr["start"] + fr["end"]) / 2.0
        err_ang = ((angle_used - fmid + 180) % 360) - 180

        cmd = "detener"
        for rg in self.angle_regions:
            if rg["start"] <= angle_used <= rg["end"]:
                cmd = rg["name"]
                break

        self.prev_cmd   = cmd
        self.last_angle = angle_used
        self.last_dist  = dist
        self.last_err_d = err_dist
        self.last_err_a = err_ang

        return cmd, angle_used, dist, err_dist, err_ang

    def next_command(self, loc_mat, corners, ids):
        raise NotImplementedError("Usa PlannerCommandGenerator o
PlanificadorCommandGenerator")

class PlanificadorCommandGenerator(CommandGenerator):
    global MODO, planner_habitacion_a, planner_habitacion_b, planner_habitacion_c

    def __init__(self,
                 id_space: np.ndarray,
                 subregion,
                 debug_rows, debug_cols,
                 angle_regions, reach_threshold, smooth_alpha,
                 goal_id: int,
                 max_steps: int = 200):
        super().__init__(subregion, debug_rows, debug_cols,
                         angle_regions, reach_threshold, smooth_alpha)

        # Mapa de ID ↔ posición
        self.id_space = id_space
        self.rows, self.cols = id_space.shape
        self.id_to_pos = {
```

```
            int(id_): tuple(idx)
            for idx, id_ in np.ndenumerate(id_space) if id_ != 0
        }

        # Historial de nodos para penalizar repeticiones
        self.visited_history = deque(maxlen=200)
        self.missed_target_count = 0

        # Parámetros globales
        self.goal_id = goal_id
        self.max_steps = max_steps

        #Trayectorias deseadas
        self.trayectoria_ab = ([46,57,68,79,90,101,112,123,134,145,156,167,178])
        self.trayectoria_ac = ([46,57,68,79,90,101,102,103,104,105,107,96,85,76,63,52])
        self.trayectoria_ba = ([178,167,156,145,134,123,112,101,90,79,68,57,46])
        self.trayectoria_bc =
([178,167,156,145,134,123,112,101,102,103,104,105,106,107,96,85,74,63,52])
        self.trayectoria_ca = ([52,63,74,85,96,107,106,105,104,103,102,101,90,79,68,57,46])
        self.trayectoria_cb =
([52,63,74,85,96,107,106,105,104,103,102,101,112,123,134,145,156,167,178])

    def _visible_ids(self, loc_mat):
        detected = {int(x) for x in np.unique(loc_mat) if x != 0}
        blocked  = {int(loc_mat[r,c]) for (r,c) in self.subregion if loc_mat[r,c] != 0}
        return detected - blocked

    def _in_target_region(self, target_id, loc_mat):
        pos = np.argwhere(loc_mat == target_id)
        if not pos.size:
            return False
        r, c = tuple(pos[0])
        cr, cc = self.centroid_cell
        return (np.hypot(c - cc, cr - r) <= self.reach_threshold
                or self._in_corner(target_id, loc_mat))

    def _in_corner(self, target_id, loc_mat):
        return False

    def _plan_next(self):
        """Pide al planificador correspondiente el siguiente ID."""
        if MODO == 0:
            return planner_habitacion_a.predict_next_id(current_id=self.current_target_id)
        elif MODO == 1:
            return planner_habitacion_b.predict_next_id(current_id=self.current_target_id)
        else:
            return planner_habitacion_c.predict_next_id(current_id=self.current_target_id)

    def _simulate_path(self, planner, start_id):
```

```
    """
    Simula la política de Q-Learning desde start_id hasta self.goal_id:
    - devuelve (L, turns, D)
    - ó (None, None, None) si no llega en self.max_steps o cierra ciclo.
    """
    path = [start_id]
    current = start_id

    for _ in range(self.max_steps):
        nxt = planner.predict_next_id(current_id=current)
        if nxt is None or nxt in path:
            return None, None, None
        path.append(nxt)
        if nxt == self.goal_id:
            break
        current = nxt
    else:
        return None, None, None

