UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Plataforma robótica con capacidad de reconocimiento de patrones y navegación autónoma con microcontroladores

Informe de Proyecto de Grado presentado por

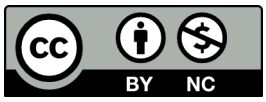
Enrique Rodriguez, Germán Cabarro, Guillermo Lara

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Jorge Visca
Ewelina Bakala

Montevideo, 24 de abril de 2026



Agradecimientos

A nuestros supervisores Jorge Visca y Ewelina Bakala, por su guía y apoyo constante durante el desarrollo de este proyecto.

A nuestras familias, por su paciencia y comprensión durante las largas jornadas de trabajo.

A la Facultad de Ingeniería de la Universidad de la República, por brindarnos los recursos y el espacio para cursar nuestra carrera y este proyecto.

A la comunidad de código abierto, cuyas bibliotecas y herramientas han hecho posible este trabajo.

Glosario

buck converter Conversor de voltaje DC-DC de tipo reductor (step-down) que disminuye el voltaje de entrada a un nivel inferior mediante conmutación de alta frecuencia, con alta eficiencia energética.

centroide Punto que representa el centro de masa de una región de píxeles, calculado como el promedio ponderado de las coordenadas de todos los píxeles que la componen.

cuantización Técnica de optimización de modelos de aprendizaje automático que reduce la precisión numérica de los pesos y activaciones (por ejemplo, de 32 bits flotantes a 8 bits enteros), reduciendo el tamaño del modelo y acelerando la inferencia.

código abierto Modelo de desarrollo en el que el código fuente de un software (o los diseños de un hardware) son públicamente accesibles, permitiendo su estudio, modificación y redistribución.

data augmentation Técnica para incrementar artificialmente el tamaño y diversidad de un conjunto de datos de entrenamiento mediante transformaciones como rotaciones, cambios de escala, variaciones de brillo y recortes aleatorios.

distancia euclidiana Medida de distancia geométrica entre dos puntos en el plano, calculada como $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

edge computing Paradigma de procesamiento distribuido en el que el cómputo se realiza en el propio dispositivo o en su proximidad, sin necesidad de enviar datos a servidores remotos o la nube.

encoder Sensor que convierte el movimiento rotacional o lineal de un eje en señales eléctricas, permitiendo medir la velocidad, dirección y posición angular de un motor con precisión.

filtro complementario Algoritmo de fusión sensorial que combina señales de baja y alta frecuencia provenientes de distintos sensores (por ejemplo, acelerómetro y giroscopio) mediante filtros pasa-bajos y pasa-altos complementarios.

filtro de Kalman Algoritmo recursivo de estimación óptima del estado de un sistema dinámico a partir de mediciones ruidosas, ampliamente utilizado en robótica móvil para fusión sensorial y localización.

flood-fill Algoritmo que, partiendo de un píxel semilla, expande la exploración hacia píxeles vecinos conectados que cumplan una condición determinada, permitiendo determinar el área y propiedades geométricas de una región.

fusión sensorial Combinación de datos provenientes de múltiples sensores heterogéneos para obtener estimaciones de estado más robustas y precisas que las que proporcionaría cualquiera de los sensores por separado.

handshake Intercambio inicial de mensajes entre dos dispositivos para establecer sincronización, verificar compatibilidad y acordar parámetros de comunicación antes de iniciar la operación.

inferencia Proceso de utilizar un modelo de aprendizaje automático ya entrenado para generar predicciones a partir de nuevos datos de entrada.

invariancia a rotación Propiedad de un algoritmo de reconocimiento de patrones que le permite identificar correctamente un objeto o patrón independientemente de la orientación angular con que aparezca en la imagen.

microcontrolador Circuito integrado que concentra en un único chip un procesador, memoria (flash, RAM, EEPROM) y periféricos de entrada/salida, diseñado para ejecutar una aplicación específica de forma autónoma.

MobileNet Arquitectura de red neuronal convolucional diseñada para operar eficientemente en dispositivos con recursos limitados, mediante el uso de convoluciones separables en profundidad.

motor con reductora Motor de corriente continua (DC) que incorpora una caja reductora mecánica, la cual disminuye la velocidad de rotación y aumenta el torque disponible en el eje de salida.

odometría Técnica de estimación de la posición y orientación de un robot a partir del conteo de revoluciones de sus ruedas, utilizando la cinemática del sistema de tracción.

pivot turn Maniobra de giro en el lugar en la que las dos ruedas del robot giran en sentidos opuestos, permitiendo rotar sobre su propio eje sin desplazamiento traslacional.

profiling Técnica de análisis de rendimiento que consiste en medir los tiempos de ejecución y el consumo de recursos de cada etapa de un programa, con el fin de identificar cuellos de botella.

programación tangible Modalidad de interacción en la que los programas se construyen manipulando objetos físicos (tarjetas, fichas, bloques), en lugar de escribir código en una pantalla.

pruning (poda) Técnica de optimización de redes neuronales que elimina conexiones o neuronas de bajo impacto, reduciendo el tamaño y la complejidad del modelo con una pérdida mínima de precisión.

puente H Circuito electrónico que permite controlar la dirección y velocidad de un motor DC mediante cuatro transistores configurados en forma de H, posibilitando hacer girar el motor en ambos sentidos.

sistema embebido Sistema computacional dedicado a una función específica, integrado dentro de un dispositivo mayor, generalmente con restricciones de recursos, consumo energético y tamaño físico.

tracción diferencial Configuración de locomoción de un robot con dos ruedas motrices independientes, que permite avanzar, retroceder y girar mediante la aplicación de velocidades asimétricas entre ambas ruedas.

transfer learning Técnica de aprendizaje automático que reutiliza un modelo pre-entrenado en una tarea como punto de partida para una tarea diferente, reduciendo los datos y el tiempo de entrenamiento necesarios.

umbralización binaria Proceso de conversión de una imagen en escala de grises a una imagen binaria (blanco y negro) mediante la comparación de cada píxel con un valor de umbral predefinido.

Siglas

CMOS Complementary Metal-Oxide-Semiconductor (Semiconductor Complementario de Óxido Metálico).

CNN Convolutional Neural Network (Red Neuronal Convolucional).

DSP Digital Signal Processor (Procesador de Señal Digital).

EEPROM Electrically Erasable Programmable Read-Only Memory.

GPIO General Purpose Input/Output (Entrada/Salida de Propósito General).

ML Machine Learning (Aprendizaje Automático).

NCC Normalized Cross-Correlation (Correlación Cruzada Normalizada).

OID Optical Image Detection (Detección de Imagen Óptica).

PCB Printed Circuit Board (Placa de Circuito Impreso).

PID Controlador Proporcional-Integral-Derivativo.

PSRAM Pseudo-Static Random Access Memory.

PWM Pulse Width Modulation (Modulación por Ancho de Pulso).

SAD Sum of Absolute Differences (Suma de Diferencias Absolutas).

SoC System-on-Chip (Sistema en un Chip).

SRAM Static Random Access Memory (Memoria de Acceso Aleatorio Estática).

SSD Sum of Squared Differences (Suma de Cuadrados de las Diferencias).

TTL Transistor-Transistor Logic.

UART Universal Asynchronous Receiver-Transmitter (Receptor-Transmisor Asíncrono Universal).

Resumen

El proyecto presenta el diseño, implementación y experimentación de una plataforma robótica de código abierto capaz de leer patrones visuales impresos en tarjetas físicas y corregir su trayectoria en tiempo real. El sistema fue desarrollado íntegramente sobre microcontroladores, sin dependencia de computadoras de propósito general, servidores remotos ni servicios en la nube.

La arquitectura del sistema es distribuida y consta de dos unidades de procesamiento independientes que se comunican mediante el protocolo UART (Receptor-Transmisor Asíncrono Universal): un módulo ESP32-CAM, responsable del subsistema de visión, y una placa Arduino Nano, encargada del control de los motores. Ambos módulos fueron programados en C/C++ con el framework Arduino.

Para el reconocimiento de patrones se implementaron dos algoritmos distintos. El primero, basado en template matching con invariancia a rotación, emplea la métrica SAD (suma de diferencias absolutas) y evalúa hasta siete ángulos de rotación con caché. Si bien este enfoque demostró robustez para la detección de patrones gráficos, su costo computacional resultó elevado dado el hardware. El segundo algoritmo, diseñado para superar estas limitaciones, se basa en la detección de pares de puntos de diferente tamaño: el identificador de tarjeta queda codificado en la distancia euclidiana entre los centroides, y el ángulo de corrección de trayectoria en la dirección espacial entre ellos. Este diseño redujo el tiempo de procesamiento, logrando una mejora de rendimiento respecto al template matching.

Complementariamente, se desarrollaron aplicaciones web para el diseño e impresión de tarjetas de programación para ambos algoritmos.

La experimentación confirmó la viabilidad de la arquitectura propuesta como alternativa de código abierto frente a soluciones comerciales propietarias. A diferencia de robots educativos analizados en el mercado, el prototipo desarrollado combina reconocimiento de patrones visuales, corrección activa de trayectoria y arquitectura documentada y replicable. Entre los principales desafíos identificados se destacan la sensibilidad de los algoritmos a variaciones de iluminación y el elevado costo computacional del procesamiento de imágenes en microcontroladores de recursos limitados.

Palabras clave: robótica, reconocimiento de patrones, microcontroladores, control PID

Índice general

Capítulo 1

Introducción

Este proyecto surge del interés en el desarrollo de plataformas de robótica móvil, integrando módulos de percepción visual y control de movimiento sobre sistemas embebidos basados en microcontroladores. En particular, se busca abordar problemáticas vinculadas a la detección de patrones visuales embebidos en tarjetas de programación y a la corrección de trayectoria en robots móviles que operan de forma autónoma, considerando las limitaciones propias de este tipo de hardware.

Asimismo, este trabajo se enmarca en el creciente interés por la robótica educativa como herramienta para acercar conceptos de programación y pensamiento computacional de forma tangible e interactiva. A diferencia de los enfoques tradicionales, la robótica permite materializar algoritmos en acciones concretas, facilitando la comprensión de conceptos abstractos y promoviendo un aprendizaje activo y experimental.

Una plataforma de este tipo puede resultar especialmente valiosa en contextos educativos, ya que permite fomentar el desarrollo del pensamiento algorítmico en niños, promoviendo habilidades de razonamiento lógico, resolución de problemas y estructuración de ideas desde edades tempranas.

Actualmente, gran parte de las soluciones disponibles en el mercado presentan costos elevados, dependen de tecnologías propietarias y/o patentadas, con esquemas de control cerrados, lo que limita su accesibilidad, auditoría y adaptación a distintos contextos educativos.

Frente a este escenario, el presente proyecto propone el diseño y desarrollo de un robot móvil capaz de leer patrones impresos en tarjetas y corregir su trayectoria en tiempo real en función del análisis realizado. De esta manera, se busca fomentar el desarrollo de soluciones tecnológicas abiertas que combinen bajo costo, precisión y autonomía, contribuyendo al acceso a herramientas de robótica educativa más flexibles y accesibles.

1.1. Objetivo del proyecto

El objetivo principal de este proyecto de grado es diseñar y construir un prototipo base para un robot móvil capaz de tomar decisiones a partir de la información visual obtenida por una cámara, reconociendo patrones en imágenes y desplazándose en función de dichos reconocimientos.

A diferencia de las soluciones comerciales cerradas, a menudo costosas y patentadas, la arquitectura propuesta se caracterizará por ser replicable, extensible y documentada.

Para alcanzar este objetivo general, se definen los siguientes objetivos específicos:

- Diseñar y construir un prototipo capaz de desplazarse, capturar imágenes, procesarlas y reconocer patrones visuales.
- Diseñar y fabricar un conjunto de tarjetas de programación con patrones visuales que, al ser detectados por el robot, permitan la activación de sus motores para la ejecución de distintos comportamientos de movimiento. Estas tarjetas constituyen la interfaz de interacción con el sistema, permitiendo a los usuarios construir secuencias de acciones y definir recorridos personalizados. En este sentido, su utilización favorece el desarrollo del pensamiento algorítmico, al promover la comprensión de la lógica secuencial mediante la composición de instrucciones.
- Diseñar una interfaz de comunicación entre el sistema de visión y los motores.
- Implementar un algoritmo de reconocimiento de patrones visuales, priorizando la robustez y precisión del control de trayectoria del robot. Este componente constituye uno de los núcleos funcionales del sistema, ya que es responsable de identificar las tarjetas de programación y traducir dicha información en comandos de movimiento, permitiendo la navegación autónoma y la ejecución coherente de las instrucciones.
- Implementar un algoritmo de control de motores capaz de corregir la trayectoria del robot en función de la retroalimentación visual.

1.2. Organización del documento

El presente informe se estructura en siete capítulos principales, además de las secciones de referencias y anexos, con el fin de describir de manera ordenada y detallada el desarrollo del proyecto.

El **Capítulo 1** introduce el contexto general del proyecto, su motivación y los objetivos perseguidos. Se presenta el problema de investigación y se justifica la relevancia de desarrollar una plataforma robótica basada en reconocimiento visual de patrones. Asimismo, se describe la organización del documento para guiar la lectura del informe.

El **Capítulo 2** está dedicado al estado del arte, donde se analiza el panorama tecnológico actual en el ámbito de robots con reconocimiento visual de patrones. Se examinan tres dimensiones complementarias: primero, las tecnologías fundamentales de reconocimiento de patrones disponibles para sistemas embebidos, incluyendo sensores propietarios (sensor OID), técnicas algorítmicas deterministas (template matching) y enfoques basados en aprendizaje automático (TensorFlow Lite, TinyML, Edge Impulse); segundo, productos comerciales representativos como Qobo, TaleBot Pro, Sphero Indi, Bee Bot y Cubetto; y tercero, una comparación sistemática de capacidades, fortalezas y limitaciones de cada solución.

El **Capítulo 3** presenta el análisis de la problemática, delimitando el alcance técnico del proyecto mediante la identificación de requerimientos, restricciones y obstáculos que definieron las

decisiones de diseño. Se establecen los requerimientos de hardware que determinaron la selección de componentes, las restricciones de software que guiaron la elección de lenguajes de programación y algoritmos, y los principales desafíos encontrados durante la fase de investigación y desarrollo.

El **Capítulo 4** describe el diseño de la solución propuesta, detallando la arquitectura distribuida del sistema con dos unidades de procesamiento independientes que se comunican mediante el protocolo serial UART. Se presentan diagramas de conexión de componentes, esquemas de hardware y se detallan los módulos principales del sistema. Adicionalmente, se describe la arquitectura de software, el protocolo de comunicación implementado y el diseño de las tarjetas de programación tangibles junto con sus patrones visuales. Para finalizar este capítulo, se describen algunas dificultades encontradas en la etapa de análisis para el desarrollo de la solución.

El **Capítulo 5** documenta la implementación del sistema, organizada en cinco etapas. La primera etapa describe la implementación de un algoritmo de comunicación entre los distintos microcontroladores. La segunda etapa abordó el problema de detección de tarjetas y determinación de orientación mediante template matching con invariancia a rotación. La tercera etapa surgió como respuesta a las limitaciones de procesamiento, diseñando un algoritmo completamente nuevo basado en la detección de pares de puntos de diferentes tamaños, donde la distancia codifica el identificador y la disposición espacial determina el ángulo de rotación. Finalmente, la cuarta etapa describe el algoritmo que se utiliza para el control de motores, el cual es ejecutado dentro de la Arduino Nano, junto con todas sus funcionalidades, y la quinta etapa describe las aplicaciones desarrolladas para la creación de las tarjetas de patrones utilizadas por los algoritmos de reconocimiento de patrones.

El **Capítulo 6** se centra en la experimentación y validación del sistema. Se describen las herramientas de depuración utilizadas: sistema de logging vía UART, instrumentación de código con marcadores de tiempo, analizador lógico para inspección de señales, y simuladores en Python para validación de algoritmos. Se presentan las pruebas realizadas sobre los enfoques implementados (template matching y detección de pares de puntos), incluyendo métricas de desempeño como tiempos de ejecución por etapa y análisis comparativo entre algoritmos y problemas encontrados en la etapa de experimentación.

Finalmente, el **Capítulo 7** presenta las conclusiones generales del proyecto, sintetizando los principales logros alcanzados y las lecciones aprendidas durante el desarrollo. Se plantean posibles líneas de trabajo futuro orientadas a mejorar y ampliar las capacidades del robot desarrollado, incluyendo optimizaciones de hardware, mejoras en los algoritmos de detección y el desarrollo de tarjetas de programación más sofisticadas.

El documento concluye con la sección de referencias, donde se enumeran las fuentes consultadas, y con cuatro anexos:

Anexo A - Diagramas de construcción adicionales: presenta los diagramas de esquema y PCB utilizados en el proyecto, complementando los diagramas de conexión presentados en el capítulo de diseño.

Anexo B - Aplicaciones web: describe las herramientas web desarrolladas para facilitar la generación de patrones de prueba, incluyendo el sitio generador de tarjetas para template matching y otras utilidades de prototipado rápido.

Anexo C - Entorno de desarrollo: detalla el entorno de desarrollo utilizado, incluyendo los lenguajes de programación (C/C++ con framework Arduino, Python para scripts auxiliares), herramientas de desarrollo (PlatformIO, Visual Studio Code), configuración del espacio de trabajo y las bibliotecas de software empleadas en cada componente del sistema.

Anexo D - Pseudocódigos: detalla los pseudocódigos de los algoritmos de reconocimiento de patrones basados en pares de puntos y de control de motores.

Anexo E - Templates utilizados: detalla los templates utilizados para las pruebas del algoritmo de reconocimiento de patrones basado en template matching.

Capítulo 2

Estado del arte

El presente capítulo analiza el panorama tecnológico actual en el ámbito de robots educativos, con especial énfasis en sistemas que utilizan reconocimiento de patrones visuales, procesamiento embebido y control de movimiento basado en retroalimentación sensorial. El objetivo de este análisis es contextualizar las decisiones de diseño del proyecto propuesto dentro del marco de soluciones existentes, identificar brechas tecnológicas y justificar el enfoque adoptado.

La estructura del capítulo se organiza en tres dimensiones complementarias:

Primero, se examinan las tecnologías fundamentales de reconocimiento de patrones disponibles para sistemas embebidos: desde sensores propietarios especializados (sensor OID) hasta técnicas algorítmicas deterministas (template matching) y enfoques basados en aprendizaje automático (TensorFlow Lite, TinyML, Edge Impulse). Esta revisión permite evaluar el espectro de alternativas desde la perspectiva de requisitos computacionales, latencia de procesamiento, complejidad de implementación y robustez ante variaciones ambientales.

Segundo, se analizan productos comerciales representativos del mercado actual de robots educativos: Qobo, TaleBot Pro, Sphero Indi, Bee Bot y Cubetto. Estos robots fueron seleccionados por cumplir con al menos uno de los siguientes criterios de relevancia para el proyecto:

- Reconocimiento de patrones
- Corrección de trayectoria
- Arquitectura de código abierto
- Programabilidad mediante interfaces tangibles

Tercero, se establece una comparación sistemática de las capacidades de los robots analizados, identificando las fortalezas y limitaciones de cada solución.

Finalmente, el capítulo concluye con un resumen que sintetiza las principales conclusiones del análisis del estado del arte y articula la transición hacia la solución propuesta.

2.1. Sensor OID

El sensor OID [1] es un decodificador de imágenes de alto rendimiento, diseñado para interpretar patrones gráficos y convertir la información visual en instrucciones precisas para el control de movimiento y comportamiento. Este dispositivo se utiliza principalmente en el reconocimiento de patrones, siendo especialmente útil en aplicaciones que requieren una transformación eficiente de datos visuales a acciones. Incorpora un módulo SoC altamente integrado que combina un sensor CMOS y el decodificador OID. Como dispositivo de entrada (INPUT Device), ofrece una solución eficaz que combina alto rendimiento y bajo costo.

Se caracterizan por tener una interfaz de comunicación sencilla de dos hilos, que permite una integración fácil con circuitos posteriores (backend IC), utilizando pines de entrada/salida de propósito general (GPIO). Esta arquitectura simplificada facilita su uso en diversos sistemas electrónicos.

Gracias a su diseño modular, el sensor proporciona una gran flexibilidad de integración, siendo compatible con una amplia gama de circuitos posteriores, que incluyen desde controladores de voz de baja complejidad hasta procesadores DSP basados en ARM de 32 bits, adecuados para aplicaciones de alto rendimiento.

Este tipo de sensor, además de su uso en diversas aplicaciones industriales y tecnológicas, es comúnmente empleado en robots educativos para el reconocimiento de patrones y la ejecución de acciones automáticas, como se menciona en algunos ejemplos de la sección 2.4.

El sensor OID, además, es un producto cerrado patentado de origen propietario. [2–4]

Figura 2.1: Sensor OID

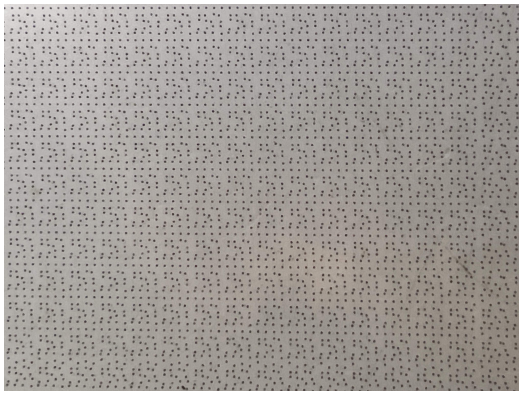


Figura 2.2: Ejemplo de patrones que reconoce el sensor

2.2. Template matching

Template matching es una técnica fundamental de visión por computadora que permite detectar y localizar un patrón de referencia (template) dentro de una imagen más grande. El principio básico consiste en deslizar el template sobre la imagen objetivo, calculando en cada posición una métrica de similitud que indica qué tan bien coincide el patrón con esa región de la imagen. [5, 6]

El proceso de template matching puede describirse formalmente de la siguiente manera. Sea T un template de dimensiones $m \times n$ píxeles y I una imagen de entrada de dimensiones $M \times N$ píxeles, donde $M \geq m$ y $N \geq n$. El objetivo es encontrar la posición (x, y) en I donde el template T presenta la mejor coincidencia.

