

# Diseño de un robot futbolista de control híbrido usando un ESP32 y visión por computador en un portátil

Chacón Gómez. José Daniel  
Estudiante de Ingeniería en Informática  
Universidad Nacional Experimental del Táchira (UNET)  
San Cristóbal, Táchira, Venezuela  
josedaniel.chacon@unet.edu.ve

Castillo Gimenez. Alba Patricia  
Estudiante de Ingeniería Mecánica  
Universidad Nacional Experimental del Táchira (UNET)  
San Cristóbal, Táchira, Venezuela  
alba.castillo@unet.edu.ve

**Abstract**—Este artículo presenta el *Auto Soccer Bot*, un robot móvil de bajo costo para tareas de fútbol robótico construido alrededor de un ESP32-CAM y una pila de percepción ejecutada en un portátil. El sistema sigue un paradigma de control híbrido con dos modos: (i) teleoperación manual impulsada por gestos de mano basados en visión (MediaPipe) y (ii) un modo automático en el que el ESP32-CAM transmite video MJPEG por Wi-Fi a una aplicación en el host que realiza detección híbrida del balón (umbrales de color HSV más YOLO) y alimenta una máquina de estados finitos (FSM) para el control. Todos los comandos se transportan a través de una API HTTP (control) desacoplada del servicio de transmisión MJPEG. La arquitectura enfatiza baja latencia mediante retención del último frame y envío de comandos con limitación de tasa y deduplicación. Además del detector de balón en vivo, entrenamos y validamos modelos YOLOv11 para las clases *goal* y *opponent* (mAP@0.5 de validación = 0.991), que aún no están integrados en la máquina de estados del modo automático. Detallamos el hardware, el firmware y el software del host, y reportamos el estado actual del proyecto: control manual robusto y seguimiento fiable del balón en el lazo automático, dejando la fusión de decisiones multiobjeto para integración futura.

**Index Terms**—Robótica móvil, ESP32-CAM, Visión por computador, HTTP, Teleoperación, Control por gestos de la mano, Detección de objetos, YOLO

## I. INTRODUCCIÓN

La robótica móvil autónoma—y el fútbol robótico en particular—ofrece un entorno compacto para integrar percepción, decisión y control. Un reto persistente es habilitar percepción visual avanzada en plataformas de bajo costo sin sobrecargar el microcontrolador a bordo.

Muchos robots educativos implementan una autonomía sencilla (p.ej., seguidor de línea) o control remoto básico. Rara vez combinan (i) teleoperación intuitiva, (ii) autonomía con visión en el lazo y (iii) una separación limpia entre percepción de alto nivel y actuación de bajo nivel en hardware restringido.

Este trabajo introduce el *Auto Soccer Bot*, que acopla un robot ESP32-CAM de bajo costo con una pila de percepción y decisión en portátil. El diseño distribuido descarga la visión intensiva en cómputo al portátil, mientras el ESP32 se enfoca en transmisión en tiempo real y control de motores. El sistema ofrece dos modos: uno manual por gestos (MediaPipe) y

otro automático que ingiere el flujo MJPEG del ESP32-CAM y ejecuta un detector híbrido (HSV + YOLO) dentro de un controlador de estados finitos para seguir el balón. La comunicación usa una API HTTP simple en red local, desacoplando el plano de control del flujo de video.

**Estado de implementación.** La teleoperación manual es robusta y en tiempo real; el modo automático logra seguimiento fiable del balón. Más allá del detector en vivo, se entrenaron y validaron modelos YOLOv11 para *goal* y *opponent* (mAP@0.5 = 0.991 en validación), pero aún no se conectan a la máquina de estados del modo automático; la fusión de decisiones multiobjeto queda para trabajo futuro.

**Contribuciones.** Este trabajo ofrece: (i) una arquitectura híbrida teleoperación/autonomía de bajo costo sobre ESP32-CAM; (ii) ingesta MJPEG de baja latencia con retención del último frame y transporte HTTP resiliente; (iii) una tubería de percepción HSV+YOLO integrada con una FSM para seguimiento del balón; y (iv) detectores entrenados y validados para *goal* y *opponent* para habilitar comportamientos multiobjeto.

**Estructura del artículo.** La Sección II revisa trabajos relacionados. La Sección III presenta la arquitectura y el esquema de comunicación. La Sección IV detalla el hardware. La Sección V describe el firmware y las aplicaciones del host. La Sección VI reporta experimentos, y la Sección VII concluye.

## II. TRABAJOS RELACIONADOS

El diseño del *Auto Soccer Bot* se apoya en paradigmas consolidados de robótica móvil. La elección central—descargar cómputo desde un robot restringido hacia una estación base más potente—es una estrategia bien documentada para crear sistemas autónomos efectivos y de bajo costo [4]. En este modelo, el robot actúa como plataforma móvil de sensores y actuadores, transmitiendo datos crudos o mínimamente procesados por un enlace inalámbrico a un host que realiza tareas intensivas como decisión basada en IA y procesamiento de video [4], [5]. Este enfoque, prototipado en sistemas como la plataforma de búsqueda y rescate “SAVIOUR” [4], permite

usar microcontroladores económicos como el ESP32-CAM, que sobresalen en transmisión de video pero no pueden ejecutar localmente modelos de visión complejos [5].