    # longitud en saltos
    L = len(path) - 1

    # contar giros
    dirs = []
    for a, b in zip(path, path[1:]):
        ra, ca = self.id_to_pos[a]
        rb, cb = self.id_to_pos[b]
        dirs.append((rb - ra, cb - ca))
    turns = sum(1 for i in range(1, len(dirs)) if dirs[i] != dirs[i-1])

    # distancia euclidiana start→goal
    start_pos = np.array(self.id_to_pos[start_id])
    goal_pos  = np.array(self.id_to_pos[self.goal_id])
    D = np.linalg.norm(goal_pos - start_pos)

    return L, turns, D

def _hist_penalty(self, vid):
    """ Penalización ≥1 según cuán reciente aparece en visited_history """
    if vid not in self.visited_history:
        return 1.0
    idx = list(self.visited_history).index(vid)
    return 1.0 + (idx / len(self.visited_history)) * 2.0

def _best_visible_target(self, visibles):
    """
    Para cada vid ∈ visibles calcula:
      cost = α·L + β·turns + γ·(pen_hist−1) + δ·D
    y elige el vid de menor coste.
```

```
        """
        planner = {
            0: planner_habitacion_a,
            1: planner_habitacion_b,
            2: planner_habitacion_c
        }[MODO]

        # Pesos ajustables
        α, β, γ, δ = 1.0, 1.0, 1.0, 0.5

        best_id, best_cost = None, float('inf')
        for vid in visibles:
            L, turns, D = self._simulate_path(planner, vid)
            if L is None:
                continue
            hist_p = self._hist_penalty(vid)
            cost = α*L + β*turns + γ*(hist_p - 1.0) + δ*D
            if cost < best_cost:
                best_cost, best_id = cost, vid

        return best_id, best_cost

    def next_command(self, loc_mat, corners=None, ids=None):
        visibles = self._visible_ids(loc_mat)
        if not visibles:
            return "detener", 0.0, 0.0, 0.0, 0.0

        # 1) inicializar target si no hay
        if self.current_target_id is None:
            self.current_target_id = random.choice(list(visibles))
            self.visited_history.append(self.current_target_id)
            print(f"[Planificador] Nuevo target inicial: {self.current_target_id}")

        # 2) si alcanzamos el target actual
        if self._in_target_region(self.current_target_id, loc_mat):
            next_id = self._plan_next()
            if next_id not in visibles:
                fb, cost = self._best_visible_target(visibles)
                print(f"[Planificador] Planner sugirió {next_id}, fallback a {fb} (costo {cost:.2f})")
                next_id = fb or self.current_target_id
            else:
                print(f"[Planificador] Target {self.current_target_id} alcanzado → planificando
{next_id}")
            self.current_target_id = next_id
            self.missed_target_count = 0
            self.visited_history.append(self.current_target_id)

        else:
            # 3) si perdimos el target
```

```
        if self.current_target_id not in visibles:
            self.missed_target_count += 1
            if self.missed_target_count >= 100:
                best_id, cost = self._best_visible_target(visibles)
                if best_id is None:
                    fallback = min(visibles, key=lambda nid: abs(nid - (self.current_target_id or
0)))
                    print(f"[Planificador] 100 misses → sin candidato válido, fallback
{fallback}")
                    best_id = fallback
                else:
                    print(f"[Planificador] 100 misses → candidato óptimo {best_id} (costo
{cost:.2f})")
                self.current_target_id = best_id
                self.missed_target_count = 0
                self.visited_history.append(self.current_target_id)
            # si aún no llegan 100 misses, seguimos con el mismo target

        # 4) genera comando hacia current_target_id
        return self._compute_control(self.current_target_id, loc_mat)

class LiveCameraTrajectoryController:
    global MODELO_ML
    def __init__(self,
                 subregion,
                 id_space,
                 debug_rows: int = 6,
                 debug_cols: int = 12,
                 angle_regions=ANGLE_REGIONS,
                 reach_threshold=REACH_THRESHOLD,
                 smooth_alpha=ANGLE_SMOOTH_ALPHA,
                 goal_id: int = HABITACION_C,
                 max_steps: int = 200):