Para cada posición candidata (x, y) en la imagen, se extrae una ventana $W_{x,y}$ del mismo tamaño que el template y se calcula una medida de similitud $S(x, y)$. La posición que maximiza (o minimiza, según la métrica utilizada) esta función de similitud corresponde a la mejor coincidencia:

$$(x^*, y^*) = \arg \max_{x,y} S(x, y) \tag{2.1}$$

Existen diversas métricas para calcular la similitud entre el template y las regiones de la imagen. Entre las más utilizadas se encuentran:

2.2.1. Sum of Absolute Differences (SAD)

Calcula la suma de las diferencias absolutas entre píxeles correspondientes. Es computacionalmente eficiente y ampliamente utilizada en sistemas embebidos debido a su simplicidad:

$$SAD(x, y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |T(i, j) - I(x + i, y + j)| \tag{2.2}$$

Valores menores de SAD indican mayor similitud, siendo 0 una coincidencia perfecta.

2.2.2. Sum of Squared Differences (SSD)

Similar a SAD pero utilizando el cuadrado de las diferencias, lo que penaliza más fuertemente las discrepancias grandes:

$$SSD(x, y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [T(i, j) - I(x + i, y + j)]^2 \tag{2.3}$$

2.2.3. Normalized Cross-Correlation (NCC)

Método más robusto a cambios de iluminación que normaliza los valores de intensidad:

$$NCC(x, y) = \frac{\sum_{i,j} T(i, j) \cdot I(x + i, y + j)}{\sqrt{\sum_{i,j} T(i, j)^2} \cdot \sqrt{\sum_{i,j} I(x + i, y + j)^2}} \tag{2.4}$$

La correlación cruzada normalizada produce valores entre -1 y 1, donde 1 indica una correlación perfecta.

2.2.4. Ventajas

El template matching presenta ventajas significativas para aplicaciones en robótica:

- **No requiere entrenamiento:** A diferencia de técnicas de aprendizaje automático, no necesita conjuntos de datos de entrenamiento extensos.
- **Eficiencia computacional:** Especialmente con métricas como SAD, puede ejecutarse en tiempo real en microcontroladores.
- **Precisión determinista:** Produce resultados predecibles y repetibles.

Sin embargo, la técnica de template matching básica presenta limitaciones importantes:

- **Sensibilidad a rotaciones:** Un template rotado no coincidirá correctamente con su versión sin rotar en la imagen.
- **Sensibilidad a cambios de escala:** El template debe tener aproximadamente el mismo tamaño que el objeto en la imagen.
- **Sensibilidad a iluminación:** Variaciones en la iluminación pueden afectar significativamente los resultados, especialmente con métricas simples como SAD.
- **Costo computacional:** La búsqueda exhaustiva en toda la imagen puede ser costosa, especialmente al considerar múltiples rotaciones y escalas.

2.3. Aprendizaje Automático

En el contexto de la visión por computadora y el reconocimiento de patrones, la inteligencia artificial y el aprendizaje automático (Machine Learning, ML) han emergido como alternativas poderosas a las técnicas clásicas de procesamiento de imágenes. Mientras que el template matching se basa en comparaciones deterministas de patrones predefinidos, los enfoques basados en ML permiten que el sistema aprenda a reconocer patrones a partir de ejemplos, ofreciendo potencialmente mayor robustez ante variaciones de iluminación, escala, oclusiones parciales y deformaciones.

En aplicaciones de robótica educativa, el uso de técnicas de ML presenta tanto oportunidades como desafíos. Por un lado, modelos como las redes neuronales convolucionales (CNN) pueden lograr tasas de reconocimiento superiores al 95 % en tareas de clasificación de imágenes, incluso en condiciones variables. Por otra parte, estos modelos suelen exigir recursos computacionales considerables —procesadores de alto rendimiento, amplias cantidades de memoria y, con frecuencia, aceleradores especializados— que superan las capacidades de los microcontroladores de bajo costo.

Estas secciones exploran tres tecnologías clave que han democratizado el uso de ML en sistemas embebidos: TensorFlow Lite, TinyML y Edge Impulse. Seleccionamos estas tres tecnologías porque representan enfoques complementarios dentro del ecosistema de aprendizaje automático: TensorFlow Lite ofrece una infraestructura madura y ampliamente adoptada para desplegar modelos optimizados en dispositivos con recursos limitados; TinyML define el marco conceptual y el conjunto de herramientas orientadas a llevar modelos ligeros a microcontroladores de baja potencia; y Edge Impulse proporciona una plataforma integrada, accesible incluso para no expertos, que simplifica el ciclo completo de desarrollo, desde la adquisición de datos hasta el despliegue en hardware real. Adicionalmente, estas tecnologías cuentan con comunidades grandes y activas, lo que se traduce en abundante documentación, soporte continuo, ejemplos prácticos y una evolución sostenida en el tiempo, factores especialmente relevantes en contextos académicos y de investigación aplicada. En conjunto, estas tecnologías cubren tanto los aspectos teóricos como prácticos del ML embebido y permiten ilustrar diferentes niveles de complejidad, casos de uso y requisitos de hardware.

2.3.1. TensorFlow Lite

TensorFlow [7] Lite es un framework de código abierto desarrollado por Google diseñado específicamente para ejecutar modelos de aprendizaje automático en dispositivos con recursos limitados, incluyendo dispositivos móviles, microcontroladores y sistemas embebidos. Constituye una versión optimizada de TensorFlow, el popular framework de ML, adaptada para inferencia (predicción) en lugar de entrenamiento, eliminando componentes innecesarios y aplicando optimizaciones agresivas de tamaño y velocidad.

El flujo de trabajo típico con TensorFlow Lite involucra tres etapas principales. Primero, se entrena un modelo de ML utilizando TensorFlow completo en una computadora o servidor con recursos abundantes, empleando conjuntos de datos etiquetados relevantes para la tarea (por ejemplo, imágenes de diferentes patrones de tarjetas de programación con sus correspondientes etiquetas). Segundo, el modelo entrenado se convierte al formato TensorFlow Lite (.tflite) mediante un proceso de optimización que incluye cuantización —reducción de la precisión numérica de 32 bits flotantes a 8 bits enteros— lo que típicamente reduce el tamaño del modelo en un 75 % y acelera la inferencia en 2-4 veces con mínima pérdida de precisión. Tercero, el modelo optimizado se despliega en el dispositivo objetivo, donde el intérprete de TensorFlow Lite ejecuta las predicciones.

En el contexto de visión por computadora en sistemas embebidos, TensorFlow Lite permite

implementar modelos como MobileNet, diseñados específicamente para dispositivos con recursos limitados. MobileNet es una arquitectura de CNN que utiliza convoluciones separables en profundidad (depthwise separable convolutions) para reducir drásticamente el número de operaciones necesarias, logrando un balance favorable entre precisión y eficiencia computacional. Una versión cuantizada de MobileNet V2 puede ocupar aproximadamente 3.4 MB y ejecutarse en microcontroladores con al menos 512 KB de RAM y varios MB de memoria flash.

2.3.2. TinyML

TinyML (Tiny Machine Learning) [8] es un paradigma emergente que se refiere a la intersección del aprendizaje automático, sistemas embebidos y edge computing, enfocándose específicamente en la ejecución de modelos de ML en microcontroladores con recursos limitados —típicamente dispositivos con menos de 1 MB de RAM y consumo energético del orden de milivatios. Este enfoque representa un cambio fundamental en la forma de desplegar inteligencia artificial, trasladando el procesamiento desde la nube hacia el dispositivo final, lo que ofrece ventajas críticas en latencia, privacidad, conectividad y eficiencia energética.

El concepto de TinyML se sustenta en varias innovaciones técnicas clave. Primero, la cuantización agresiva de modelos, donde los pesos y activaciones de las redes neuronales se representan con enteros de 8 bits o incluso menos (4 bits, 2 bits, o valores binarios), reduciendo drásticamente los requisitos de memoria y acelerando las operaciones aritméticas en microcontroladores sin unidades de punto flotante dedicadas. Segundo, arquitecturas de redes neuronales especialmente diseñadas para eficiencia, como MobileNet, SqueezeNet y EfficientNet, que minimizan el número de parámetros y operaciones necesarias mientras mantienen capacidad de representación adecuada. Tercero, técnicas de poda (pruning) que eliminan conexiones redundantes o poco importantes de las redes, reduciendo su complejidad sin degradación significativa del rendimiento.

En aplicaciones de robótica, TinyML abre posibilidades fascinantes. Un robot equipado con un microcontrolador ejecutando un modelo TinyML puede realizar reconocimiento de voz para comandos simples (detección de palabras clave), clasificación de gestos mediante sensores inerciales (acelerómetros y giroscopios), detección de anomalías en patrones de movimiento, o clasificación básica de objetos mediante visión por computadora. Estos sistemas pueden operar de forma autónoma sin requerir conectividad constante a servidores remotos, reduciendo latencias a milisegundos y permitiendo funcionamiento en entornos sin red.

Los requisitos de hardware para TinyML varían según la complejidad de la tarea, pero generalmente incluyen al menos 512 KB de memoria flash para almacenar el modelo, 128 KB de RAM para buffers de entrada y activaciones intermedias, y procesadores con velocidades mínimas de 100 MHz. Tomando como ejemplo un microcontrolador conocido como lo es una ESP32-CAM (Ver 4.1.6), con su ESP32 de doble núcleo a 240 MHz, 520 KB de SRAM y 4 MB de PSRAM, se encuentra en el rango de dispositivos compatibles con TinyML, especialmente para tareas de visión por computadora.

2.3.3. Edge Impulse

Edge Impulse [9] es una plataforma de desarrollo integral diseñada para simplificar y democratizar la creación de soluciones de TinyML, ofreciendo un flujo de trabajo completo desde la recolección de datos hasta el despliegue de modelos en dispositivos embebidos. La plataforma se distingue por su enfoque "low-code"que permite a desarrolladores sin experiencia profunda en ML crear mode-

los funcionales mediante una interfaz web intuitiva, al tiempo que proporciona flexibilidad para usuarios avanzados que deseen personalizar arquitecturas y parámetros.

El flujo de trabajo en Edge Impulse consta de varias etapas bien definidas. La primera etapa es la recolección de datos (data acquisition), donde los usuarios capturan muestras directamente desde el dispositivo objetivo (por ejemplo, capturando imágenes desde el ESP32-CAM) o cargan datos existentes. La plataforma facilita la captura sincronizada desde múltiples sensores y ofrece herramientas para balancear clases y aumentar conjuntos de datos. La segunda etapa es la de diseño (impulse design), donde se configura el pipeline de procesamiento: extracción de características, arquitectura del modelo y parámetros de salida. Para visión por computadora, Edge Impulse ofrece bloques de procesamiento predefinidos como redimensionamiento de imágenes, conversión a escala de grises, y extracción de características mediante transfer learning de modelos pre-entrenados como MobileNetV2.

La tercera etapa es el entrenamiento, donde la plataforma ejecuta el proceso de entrenamiento en sus servidores cloud, liberando al usuario de la necesidad de hardware potente local. Edge Impulse optimiza automáticamente hiperparámetros mediante búsqueda heurística y aplica técnicas de data augmentation (rotaciones, cambios de brillo, recortes aleatorios) para mejorar la robustez del modelo. La cuarta etapa es el despliegue, donde el modelo entrenado se convierte automáticamente al formato apropiado para la plataforma objetivo (incluyendo Arduino, ESP32, STM32, entre otros) y se genera código de integración optimizado que puede incorporarse directamente en el proyecto.

Las ventajas de Edge Impulse en contextos de prototipado son significativas. La plataforma reduce dramáticamente la curva de aprendizaje para ML embebido, permitiendo que estudiantes o desarrolladores con conocimientos básicos creen sistemas funcionales de manera más ágil. El proceso de recolección de datos directamente desde el dispositivo elimina complejidades de sincronización y formato. Las herramientas de validación integradas (matriz de confusión, pruebas en vivo, análisis de características) facilitan la evaluación y depuración del modelo.

2.4. Productos existentes

En esta sección se presenta un relevamiento de productos disponibles en el mercado cuyos objetivos y funcionalidades resultan comparables con los del prototipo propuesto. En particular, se relevan productos que se puedan programar mediante elementos físicos sin la necesidad de utilizar computadoras, tablets ni software adicional, lo cual le permite a los usuarios interactuar de forma directa y tangible con los conceptos básicos de programación. Este análisis tiene como finalidad establecer una comparación sistemática entre dichas soluciones, identificando similitudes y diferencias, así como sus principales fortalezas y debilidades, con el propósito de contextualizar el desarrollo dentro del estado del arte.

2.4.1. Qobo

Qobo [10] es un robot educativo cuyo funcionamiento se basa en la lectura secuencial de tarjetas mediante sensores OID integrados en la parte inferior del robot. Cada tarjeta contiene un patrón específico que codifica una instrucción, como avanzar o realizar giros de 90 grados. Al disponer las tarjetas en una fila o camino, el robot interpreta la secuencia y ejecuta las acciones en el orden definido, permitiendo a los usuarios comprender de manera tangible el concepto de programación secuencial. Los patrones impresos en estas tarjetas ademas permiten que el robot corrija su trayectoria a medida que se desplaza.

El robot incorpora además mecanismos de retroalimentación sonora y visual, como luces o sonidos, que indican la correcta lectura de las tarjetas o la ejecución de una acción.

Figura 2.3: Robot Qobo

2.4.2. Tale-Bot Pro

El Tale-Bot Pro es un robot educativo programable que permite definir secuencias de acciones mediante botones físicos integrados en el dispositivo. A través de estos controles, los usuarios pueden ingresar comandos como avanzar, girar, repetir acciones o reproducir sonidos, los cuales son almacenados y posteriormente ejecutados por el robot. [11, 12]

El robot es capaz de desplazarse hacia adelante y realizar giros de 90 grados tanto a la derecha como a la izquierda, lo que permite construir trayectorias discretas sobre una superficie.

Adicionalmente, el Tale-Bot Pro incorpora sensores basados en tecnología OID, que le permiten detectar y reconocer patrones impresos en superficies específicas. Mediante esta funcionalidad, el robot puede interpretar instrucciones externas, tales como cambiar de dirección o ejecutar acciones específicas al pasar sobre un determinado patrón, así como emitir distintas frases y sonidos.

El sistema permite almacenar secuencias de comandos, facilitando la repetición de programas sin necesidad de re configuración manual.

No obstante, el Tale-Bot Pro no cuenta con capacidades de navegación autónoma avanzada ni mecanismos de corrección de trayectoria en tiempo real, por lo que pueden producirse desviaciones durante el desplazamiento.

Figura 2.4: Tale-Bot Pro

2.4.3. Sphero Indi

El Sphero Indi es un robot educativo que se basa en el reconocimiento de tarjetas de colores las cuales representan diferentes instrucciones o comportamientos. Estas tarjetas se disponen sobre el suelo formando recorridos o secuencias, y el robot, mediante sensores de color integrados en su base, interpreta cada tarjeta al pasar sobre ella. Dependiendo del color detectado, el Indi ejecuta acciones predefinidas como avanzar, girar, detenerse o reproducir sonidos. Este mecanismo permite a los usuarios construir programas de forma visual y concreta, facilitando la comprensión de conceptos fundamentales como secuencias [13].

El robot incorpora retroalimentación mediante luces LED de distintos colores y sonidos, que informan al usuario sobre el estado del robot, la lectura correcta de una tarjeta o la ejecución de una acción específica.

Sphero Indi no es capaz de desplazarse de manera centrada sobre las tarjetas de programación, presentando desviaciones significativas respecto al eje medio de las mismas, pudiendo generar errores de trayectoria [14].

Figura 2.5: Sphero Indi

2.4.4. Bee Bot

Bee Bot es un robot educativo en forma de "abeja" que se programa mediante botones físicos para conseguir que efectúe movimientos determinados sobre una cuadrícula. Las secuencias de movimientos que puede realizar son: adelante, atrás, izquierda, derecha, pausa y curvas de noventa grados.

Bee Bot no dispone de mecanismos que le permita corregir una trayectoria ya que solo es capaz de moverse en línea recta y no de realizar curvas [15].

Figura 2.6: Bee Bot

El funcionamiento de Bee Bot es muy sencillo. Bee bot tiene distintas acciones para cada uno de los comandos [16].

- Adelante: avanza 15 cm
- Atrás: retrocede 15 cm
- Giro a la derecha: realiza un giro a la derecha sobre sí misma de 90 grados
- Giro a la izquierda: realiza un giro a la izquierda sobre sí misma de 90 grados
- Pausa: hace una pausa en su recorrido
- Go: ejecuta las órdenes programadas
- X: borra toda la secuencia de órdenes

Figura 2.7: Botones del Bee Bot

2.4.5. Cubetto

Cubetto es un robot educativo de madera que se programa mediante fichas físicas de comando las cuales se insertan en un panel de control dedicado, los usuarios pueden construir secuencias de instrucciones que el robot ejecuta posteriormente.

El robot es capaz de desplazarse hacia adelante y realizar giros de 90 grados tanto hacia la derecha como hacia la izquierda, siguiendo las instrucciones definidas por la secuencia de fichas colocadas en su tablero de programación.

Cubetto interpreta las fichas mediante un sistema de lectura integrado en su panel de control, el cual traduce cada ficha en una acción específica. Estas fichas representan comandos de movimiento y control, permitiendo definir recorridos sobre distintos mapas físicos diseñados para acompañar el proceso educativo [17–19].

El robot también cuenta con la capacidad de almacenar y ejecutar la secuencia de comandos definida previamente, reproduciendo el programa completo cada vez que se inicia la ejecución. Sin embargo, Cubetto no dispone de sensores de corrección de trayectoria ni retroalimentación del entorno, por lo que ejecuta los movimientos de manera abierta, sin ajustar su posición para mantenerse dentro de los límites del mapa [20].

Una de las mayores fortalezas de Cubetto reside en el enfoque abierto de su desarrollo: el código utilizado para su programación es público y de acceso libre. Los manuales de construcción del hardware están bien documentados y son replicables. Esto no solo favorece la transparencia del funcionamiento del robot, sino que también habilita la posibilidad de estudiar sus funcionalidades [21].

Figura 2.8: Cubetto

2.4.6. Comparación de funcionalidades clave

A continuación se presenta una tabla comparativa que detalla las principales características de cada robot, junto con una breve descripción de cada una de ellas, con el fin de facilitar su análisis y comparación.

Las definiciones utilizadas para la comparación son las siguientes:

- Reconoce patrones: Lectura de símbolos/códigos impresos.
- Reconoce colores: Uso de sensores de color para disparar comportamientos.
- Corrige trayectoria: Capacidad del robot de ajustar activamente su movimiento para compensar desviaciones físicas.
- Es programable: Permite definir secuencias de acciones.
- Es de código abierto: Hardware o software documentado y modificable por la comunidad.

Característica	**TaleBot Pro**	**Qobo**	**Sphero Indi**	**Bee Bot**	**Cubetto**
Reconoce patrones	✓	✓	×	×	×
Reconoce colores	×	×	✓	×	×
Corrige trayectoria	×	✓	×	×	×
Es programable	✓	✓	✓	✓	✓
Es de código abierto	×	×	×	×	✓

Tabla 2.1: Comparación de robots según sus capacidades

Capítulo 3

Análisis de la problemática

El desarrollo de esta plataforma presenta un conjunto de desafíos técnicos que requieren un análisis detallado. A diferencia de las implementaciones convencionales que dependen de computadoras de propósito general o servicios en la nube, la naturaleza embebida de esta solución impone restricciones fundamentales en cuanto a capacidad de procesamiento, memoria disponible y recursos computacionales.

Este capítulo tiene como objetivo delimitar el alcance técnico del proyecto mediante la identificación y análisis de los requerimientos, restricciones y obstáculos que definieron las decisiones de diseño e implementación del sistema. En primer lugar, se establecen los requerimientos de hardware que determinan la selección de componentes y la arquitectura física de la plataforma. Posteriormente, se detallan las restricciones de software que guiaron la elección de lenguajes de programación, algoritmos de procesamiento de imágenes y estrategias de control. Finalmente, se documentan los principales problemas encontrados durante la fase de investigación y desarrollo, incluyendo las limitaciones de documentación técnica, desafíos de hardware y complejidad en las pruebas y depuración del sistema embebido.

3.1. Requerimientos de hardware

En la sección 2.3 se presentan alternativas de aprendizaje automático orientadas a la detección de patrones. Sin embargo, estos algoritmos pueden implicar un alto costo computacional al intentar maximizar la precisión en las detecciones.

Con el fin de superar esta limitación, se propone el desarrollo de algoritmos deterministas y computacionalmente eficientes, diseñados para ejecutarse en microcontroladores de bajo costo. De este modo, se busca evaluar la viabilidad del sistema robótico en este tipo de entornos y determinar si es posible realizar reconocimiento de patrones visuales directamente sobre hardware embebido con recursos limitados.

En este contexto, el desarrollo de la solución se encuentra condicionado por un conjunto de decisiones de diseño en términos de componentes y arquitectura de hardware, las cuales responden a los objetivos de bajo costo, portabilidad y aplicabilidad en entornos educativos:

- **Utilización de microcontroladores embebidos:** Se opta por implementar la solución exclusivamente mediante microcontroladores embebidos, evitando la dependencia de computadoras o sistemas basados en arquitecturas convencionales (por ejemplo, Raspberry Pi). Esta decisión permite reducir costos, consumo energético y complejidad del sistema, facilitando además su replicabilidad en contextos educativos con recursos limitados.