        # Localizador ArUco
        self.localizer = ArucoLocalizerPure(
            debug_rows=debug_rows,
            debug_cols=debug_cols
        )

        # Generador de comandos con planificación avanzada
        self.command_generator = PlanificadorCommandGenerator(
            id_space=id_space,
            subregion=subregion,
            debug_rows=debug_rows,
            debug_cols=debug_cols,
            angle_regions=angle_regions,
            reach_threshold=reach_threshold,
            smooth_alpha=smooth_alpha,
```

```
            goal_id=goal_id,
            max_steps=max_steps
        )

        # Parámetros de visualización y control
        self.angle_regions = angle_regions
        self.debug_mode    = DEBUG_MODE,
        self.debug_rows    = debug_rows
        self.debug_cols    = debug_cols
        self.modelo = tf.keras.models.load_model(model_path)

    def process_frame(self):
        ret, frame = self.localizer.cap.read()
        if not ret:
            return None, None, None

        # Obtener matriz de localización e IDs detectados
        loc_mat, corners, ids = self.localizer.obtener_matriz_localizacion(frame)

        # Calcular siguiente comando
        cmd, angle, dist, err_d, err_a = self.command_generator.next_command(loc_mat,
corners, ids)

        if MODELO_ML:
            # Preprocesamiento
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            resized = cv2.resize(gray, (img_width, img_height))
            normalized = resized.astype('float32') / 255.0
            input_tensor = np.expand_dims(normalized, axis=(0, -1))
            # Predicción
            preds = self.modelo.predict(input_tensor)
            idx = np.argmax(preds[0])
            clase_pred = clases[idx]
            confianza = preds[0][idx]
            print(f"Clase: {clase_pred}, Confianza: {confianza}")
            cmd = clase_pred
            if cmd == 'izquierda':
                cmd = 'derechar'
            elif cmd == 'derecha':
                cmd = 'izquierda'

        # Construir respuesta JSON
        response = {
            "comando":    cmd,
            "target_id":  self.command_generator.current_target_id,
            "angulo_2d":  angle,
            "distancia":  dist,
            "error_dist": err_d,
            "error_ang":  err_a
```

```
        }
        # Validar si el target está dentro de los IDs detectados
        valid = (
            ids is not None
            and len(ids) > 0
            and self.command_generator.current_target_id in ids.flatten()
        )

        # Generar imagen anotada para salida
        annotated = None
        if valid:
            annotated = frame.copy()

        return json.dumps(response), response, annotated

    def close(self):
        if hasattr(self, 'localizer') and hasattr(self.localizer, 'cap'):
            self.localizer.cap.release()

# Manejo de señal para shutdown limpio
shutdown_event = threading.Event()

def handle_shutdown(signum, frame):
    print("\n[Main] Señal de cierre recibida, iniciando desconexion...")
    shutdown_event.set()
    sys.exit(0)

class HTTPServerWrapper:
    def __init__(self, addr, controller):
        self.server_address       = addr
        self.trajectory_controller = controller
        class Handler(BaseHTTPRequestHandler):
            def do_GET(self):
                if self.path.strip("/") == "K":
                    max_attempts = 100
                    attempt = 0
                    annotated = None
                    while attempt < max_attempts:
                        data_str, data_dict, annotated = \
                            self.server.trajectory_controller.process_frame()
                        if annotated is not None:
                            break
                        attempt += 1
                    if annotated is not None:
                        data_str, ctype = data_str, "application/json"
                        if self.server.data_log:
                            timestamp = time.strftime("%Y%m%d_%H%M%S")
                            img_path  = os.path.join(self.server.save_path,
                                                     f"{timestamp}.jpg")
```

```
                    json_path = os.path.join(self.server.save_path,
                                             f"{timestamp}.json")
                    cv2.imwrite(img_path, annotated)
                    with open(json_path, 'w') as f:
                        json.dump(data_dict, f, indent=4)
                else:
                    data_str, ctype = "SN", "text/plain"
            else:
                data_str, ctype = "SN", "text/plain"
            resp = data_str.encode()
            self.send_response(200)
            self.send_header("Content-Type", ctype)
            self.send_header("Content-Length", str(len(resp)))
            self.end_headers()
            print(f"[SERVIDOR] {resp}")
            self.wfile.write(resp)

        def log_message(self, fmt, *args): pass

    self.httpd = HTTPServer(addr, Handler)
    self.httpd.data_log   = DATA_LOG
    self.httpd.save_path  = SAVE_PATH
    self.httpd.trajectory_controller = controller

def start(self):
    print(f"Servidor HTTP en http://{self.server_address[0]}:{self.server_address[1]}")
    try:
        self.httpd.serve_forever()
    except KeyboardInterrupt:
        pass
    finally:
        self.httpd.server_close()
        self.trajectory_controller.localizer.cap.release()
        print("Servidor detenido.")

# Manejo de cierre mejorado
shutdown_event = threading.Event()

def handle_shutdown(signum, frame):
    print("\n[Main] Señal de cierre recibida, iniciando shutdown...")
    shutdown_event.set()
    # Salir del programa
    sys.exit(0)

# Registrar los handlers
signal.signal(signal.SIGINT, handle_shutdown)
signal.signal(signal.SIGTERM, handle_shutdown)

if __name__ == "__main__":
```

```
    try:
        # Define la subregión (celdas)
        subregion = {
            (4,4),(4,5),(4,6),(4,7),
            (5,4),(5,5),(5,6),(5,7)
        }

        controller = LiveCameraTrajectoryController(subregion, id_space)

        # Arranca servidor HTTP en hilo aparte
        server = HTTPServerWrapper(SERVER_ADDRESS, controller)
        server_thread = threading.Thread(target=server.start, daemon=True)
        server_thread.start()

        # Bucle principal
        while not shutdown_event.is_set():
            time.sleep(0.1)  # Pequeña pausa para reducir carga CPU

    except Exception as e:
        print(f"[ERROR] {str(e)}")

    finally:
        # Limpieza garantizada
        print("\n[Main] Realizando limpieza final...")
        if 'controller' in locals():
            controller.localizer.cap.release()
        if 'server' in locals():
            server.httpd.server_close()
        print("Aplicación cerrada correctamente")
        os._exit(0)  # Fuerza la salida de todos los hilos
```

---

*/*Detector Arucos*/*

```
import cv2
import numpy as np

CAMERA_SOURCE = 0

class ArucoLocalizer:
    def __init__(self, debug_rows=12, debug_cols=24,
aruco_dict=cv2.aruco.DICT_5X5_250):
        self.cap = cv2.VideoCapture(CAMERA_SOURCE)
        if not self.cap.isOpened():
            raise RuntimeError("No se pudo acceder a la camara.")

        self.debug_rows = debug_rows
        self.debug_cols = debug_cols
```

```
        self.cell_size = None

        self.aruco_dict = cv2.aruco.getPredefinedDictionary(aruco_dict)
        self.aruco_params = cv2.aruco.DetectorParameters()
        self.detector = cv2.aruco.ArucoDetector(self.aruco_dict, self.aruco_params)

    def calcular_cell_size(self, frame):
        height, width = frame.shape[:2]
        self.cell_size = min(width // self.debug_cols, height // self.debug_rows)

    def localizar_en_subregiones(self, cx, cy, frame_width, frame_height):
        if cx < 0 or cy < 0 or cx >= frame_width or cy >= frame_height:
            return None

        cell_width = frame_width / self.debug_cols
        cell_height = frame_height / self.debug_rows
        col = int(cx // cell_width)
        row = int(cy // cell_height)
        return row, col

    def obtener_matriz_localizacion(self, frame):
        if self.cell_size is None:
            self.calcular_cell_size(frame)

        height, width = frame.shape[:2]
        loc_mat = np.full((self.debug_rows, self.debug_cols), -1, dtype=int)

        corners, ids, _ = self.detector.detectMarkers(frame)

        if ids is not None:
            for corner, id_ in zip(corners, ids.flatten()):
                cx = np.mean(corner[0][:, 0])
                cy = np.mean(corner[0][:, 1])
                region = self.localizar_en_subregiones(cx, cy, width, height)
                if region:
                    rr, cc = region
                    loc_mat[rr, cc] = id_

        return loc_mat, corners, ids
```

Anexo 17. Script para generar el modelo de inferencia con Q-Learning. El entrenamiento del algoritmo se hace de manera simulada para obtener la tabla Q.