- **Independencia de servidores remotos:** La arquitectura propuesta prescinde del uso de servidores externos o servicios en la nube. Esto permite garantizar la operatividad del sistema en entornos sin conectividad, simplificar su despliegue y evitar dependencias de infraestructura adicional, lo cual resulta especialmente relevante en aplicaciones educativas.

- **Autosuficiencia del sistema:** El sistema se diseña para operar de manera autónoma en términos de procesamiento y control, ejecutando toda la lógica localmente en los dispositivos embebidos. De esta forma, se elimina la necesidad de conectividad externa (por ejemplo, Wi-Fi o Bluetooth) para su funcionamiento, lo que incrementa la confiabilidad y reduce la latencia. Asimismo, el almacenamiento de datos y parámetros de configuración se realiza en memoria local, evitando dependencias de bases de datos externas.

- **Restricciones físicas y constructivas:** La plataforma se concibe con un tamaño compacto que permita su operación en espacios reducidos, como mesas o escritorios de aula. Esta característica favorece su uso en entornos educativos, donde la portabilidad, facilidad de manipulación y adaptación al espacio disponible son factores clave.

3.2. Requerimientos de software

Como limitante en cuanto al software a utilizar, tenemos los siguientes puntos fundamentales:

- Lenguaje de Programación Embebido: La solución debe estar implementada exclusivamente en C/C++, evitando lenguajes de alto nivel (Python, JavaScript, Node) que presentan mayor consumo de memoria y overhead de ejecución incompatible con microcontroladores de recursos de almacenamiento acotados.

- Procesamiento Local en el Dispositivo: Todo el procesamiento de datos, análisis de imágenes y toma de decisiones debe realizarse completamente en el dispositivo.

- Modelos remotos: No está permitido el uso de modelos de inteligencia artificial o aprendizaje automático que se ejecuten de forma remota. Los algoritmos de detección deben ser deterministas y basados en reglas. La justificación de esto es que el sistema debe funcionar como un dispositivo completamente autónomo, similar a plataformas robóticas comerciales que existen actualmente como los robots Qobo o TaleBot Pro. Garantizando:

    1. Independencia operacional: El robot puede funcionar en cualquier entorno sin necesidad de conexión a internet o servicios externos.
    2. Predictibilidad y confiabilidad: Comportamiento consistente y reproducible.
    3. Baja latencia: Respuesta inmediata sin overhead de comunicación remota.
    4. Privacidad y seguridad: Ningún dato es transmitido a servidores externos.

- Dependencias Pesadas: Evitar el uso de frameworks, bibliotecas de alto nivel o sistemas operativos pesados. Se permiten bibliotecas open-source livianas comprobadas de bajo nivel. El preprocesado de imágenes debe implementarse sin bibliotecas externas pesadas (OpenCV).

3.3. Justificación del enfoque propuesto

El análisis del estado del arte revela dos tendencias dominantes en el desarrollo de robots autónomos para enseñanza de programación: por un lado, soluciones comerciales basadas en sensores propietarios (OID) que ofrecen reconocimiento confiable de patrones pero con limitaciones de accesibilidad, costo y apertura tecnológica; por otro lado, enfoques basados en aprendizaje automático (TensorFlow Lite, TinyML, Edge Impulse) que prometen mayor robustez y adaptabilidad, pero que imponen requisitos computacionales, energéticos y de desarrollo significativamente superiores.

La comparación de productos existentes evidencia que ninguno de los robots analizados combina simultáneamente las siguientes características: reconocimiento de patrones visuales, corrección activa de trayectoria mediante retroalimentación continua, y arquitectura de código abierto que permita su estudio, modificación y replicación en contextos con recursos limitados. Cubetto se destaca por su naturaleza abierta, pero carece de sensores de retroalimentación para corrección de trayectoria. Qobo incorpora corrección de trayectoria, pero utiliza tecnología propietaria (sensor OID) de costo elevado y naturaleza cerrada. Los robots restantes (TaleBot Pro, Sphero Indi, Bee Bot) tampoco ofrecen corrección activa de trayectoria ni son de código abierto.

Teniendo en cuenta el relevamiento hecho en el estado del arte y los requisitos de capacidad de procesamiento, cómputo y la necesidad de tener una arquitectura basada en microcontroladores con algoritmos completamente embebidos, el proyecto presentado en este trabajo busca llenar este vacío mediante una solución que integra reconocimiento de patrones visuales, corrección continua de trayectoria y arquitectura completamente abierta, utilizando componentes de bajo costo y amplia disponibilidad. La solución propuesta cumple con las siguientes capacidades identificadas en la comparación:

- **Reconoce patrones:** Mediante visión por computadora con módulo ESP32-CAM (Ver Capítulo 4) y algoritmo personalizado de procesamiento de imágenes.

- **Corrige trayectoria:** A través de control PID que ajusta continuamente el movimiento del robot basándose en la información visual procesada en tiempo real.

- **Es programable:** Permite definir secuencias de acciones mediante tarjetas físicas interpretadas por el sistema de visión.
- **Es de código abierto:** Hardware y software completamente documentados y accesibles para estudio, modificación y replicación.

Capítulo 4

Diseño de la solución

En este capítulo se describe el diseño de la solución propuesta, detallando la arquitectura general del sistema. Se presentan diagramas sobre la organización de los componentes de hardware y software, así como el flujo de comunicación entre los distintos subsistemas que conforman el robot móvil. A lo largo del capítulo se analizan los principales componentes del sistema y su rol dentro de la arquitectura global.

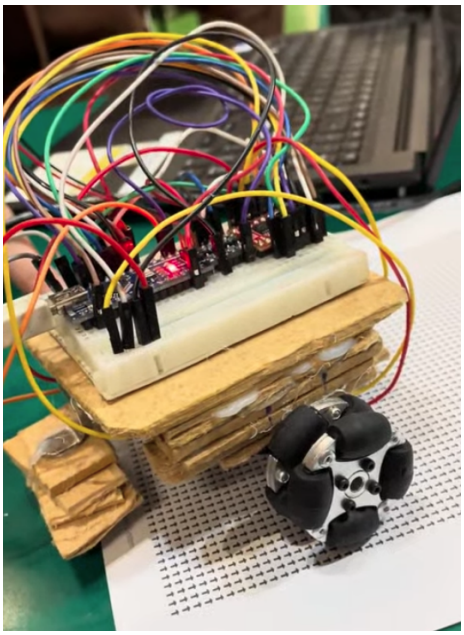


Figura 4.1: Prototipo ejecutando algoritmo de reconocimiento de patrones basado en template matching

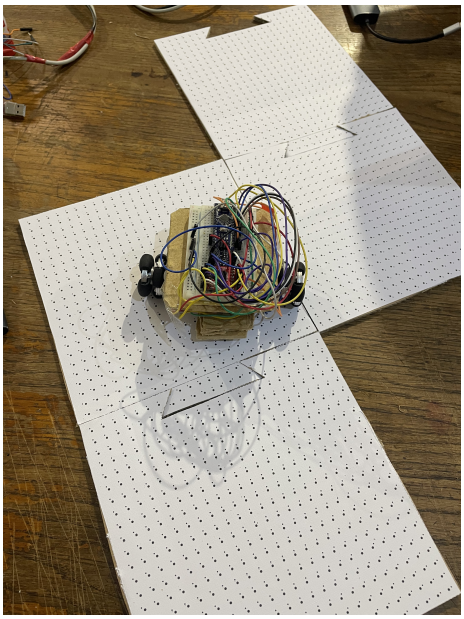


Figura 4.2: Prototipo ejecutando algoritmo de reconocimiento de patrones basado en pares de puntos

4.1. Diagrama de componentes de hardware

La plataforma robótica desarrollada consiste en una arquitectura distribuida con dos unidades de procesamiento independientes; la comunicación entre ambos se realiza mediante el protocolo serial UART.

El ESP32-CAM se encarga del procesamiento de la información sensorial, ejecutando un algoritmo de detección. El Arduino Nano interpreta esta información, ejecuta una estrategia de control y genera señales PWM para el motor driver; este, a su vez, suministra la potencia a los dos motores DC que efectúan el desplazamiento.

Para depuración, cada microcontrolador dispone de su interfaz USB/FTDI que facilita el proceso de pruebas y trazabilidad del sistema en tiempo real.

A continuación, se presenta un diagrama de conexión de componentes de alto nivel y una descripción de cada componente de hardware que conforma el sistema.

(para ver en detalle los diagramas de conexión, esquemas y PCB, dirigirse al Anexo A.)

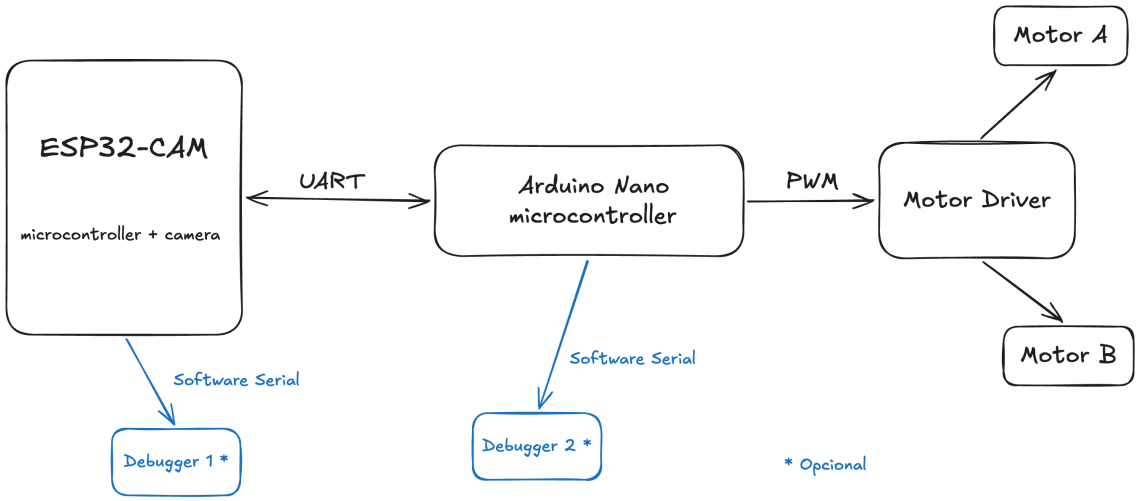


Figura 4.3: Diagrama de componentes de alto nivel

4.1.1. Arduino Nano

El Arduino Nano es una placa de desarrollo compacta basada en el microcontrolador ATmega328P. Opera a 5 V con un reloj de 16 MHz y cuenta con 32 KB de memoria flash, 2 KB de SRAM y 1 KB de EEPROM. Dispone de 14 pines de entrada/salida digital (de los cuales 6 pueden usarse como salidas PWM), 8 entradas analógicas y comunicación serial mediante UART.

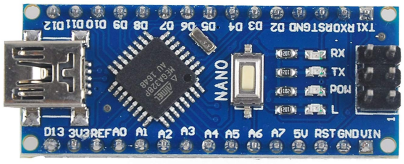


Figura 4.4: Arduino Nano

La Arduino Nano actúa como la unidad de control central del robot. Sus responsabilidades principales son:

- Control de motores: Genera señales PWM y direccionales para comandar el puente H (Motor Driver) que controla los dos motores DC del sistema de tracción diferencial. Implementa control proporcional-integral-derivativo (PID) para corrección de trayectoria.
- Procesamiento de comandos: Recibe estructuras de datos desde el ESP32-CAM vía comunicación serial UART a 115200 baud, conteniendo información sobre detección de la cámara y ángulo de rotación necesario.
- Lógica de navegación: Implementa algoritmos de seguimiento que convierten los datos de visión (ángulo de rotación) en comandos de movimiento diferencial para los motores, aplicando velocidades asimétricas para ejecutar giros proporcionales al error angular detectado.
- Sincronización del sistema: Coordina el handshake inicial con el ESP32-CAM y administra el ciclo de solicitud-respuesta para procesamiento continuo de frames.

4.1.2. Puente H (Motor Driver)

El TB6612FNG es un driver dual para motores DC con capacidad para controlar dos motores de forma independiente. Utiliza tecnología MOSFET de baja resistencia para alta eficiencia energética. Cada canal puede proporcionar hasta 1.2A de corriente continua (3.2A pico) con control bidireccional completo.

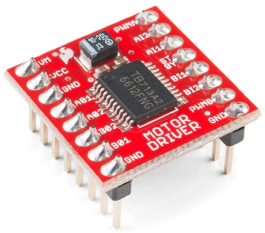


Figura 4.5: Motor Driver TB6612FNG

4.1.3. Motores

Dos motores de corriente continua (DC) con reductora incorporada (gear motors) configurados en un sistema de tracción diferencial. Cada motor acciona una rueda independiente.

Los motores constituyen el sistema de actuación y locomoción del robot.

Tracción diferencial: La configuración de dos motores independientes permite al robot: Avanzar, Retroceder, Girar (velocidades asimétricas entre motores) y Pivote (un motor gira en dirección opuesta al otro para rotación sobre su eje).

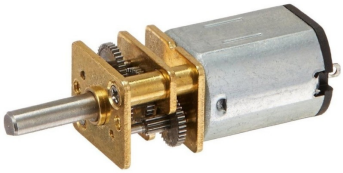


Figura 4.6: DC motor

4.1.4. Conversor de voltaje

Módulo regulador de voltaje DC-DC step-down (buck converter) que reduce el voltaje de la batería principal a niveles seguros para los componentes electrónicos. Posee capacidad de regulación eficiente mediante conmutación.

El conversor de voltaje proporciona regulación y distribución de energía para los diferentes subsistemas brindando protección a los componentes. Previene daños por sobrevoltaje en los componentes electrónicos si se utiliza una batería con voltaje superior al nominal.

4.1.5. Debuggers USB

El módulo USB-TTL CP2102 es un adaptador que convierte la interfaz USB del ordenador a señales TTL seriales (UART). Incluye un chipset CP2102 que implementa el puente $USB \leftrightarrow UART$ y pines de salida TX, RX, VCC y GND accesibles mediante cables Dupont. Es muy usado para programar y monitorizar microcontroladores que no disponen de un puerto USB integrado o cuando se necesita una segunda conexión serial.

En nuestro prototipo se usaron dos módulos CP2102: uno conectado a la ESP32-CAM y otro al Arduino Nano para tareas de depuración y monitoreo independientes:

- Monitorización serial en tiempo real: Recibe los mensajes de depuración serial generados por el ESP32-CAM y por el Arduino Nano para analizar el flujo de datos (handshake, comandos REQ/ACK, SimpleCommand, logs de diagnóstico, tiempos y profiling).
- Diagnóstico y pruebas: Facilita la inspección de valores intermedios (ángulos detectados, estados del PID, valores de sensores) y la inyección manual de comandos durante pruebas.

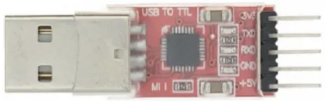


Figura 4.7: Módulo Usb A Ttl Cp2102

4.1.6. ESP32-CAM

El ESP32-CAM es un módulo compacto que integra un microcontrolador ESP32 dual-core con cámara OV2640 de 2 megapíxeles. Incluye memoria PSRAM externa para buffering de imágenes, ranura para tarjeta microSD, LED flash integrado, y capacidades WiFi/Bluetooth (no utilizadas en este proyecto).

Figura 4.8: ESP32Cam

Funcionalidad dentro del sistema:

El ESP32-CAM funciona como el subsistema de visión artificial y procesamiento de imágenes.

- Captura de imágenes: Configura y controla la cámara OV2640 en modo FRAMESIZE 96X96 pixeles en escala de grises para maximizar velocidad de procesamiento. Captura frames bajo demanda al recibir comando REQ del Arduino Nano.
- Procesamiento de imagen en tiempo real: Implementa y ejecuta el algoritmo de detección de patrones.
- Comunicación de alto nivel: Transmite estructura SimpleCommand de 8 bytes al Arduino Nano vía UART (115200 baud) conteniendo la información necesaria especificada en el protocolo de comunicación, entre estos datos enviados se destaca detected_card y rotation_angle.

4.1.7. Dimensiones

El prototipo construido, integrando todos los componentes de hardware descritos en las secciones anteriores, resulta en un robot de dimensiones compactas: 13 cm de ancho, 9 cm de largo y 6 cm de alto. Estas medidas permiten que el robot opere en espacios reducidos, como una mesa o escritorio de aula, cumpliendo con las restricciones físicas y constructivas establecidas en los requerimientos del proyecto.

4.2. Componentes de software

Toda la plataforma fue desarrollada utilizando diversas herramientas de software. En particular, para el proceso de desarrollo se emplearon, entre otras, Visual Studio Code [22] y PlatformIO [23], las cuales fueron configuradas con distintas tareas, configuraciones y extensiones según las necesidades del proyecto. Tanto estas configuraciones como información adicional sobre el entorno de desarrollo se detallan en el Anexo C.

4.2.1. Algoritmo de reconocimiento de patrones

El algoritmo de reconocimiento de patrones constituye uno de los componentes fundamentales del sistema. Su función principal es llevar a cabo el proceso de extracción de información a partir de los patrones visuales, el cual comprende las siguientes etapas:

- Captura de una imagen del patrón a analizar.
- Pos-procesado de la imagen para eliminar ruido e inconsistencias de la misma mediante la técnica de binary thresholding [24].
- Ejecución del proceso de extracción de datos, que incluye la determinación del ángulo de rotación necesario para la corrección de trayectoria y la detección de la tarjeta sobre la cual se encuentra el robot.
- Envío de la información obtenida al algoritmo de movimiento, responsable de la gestión y el funcionamiento de los motores.

4.2.2. Algoritmo de control de movimiento

Este componente, junto con el algoritmo de reconocimiento de patrones, constituye uno de los elementos centrales del sistema. Su función principal es controlar el funcionamiento de los motores, garantizando un movimiento preciso y estable del robot.

El algoritmo de control comprende las siguientes etapas:

- Recepción del ángulo de rotación requerido y de la información de la tarjeta sobre la cual se encuentra posicionado el robot.
- Ajuste del controlador PID con el objetivo de asegurar un movimiento fluido y preciso. Dicho controlador es retroalimentado en cada iteración del bucle principal, permitiendo su adaptación a los distintos ángulos de rotación recibidos.
- Activación de los motores en función de la señal de control generada por el controlador PID.

4.2.3. Protocolo de comunicación

Para poder coordinar ambos microcontroladores, se optó por desarrollar un protocolo híbrido bidireccional propio que se articula en dos fases diferenciadas. La primera, de inicialización (handshake), establece la sincronización entre módulos al arranque y transfiere la configuración del algoritmo antes de comenzar la operación. La segunda, la fase operativa, consiste en un ciclo

continuo donde el Arduino Nano solicita procesamiento al ESP32-CAM y este responde con los resultados de detección.

La combinación de comandos ASCII —fáciles de depurar durante el desarrollo— en la fase de inicialización, con estructuras binarias serializadas en la fase operativa, equilibra dos necesidades contrapuestas: la legibilidad necesaria durante el desarrollo y la eficiencia en ancho de banda y latencia que exige el ciclo de control en tiempo real.

Para más detalles en cuanto a la implementación del protocolo ver 5.1.

4.3. Funcionamiento con tarjetas y patrones

El prototipo fue diseñado para operar de forma tal que el movimiento del robot se programa mediante tarjetas físicas con patrones que son colocadas en el suelo, de modo que al posicionarse sobre ellas el robot lee e interpreta dicho patrón para determinar su siguiente acción.

Cada tarjeta codifica dos piezas de información fundamentales para el control del sistema:

- **Identificador de tarjeta**: un número entero que identifica de forma única a la tarjeta y permite asociarla a una acción o comando específico.
- **Ángulo de rotación**: un valor en grados que representa la orientación relativa de la tarjeta respecto a la posición canónica, y que se utiliza para corregir la trayectoria del robot.

Con estos dos datos, el módulo de control de motores puede tanto corregir la orientación del robot aplicando el ángulo recibido, como ejecutar comportamientos específicos en función del identificador de tarjeta detectado.

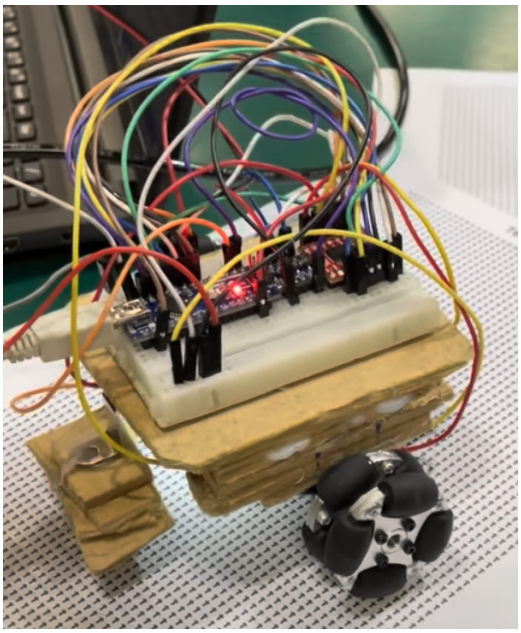


Figura 4.9: Prototipo posicionado sobre una tarjeta de template matching.

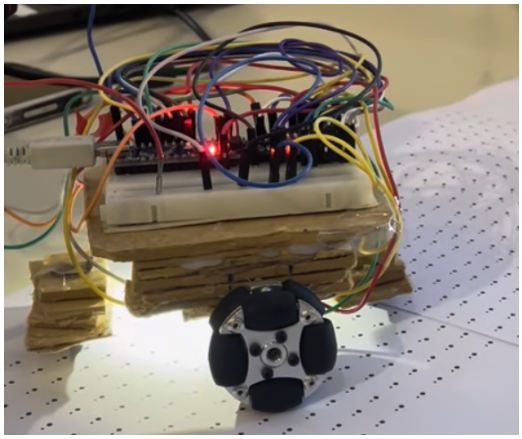


Figura 4.10: Prototipo posicionado sobre una tarjeta de par de puntos.