```
-----------------------------------------------------------------------------------------------------
/*Script de entrenamiento algoritmo Q-Learning */
-----------------------------------------------------------------------------------------------------

import numpy as np
```

```
import matplotlib.pyplot as plt
import pickle
import random
import os

class QLearningNavigator:
    def __init__(self, rows=17, cols=11, valid_nodes=None,
                 alpha=0.1, gamma=0.9, epsilon=1.0, min_epsilon=0.1, decay=0.995,
                 episodes=2000, max_steps=200,
                 reward_move=-1, reward_forb=-100, reward_goal=100):

        self.rows = rows
        self.cols = cols
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.min_epsilon = min_epsilon
        self.decay = decay
        self.episodes = episodes
        self.max_steps = max_steps
        self.R_move = reward_move
        self.R_forb = reward_forb
        self.R_goal = reward_goal

        self.ACTIONS = [
            (-1, 0), (1, 0), (0, -1), (0, 1),
            (-1, -1), (-1, 1), (1, -1), (1, 1)
        ]
        self.num_actions = len(self.ACTIONS)

        self.valid_nodes = set(valid_nodes or [(r, c) for r in range(rows) for c in range(cols)])
        self.forbidden = {
            (r, c)
            for r in range(rows)
            for c in range(cols)
            if (r, c) not in self.valid_nodes
        }

    def is_valid(self, pos):
        return pos in self.valid_nodes

    def step(self, state, action, goal):
        dr, dc = self.ACTIONS[action]
        nr, nc = state[0] + dr, state[1] + dc
        next_pos = (nr, nc)

        if not self.is_valid(next_pos):
            return state, self.R_forb, False
        if next_pos == goal:
```

```
            return next_pos, self.R_goal, True
        return next_pos, self.R_move, False

    def choose_action(self, state, Q):
        if np.random.rand() < self.epsilon:
            return np.random.randint(self.num_actions)
        r, c = state
        return int(np.argmax(Q[r, c]))

    def train(self, goal):
        Q = np.zeros((self.rows, self.cols, self.num_actions), dtype=np.float32)
        rewards, steps = [], []

        for ep in range(1, self.episodes + 1):
            state = random.choice(list(self.valid_nodes - {goal}))
            total_reward = 0

            for step_i in range(self.max_steps):
                action = self.choose_action(state, Q)
                next_state, reward, done = self.step(state, action, goal)

                r, c = state
                Q[r, c, action] += self.alpha * (reward + self.gamma * np.max(Q[next_state]) - Q[r,
c, action])
                state = next_state
                total_reward += reward

                if done:
                    break

            self.epsilon = max(self.min_epsilon, self.epsilon * self.decay)
            rewards.append(total_reward)
            steps.append(step_i + 1)

            if ep % 100 == 0:
                avg_r = np.mean(rewards[-100:])
                avg_s = np.mean(steps[-100:])
                print(f"[Epi {ep:4d}] AvgReward: {avg_r:7.2f}, AvgSteps: {avg_s:6.2f}, Epsilon:
{self.epsilon:.3f}")

        with open('q_model_improved_goal_151.pkl', 'wb') as f:
            pickle.dump(Q, f)
        print("\n☑ Modelo guardado en 'q_model_improved.pkl'")
        return Q, rewards, steps

    def evaluate_policy(self, Q, goal):
        success, optimal = 0, 0
        total = len(self.valid_nodes) - 1
        for start in self.valid_nodes - {goal}:
```

```
            state = start
            for steps in range(1, self.max_steps + 1):
                r, c = state
                action = int(np.argmax(Q[r, c]))
                state, _, done = self.step(state, action, goal)
                if done:
                    success += 1
                    if steps < 25:
                        optimal += 1
                    break
        print(f"\n Evaluation: {success}/{total} reached goal ({optimal} optimally)")

    def plot_policy(self, Q, goal):
        arrow_map = {0: '↑', 1: '↓', 2: '←', 3: '→', 4: '↖', 5: '↗', 6: '↙', 7: '↘'}
        grid = np.full((self.rows, self.cols), ' ', dtype='<U2')
        for r in range(self.rows):
            for c in range(self.cols):
                if (r, c) in self.forbidden:
                    grid[r, c] = '■'
                elif (r, c) == goal:
                    grid[r, c] = 'G'
                else:
                    a = int(np.argmax(Q[r, c]))
                    grid[r, c] = arrow_map[a]

        print("\n Politica aprendida (G=goal, ■=forbidden):\n")
        for row in grid:
            print(' '.join(row))

if __name__ == "__main__":
    mode = 'train'        # 'train' or 'test'
    goal = (15,1)         # nodo objetivo
    start = (10, 3)       # Nodo inicial para prueba de algoritmo
    do_eval = True

    rows, cols = 17, 11
    forbidden = {(r, c) for r in range(rows) for c in range(cols)
                 if r in (0, rows - 1) or c in (0, cols - 1)}
    valid_nodes = [(r, c) for r in range(rows) for c in range(cols)
                   if (r, c) not in forbidden]

    nav = QLearningNavigator(rows=rows, cols=cols, valid_nodes=valid_nodes)

    if goal not in nav.valid_nodes:
        raise ValueError("No es un nodo valido")

    if mode == 'train':
        Q, rewards, steps = nav.train(goal)
        plt.plot(rewards)
```

```
        plt.title('Recompensas por episodio')
        plt.grid()
        plt.show()
        plt.plot(steps)
        plt.title('Pasos por episodio')
        plt.grid()
        plt.show()
        nav.plot_policy(Q, goal)

    elif mode == 'test':
        if not os.path.exists('q_model_improved_goal_19.pkl'):
            raise FileNotFoundError("No se encuentra modelo, entrene primero")
        with open('q_model_improved_goal_19.pkl', 'rb') as f:
            Q = pickle.load(f)

        nav.plot_policy(Q, goal)
        if do_eval:
            nav.evaluate_policy(Q, goal)
```

Anexo 18. Clase que utiliza la tabla Q generada. Esta clase es utilizada por el algoritmo para generar una trayectoria hacia el lugar determinado por el nodo objetivo.

```
-----------------------------------------------------------------------------------------------------------
/*Clase para planeamiento de trayectoria con el modelo Q-Learning entrenado*/
-----------------------------------------------------------------------------------------------------------
import pickle
import numpy as np
import cv2
import time

HABITACION_A = (1,1)
HABITACION_B = (1,9)
HABITACION_C = (15,1)

class TrajectoryPlannerTo41:
    def __init__(self, model_path='q_model_to_habitacion_goal_19.pkl', cell_size=40,
step_delay=0.4,target = HABITACION_A):
        self.ROWS, self.COLS = 17, 11
        self.TARGET_NODE = target
        self.TARGET_ID = self.node_to_id(self.TARGET_NODE)
        self.FORBIDDEN = {(r, c) for r in range(self.ROWS) for c in range(self.COLS)
                          if r in (0, self.ROWS - 1) or c in (0, self.COLS - 1)}
        self.VALID_NODES = {(r, c) for r in range(self.ROWS) for c in range(self.COLS)
                            if (r, c) not in self.FORBIDDEN}
        self.ACTIONS = [
            (-1, 0), (1, 0), (0, -1), (0, 1),
            (-1, -1), (-1, 1), (1, -1), (1, 1)
        ]
```