4.3.1. Diseño físico de las tarjetas

Las tarjetas son elementos físicos impresos que se ubican en el suelo formando un recorrido. El robot se desplaza por encima de ellas, y su cámara inferior captura la imagen del patrón que se encuentra directamente bajo el robot.

Se diseñaron dos familias de tarjetas, cada una asociada a uno de los algoritmos de reconocimiento desarrollados en el proyecto:

Tarjetas de template matching

Contienen imágenes o patrones gráficos predefinidos (flechas, círculos, líneas onduladas, entre otros) que el algoritmo intenta emparejar contra plantillas de referencia almacenadas en el sistema. La identidad de la tarjeta queda determinada por el template que mejor coincide, y la orientación se obtiene del ángulo que minimiza el error de emparejamiento.

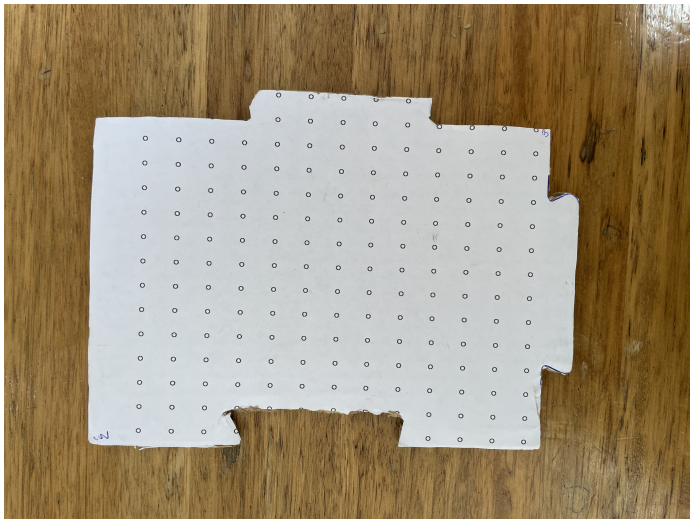


Figura 4.11: Tarjeta de template matching con patrón circulo

Tarjetas de par de puntos

Contienen exactamente dos puntos circulares de diferente tamaño impresas sobre fondo blanco. El identificador de tarjeta está codificado en la **distancia** entre los centros de ambos puntos, mientras que el ángulo de rotación queda definido por la **dirección espacial** que va desde la mancha más pequeña hacia la más grande.

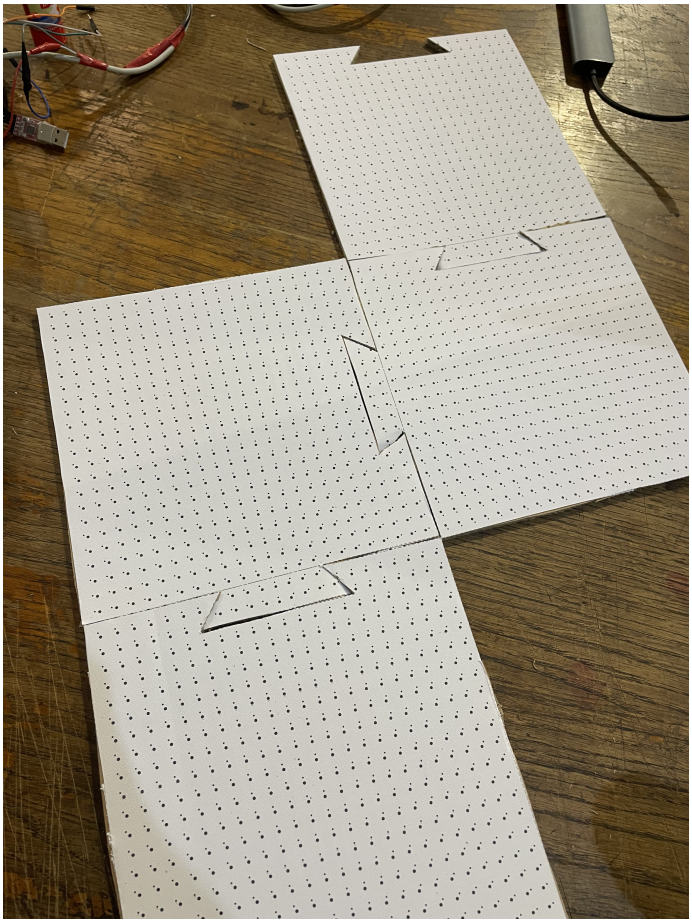


Figura 4.12: Recorrido formado con tarjetas de pares de puntos

4.4. Dificultades encontradas

Durante la fase de investigación y diseño de la solución, se identificaron diversos desafíos técnicos y limitaciones que requirieron análisis y adaptación de la estrategia inicial. Los principales problemas encontrados se clasifican en las siguientes categorías:

4.4.1. Limitaciones en la documentación y recursos disponibles

La mayoría de la documentación existente sobre detección de patrones y visión por computadora está orientada a lenguajes de alto nivel (Python, JavaScript) y bibliotecas especializadas como OpenCV. Esta situación presentó un desafío significativo:

- **Brecha de implementación**: Los algoritmos documentados en OpenCV y bibliotecas similares no son directamente trasladables a microcontroladores debido a sus requerimientos de memoria y dependencias de bibliotecas externas.

4.4.2. Limitaciones de hardware y tecnología propietaria

Durante la investigación inicial, se evaluó replicar el enfoque utilizado por robots educativos comerciales como Qobo, que emplean sensores de identificación óptica (OID) para la detección de patrones. Sin embargo, se identificaron las siguientes limitaciones:

- **Tecnología propietaria**: Qobo y productos similares utilizan sensores OID propietarios con algoritmos de detección cerrados, no disponibles para implementación independiente.
- **Indisponibilidad de sensores OID**: Se intentó adquirir sensores OID comerciales sin éxito, debido a restricciones de distribución y disponibilidad en el mercado.

Ante estas limitaciones, se optó por desarrollar una solución basada en visión por computadora utilizando la cámara ESP32-CAM y algoritmos de procesamiento de imágenes implementados manualmente.

Capítulo 5

Implementación

El presente capítulo tiene por propósito describir la arquitectura y el funcionamiento interno de los principales componentes lógicos del sistema. Se detallan, en particular, el mecanismo de comunicación implementado entre las unidades de procesamiento, los algoritmos de reconocimiento de patrones desarrollados, el algoritmo de control de motores y las aplicaciones web empleadas para generar los distintos patrones. El énfasis se sitúa en las decisiones de diseño, la estructura de los componentes y los criterios de operación adoptados.

La evolución del sistema puede sintetizarse en dos etapas principales.

La **primer etapa** abordó el problema central del proyecto: la detección de tarjetas de programación tangibles y la determinación de su orientación en el espacio. Se exploró el uso de template matching con invariancia a rotación, desarrollando múltiples variantes del algoritmo para optimizar el balance entre precisión y rendimiento computacional. Esta etapa evidenció tanto las capacidades como las limitaciones del microcontrolador ESP32-CAM, llevando al desarrollo de técnicas de optimización específicas como el guardado en memoria de templates rotados y la búsqueda con terminación temprana.

La **segunda etapa** surgió como respuesta a las restricciones de procesamiento identificadas durante las pruebas con template matching. Se diseñó e implementó un algoritmo completamente nuevo basado en la detección de pares de puntos de diferentes tamaños, donde la distancia entre ellos codifica el identificador de la tarjeta, mientras que su disposición espacial determina el ángulo de rotación. Esta solución demostró ser más eficiente computacionalmente que el template matching, permitiendo tiempos de respuesta adecuados para la interacción en tiempo real requerida.

A continuación se presenta el protocolo de comunicación implementado y una descripción pormenorizada de cada una de las dos etapas mencionadas, incluyendo los fundamentos teóricos, las decisiones de diseño y los resultados obtenidos en las distintas iteraciones de desarrollo.

Todo el código fuente de los algoritmos desarrollados se encuentran disponibles de forma pública en el Gitlab de la facultad de ingeniería [25].

5.1. Protocolo de comunicación

El sistema está compuesto por dos microcontroladores que se comunican mediante protocolo UART utilizando los pines TX/RX:

- Arduino Nano el cual ejecuta el algoritmo de control de movimiento.
- ESP32-CAM la cual que ejecuta el algoritmo de reconocimiento de patrones.

Para poder comunicar ambos microcontroladores se optó por desarrollar un protocolo híbrido propio, que combina comandos ASCII para la fase de handshake, y estructuras binarias serializadas para la fase operativa. Esta decisión de diseño equilibra las necesidades de:

- Depuración y desarrollo: Comandos ASCII son legibles y facilitan la depuración mediante monitores serie estándar
- Eficiencia operativa: Estructuras binarias minimizan overhead de transmisión en el ciclo crítico de control
- Robustez: Comandos terminados en newline (\n) permiten sincronización y detección de errores de framing

Comandos del Protocolo de Comunicación

En la implementación fue necesario utilizar siete comandos para poder controlar correctamente el handshake inicial y además mantener una operación cíclica. Para más detalles ver tabla 5.1.

Todos los comandos ASCII siguen el formato:

```
<COMANDO>\n
```

Donde `\n` es el carácter de nueva línea, utilizado como delimitador para sincronización y parsing mediante la función

```
Serial.readStringUntil('\n');
```

Comando	**Valor Literal**	**Dirección**	**Tipo**	**Tamaño**	**Descripción**
`HANDSHAKE_INIT_COMMAND`	`"INIT_HANDSHAKE\n "`	Arduino → ESP32	ASCII	15 bytes	Señal de inicio de handshake
`REQUEST_CONFIG_COMMAND`	`"502\n "`	ESP32 → Arduino	ASCII	4 bytes	Solicitud de configuración
`AlgorithmConfig`	(binario)	Arduino → ESP32	Binary	32 bytes	Estructura de configuración
`CONFIG_RECEIVED_COMMAND`	`"503\n "`	ESP32 → Arduino	ASCII	4 bytes	Confirmación de config
`SIMPLE_INITIALIZED_COMMAND`	`"INIT\n "`	ESP32 → Arduino	ASCII	5 bytes	Sistema listo
`SIMPLE_REQUEST_COMMAND`	`"REQ\n "`	Arduino → ESP32	ASCII	4 bytes	Petición de frame
`SimpleCommand`	(binario)	ESP32 → Arduino	Binary	6 bytes	Resultado de detección

Tabla 5.1: Tabla Resumen de Comandos

Protocolo de inicialización y Configuración (Handshake)

El protocolo de handshake que implementamos es un intercambio en cuatro pasos que permite sincronizar el tiempo entre las partes y validar los parámetros antes de empezar a comunicarse. (Ver figura 5.1)

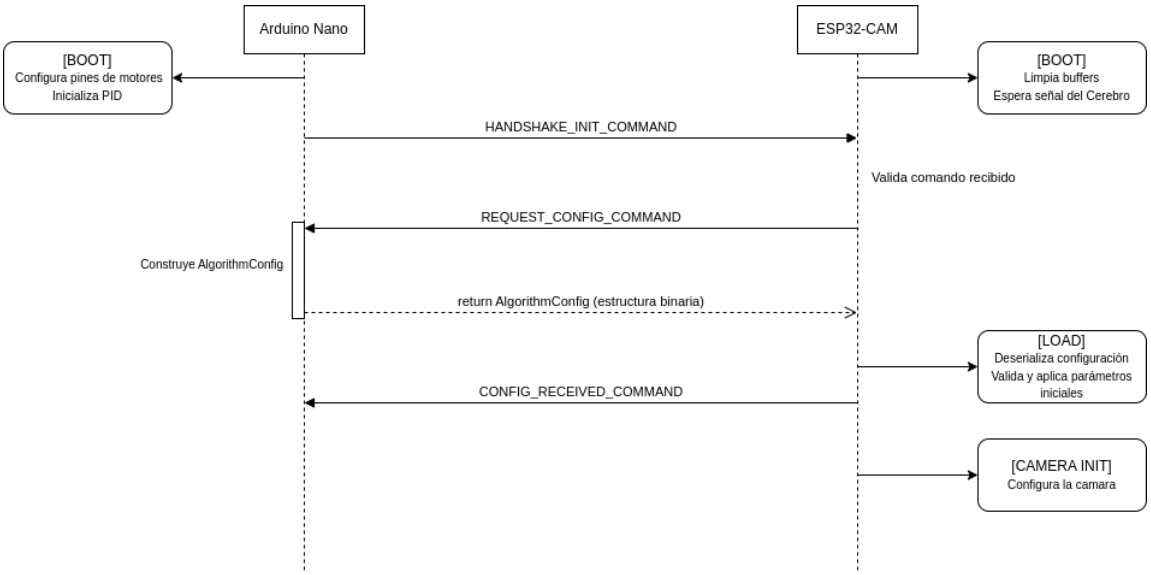


Figura 5.1: Diagrama de Comunicación del handshake

Protocolo de Operación Cíclica

Una vez completada la fase de inicialización, el sistema entra en un ciclo de operación continuo basado en el paradigma petición-respuesta (request-response). Este modelo garantiza la sincronización explícita entre la adquisición de datos de visión y la aplicación de comandos de control a los actuadores. El ciclo operativo de control se implementó de la siguiente manera:

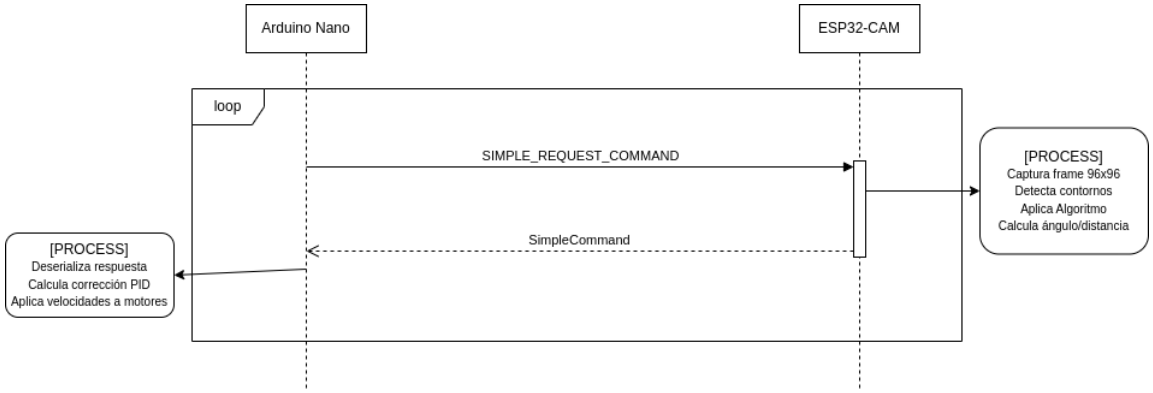


Figura 5.2: Diagrama de comunicación del ciclo de captura y control

5.2. Primera etapa: Template Matching

En el desarrollo del presente proyecto se implementaron y evaluaron distintas variantes principales de algoritmos de template matching, cada una optimizada para abordar desafíos específicos del sistema de reconocimiento de patrones en el robot móvil. En particular, se prestó atención principalmente al problema de las rotaciones de las tarjetas que contienen los patrones a reconocer. Estas variantes nos permitieron iterar sobre nuestro algoritmo y definirlo.

Se utilizaron varias variantes de algoritmos de comparación para realizar pruebas, por ejemplo, los algoritmos SSD (2.2.2) y con NCC (2.2.3).

Por otro lado, la técnica de template matching se puede hacer invariante a la rotación e invariante a la escala.

Debido a las características del prototipo y las limitaciones previamente mencionadas, la variante de template matching implementada fue con invarianza a rotación, debido a la necesidad del robot de poder corregir la trayectoria. La invarianza a escala no se pudo implementar debido a la capacidad de procesamiento limitada.

Template matching con SAD

La implementación inicial consiste en crear un algoritmo de template matching básico utilizando la métrica Sum of Absolute Differences (SAD). Este enfoque fue seleccionado por su eficiencia computacional y simplicidad, características cruciales para la ejecución en tiempo real en el microcontrolador ESP32-CAM.

El algoritmo opera en imágenes en escala de grises, lo que reduce el consumo de memoria en un 50 % comparado con formatos de color. Los templates se almacenan como matrices binarias de 28×28 píxeles, donde cada píxel tiene valor 0 (negro) o 255 (blanco).

El proceso de búsqueda se realiza deslizando el template sobre la imagen capturada, calculando el SAD promedio por píxel en cada posición. El algoritmo recorre la imagen completa con un paso configurable (search_region_size), permitiendo ajustar el balance entre velocidad de procesamiento y exhaustividad de la búsqueda. Este parámetro configurable posibilita adaptaciones futuras según las restricciones de tiempo real del sistema.

La validación de coincidencias se realiza mediante umbrales duales: un umbral para píxeles negros (T_{black}) y otro para píxeles blancos (T_{white}). El cálculo se basa en el SAD promedio de cada categoría de píxeles, permitiendo una comparación normalizada independiente del tamaño del template. Una coincidencia se considera válida cuando:

$$\overline{SAD}_{black} < T_{black} \quad \text{y} \quad \overline{SAD}_{white} < T_{white} \tag{5.1}$$

donde $\overline{SAD}_{black}$ y $\overline{SAD}_{white}$ representan el SAD promedio por píxel para píxeles negros y blancos respectivamente.

Los umbrales se calibraron empíricamente mediante pruebas en condiciones reales de iluminación. Cada template (ver Anexo E) tiene umbrales específicos optimizados según sus características visuales: el template de flecha utiliza $T_{black} = 46$ y $T_{white} = 12$ debido a su patrón predecible, mientras que el template de círculo requiere $T_{black} = 120$ y $T_{white} = 40$ por su mayor complejidad geométrica. Los templates de círculo negro y líneas onduladas utilizan valores intermedios ($T_{black} = 40$, $T_{white} = 40$).

Detección de rotación

La segunda iteración extiende el algoritmo básico con el objetivo de manejar rotaciones de las tarjetas de programación. Resulta fundamental para la corrección de trayectoria que el sistema pueda reconocer los patrones independientemente de su rotación.

El algoritmo implementa una búsqueda multi-angular utilizando un conjunto optimizado de 7 ángulos de rotación:

$$\Theta = \{-45°, -30°, -15°, 0°, 15°, 30°, 45°\} \tag{5.2}$$

Este conjunto de ángulos fue seleccionado tras experimentación empírica, balanceando cobertura angular y costo computacional.

Para cada ángulo $\theta \in \Theta$, se genera una versión rotada del template T_θ aplicando una transformación de rotación bilineal:

$$T_\theta(i', j') = T\left(\cos\theta \cdot i' - \sin\theta \cdot j', \sin\theta \cdot i' + \cos\theta \cdot j'\right) \tag{5.3}$$

El algoritmo calcula el SAD para cada template rotado y selecciona el ángulo que produce el menor error:

$$(\theta^*, x^*, y^*) = \arg\min_{\theta,x,y} SAD_\theta(x, y) \tag{5.4}$$

Para mejorar el rendimiento, se implementó un sistema de caché de templates rotados. Los templates rotados se pre-calculan y almacenan en la memoria PSRAM del ESP32-CAM durante la inicialización, evitando recalcular las rotaciones en cada frame.

Estrategia de búsqueda optimizada

La tercera iteración implementa una estrategia de búsqueda optimizada denominada "Simple First-Match Search"para mejorar el rendimiento en tiempo real. Esta aproximación prioriza la velocidad de respuesta sobre la precisión sub-píxel, una decisión de diseño motivada por las limitaciones de procesamiento del ESP32-CAM y los requisitos de respuesta en tiempo real del sistema.

El algoritmo prueba los ángulos de rotación en un orden de prioridad específico, comenzando por 0° (sin rotación) y expandiendo hacia ángulos mayores:

$$\text{Orden de prueba: } \{0°, -15°, 15°, -30°, 30°, -45°, 45°\} \tag{5.5}$$

Para cada ángulo θ en el orden de prueba, el algoritmo recorre la imagen buscando posiciones donde tanto el SAD de píxeles negros como el de píxeles blancos cumplan simultáneamente con sus umbrales respectivos:

$$\text{Match válido si: } SAD_{black}(x, y, \theta) \leq T_{black} \land SAD_{white}(x, y, \theta) \leq T_{white} \tag{5.6}$$

En cuanto se encuentra una posición que satisface ambas condiciones, el algoritmo retorna inmediatamente ese resultado como la mejor coincidencia, evitando el costo computacional de escanear toda la imagen en múltiples rotaciones. Esta estrategia de 'salida temprana' reduce significativamente el tiempo de procesamiento promedio, especialmente en escenarios donde el template está orientado cerca de 0°.

Además, se implementaron mecanismos de validación de calidad para garantizar la confiabilidad de las detecciones:

- **Filtrado por umbrales duales SAD:** Se descartan detecciones que no cumplan simultáneamente con los umbrales calibrados para píxeles negros y blancos. Cada template tiene umbrales específicos (T_{black} y T_{white}) ajustados empíricamente según sus características visuales. Solo se acepta una detección cuando $SAD_{black} \leq T_{black}$ y $SAD_{white} \leq T_{white}$ simultáneamente.

- **Selección por frecuencia de aparición:** Cuando se detectan múltiples templates en una misma imagen, el sistema selecciona aquel con mayor frecuencia de aparición (match_count). Este criterio permite identificar el template dominante en la escena, reduciendo falsos positivos causados por coincidencias parciales o ruido visual.

- **Cálculo de confianza normalizado:** La confianza de cada detección se calcula mediante un score de correlación normalizado basado en el SAD ponderado. Para imágenes binarias, la confianza se define como $confidence = 1{,}0 - \frac{SAD_{weighted}}{255{,}0}$, donde $SAD_{weighted} = \max(SAD^{w}_{black}, SAD^{w}_{white})$ representa el peor caso entre los SAD ponderados por pesos espaciales (enfatizando el centro del template). Este score se escala a un valor entero de 8 bits (0-255) para transmisión al módulo de control.