```
    with open(model_path, 'rb') as f:
        self.Q = pickle.load(f)
    self.cell_size = cell_size
    self.step_delay = step_delay

def id_to_node(self, id_):
    return (id_ // self.COLS, id_ % self.COLS)

def node_to_id(self, node):
    return node[0] * self.COLS + node[1]

def predict_next_id(self, current_id):
    print(f"[Nodo] {current_id}")
    current = self.id_to_node(current_id)
    if current not in self.VALID_NODES:
        print(f" ⚠ Nodo inválido: {current} (ID {current_id})")
        return None
    r, c = current
    action = int(np.argmax(self.Q[r, c]))
    dr, dc = self.ACTIONS[action]
    next_node = (r + dr, c + dc)
    if next_node in self.VALID_NODES:
        return self.node_to_id(next_node)
    else:
        return None

def get_trajectory_length_to_target(self, start_id, max_steps=100):
    current = self.id_to_node(start_id)
    if current not in self.VALID_NODES:
        print(f" ⚠ Nodo inicial inválido: {current}")
        return None
        print(f"NODO {current}")
    steps = 0
    for _ in range(max_steps):
        r, c = current
        action = int(np.argmax(self.Q[r, c]))
        dr, dc = self.ACTIONS[action]
        next_node = (r + dr, c + dc)
        print(f"ID_INIT {start_id} , [NEXT_NODE] {next_node}")
        if next_node not in self.VALID_NODES or next_node == current:
            return None  # camino bloqueado o ciclo
        steps += 1
        current = next_node
        if current == self.TARGET_NODE:
            print(f"[TARGET_NODE] {self.TARGET_NODE}")
            return steps
    return None  # no se alcanzó el objetivo dentro del límite de pasos
```

Anexo 19: Script de entrenamiento del modelo de red neuronal convolucional. Este script se ejecuta en Google Colab para hacer el entrenamiento con las imágenes preparadas para el mismo.

```
from google.colab import drive
drive.mount('/content/drive')

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/drive/MyDrive/Entrenamiento/CNN
# %ls
# %pwd

# Commented out IPython magic to ensure Python compatibility.
# %ls

"""Se carga la base los datos de entrenamiento (Imagenes)"""

!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="gJad5CHXdquyXYPwHiHN")
project = rf.workspace("tesis-silla").project("path_bw")
version = project.version(2)
dataset = version.download("folder")

"""IMPORTAR LIBRERIAS NECESARIAS"""

import os
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from tensorflow.keras.callbacks import ModelCheckpoint
import numpy as np
from tensorflow.keras.metrics import Precision, Recall, AUC

# Commented out IPython magic to ensure Python compatibility.
# %ls

# Transformaciones
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])
# Clases que queremos mantener
clases_permitidas = ['adelante', 'derecha', 'izquierda']
def load_dataset_filtrado(path, clases_permitidas):
```

```
    # Cargar todas las muestras
    dataset = datasets.ImageFolder(root=path, transform=transform)
    # Filtrar solo las muestras cuya carpeta (clase) esté permitida
    muestras_filtradas = [
        (ruta, dataset.class_to_idx[clase])
        for ruta, clase in [
            (s[0], s[0].split(os.sep)[-2]) for s in dataset.samples
        ]
        if clase in clases_permitidas
    ]
    # Actualizar samples y targets
    dataset.samples = muestras_filtradas
    dataset.targets = [s[1] for s in muestras_filtradas]
    # Redefinir class_to_idx y classes solo con las permitidas
    dataset.classes = clases_permitidas
    dataset.class_to_idx = {cls: i for i, cls in enumerate(clases_permitidas)}
    return dataset
# Cargar datasets

train_dataset = load_dataset_filtrado('PATH_BW-2/train', clases_permitidas)
valid_dataset = load_dataset_filtrado('PATH_BW-2/valid', clases_permitidas)
test_dataset = load_dataset_filtrado('PATH_BW-2/test', clases_permitidas)

# Loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)

# Verificación
print("Clases filtradas:", train_dataset.classes)
print("Imágenes en entrenamiento:", len(train_dataset))

# ========== [1] CONFIGURACIÓN ==========
data_dir = '/content/drive/MyDrive/Entrenamiento/CNN/PATH_BW-2'
model_path = '/content/drive/MyDrive/Entrenamiento/CNN/modelo_pista_cnn.keras'
img_height, img_width = 200, 300
batch_size = 32
clases_permitidas = ['adelante', 'derecha', 'izquierda']