5.3. Segunda etapa: Algoritmo de pares de puntos

El algoritmo de pares de puntos presentado en esta sección fue diseñado íntegramente por nosotros, constituyendo una contribucion original para el proyecto. Su concepción surge de la analogía con los *campos vectoriales*, una herramienta fundamental de la física y la matemática utilizada para modelar fenómenos donde a cada punto del espacio se le asocia una magnitud con dirección y sentido (como ocurre, por ejemplo, con los campos gravitacionales, electromagnéticos o los flujos de fluidos). Inspirándonos en esta idea, trasladamos el concepto al dominio del reconocimiento visual: así como un campo vectorial codifica dirección mediante la relación geométrica entre puntos del espacio, nuestro algoritmo codifica tanto la identidad de una tarjeta como su ángulo de orientación a partir de la relación geométrica entre dos puntos detectados en la imagen. La distancia euclidiana entre ambos centroides actúa como el *módulo* del vector, determinando qué tarjeta se está leyendo, mientras que el ángulo formado por el segmento que los une actúa como la *dirección* del vector, determinando la corrección de trayectoria a aplicar. A lo largo de esta sección se describen en detalle los principios de funcionamiento del algoritmo, sus etapas de procesamiento y los criterios de diseño que guiaron su implementación.

La implementación del algoritmo se basa en la detección de patrones compuestos por pares de puntos, uno de mayor tamaño que el otro, separados por una distancia característica que denotaremos como d. Su funcionamiento puede describirse de forma estructurada. Al iniciar, se localiza el primer punto válido que cumpla las restricciones de tamaño (umbrales de área). A continuación se busca un segundo punto válido que no comparta componentes con el primero y que cumpla las restricciones espaciales requeridas (restricción distancia mínima y máxima entre ambos puntos).Una vez identificado el par de puntos, se calculan las magnitudes de interés:

- d: distancia euclidiana entre los centroides de ambos puntos.
- θ: ángulo de rotación definido a partir de la disposición espacial de los centroides en la imagen.

Un desglose más específico de las etapas principales del algoritmo es la siguiente:

1. **Preprocesado y umbralización:** la imagen en escala de grises se binariza mediante una tabla de umbrales por píxel $T(x, y)$.
2. **Búsqueda en cuadrados concéntricos:** se inspeccionan cuadrados concéntricos crecientes desde el centro de la imagen hasta encontrar un píxel negro semilla perteneciente al primer punto.
3. **Exploración del primer punto:** se realiza un *flood-fill* desde la semilla para obtener el área del punto y su centroide.
4. **Búsqueda radial del segundo punto:** desde el centroide del primer punto se explora en cuadrados concéntricos hasta una distancia máxima permitida, ignorando píxeles ya visitados del primer punto.
5. **Exploración del segundo punto:** se repite el *flood-fill* para el segundo punto y se obtienen sus propiedades geométricas.

6. **Clasificación de tarjeta:** se compara la distancia medida entre ambos centroides con una tabla de distancias preconfigurada, considerando una tolerancia permitida.

7. **Cálculo del ángulo de rotación:** primero se computa el ángulo entre los dos centroides usando la función atan2(dy, dx) y se convierte a grados en el rango $[-180°, 180°]$. Luego se computa la longitud del segmento de recta que une ambos centroides del par mediante distancia euclidiana.

Si todas las validaciones anteriores se satisfacen, la distancia medida se asocia con una entrada de la tabla de referencia correspondiente a una tarjeta concreta; el algoritmo finalmente retorna dicho identificador de tarjeta junto con el ángulo de rotación.
Para visualizar el pseudocódigo (ver Anexo D).

5.4. Algoritmo de control de motores

El módulo de control de motores implementado en el Arduino Nano recibe los datos de detección del módulo de visión (ESP32-CAM) y ejecuta acciones de movimiento basadas en el ángulo de rotación detectado. El sistema utiliza un controlador PID (Proporcional-Integral-Derivativo) para mantener al robot orientado hacia el objetivo, implementando además un mecanismo de giro en el lugar (pivot turn) para correcciones angulares grandes.

El algoritmo se compone de las siguientes etapas principales:

1. **Inicialización del sistema:** Configuración de pines del driver de motores TB6612FNG, inicialización del controlador PID con parámetros $K_p = 2{,}0$, $K_i = 0{,}5$, $K_d = 1{,}0$ (ver 5.4.1), y establecimiento del protocolo de comunicación serial con el módulo de visión a 115200 baudios.

2. **Handshake y configuración:** Establecimiento de comunicación con el ESP32-CAM mediante envío del comando `INIT_HANDSHAKE`, seguido de la transmisión de la estructura de configuración `AlgorithmConfig` que incluye parámetros de umbralización, detección de puntos, y tabla de distancias para clasificación de tarjetas.

3. **Solicitud de procesamiento:** Envío del comando `SIMPLE_REQUEST_COMMAND` al módulo de visión para solicitar la captura y procesamiento de un nuevo frame.

4. **Recepción de datos de detección:** Recepción de la estructura `SimpleCommand` (8 bytes) que contiene el ID de tarjeta detectada (`detected_card`), el ángulo de rotación (`rotation_angle`), y un flag de validez (`valid_angle`).

5. **Determinación del ángulo de control:** Si la detección es válida, se utiliza el nuevo ángulo recibido y se almacena como último ángulo válido. Si la detección es inválida pero existe un ángulo válido previo, se reutiliza el último ángulo conocido para mantener continuidad en el movimiento. Si no hay detección válida ni ángulo previo, se detienen los motores.

6. **Selección del modo de control:** Para ángulos mayores a 70° con detección válida, se activa el modo *pivot turn* donde los motores giran en direcciones opuestas para rotar en el lugar. Para ángulos menores, se utiliza control PID diferencial.

7. **Cálculo de velocidades con PID:** En modo PID, se calcula la corrección necesaria usando el ángulo como entrada (`input = rotation_angle`) y cero como valor de salida deseado (`setpoint = 0`). La salida del PID se aplica diferencialmente a los motores: `leftSpeed = baseSpeed - output`, `rightSpeed = baseSpeed + output`.

8. **Limitación y aplicación de velocidades:** Las velocidades calculadas se limitan al rango $[\texttt{minSpeed}, \texttt{maxSpeed}] = [15, 55]$ PWM para garantizar que los motores superen la fricción estática pero no excedan la velocidad máxima segura. Se configuran las señales de dirección (pines `AIN1/AIN2` y `BIN1/BIN2`) y PWM (`PWMA/PWMB`) del driver TB6612FNG.

9. **Ejecución temporal del control:** Las velocidades calculadas se mantienen aplicadas durante 500ms antes de solicitar un nuevo frame, permitiendo que el robot ejecute la corrección antes de re-evaluar su orientación.

10. **Verificación de timeout de seguridad:** Si no se reciben comandos válidos del módulo de visión durante más de 2 segundos (`COMMAND_TIMEOUT = 2000ms`), se detienen los motores automáticamente.

El sistema implementa dos estrategias de control complementarias:

Control PID diferencial: Utilizado para correcciones finas ($|\theta| \leq 70°$), aplica la salida del controlador PID de forma diferencial a ambos motores. El motor izquierdo reduce su velocidad proporcionalmente a la corrección negativa mientras el motor derecho la aumenta, generando un giro gradual. Los límites de salida del PID están configurados en ± 50 unidades PWM para evitar oscilaciones bruscas.

Pivot turn: Activado para correcciones grandes ($|\theta| > 70°$ con detección válida), hace que los motores giren en direcciones opuestas a velocidad fija (`pivotSpeed = 30` PWM). Para ángulos positivos (giro a la derecha), el motor izquierdo avanza y el derecho retrocede; para ángulos negativos (giro a la izquierda), se invierte la configuración.

Ver Anexo D para visualizar el pseudocódigo.

5.4.1. Constantes PID

Para determinar las constantes del controlador PID mencionado en la sección 5.4, se empleó el método de *ajuste manual* [26]. Este enfoque consiste en un proceso iterativo de prueba y error guiado por el efecto conocido de cada parámetro sobre la respuesta del sistema.

El proceso de ajuste se realizó en condiciones reales de operación siguiendo estos pasos:

1. Se fijó $K_i = 0$ y $K_d = 0$, y se incrementó K_p gradualmente hasta obtener una respuesta de orientación suficientemente rápida sin inducir oscilaciones sostenidas.
2. Con K_p estabilizado, se introdujo K_d para atenuar el sobre-impulso y suavizar las correcciones bruscas al cambiar de dirección.
3. Finalmente se añadió un valor reducido de K_i para eliminar el error de estado estacionario que se manifestaba como una desviación angular residual cuando el robot circulaba en línea recta.
4. Los parámetros se refinaron iterativamente verificando la estabilidad del sistema ante perturbaciones en las condiciones reales de la pista.

Los valores finales obtenidos mediante este proceso son $K_p = 2{,}0$, $K_i = 0{,}5$ y $K_d = 1{,}0$. La Tabla 5.2 resume el rol funcional de cada parámetro en el contexto de este sistema.

Tabla 5.2: Parámetros del controlador PID y su efecto en el sistema.

Parámetro	**Tipo**	**Efecto en el sistema**
$K_p = 2{,}0$	Proporcional	Genera una corrección directamente proporcional al ángulo de error. Un valor de 2,0 produce una respuesta suficientemente agresiva para corregir desviaciones medianas sin inducir oscilaciones en el ciclo.
$K_i = 0{,}5$	Integral	Acumula el error a lo largo de los ciclos y elimina la desviación residual en estado estacionario.
$K_d = 1{,}0$	Derivativo	Anticipa cambios bruscos en el ángulo y amortigua el sobre-impulso, mejorando la estabilidad ante correcciones abruptas.

5.5. Aplicaciones para diseño de tarjetas y patrones

Para poder diseñar las tarjetas y los distintos patrones a utilizar, se crearon aplicaciones web que permiten realizar estos patrones de forma sencilla y amigable. Estas aplicaciones además permiten descargar lo generado para posterior impresión (ver Anexo B para más detalles).

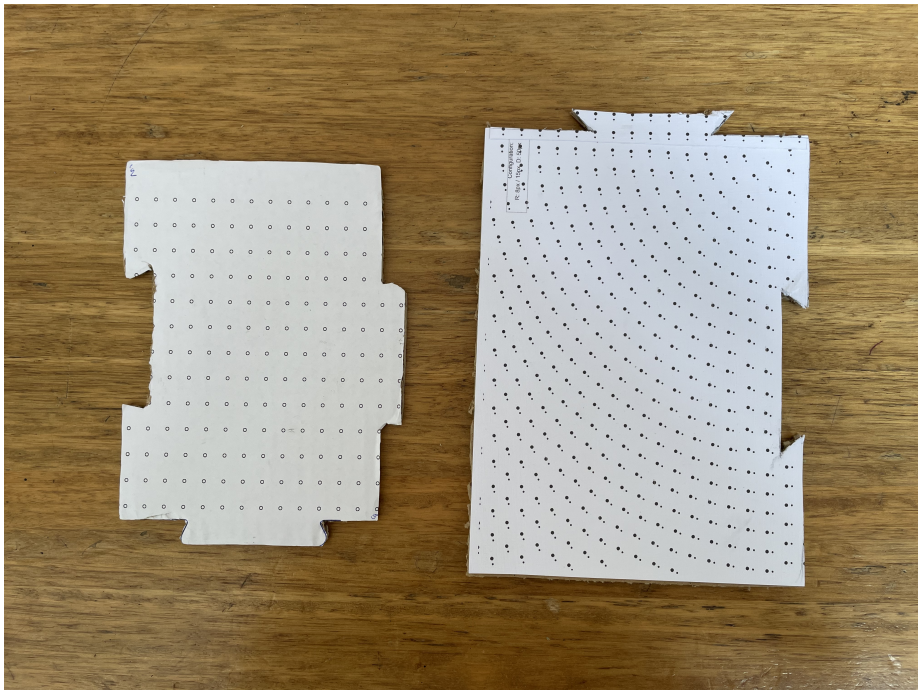


Figura 5.3: Tarjetas con patrones

5.5.1. Template matching

Para el algoritmo de template matching se dispone de un portal en el cual se cargan los templates que serán utilizados por el algoritmo. A partir de estos templates configurados, el sistema permite definir y seleccionar las posiciones en las que se ubicarán los distintos templates dentro de la tarjeta.

En el Anexo B se presenta el sitio con el cual se generan fácilmente estos patrones.

5.5.2. Pares de puntos

Para el algoritmo de reconocimiento basado en pares de puntos se dispone de un portal que permite la definición manual de dichos pares. En este entorno es posible configurar previamente el volumen de cada punto, así como la distancia entre ellos. Este portal facilita significativamente el proceso de diseño, ya que además ofrece funcionalidades para copiar, pegar, rotar y arrastrar los pares de puntos. De esta manera, a medida que se construye la tarjeta, es posible visualizar de forma anticipada su apariencia final una vez impresa.

En el Anexo B se presenta el sitio con el cual se generan fácilmente estos patrones.

Capítulo 6

Experimentación

El presente capítulo describe el proceso de experimentación llevado a cabo para evaluar el desempeño, la viabilidad y las limitaciones de los algoritmos de detección implementados. En particular, se analizan los dos enfoques ejecutados sobre la ESP32-CAM: el algoritmo basado en template matching y el algoritmo optimizado basado en la detección de pares de puntos.

La experimentación se orientó principalmente a caracterizar el comportamiento temporal de cada algoritmo, identificar cuellos de botella en el procesamiento y comparar su rendimiento relativo en condiciones controladas. Para ello, se instrumentó el código con mecanismos de profiling, permitiendo medir los tiempos de ejecución de cada etapa del ciclo de procesamiento, desde la captura de imagen hasta la transmisión de los resultados.

Finalmente, el capítulo presenta los resultados obtenidos a partir de las pruebas experimentales, incluyendo un análisis comparativo de tiempos de ejecución, tasas de detección y precisión, junto con una discusión de las principales limitaciones observadas en cada enfoque.

6.1. Herramientas de depuración

Durante el desarrollo y experimentación del proyecto, se utilizaron diversas herramientas de depuración tanto a nivel software como hardware para capturar, analizar y visualizar el comportamiento de los algoritmos de detección implementados en el ESP32-CAM y de la detección de movimiento.

6.1.1. Software

Sistema de logging vía UART

Los módulos ESP32-CAM y Arduino Nano se comunican vía UART a 115200 baudios para la transmisión de datos de detección y a 9600 baudios para el envío de información de profiling en tiempo real, almacenando además la información recibida en archivos de texto para futuras referencias. Se instrumentó el código con marcadores de tiempo de alta precisión utilizando la función `micros()` de la ESP32 para medir cada fase del ciclo de procesamiento:

- Tiempo de espera por comando del algoritmo de movimiento (`wait_for_arduino`)
- Tiempo de captura de frame de cámara (`frame_capture`)
- Tiempo de umbralización binaria (`binary_threshold`)
- Tiempo de procesamiento del algoritmo principal (`process_pairs` o `template_matching`)
- Tiempo de envío de resultados vía UART (`send_detection`)
- Tiempo total del ciclo completo (`loop_total`)

Script de monitoreo serial

Se desarrolló un script en Python (`serial_monitor.py`) para la captura automatizada de datos de depuración del ESP32-CAM. Este script implementa:

- Detección automática del puerto serial USB (con soporte específico para adaptadores CP2102, CH340 y FTDI)
- Captura continua de logs seriales con timestamp
- Parseo de matrices de píxeles transmitidas por el ESP32-CAM
- Generación automática de imágenes PNG a partir de datos de píxeles capturados
- Almacenamiento organizado de logs y capturas en directorios estructurados

El script utiliza las bibliotecas `pyserial` para comunicación serial, `numpy` para procesamiento de matrices y `Pillow` para generación de imágenes.

Script de visualización de métricas

Para el análisis cuantitativo del desempeño de los algoritmos, se desarrolló un script de visualización en Python (`visualizeMetrics.py`) que procesa los datos de profiling capturados y genera gráficos estadísticos detallados. El script implementa:

- Conversión de logs de métricas en formato JSON
- Cálculo de estadísticas descriptivas (media, mediana, desviación estándar, mínimo, máximo)
- Generación de múltiples visualizaciones:
  - Series temporales de cada fase de procesamiento
  - Histogramas de distribución de tiempos
  - Diagramas de caja (boxplots) para comparación de métricas
  - Gráficos de área apilada mostrando el desglose temporal por componente
  - Gráficos de torta (pie charts) de distribución porcentual de tiempo
  - Mapas de calor (heatmaps) de correlación entre métricas
  - Resumen con estadísticas consolidadas
- Exportación de gráficos en alta resolución (300 DPI) formato PNG

El script utiliza `matplotlib` para generación de gráficos y `numpy` para análisis estadístico. Todas las visualizaciones presentadas en la sección de experimentación fueron generadas con esta herramienta.

6.1.2. Hardware

Para la captura de logs de depuración del ESP32-CAM y del arduino nano se utilizaron un par de Módulos USB a TTL CP2102. Este adaptador serial permite la comunicación entre el componente que lo utiliza y la computadora de desarrollo mediante cables Dupont. El CP2102 proporciona:

- Conversión USB a UART con soporte para velocidades de hasta 1 Mbps (utilizado a 115200 baud)
- Niveles lógicos de 3.3V compatibles con ESP32-CAM
- Aislamiento galvánico para protección del microcontrolador
- Indicadores LED de transmisión (TX) y recepción (RX) para monitoreo visual
- Driver USB-serial ampliamente soportado en Windows, macOS y Linux

La conexión para la ESP32-CAM se realizó mediante cables Dupont entre los pines TX del ESP32-CAM y RX del CP2102 (y viceversa), junto con la conexión de tierra (GND) común. Este método de depuración permitió capturar logs en tiempo real sin interferir con la comunicación UART principal entre el ESP32-CAM (algoritmo de detección de patrones) y el Arduino Nano (algoritmo de control de movimiento), utilizando el puerto serial de depuración del ESP32.

6.2. Pruebas realizadas

Para evaluar el desempeño de los algoritmos implementados, se realizaron pruebas experimentales exhaustivas tanto para el algoritmo de template matching como para el algoritmo de pares de puntos. Las métricas recopiladas incluyen tiempos de ejecución detallados de cada etapa del procesamiento y tasas de detección.

Las pruebas se ejecutaron en condiciones controladas utilizando el mismo hardware (ESP32-CAM con resolución 96x96 píxeles en escala de grises) y tarjetas de programación acordes para cada algoritmo.

6.2.1. Metodología de pruebas

El proceso de captura de métricas se estructuró de la siguiente manera:

1. **Instrumentación del código:** Se insertaron marcadores de tiempo utilizando la función `micros()` de la ESP32 en puntos estratégicos del flujo de ejecución para medir:
   - Tiempo de espera por comando del algoritmo de control de movimiento (wait_for_arduino)
   - Tiempo de captura de frame de cámara (frame_capture)
   - Tiempo de umbralización binaria (binary_threshold)
   - Tiempo de procesamiento del algoritmo principal (process_pairs o template_matching)
   - Tiempo de envío de resultados vía UART (send_detection)
   - Tiempo total del ciclo completo (loop_total)
2. **Captura de datos:** Los datos de tiempo se transmitieron vía UART al monitor serial y se guardaron en archivos de texto plano para análisis posterior.
3. **Procesamiento de métricas:** Se desarrollaron scripts en Python para parsear los logs, calcular estadísticas descriptivas (media, mediana, desviación estándar, mínimo, máximo) y generar visualizaciones gráficas.
4. **Análisis de detecciones:** Para el algoritmo de pares de puntos, se registró adicionalmente información sobre cada detección: ID de tarjeta detectada, ángulo de rotación calculado y validez de la detección.

6.2.2. Tiempos de ejecución

Algoritmo de reconocimiento basado en template matching

El algoritmo de template matching con detección de rotación fue evaluado en tres configuraciones distintas, variando el número de templates habilitados: 2, 3 y 4 templates y se prueba en modo velocidad (Ver sección 5.1). Se distingue la iteración 1 del algoritmo —que incluye la inicialización del caché de templates rotados— de demás iteraciones (iteración 2 en adelante).

Inicialización del caché de rotaciones (Iteración 1) En la primer iteración de cada sesión, el sistema pre-computa y almacena en PSRAM todos los templates habilitados, incluyendo sus rotaciones. La Tabla 6.1 resume los costos de esta fase inicial.

Métrica	**2 templates**	**3 templates**	**4 templates**
Templates cacheados incluyendo rotaciones (cant.)	8	12	16
Tiempo de cache init (ms)	369	493	738
Tiempo promedio por rotación (µs)	195	199	202
Loop total Iteración 1 (ms)	8 023	22 231	22 449

Tabla 6.1: Métricas de la Iteración 1: inicialización del caché de templates rotados

El tiempo de inicialización del caché escala linealmente con la cantidad de templates (369 ms → 493 ms → 738 ms), a razón de aproximadamente **123 ms por template adicional**. Cada rotación individual toma **200 µs**, valor prácticamente constante e independiente de la configuración utilizada y despreciable en la práctica.

Rendimiento en régimen permanente (iteraciones estables)

Se denominan iteraciones estables del algoritmo a aquellas que no cachean nada en memoria.

La Tabla 6.2 presenta las estadísticas descriptivas de las iteraciones estables para cada configuración. Los valores corresponden a los promedios de la iteración 2 en adelante.