# ========== [2] CARGA Y FILTRO ==========
train_datagen = ImageDataGenerator(rescale=1./255)
valid_datagen = ImageDataGenerator(rescale=1./255)
test_datagen  = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    os.path.join(data_dir, 'train'),
    target_size=(img_height, img_width),
    color_mode='grayscale',
    batch_size=batch_size,
```

```
    classes=clases_permitidas,
    class_mode='categorical'
)

valid_generator = valid_datagen.flow_from_directory(
    os.path.join(data_dir, 'valid'),
    target_size=(img_height, img_width),
    color_mode='grayscale',
    batch_size=batch_size,
    classes=clases_permitidas,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    os.path.join(data_dir, 'test'),
    target_size=(img_height, img_width),
    color_mode='grayscale',
    batch_size=batch_size,
    classes=clases_permitidas,
    class_mode='categorical',
    shuffle=False
)

# ========== [3] MODELO CONVOLUCIONAL ==========
modelo = tf.keras.Sequential([
    tf.keras.Input(shape=(img_height, img_width, 1)),  # escala de grises
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

modelo.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy', Precision(), Recall(), AUC()])
modelo.summary()

# ========== [4] ENTRENAMIENTO ==========
epochs = 50  # Puedes ajustarlo según necesidad
# Callback para guardar el mejor modelo basado en val_accuracy
checkpoint = ModelCheckpoint(
    filepath='mejor_modelo.keras',
    monitor='val_accuracy',      # Puedes cambiar a 'val_loss' si prefieres
    save_best_only=True,
    mode='max',
    verbose=1
```

```
)
history = modelo.fit(
    train_generator,
    epochs=epochs,
    validation_data=valid_generator,
    callbacks=[checkpoint]
)

modelo.save(model_path)  # Guarda en formato .keras, pero con extensión .pt si lo deseas
print(f"Modelo guardado en: {model_path}")

# ========= [6] VISUALIZACIÓN DEL HISTORIAL ==========
import matplotlib.pyplot as plt

# Mostrar qué hay disponible
print("Métricas disponibles en el historial:")
for k in history.history.keys():
    print("-", k)

# Buscar métricas en pares: entrenamiento y validación
metricas = {}
for key in history.history.keys():
    if not key.startswith('val_'):
        val_key = 'val_' + key
        if val_key in history.history:
            metricas[key] = val_key

# Graficar dinámicamente cada métrica y su validación
n = len(metricas)
plt.figure(figsize=(6 * n, 5))

for i, (train_key, val_key) in enumerate(metricas.items()):
    plt.subplot(1, n, i + 1)
    plt.plot(history.history[train_key], label='Entrenamiento')
    plt.plot(history.history[val_key], label='Validación')
    plt.title(f'{train_key.capitalize()} por Época')
    plt.xlabel('Época')
    plt.ylabel(train_key.capitalize())
    plt.legend()

plt.tight_layout()
plt.show()

# Obtener batch del test
test_images, test_labels = next(test_generator)
preds = modelo.predict(test_images)
pred_classes = np.argmax(preds, axis=1)
true_classes = np.argmax(test_labels, axis=1)
class_names = list(test_generator.class_indices.keys())
```

```
# Mostrar en subplots de N imágenes (2 filas x 5 columnas)
num_imgs = 20
for start in range(0, num_imgs, 10):

    end = start + 10
    fig, axes = plt.subplots(2, 5, figsize=(15, 6))
    fig.suptitle("Predicciones vs Reales", fontsize=16)

    for i, ax in enumerate(axes.flat):
        idx = start + i
        if idx >= len(test_images):
            ax.axis('off')
            continue
        img = test_images[idx]
        # Usa cmap='gray' si son imágenes en escala de grises
        ax.imshow(img.squeeze(), cmap='gray')
        pred = class_names[pred_classes[idx]]
        true = class_names[true_classes[idx]]
        ax.set_title(f"Pred: {pred}\nReal: {true}", fontsize=10)
        ax.axis('off')
    plt.tight_layout(rect=[0, 0, 1, 0.95])
    plt.show()
```