Métrica	**2 templates**	**3 templates**	**4 templates**
Loop total — media (ms)	**14 548**	**21 628**	**23 455**
Template matching — media (ms)	**14 297**	**21 378**	**23 204**
Overhead de comunicación — media (ms)	250	251	251
Costo de rotaciones por frame — media (ms)	0.87	1.34	1.42
rotation_single individual — media (µs)	162.7	162.0	161.8
Rotaciones durante matching — media (n)	5.3	8.3	8.8
FPS equivalente	**0.069**	**0.046**	**0.043**

Tabla 6.2: Métricas de tiempo del algoritmo de template matching

Del análisis de estos resultados se observa:

- **Escalabilidad con número de templates:** El costo de template matching escala con la cantidad de patrones habilitados. Pasar de 2 a 3 templates agrega **+7 080 ms** (+49,5 %), mientras que de 3 a 4 templates agrega **+1 827 ms** (+8,5 %). La diferencia entre estos incrementos se explica por la terminación temprana: en la configuración de 4 templates, varios frames presentaron detecciones exitosas en ángulo 0°, acortando la búsqueda para ese patrón. En ausencia total de detecciones, cada template adicional aporta aproximadamente **7 100 ms** de costo adicional por ciclo, confirmando la complejidad lineal en N_t.
- **Variabilidad según detección:** La configuración de 4 templates muestra una desviación estándar de 3 513 ms, significativamente mayor que las de 2 y 3 templates (∼10 ms). Esto se debe al mecanismo de *first-match search*: cuando se detecta un patrón en ángulo 0° el algoritmo omite los 6 ángulos restantes para ese template, reduciendo el tiempo de 28 474 ms (sin matches) a 21 448 ms (con match en ángulo 0°). Las configuraciones de 2 y 3 templates, al no generar matches en los frames de prueba (excepto Frame 1), muestran tiempos virtualmente constantes.
- **Costo de rotaciones — marginal:** El costo total de las rotaciones computadas durante la fase de matching (suma de los tiempos `rotation_single` por frame) es de **0.87–1.42 ms** por frame, representando menos del **0.01 %** del tiempo total del ciclo. Esto confirma que el caché de templates rotados elimina efectivamente el cuello de botella geométrico. Cada rotación individual tarda **162 µs**, valor estable independientemente de la configuración.
- **Procesamiento exhaustivo:** En cada iteración, el algoritmo evalúa por cada template:
  - 7 ángulos de rotación: $-45°, -30°, -15°, 0°, 15°, 30°, 45°$
  - SAD (Sum of Absolute Differences) sobre píxeles negros y blancos por separado
  - Recorrido completo de la imagen 96×96 con el template rotado en cada posición válida
  - Terminación temprana si se encuentra por debajo del umbral de aceptación en el ángulo actual

Análisis de escalabilidad La Tabla 6.3 cuantifica el impacto incremental de agregar un template adicional al sistema, tomando como baseline la configuración de 2 templates.

Métrica	**Baseline (2 tmpl)**	**+1 tmpl (3)**	**+2 tmpl (4)**
Loop total (ms)	14 548	+7 080 (+48,7 %)	+8 907 (+61,2 %)
Template matching (ms)	14 297	+7 080 (+49,5 %)	+8 907 (+62,3 %)
Overhead comunicación (ms)	250	+0.4 (+0,2 %)	+0.3 (+0,1 %)
Costo rotaciones/frame (ms)	0.87	+0.47 (+54,0 %)	+0.56 (+64,1 %)
Cache init (ms)	369	+124 (+33,6 %)	+369 (+100 %)

Tabla 6.3: Impacto incremental de templates adicionales respecto al baseline de 2 templates

Algoritmo de reconocimiento basado en pares de puntos

El algoritmo de pares de puntos fue evaluado durante 118 ciclos completos de procesamiento. La Tabla 6.4 presenta las estadísticas descriptivas de cada fase del procesamiento.

Fase	**Media (ms)**	**Mediana (ms)**	**Desv. Est. (ms)**	**Mín (ms)**	**Máx (ms)**
Espera Algoritmo control de movimiento	**218.09**	260.12	76.91	0.12	280.12
Captura Frame	0.012	0.012	0.001	0.010	0.021
Umbralización	1.021	1.015	0.016	0.989	1.130
Procesamiento Pares	60.05	11.37	143.54	2.32	1390.18
Envío Detección	0.533	0.533	0.000	0.532	0.535
Total por Ciclo	**366.19**	389.44	143.59	87.47	1652.29

Tabla 6.4: Métricas de tiempo del algoritmo de pares de puntos (118 ciclos)

Del análisis de estos resultados se desprenden las siguientes observaciones:

- **Tiempo total promedio:** El ciclo completo de procesamiento tiene una duración media de **366.2 ms**, lo que equivale a una tasa de procesamiento de aproximadamente **2.73 FPS**.
- **Fase más costosa:** La espera al algoritmo de control de movimiento representa el **59.6 %** del tiempo total del ciclo, siendo la fase de mayor duración. Esta espera corresponde al tiempo en que el ESP32-CAM permanece bloqueado aguardando que la Arduino Nano le solicite un nuevo resultado de detección. Dado que el módulo de control de movimiento opera sobre una cadencia fija impuesta por su propio lazo de control PID y tiempos de actuación de motores, el ESP32-CAM queda inactivo hasta recibir dicha solicitud. En consecuencia, el cuello de botella del ciclo no es el procesamiento de visión, sino el ritmo al que el controlador de movimiento demanda nuevos datos.

  Si se considera únicamente el tiempo de procesamiento activo del ESP32-CAM (excluyendo la espera), las fases algorítmicas suman en promedio **61.6 ms**, de los cuales el **procesamiento de pares representa el 97.5 %**. Esta fase incluye:

  - **Búsqueda en anillos concéntricos:** tiempo mediano de 465 µs (primer punto) y 524 µs (segundo punto) por ciclo con detección.
  - **Flood-fill** para extracción de cada candidato: mediana de 5 µs (primer punto) y 4 µs (segundo punto).
  - **Cálculo de centroides:** aproximadamente 1 µs por candidato, operación de costo despreciable.
  - **Clasificación por distancia euclidiana:** media de 5.7 µs, ejecutada únicamente cuando se detecta un par válido.
  - **Cálculo de ángulo de rotación** mediante atan2: media de 18.8 µs.

La Tabla 6.5 detalla las estadísticas de cada sub-operación interna.

Sub-operación	**N**	**Media (µs)**	**Mediana (µs)**	**Máx (µs)**
Búsqueda 1er pto (anillos)	119	576	465	2271
Flood-fill 1er punto	480	76	5	7872
Centroide 1er punto	480	0.6	1	3
Búsqueda 2do pto (anillos)	118	2489	524	7128
Flood-fill 2do punto	465	35	4	936
Centroide 2do punto	465	0.6	1	1
Clasificación (dist. euclidiana)	87	5.7	5	50
Ángulo de rotación (atan2)	85	18.8	20	111

Tabla 6.5: Tiempos de sub-operaciones internas del procesamiento de pares (118 ciclos)

- **Comunicación UART:** El envío de la estructura de detección vía UART tiene un tiempo consistente de **0.533 ms**, confirmando estabilidad del protocolo de comunicación.

- **Umbralización binaria:** La conversión a imagen binaria mediante lookup table tiene un costo de **1.021 ms**, representando el 1.7 % del tiempo de procesamiento activo. Altamente estable (desviación estándar de 0.016 ms).

- **Captura de frame:** La captura desde la cámara OV2640 es prácticamente instantánea (**0.012 ms promedio**) gracias al acceso directo a memoria DMA.

La Figura 6.1 muestra la distribución porcentual del tiempo de ejecución entre las diferentes fases del procesamiento.

La Figura 6.2 presenta una visualización de los tiempos de cada fase a lo largo de los 118 ciclos de prueba, permitiendo observar la variabilidad temporal del procesamiento.

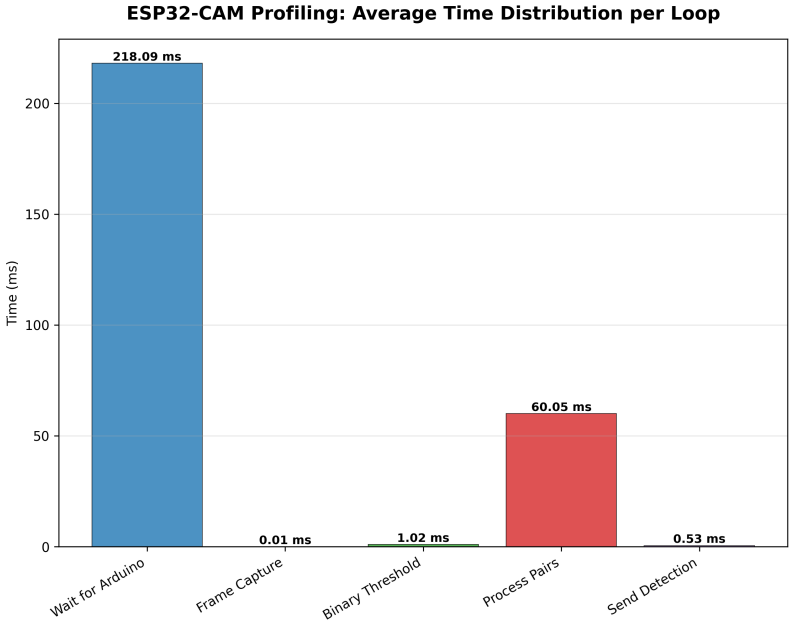


Figura 6.1: Distribución de tiempo de ejecución por fase

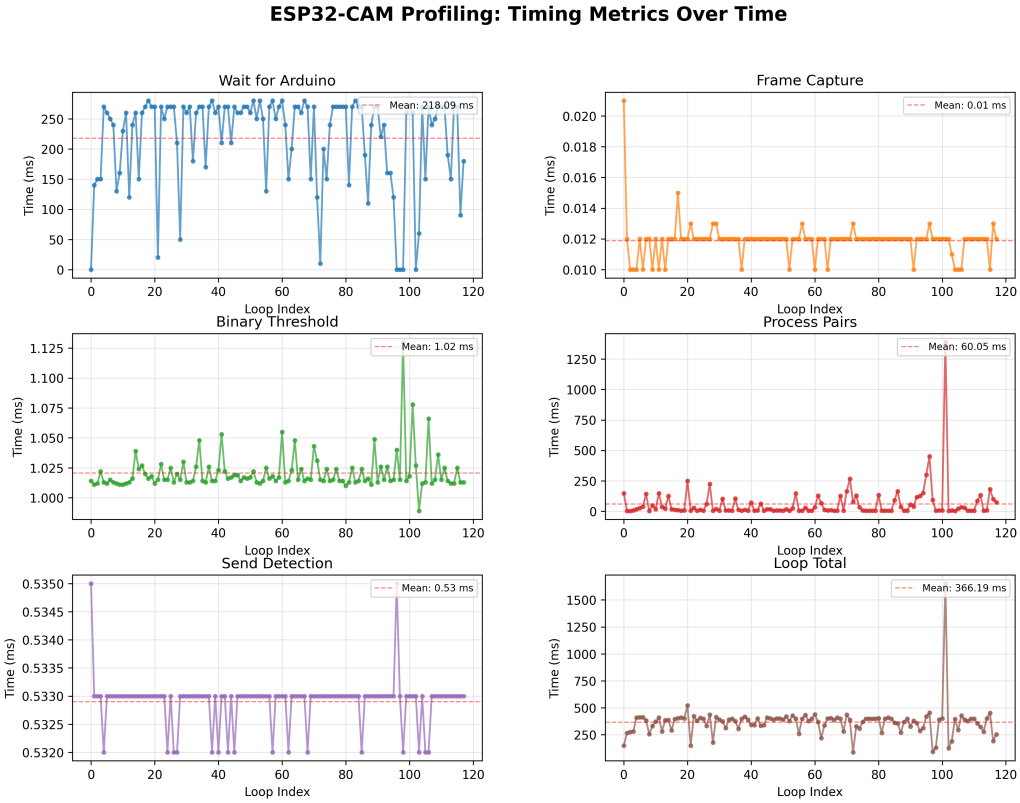


Figura 6.2: Tiempos de ejecución por ciclo del algoritmo de pares de puntos

Comparación entre algoritmos

La Tabla 6.6 presenta una comparación directa entre ambos algoritmos en términos de velocidad de procesamiento total por ciclo.

Algoritmo	**Tiempo/Ciclo (s)**	**FPS**	**Speedup**
Template Matching (2 templates)	14.297	0.07	1.0× (baseline)
Pares de Puntos	0.366	2.73	**39.1×**

Tabla 6.6: Comparación de rendimiento entre algoritmos

El algoritmo de pares de puntos demuestra una **mejora de rendimiento de 39.1 veces** respecto al template matching, reduciendo el tiempo de ciclo de 14.3 segundos a 366 ms. Esta mejora sustancial se atribuye a:

1. **Búsqueda dirigida vs. exhaustiva:** El algoritmo de pares busca específicamente dos puntos desde el centro de la imagen en anillos concéntricos, evitando recorrer un píxel múltiples veces, mientras que template matching recorre exhaustivamente toda la imagen múltiples veces.

2. **Complejidad algorítmica:** Template matching tiene complejidad $O(W \times H \times N_t \times N_r \times T_w \times T_h)$ donde W, H son dimensiones de la imagen, N_t número de templates, N_r número de rotaciones, y T_w, T_h dimensiones del template. Para 96×96, 4 templates, 7 rotaciones, y templates 28×28: ∼49 millones de comparaciones SAD por ciclo.

   El algoritmo de pares tiene complejidad $O(R_{max}^2)$ para búsqueda en anillos concéntricos más $O(A)$ para flood-fill de cada punto, donde R_{max} es el radio máximo de búsqueda y A es el área del punto. En el peor caso: ∼18,000 píxeles examinados.

3. **Operaciones por píxel:** Template matching realiza múltiples restas y sumas acumuladas por píxel analizado, mientras que pares de puntos realiza principalmente comparaciones binarias (negro/blanco) y operaciones de marcado de visitados.

4. **Terminación temprana:** El algoritmo de pares de puntos detiene la búsqueda en cuanto localiza un par válido, sin necesidad de procesar el resto de la imagen. En la mayoría de los ciclos el par se encuentra rápidamente cerca del centro.

6.3. Problemas encontrados

Durante el desarrollo e implementación del sistema se identificaron diversos problemas técnicos que afectaron el desempeño y la confiabilidad del prototipo. Estos problemas se pueden clasificar en cuatro categorías principales: algorítmicos y de procesamiento, de hardware y eléctricos, de comunicación entre módulos, y de construcción mecánica.

6.3.1. Problemas algorítmicos y de procesamiento

Sensibilidad a las condiciones de iluminación

Uno de los desafíos más significativos enfrentados fue la sensibilidad de ambos algoritmos a las variaciones en las condiciones de iluminación. Tanto el algoritmo reconocimiento de patrones basado en template matching como el de detección de pares de puntos mostraron degradación en su desempeño ante cambios en la intensidad o uniformidad de la luz ambiente. Este problema se manifestó especialmente en:

- Presencia de sombras que alteran la umbralización binaria en los algoritmos de reconocimiento de patrones.
- Variaciones en el contraste que afectan la métrica SAD del algoritmo de reconocimiento de patrones basado en template matching.
- Reflejos sobre las superficies de las tarjetas que generan regiones de alta intensidad no deseadas.

Para mitigar este problema fue necesario implementar calibración manual de umbrales y ajuste del brillo del LED integrado en la ESP32-CAM, aunque estas soluciones no eliminaron completamente la dependencia de condiciones controladas de iluminación, por lo que fue necesario también una umbralización dinámica basada en distancias para eliminar sombras fuertes en los bordes del frame capturado.

Alto costo computacional

Los algoritmos de reconocimientos de patrones implementados, especialmente el de *template matching* con detección de rotación, presentaron un alto costo computacional. El barrido exhaustivo de la imagen combinado con la evaluación de múltiples ángulos de rotación resultó en tiempos de procesamiento que limitaron la frecuencia de actualización del sistema de control. Aunque se implementaron optimizaciones como el uso de formato de píxeles en escala de grises, caché de *templates* rotados y reducción del conjunto de ángulos evaluados, el procesamiento de cada *frame* aún requirió recursos computacionales considerables en el microcontrolador ESP32-CAM debido a las limitaciones de hardware de la misma.

En la segunda etapa, para la implementación del algoritmo de pares de puntos se logró una reducción significativa en los tiempos de procesamiento; sin embargo, el costo computacional resultante desde nuestro punto de vista continúa siendo elevado para los requerimientos de sistemas en tiempo real.

Comportamientos automáticos de la cámara

La cámara OV2640 integrada en la ESP32-CAM incluye funcionalidades automáticas de ajuste de ganancia (*auto gain control*, AGC) y exposición que, aunque son útiles en aplicaciones generales, introdujeron variabilidad no deseada en las imágenes capturadas. Estos ajustes automáticos causaron inconsistencias en la intensidad de los píxeles entre *frames* consecutivos, afectando la detección de patrones. Fue necesario desactivar estas funcionalidades y configurar manualmente los parámetros de la cámara para lograr un comportamiento predecible y consistente.

Alto consumo de memoria en el algoritmo de reconocimiento de patrones

En las etapas iniciales del desarrollo del algoritmo de reconocimiento de patrones, en particular del método basado en *template matching*, fue necesaria una depuración extensa del código. Durante este proceso se incorporaron numerosas instrucciones de depuración, lo que incrementó significativamente el consumo de memoria. Como consecuencia, el algoritmo excedió los recursos disponibles del módulo ESP32-CAM, provocando reinicios inesperados del sistema y la aparición del error conocido como *Guru Meditation Error*.

6.3.2. Problemas de hardware y eléctricos

Problemas de cableado

Durante las pruebas de navegación se identificaron problemas de confiabilidad en las conexiones eléctricas entre los componentes del robot. El movimiento del prototipo y las vibraciones generadas por los motores ocasionaron que algunas conexiones se aflojaran o desconectaran, causando interrupciones intermitentes en su funcionamiento. Este problema fue particularmente crítico en el bus de comunicación serial utilizado para el pasaje de información entre la ESP32-CAM y la Arduino Nano.

Brown-out detector

Se registraron reinicios inesperados de la ESP32-CAM debido a la activación del circuito de protección *brown-out detector* (BOD). Este problema ocurrió cuando el consumo de corriente de los motores causaba caídas momentáneas en el voltaje de alimentación, llevando el nivel por debajo del umbral de operación segura del microcontrolador. La activación del BOD provocaba el reinicio del ESP32-CAM, interrumpiendo el proceso de captura y análisis de imágenes.

Conflicto de pines compartidos

La arquitectura del módulo ESP32-CAM presenta limitaciones en la disponibilidad de pines GPIO. En particular, el pin GPIO4 es compartido entre la conectividad WiFi y el acceso a la tarjeta SD integrada. Inicialmente se contempló el uso de la tarjeta SD para el almacenamiento de datos con fines de depuración; sin embargo, esta restricción impide el uso simultáneo de ambas funcionalidades, obligando a priorizar entre almacenamiento local y conectividad inalámbrica.

Para el funcionamiento del sistema se optó por la comunicación serial en lugar de WiFi, lo que limitó las posibilidades de configuración remota y depuración durante el desarrollo. Finalmente, no se utilizó ni la conexión WiFi ni el almacenamiento en tarjeta SD, ya que los depuradores USB resultaron ser la opción más adecuada para la etapa de pruebas.

6.3.3. Problemas de comunicación entre módulos

Incompatibilidad de tipos de datos entre arquitecturas

La diferencia en las arquitecturas de los microcontroladores utilizados (ESP32 de 32 bits y ATmega328P de 8 bits) es una posible causa de problemas de compatibilidad en la comunicación serial. Específicamente, los tipos de datos primitivos como `int` y `short` tienen diferentes tamaños en memoria según la arquitectura, lo que puede resultar en errores de deserialización de las estructuras de datos intercambiadas entre los distintos microcontroladores. Este posible problema fue mitigado en las etapas iniciales de diseño de la solución al utilizar tipos de datos de mismo tamaño como lo son el `uint8` o `float`.

6.3.4. Complejidad en pruebas y depuración

La naturaleza embebida del sistema presentó desafíos significativos en el proceso de validación:

- **Limitaciones de depuración en tiempo real**: La ausencia de herramientas de depuración interactivas en microcontroladores requirió el uso exclusivo de logs seriales para diagnóstico, limitando la capacidad de inspección de estado del sistema.
- **Variabilidad de escenarios**: La validación de algoritmos de detección de imágenes requirió pruebas exhaustivas bajo múltiples condiciones de iluminación, distancia, ángulos de cámara y tipos de superficies, multiplicando la cantidad de casos de prueba necesarios.
- **Dificultad en la reproducción de errores**: Algunos comportamientos erróneos del sistema fueron difíciles de reproducir consistentemente debido a la variabilidad ambiental y la naturaleza no determinista de ciertos escenarios de captura de imagen.
- **Dificultad en la generación de patrones**: La falta de sitios específicos para diseñar tarjetas de programación con distintos patrones fue un problema que nos llevo a tener que crear aplicaciones que nos permitieran avanzar en este frente. (ver sección 5.5 para más información).

6.3.5. Problemas mecánicos y de construcción

Deslizamiento sobre las tarjetas

Durante las pruebas de navegación se observó que las ruedas del robot presentaban deslizamiento al desplazarse sobre las tarjetas impresas. Este problema se atribuyó a las propiedades de fricción del material de las tarjetas (papel común A4), que resultó insuficiente para proporcionar la tracción necesaria. El deslizamiento introducía un error acumulativo en la odometría implícita del sistema y dificultaba la ejecución precisa de maniobras de rotación y seguimiento de trayectorias.

Desafíos en el ensamblaje del prototipo

El proceso de ensamblaje del prototipo presentó diversos desafíos relacionados con el montaje de componentes, la fijación de sensores y la organización del cableado en un espacio reducido. La necesidad de iterar sobre el diseño mecánico para acomodar los módulos electrónicos, asegurar la estabilidad de la cámara y permitir el acceso a los puertos de programación extendió el tiempo de desarrollo y requirió múltiples ajustes en la estructura física del robot.

6.3.6. Problemas de calibración y ajuste

Sensibilidad del controlador PID

El ajuste de los parámetros del controlador PID utilizado para el control de motores resultó ser un proceso altamente sensible. Pequeñas variaciones en las constantes proporcional, integral y derivativa generaban comportamientos significativamente diferentes en la respuesta del sistema, desde oscilaciones hasta inestabilidad completa. La determinación de valores óptimos requirió un proceso extenso de ensayo y error.

Capítulo 7

Conclusiones y trabajo futuro

El presente capítulo sintetiza los principales resultados alcanzados a lo largo del proyecto, destacando los aportes técnicos, metodológicos y conceptuales derivados del diseño e implementación de la plataforma robótica desarrollada. En primer lugar, se exponen las conclusiones más relevantes, organizadas en términos de logros obtenidos, lecciones aprendidas y reflexiones finales sobre el proceso de trabajo.

Posteriormente, se presentan las líneas de trabajo futuro que emergen a partir de las limitaciones identificadas y de las oportunidades de mejora detectadas durante la experimentación. Estas propuestas abarcan tanto posibles optimizaciones de hardware como extensiones algorítmicas y funcionales del sistema, con el objetivo de consolidar y ampliar el alcance de la solución desarrollada.

7.1. Conclusiones

El presente proyecto de grado ha logrado diseñar, implementar y validar una plataforma robótica con capacidad de reconocimiento de patrones visuales y navegación autónoma con corrección de trayectoria basada en microcontroladores de bajo costo y de código abierto.

7.1.1. Logros principales

El trabajo realizado puede sintetizarse en los siguientes logros técnicos fundamentales:

Arquitectura de sistema embebido distribuido: Se diseñó e implementó exitosamente una arquitectura distribuida compuesta por dos unidades de procesamiento especializadas (ESP32-CAM para visión y Arduino Nano para control de motores) que se comunican mediante protocolo UART. Esta arquitectura permite la especialización de tareas y facilita el desarrollo modular del sistema.

Solución documentada y de código abierto: A diferencia de las soluciones comerciales propietarias analizadas en el estado del arte, el presente proyecto proporciona una alternativa documentada, con código fuente accesible y esquemas de hardware replicables. Esta decisión de diseño facilita la auditoría, adaptación y extensión del sistema por parte de la comunidad educativa y de investigación.

Sistema de navegación autónoma: Se implementó un sistema de navegación basado en visión por computadora capaz de reconocer patrones impresos en tarjetas físicas, extraer la información relevante asociada a dichos patrones y utilizarla para determinar las acciones de movimiento del

robot. A partir de esta información, el sistema es capaz de ejecutar desplazamientos y realizar correcciones de trayectoria de manera autónoma.

7.1.2. Lecciones aprendidas

El desarrollo del proyecto permitió extraer varias lecciones significativas sobre el diseño de sistemas embebidos de visión por computadora:

La optimización prematura de algoritmos existentes puede ser menos eficiente que el diseño de algoritmos específicos para el problema: La segunda etapa del proyecto invirtió considerable esfuerzo en optimizar template matching mediante técnicas como el guardado en memoria de templates rotados, búsqueda con terminación temprana y modos de optimización configurables. Sin embargo, el cambio de paradigma hacia el algoritmo de pares de puntos en la tercera etapa demostró que un diseño algorítmico fundamentalmente diferente, adaptado específicamente a las restricciones del hardware y los requisitos del problema, puede superar en eficiencia a las optimizaciones incrementales de enfoques clásicos.

La caracterización cuantitativa del desempeño es esencial: El desarrollo de herramientas de profiling y visualización de métricas desde las primeras etapas permitió tomar decisiones de diseño informadas por datos objetivos en lugar de intuiciones. La identificación temprana de cuellos de botella (como el 69 % del tiempo de ciclo dedicado al procesamiento de pares de puntos) guió las decisiones de optimización hacia las áreas con mayor impacto potencial.

7.1.3. Reflexiones finales

El presente proyecto demuestra que es posible desarrollar plataformas robóticas con capacidades de reconocimiento de patrones y navegación autónoma con corrección de trayectoria, utilizando microcontroladores y herramientas de código abierto. La metodología de diseño iterativo, guiada por experimentación sistemática y métricas cuantitativas, resultó fundamental para navegar el espacio de soluciones y converger hacia implementaciones viables dentro de las restricciones del hardware seleccionado.

La arquitectura distribuida propuesta, junto con el protocolo de comunicación diseñado y el conjunto de herramientas de desarrollo auxiliares, constituyen una base para futuros desarrollos en el ámbito de sistemas de navegación robótica autónoma basados en la detección de patrones.

7.2. Trabajo futuro

Los resultados obtenidos en este proyecto abren múltiples líneas de investigación y desarrollo que podrían extender significativamente las capacidades del robot construido.

7.2.1. Mejoras de hardware

Exploración de microcontroladores de mayor capacidad computacional: El proyecto identificó las limitaciones computacionales del ESP32-CAM como el principal cuello de botella para algoritmos de visión por computadora. Microcontroladores de próxima generación como el ESP32-S3 o el Raspberry Pi Pico 2 (con doble núcleo ARM Cortex-M33 a 150 MHz) podrían permitir la implementación de algoritmos más costosos computacionalmente manteniendo el perfil de bajo costo del proyecto.

Integración de sensor OID: Durante la investigación del estado del arte se identificó el sensor OID como una tecnología propietaria prometedora para el reconocimiento de patrones de alta velocidad. Obtener acceso a este sensor y realizar un análisis comparativo de desempeño frente al enfoque basado en cámara implementado en este proyecto constituiría una línea de investigación valiosa. Esta comparación permitiría evaluar cuantitativamente el trade-off entre soluciones propietarias especializadas y enfoques abiertos basados en visión por computadora.

Diseño de PCB integrado: El prototipo actual utiliza conexiones mediante cables Dupont y protoboards, lo que introduce fragilidad mecánica y la posibilidad de desconexiones accidentales. El diseño de una PCB personalizada que integre el ESP32-CAM, el Arduino Nano (o su sucesor), el driver de motores y el sistema de alimentación en un único módulo compacto mejoraría significativamente la robustez del robot y facilitaría su replicación.

Diseño de carcasa rígida mediante impresión 3D: El desarrollo de una carcasa estructural impresa en 3D permitiría integrar todos los componentes electrónicos (ESP32-CAM, Arduino Nano, driver de motores, sistema de alimentación) junto con el sistema mecánico (ruedas, motores, ejes) en un chasis compacto y robusto. Una estructura de plástico rígido (PLA o PETG) proporcionaría múltiples ventajas frente al montaje actual basado en protoboard y carton: mayor estabilidad mecánica ante vibraciones, protección de componentes electrónicos contra daños físicos y optimización del espacio interno mediante diseño específico para cada componente.

7.2.2. Mejoras algorítmicas

Implementación de aprendizaje automático: La experimentación con frameworks de machine learning optimizados para sistemas embebidos (TensorFlow Lite for Microcontrollers, Edge Impulse, TinyML) podría permitir reconocimiento de patrones más flexible ante variaciones de iluminación, perspectiva y oclusiones parciales.

Sensores adicionales: La integración de sensores adicionales con la información visual podría mejorar significativamente la precisión de navegación mediante técnicas de fusión sensorial. En particular, se identifican tres categorías de sensores con alto potencial de contribución:

- *Encoders en las ruedas*: La incorporación de encoders incrementales en los ejes de los motores permitiría obtener mediciones de odometría en tiempo real, estimando el desplazamiento y la orientación del robot a partir del conteo de revoluciones de cada rueda. Además, esto también permite mejorar la rectitud del avance en línea recta sin depender exclusivamente de la retroalimentación visual.

- *Giroscopio*: Un giroscopio proporcionaría la velocidad angular instantánea del robot, permitiendo estimar su orientación con alta frecuencia de muestreo e independientemente de las condiciones de iluminación que afectan al sistema de visión. Con este sensor sería posible detectar desvíos angulares en tiempo real y aplicar correcciones de trayectoria antes incluso de llegar a leer una nueva tarjeta, reduciendo el error acumulado entre lecturas sucesivas.

- *Fusión sensorial*: La combinación de visión, odometría y mediciones inerciales mediante algoritmos de fusión como por ejemplo el filtro de Kalman [27] permitiría obtener estimaciones de posición y orientación más robustas que cualquiera de las fuentes por separado. El filtro de Kalman, ampliamente utilizado en robótica móvil, permite combinar mediciones ruidosas de distintos sensores ponderando su confiabilidad relativa.

7.2.3. Construcción de prototipo avanzado

El prototipo desarrollado en el presente trabajo está orientado a la validación de los distintos algoritmos implementados, particularmente aquellos relacionados con la detección de patrones, la identificación de tarjetas y el control de movimiento.

La capacidad del sistema para detectar diferentes tarjetas introduce implícitamente un grado de flexibilidad en el diseño de la plataforma, lo que habilita diversas posibilidades de evolución en futuras iteraciones. En este sentido, una línea natural de desarrollo consiste en la construcción de un prototipo más completo, incorporando hardware adicional que permita ejecutar nuevas acciones, tales como la activación de luces, la emisión de sonidos u otras formas de interacción. Para ello, además de la integración del hardware correspondiente, sería necesario implementar los módulos de control encargados de interpretar las detecciones realizadas por el sistema y accionar dichos dispositivos de manera adecuada.

Asimismo, podría resultar beneficioso diseñar y fabricar una estructura física específica mediante impresión 3D, optimizada para un tamaño reducido. Este enfoque permitiría mejorar la maniobrabilidad del robot y facilitar su adaptación a distintos escenarios de uso.

Bibliografía

[1] SONiX Technology Co., Ltd. Sonix oid series optical sensor. https://www.sonix.com.tw/article-en-1406-14748, 2024.

[2] SONiX Technology Co., Ltd. Snm9s50xc3000a / snm9s50xbc3000a oid module. https://www.sonix.com.tw/article-en-4272-42949, 2024.

[3] SONiX Technology Co., Ltd. Snm9s5xxxc3000a series oid soc module with decoder datasheet. https://www.sonix.com.tw/files/1/FD5AD43B8FBD644BE050007F01007509, 2022. Data Sheet (May 12, 2022).

[4] SONiX Technology Co., Ltd. Snm9s5xxx series technical notices. https://www.sonix.com.tw/files/1/FD5AB86595F64261E050007F01006DC9, 2022.

[5] Wikipedia. Template matching. https://en.wikipedia.org/wiki/Template_matching, 2025. [Online; accessed 10-Feb-2026]
textitWikipedia, The Free Encyclopedia.

[6] H. Y. Kim and S. A. Araújo. Grayscale template-matching invariant to rotation, scale, translation, brightness and contrast. In *Proceedings of the IEEE Pacific-Rim Symposium on Image and Video Technology (PSIVT)*, volume 4872 of *Lecture Notes in Computer Science*, pages 100–113, 2007.

[7] TensorFlow. Tensorflow lite guide. https://www.tensorflow.org/lite/guide?hl=es-419, 2024.

[8] DataCamp. What is tinyml? tiny machine learning. https://www.datacamp.com/blog/what-is-tinyml-tiny-machine-learning, 2023.

[9] Edge Impulse Inc. Edge impulse. https://www.edgeimpulse.com/, 2024.

[10] Robobloq Co., Ltd. Qobo user manual (chinese). https://static-robobloq.oss-cn-shenzhen.aliyuncs.com/wiki/qobo/qobo-user-manual-cn.pdf, 2024.

[11] Matatalab. Tale-bot pro. https://matatalab.com/en/tale-botpro, 2026. [Online; accessed 25-Feb-2026].

[12] Matatalab. Lesson 4.2: Tale-bot pro robot. https://matatalab.com/en/lesson4.2, 2026. [Online; accessed 25-Feb-2026].

[13] Sphero, Inc. Sphero indi – screenless coding & learning robot for kids. https://sphero.com/pages/sphero-indi, 2026. Información oficial sobre el robot educativo Sphero Indi.

[14] YouTube. Sphero indi (short video). https://www.youtube.com/shorts/mtjQL_EKGWY, 2026. Video corto en YouTube relacionado con Sphero Indi.

[15] YouTube. Beebot (short video). https://www.youtube.com/shorts/z5y56EXsiG0, 2025. Short de YouTube relacionado con BeeBot.

[16] TIBOT - Habilitas Educación, SL. ¿qué es bee bot robot? https://www.tibot.es/blog/actividades-bee-bot/que-es-bee-bot/srsltid=AfmBOoozXUFq6yWSX9IVoZXAzComMooLP-4IKJYujhSplll_Rez4sWnd, 2024. Blog principal sobre Bee Bot Robot.

[17] Primo Toys. Cubetto educational coding toy. https://primotoys.com/, 2024.

[18] Matlo. Cubetto documentation. https://matlo.me/cubetto, 2024.

[19] Robotopia. Cubetto - kit educativo. https://robotopia.es/kits-educativos/124-cubetto.html, 2024.

[20] Primo Toys. Cubetto educational coding toy (video). https://www.youtube.com/watch?v=zcNicSJaHUs, 2017.

[21] Primo.io. Cubetto prototype documentation. https://github.com/primo-io/prototype-documentation, 2024.

[22] Microsoft. Visual studio code. https://code.visualstudio.com/, 2026. Accessed: February 19, 2026.

[23] PlatformIO. Platformio: Professional collaborative platform for embedded development. https://platformio.org/, 2026. Accessed: February 19, 2026.

[24] Wikipedia. Thresholding (image processing). https://en.wikipedia.org/wiki/Thresholding_(image_processing), 2025. Última edición consultada: marzo 2026.

[25] Facultad de Ingeniería, Universidad de la República. Nanoplataformarobotica. https://gitlab.fing.edu.uy/nanoPlataformaRobotica/nanoplataformarobotica, 2026. Repositorio GitLab.

[26] Wikipedia. Proportional–integral–derivative controller. https://en.wikipedia.org/wiki/Proportional-integral-derivative_controller#Manual_tuning, 2024. Seccion: Manual tuning. Accedido: febrero 2026.

[27] Wikipedia. Filtro de kalman. https://es.wikipedia.org/wiki/Filtro_de_Kalman, 2025. Última edición consultada: marzo 2026.

Anexo A

Diagramas de construcción adicionales

A continuación, tenemos el diagrama de conexión, esquema y el diagrama de PCB utilizados en este proyecto.

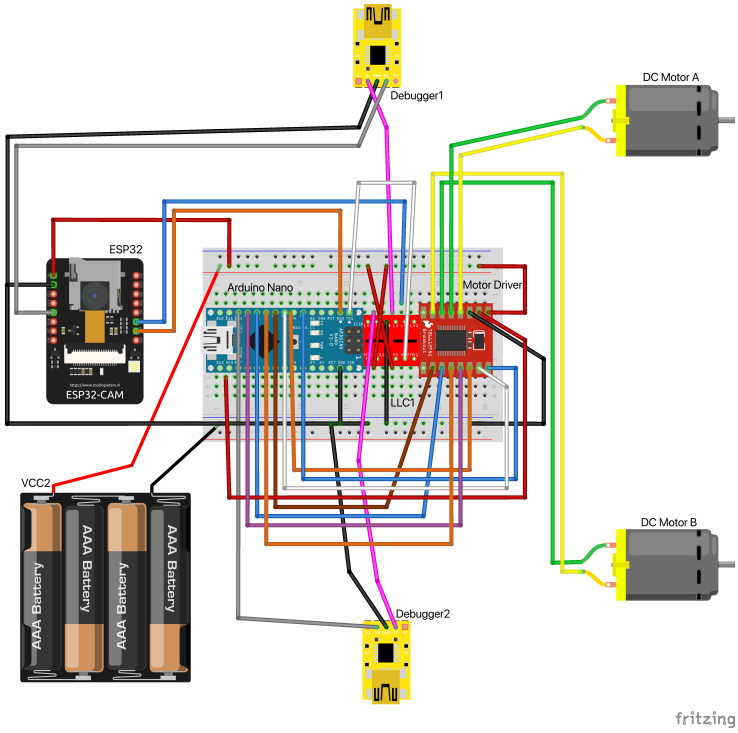


Figura A.1: Diagrama de conexión de componentes

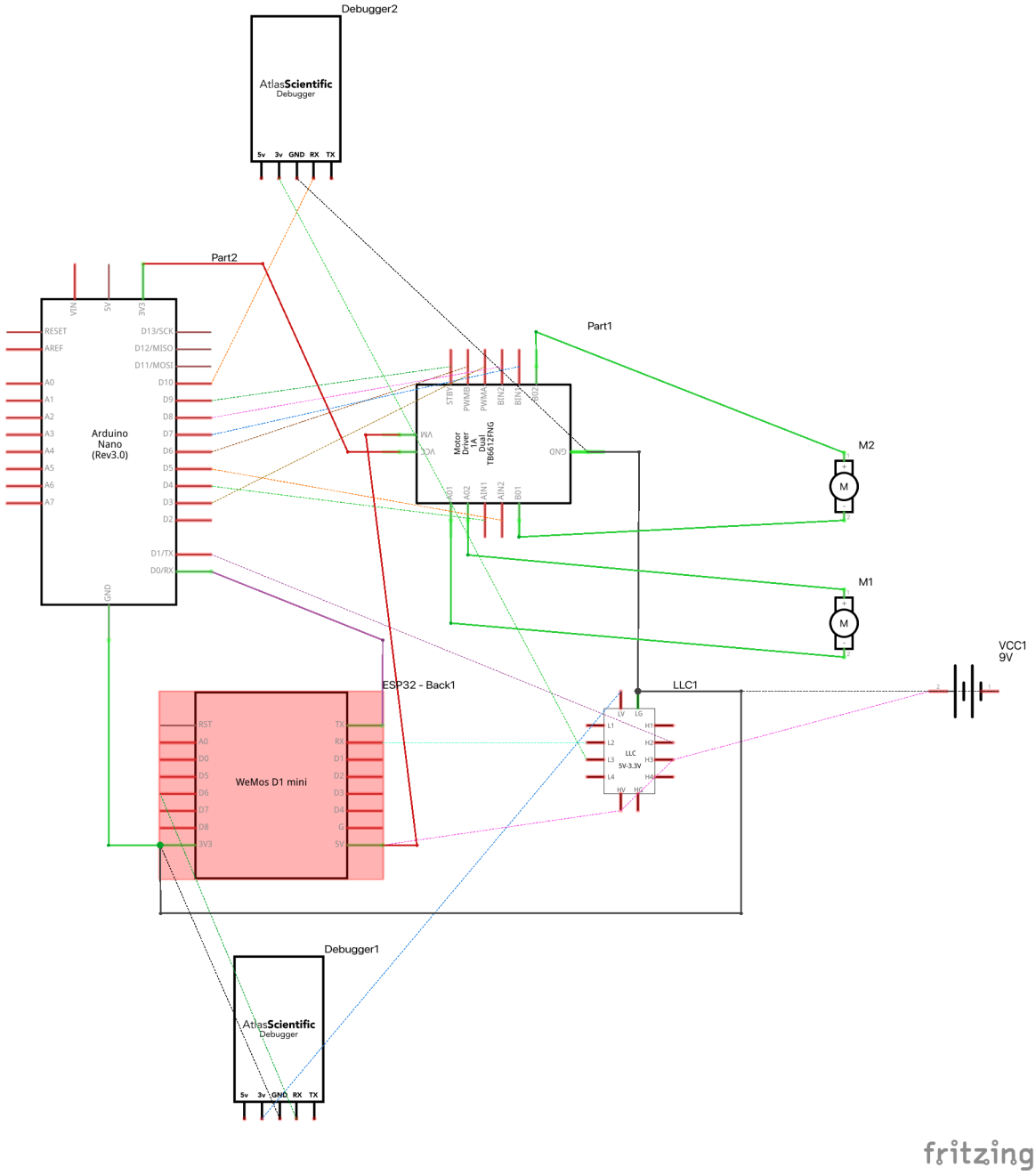


Figura A.2: Diagrama de esquema

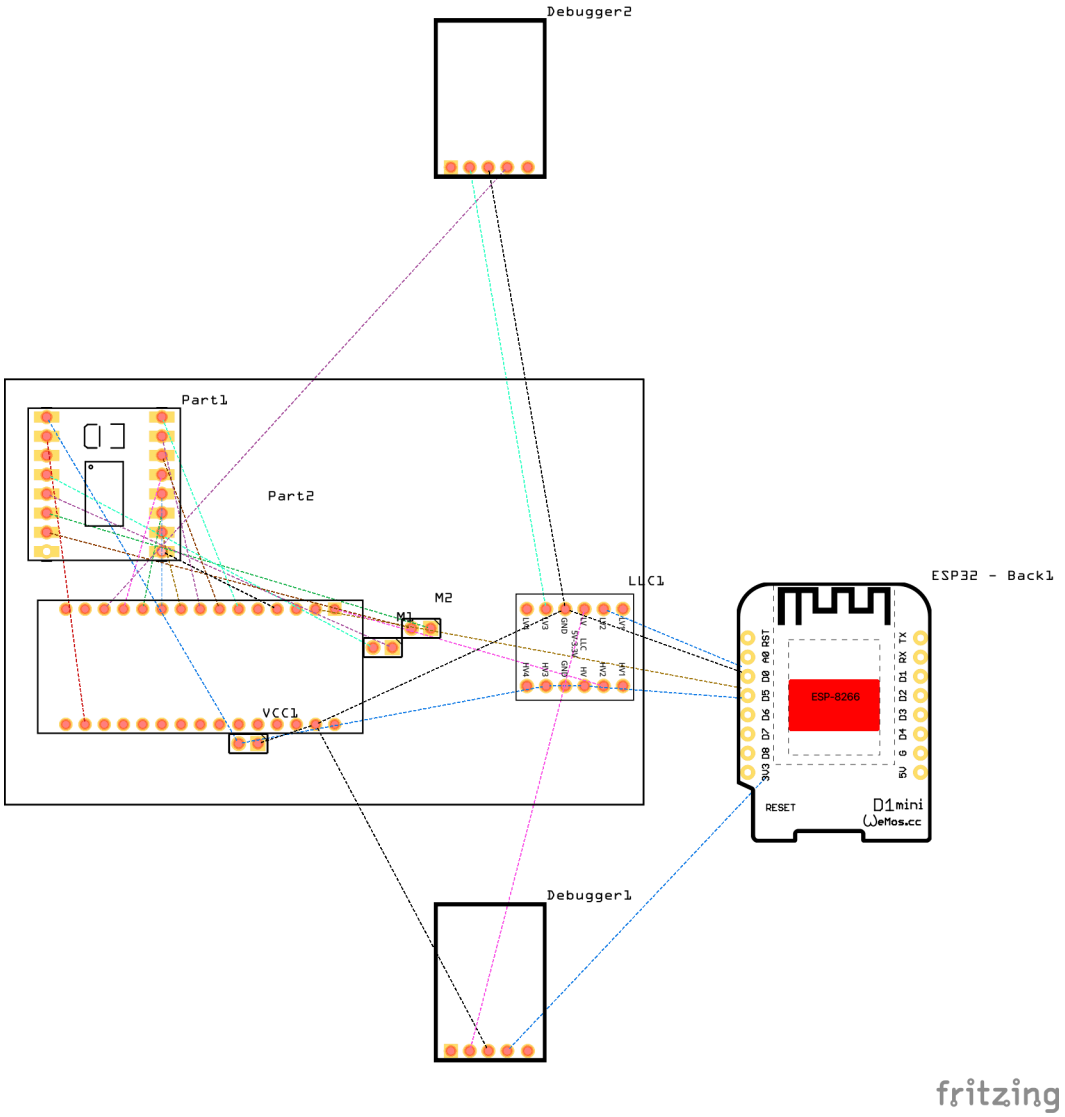


Figura A.3: Diagrama de PCB

Anexo B

Aplicaciones web

A la hora de realizar las diferentes pruebas de concepto tuvimos que generar patrones para los distintos algoritmos. Estos portales surgen por la necesidad de simplificar y acelerar la generación de distintos patrones en base a las necesidades de los distintos algoritmos. A continuación se describen brevemente estos portales y además las principales funcionalidades de los mismos.

B.1. Sitio generador de tarjetas para template matching

Para la generación de páginas con patrones a la hora de utilizar el algoritmo de template matching creamos este sitio web donde permite a los usuarios agregar distintos símbolos configurados en forma de grilla. Luego permite imprime

Las principales funcionalidades son las siguientes:

- Carga de templates desde configuración.
- Selección a nivel de grilla para generación de patrón.
- Generación de pdf para imprimir patrón.
- Configuración de hoja a nivel de escala, margenes y espacio entre templates.
- Acciones rápidas (Quick actions).

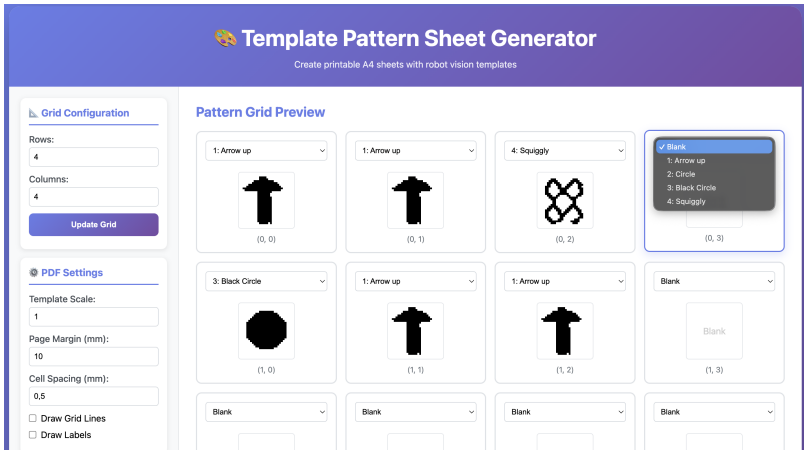


Figura B.1: Portal de generación de patrones con templates

B.2. Sitio generador de tarjetas para pares de puntos

Para la generación de páginas imprimibles a la hora de utilizar el algoritmo de pares de puntos creamos este sitio web donde permite a los usuarios agregar pares de puntos de forma sencilla y amigable.

Las principales funcionalidades son las siguientes:

- Agregado de pares de puntos con tamaños independientes.
- Funcionalidad de girar puntos a cualquier grado.
- Generación de página para imprimir patrón.
- Acciones de copiar y pegar puntos.
- Selección de múltiples pares de puntos.
- Atajos de teclado para un manejo fácil.

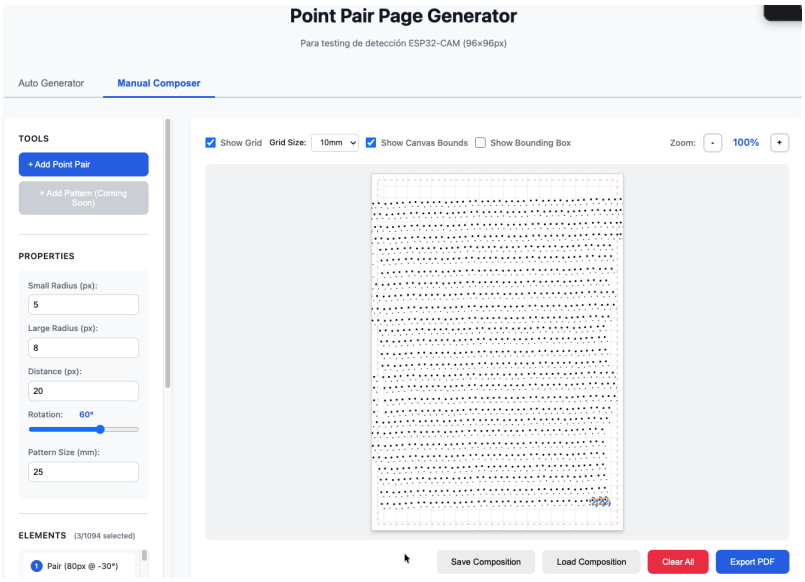


Figura B.2: Portal de generación de patrones con pares de puntos

Anexo C

Entorno de desarrollo

El presente anexo describe en detalle el entorno de desarrollo utilizado para la implementación del robot móvil con reconocimiento de patrones, incluyendo los lenguajes de programación, herramientas de desarrollo, configuración del espacio de trabajo y las bibliotecas de software empleadas en cada componente del sistema.

C.1. Lenguajes de programación

El proyecto emplea múltiples lenguajes de programación, cada uno seleccionado por su idoneidad para las diferentes capas y componentes del sistema:

C.1.1. C/C++ (Arduino Framework)

El código embebido para ambos microcontroladores (ESP32-CAM y Arduino Nano) se desarrolló utilizando C++ con el framework Arduino. Esta elección se fundamenta en varios factores:

- **Eficiencia computacional:** C/C++ proporciona control directo sobre recursos de hardware y gestión de memoria, crucial para sistemas embebidos con recursos limitados donde cada milisegundo y kilobyte son valiosos.
- **Framework Arduino:** Ofrece una capa de abstracción que simplifica el acceso a periféricos (GPIO, UART, I2C, SPI) sin sacrificar performance, con amplia documentación y comunidad de soporte.
- **Compatibilidad multiplataforma:** El mismo código base puede ejecutarse en ESP32 y AVR (Arduino Nano) con mínimas modificaciones, facilitando la portabilidad del sistema.
- **Bibliotecas especializadas:** Acceso a bibliotecas optimizadas para procesamiento de imágenes, control PID y comunicación serial.

El estándar C++11 se utiliza en ambos módulos, proporcionando características modernas como lambdas, auto type deduction, y range-based for loops, mejorando la legibilidad y mantenibilidad del código sin incrementar significativamente el overhead.

C.1.2. Python

Python se utilizó para scripts auxiliares de desarrollo, análisis y generación de contenido. Su presencia en el proyecto responde a:

- **Prototipado rápido:** Verificación de algoritmos y generación de datos de prueba antes de implementar en C++.
- **Análisis de resultados:** Scripts para procesar logs, calcular estadísticas y generar visualizaciones de rendimiento.
- **Herramientas de desarrollo:** Automatización de tareas como conversión de formatos y preprocesamiento de templates.

C.2. IDEs y herramientas

C.2.1. Visual Studio Code

Visual Studio Code (VS Code) se utilizó como entorno de desarrollo integrado principal para todo el proyecto. VS Code ofrece:

- **Soporte multiplataforma:** Funciona de manera consistente en Windows, macOS y Linux, facilitando colaboración entre desarrolladores con diferentes sistemas operativos.
- **Sistema de extensiones:** Arquitectura modular que permite agregar funcionalidad específica según las necesidades del proyecto.
- **Workspace multi-root:** Capacidad de gestionar múltiples proyectos relacionados en una única ventana, ideal para arquitecturas modulares como la del presente proyecto.
- **IntelliSense avanzado:** Autocompletado inteligente, navegación de código y refactorización para C++, TypeScript y JavaScript.
- **Depuración integrado:** Depuración visual con breakpoints, watch variables y call stack para código embebido y web.

C.2.2. PlatformIO

PlatformIO es una plataforma de desarrollo profesional para sistemas embebidos que reemplaza el Arduino IDE tradicional. Sus ventajas incluyen:

- **Gestión de dependencias:** Sistema declarativo de bibliotecas mediante archivo platformio.ini, garantizando versiones consistentes entre desarrolladores.
- **Compilación optimizada:** Toolchains profesionales con optimizaciones de compilador (O2, O3) y flags específicos de arquitectura.
- **Testing framework:** Soporte integrado para unit testing en dispositivos embebidos.
- **Monitor serial avanzado:** Captura automática de puerto y baudrate, con filtros y formateo de salida.

- **Soporte multi-board:** Configuraciones independientes para ESP32-CAM y Arduino Nano en el mismo proyecto.

La integración con VS Code se realiza mediante la extensión PlatformIO IDE, proporcionando acceso a todas las funcionalidades desde el editor.

C.3. Configuración del workspace

El proyecto utiliza una configuración de workspace multi-root de VS Code que organiza todos los componentes del sistema en una única vista unificada. El archivo `Algorithm3_TemplateMatching.code-workspace` define la estructura del proyecto y configuraciones compartidas.

C.3.1. Estructura de carpetas

El workspace está organizado en seis carpetas principales:

1. **NanoAlgoritmoTemplate (ESP32-CAM):** Código del módulo de visión por computadora, incluyendo captura de imágenes, template matching con detección de rotación y comunicación serial.

2. **NanoCerebroTemplate (Arduino Nano):** Código del módulo de control de motores, implementando control PID, máquina de estados y comunicación serial con el módulo de visión.

3. **Shared Libraries:** Biblioteca compartida conteniendo estructuras de datos comunes, protocolos de comunicación y configuraciones compartidas entre ambos microcontroladores.

4. **Template Portal:** Aplicación web Angular para diseño y generación de tarjetas de programación, con frontend responsivo y backend Node.js.

5. **Documentation:** Documentación técnica del proyecto, incluyendo diagramas de arquitectura, guías de usuario y notas de desarrollo.

C.3.2. Configuraciones de IntelliSense

El workspace configura el motor IntelliSense de C++ para proporcionar autocompletado y detección de errores precisa:

- **Defines:** Macros de preprocesador como `ARDUINO_ARCH_ESP32`, `ESP32`, `BOARD_HAS_PSRAM` y `TEMPLATE_MATCHING_MODE`.

- **Include paths:** Rutas de búsqueda para headers, incluyendo directorios de ambos módulos y la carpeta shared.

Estas configuraciones garantizan que VS Code comprenda correctamente el contexto de cada archivo, proporcionando sugerencias precisas y detectando errores antes de la compilación.

C.3.3. Tareas automatizadas

El workspace define múltiples tareas (tasks) que automatizan operaciones frecuentes de desarrollo:

- **Build tasks:** Compilación individual de ESP32-CAM y Arduino Nano mediante PlatformIO.
- **Upload tasks:** Carga de firmware a los dispositivos conectados.
- **Monitor tasks:** Apertura de monitor serial para cada dispositivo con baudrate correcto.
- **Build & Upload Both Boards:** Tarea compuesta que compila y carga ambos módulos secuencialmente, configurada como tarea de build por defecto.

Estas tareas se pueden ejecutar desde el Command Palette de VS Code o mediante atajos de teclado, agilizando el ciclo de desarrollo.

C.3.4. Configuraciones de depuración

El workspace incluye configuraciones de launch para depuración de ambos módulos mediante PlatformIO Debugger, permitiendo ejecución paso a paso, inspección de variables y análisis de call stack directamente en el hardware.

C.3.5. Extensiones recomendadas

El workspace especifica un conjunto de extensiones recomendadas que se sugieren automáticamente al abrir el proyecto:

- `platformio.platformio-ide`: Integración de PlatformIO con VS Code.
- `ms-vscode.cpptools`: IntelliSense y depuración para C/C++.
- `yzhang.markdown-all-in-one`: Herramientas para edición de documentación Markdown.
- `ms-vscode.hexeditor`: Editor hexadecimal para análisis de archivos binarios.

C.4. Bibliotecas utilizadas

C.4.1. Módulo ESP32-CAM (NanoAlgoritmoTemplate)

esp32-camera (v2.0.4)

Biblioteca oficial de Espressif para control de la cámara OV2640 integrada en el módulo ESP32-CAM. Proporciona:

- Inicialización y configuración de parámetros de cámara (resolución, formato de píxel, calidad JPEG).
- Captura de frames en diversos formatos (PIXFORMAT_GRAYSCALE, PIXFORMAT_RGB565, PIXFORMAT_JPEG).

- Control de características como brillo, contraste, saturación y balance de blancos.
- Gestión eficiente de memoria PSRAM para buffers de imagen.

El proyecto utiliza formato PIXFORMAT_GRAYSCALE (8 bits por píxel) para reducir consumo de memoria en 50 % comparado con formatos de color.

Arduino.h (ESP32 Core)

Framework Arduino para ESP32, proporcionando APIs de alto nivel para GPIO, comunicación serial (UART, I2C, SPI), timers, interrupciones y gestión de tareas mediante FreeRTOS.

Bibliotecas propias

- `TemplateConfig.h`: Definiciones de configuración del sistema, incluyendo parámetros de cámara, configuración WiFi, ángulos de rotación y umbrales de detección.
- `TemplateResponse.h`: Estructura de respuesta conteniendo ángulo detectado, nivel de confianza y datos de rotación.
- `TemplateLibrary.h`: Biblioteca de templates predefinidos (28x28 píxeles) para los 9 patrones reconocibles: Star, Circle, Arrow Up, Diamond, Triangle, Plus, X-Shape, Square, Squiggly Lines.
- `Commands.h`: Constantes de comandos para comunicación serial con Arduino Nano.

C.4.2. Módulo Arduino Nano (NanoCerebroTemplate)

Arduino.h (AVR Core)

Framework Arduino para microcontroladores AVR (ATmega328P), proporcionando funciones de control de hardware y abstracción de periféricos.

PID by Brett Beauregard (v1.2.1)

Biblioteca de control PID (Proporcional-Integral-Derivativo) altamente optimizada para microcontroladores. Implementa algoritmo PID con anti-windup, límites de salida configurables y modo automático/manual. Utilizada para control preciso de velocidad de motores basándose en el ángulo de steering detectado por el módulo de visión.

Bibliotecas propias

Las mismas bibliotecas compartidas que el ESP32-CAM (TemplateConfig.h, TemplateResponse.h, Commands.h), garantizando compatibilidad de protocolo de comunicación entre ambos módulos.

Anexo D

Pseudocódigos

El presente anexo tiene como objetivo describir, mediante pseudocódigo, el algoritmo de reconocimiento de patrones basado en pares de puntos y el algoritmo de control de motores.

D.1. Algoritmo de reconocimiento de patrones basado en pares de puntos

Algorithm 1: Detección de par de puntos

Require: Imagen binaria 96×96 (0 = negro)
Require: Configuración $config$ con:
    min_point_area
    $max\_point\_distance$
    num_cards
    $distance\_tolerance$
    $card\_distances[1..num\_cards]$
Ensure: $(card\_id, \theta)$ si es válido, `INVALIDO`

Inicializar $visitedAsFirst[p] \leftarrow false$
Inicializar $visitedAsSecond[p] \leftarrow false$
$firstAttempts \leftarrow 0$
while $firstAttempts < 6$ **do**
    $semilla_1 \leftarrow$ SearchConcentricRings(imagen, visitedAsFirst)
    if $semilla_1 = \emptyset$ **then**
        break
    end if
    $punto_1 \leftarrow$ ExploreBlobFromSeed(imagen, $semilla_1$, visitedAsFirst)
    Copiar píxeles visitados de $visitedAsFirst$ a $visitedAsSecond$
    if $punto_1.area < config.min\_point\_area$ **then**
        continue
    end if
    $secondAttempts \leftarrow 0$
    $segundoValido \leftarrow false$

```
    while secondAttempts < 6 do
        semilla_2 ← FindSecondPointRadial(imagen, punto_1.centroid, config.max_point_distance, visitedAsSecond)
        if semilla_2 = ∅ then
            break
        end if
        punto_2 ← ExploreBlobFromSeed(imagen, semilla_2, visitedAsSecond)
        if punto_2.area < config.min_point_area then
            continue
        end if
        segundoValido ← true
        break
    end while
    if not segundoValido then
        firstAttempts ← firstAttempts + 1
        continue
    end if
    d ← CalculateDistance(punto_1.centroid, punto_2.centroid)
    card_id ← DistanceToCardID(d, config)
    if card_id = 0 then
        firstAttempts ← firstAttempts + 1
        continue
    end if
    if punto_1.area ≤ punto_2.area then
        p_small ← punto_1
        p_large ← punto_2
    else
        p_small ← punto_2
        p_large ← punto_1
    end if
    θ ← CalculateRotationAngle(p_small, p_large)
    if θ < −115° or θ > 115° then
        firstAttempts ← firstAttempts + 1
        continue
    end if
    return (card_id, θ)
end while
return INVALIDO
```

D.2. Algoritmo de control de motores

Algorithm 2: Control de motores con PID y pivot turn

Require: Comunicación serial con módulo de visión
Require: Driver de motores TB6612FNG configurado
Require: Parámetros: $K_p = 2{,}0$, $K_i = 0{,}5$, $K_d = 1{,}0$
Require: Velocidades: $baseSpeed = 25$, $minSpeed = 15$, $maxSpeed = 55$
Ensure: Control continuo de motores basado en detección

INICIALIZACIÓN
Configurar pines de motor (PWMA, AIN1, AIN2, PWMB, BIN1, BIN2, STBY)
Habilitar driver de motores: $STBY \leftarrow HIGH$
Inicializar controlador PID con límites $[-50, 50]$ y período $100ms$
$lastValidAngle \leftarrow 0{,}0$
$hasValidAngle \leftarrow true$

Handshake con módulo de visión
Enviar `INIT_HANDSHAKE` vía serial
Esperar recepción de `REQUEST_CONFIG_COMMAND`
Enviar estructura `AlgorithmConfig` serializada
Esperar confirmación `CONFIG_RECEIVED_COMMAND`

CICLO PRINCIPAL
while *true* **do**

    // **Solicitar nuevo frame**
    Enviar `SIMPLE_REQUEST_COMMAND` al módulo de visión

    // **Recibir datos de detección**
    Esperar recepción de $sizeof(SimpleCommand)$ bytes (timeout 1s)
    if timeout ocurrió **then**
        continue ▷ Reintentar en siguiente iteración
    end if
    Deserializar buffer → *result* (detected_card, rotation_angle, valid_angle)

    // **Determinar ángulo a utilizar**
    $angleToUse \leftarrow 0{,}0$
    if $result.valid\_angle = true$ **then**
        $angleToUse \leftarrow result.rotation\_angle$
        $lastValidAngle \leftarrow result.rotation\_angle$
        $hasValidAngle \leftarrow true$
    else
        if $hasValidAngle = true$ **then**
            $angleToUse \leftarrow lastValidAngle$ ▷ Usar último ángulo conocido
        else

```
            continue                                  ▷ Sin datos válidos, esperar siguiente frame
        end if
    end if

    // Seleccionar modo de control
    if result.valid_angle = true and |angleToUse| > 70° then

        // Modo: Pivot Turn
        pivotSpeed ← 30
        if angleToUse > 0 then
            leftSpeed ← pivotSpeed                                          ▷ Girar derecha
            rightSpeed ← −pivotSpeed
        else
            leftSpeed ← −pivotSpeed                                         ▷ Girar izquierda
            rightSpeed ← pivotSpeed
        end if
    else

        // Modo: Control PID
        input ← angleToUse
        setpoint ← 0,0
        Calcular output usando PID(input, setpoint, K_p, K_i, K_d)
        leftSpeed ← baseSpeed − output
        rightSpeed ← baseSpeed + output
        leftSpeed ← Constrain(leftSpeed, minSpeed, maxSpeed)
        rightSpeed ← Constrain(rightSpeed, minSpeed, maxSpeed)
    end if

    // Aplicar control de motores
    SetMotorSpeed(Motor_Izquierdo, leftSpeed)
    SetMotorSpeed(Motor_Derecho, rightSpeed)
    lastCommandTime ← tiempo_actual

    // Mantener movimiento durante 500ms
    moveStartTime ← millis()
    while millis() − moveStartTime < 500 do
        delay(10)
    end while

    // Verificar timeout de seguridad
    if tiempo_actual − lastCommandTime > 2000ms then
        StopMotors
    end if
end while
```

```
function SetMotorSpeed(motor, speed)
    if speed > 0 then
        Configurar motor en dirección forward
        Aplicar PWM = speed
    else
        Configurar motor en dirección backward
        Aplicar PWM = |speed|
    end if
end function

function StopMotors
    Detener ambos motores (PWM = 0)
    Configurar direcciones en LOW
end function
```

Anexo E

Templates utilizados

Figura E.1: Template de flecha

Figura E.2: Template de círculo

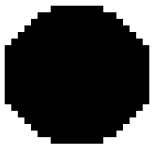


Figura E.3: Template de circulo relleno

Figura E.4: Template de lineas