Para la toma de decisiones autónoma, especialmente en entornos dinámicos como el fútbol robótico, las máquinas de estados finitos (FSM) son una herramienta probada y ampliamente usada [6], [7]. Las FSM proporcionan una forma transparente y manejable de estructurar el comportamiento del robot, definiendo estados como `APPROACH_BALL` o `DRIBBLE_TOWARDS_GOAL` y las condiciones sensoriales que disparan transiciones [6]. Los sistemas de percepción que alimentan estas FSM suelen emplear estrategias híbridas. Aunque modelos de aprendizaje profundo como YOLO [3] brindan detección robusta de objetos, pueden fusionarse con métodos más rápidos y simples para mejorar el desempeño en tiempo real. Combinar un detector primario con un algoritmo de seguimiento secundario, como un filtro de partículas o un rastreador por color, crea un sistema más resiliente frente a fallos momentáneos u oclusiones [8].

En interacción humano–robot, el control por gestos basado en visión ha emergido como una alternativa intuitiva y de bajo costo a interfaces físicas [9]. Marcos como MediaPipe de Google han acelerado esta tendencia al proporcionar detección en tiempo real de puntos de referencia de la mano desde una cámara 2D estándar [1], [10]. Esto permite mapear poses específicas (p.ej., puño cerrado) o movimientos continuos a comandos del robot, creando una experiencia de teleoperación natural y accesible sin hardware especializado [9].

### III. ARQUITECTURA DEL SISTEMA

El sistema (Fig. 1) es una arquitectura distribuida con dos agentes principales: el **robot ESP32** y el **host portátil**. Las tareas intensivas (visión y lógica de decisión) se ejecutan en el portátil, mientras el ESP32 se centra en transmisión de video y actuación de bajo nivel. Hay dos bucles de control—seleccionados ejecutando la aplicación Python correspondiente en el host.

#### A. Modo de Control Manual

En modo manual, el flujo de datos se origina en la webcam del portátil y alimenta un lazo gesto→comando:

- 1) **Sensado:** Una webcam captura frames RGB del usuario.
- 2) **Percepción:** Una aplicación Python (MediaPipe Hands) detecta puntos de referencia de la mano y clasifica el gesto de la mano derecha en comandos discretos (p.ej., `forward`, `left`, `stop`); la distancia pulgar–índice de la mano izquierda se mapea a la velocidad.
- 3) **Decisión/Codificación:** La intención clasificada se mapea a primitivas de movimiento y se serializa como JSON.
- 4) **Actuación:** El comando se envía vía HTTP POST al endpoint `/move` del robot ESP32. El firmware analiza la carga y actualiza PWM y dirección de motores.

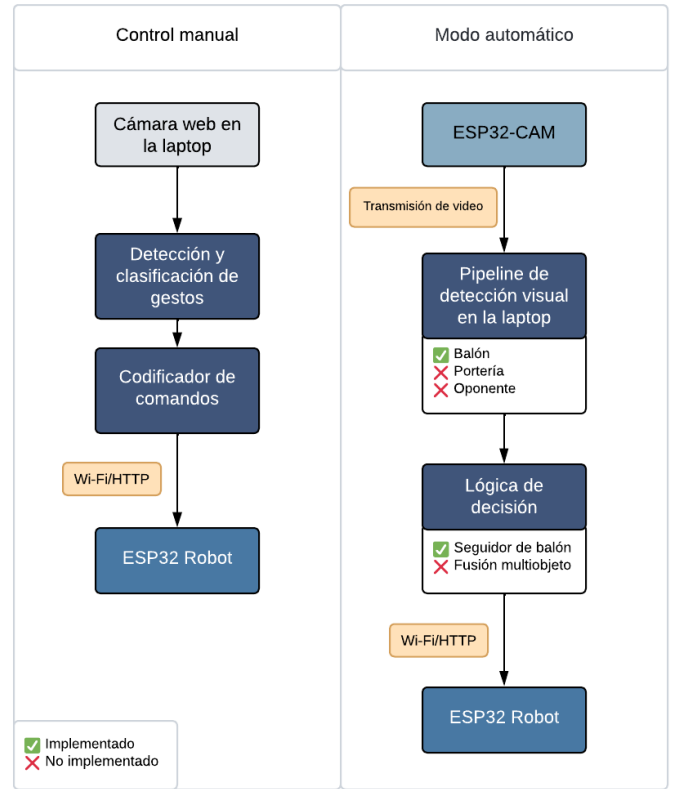


Fig. 1. Arquitectura del sistema y flujo de datos para modos manual y automático.

#### B. Modo Automático

En modo automático, el flujo de datos se origina en el ESP32-CAM a bordo:

- 1) **Sensado:** El ESP32-CAM transmite MJPEG por Wi-Fi a resolución QVGA ( $320 \times 240$ ) desde un servidor dedicado.
- 2) **Percepción:** Una aplicación en el host se suscribe al flujo y realiza detección *híbrida* del balón: umbral HSV rápido en cada frame, complementado con un detector YOLO cada  $N$  frames. (Los modelos YOLOv11 para `goal` y `opponent` están entrenados y validados, pero aún no integrados en el lazo en vivo.)
- 3) **Decisión:** La pose del balón alimenta una FSM que selecciona acciones como `SEARCHING` y `APPROACHING` para centrarlo y avanzar hacia él.
- 4) **Actuación:** La acción elegida se serializa como JSON y se emite a `/move` vía HTTP POST.

#### C. Protocolo de Comunicación

La comunicación host–robot usa HTTP en una red Wi-Fi local. El firmware del ESP32 aloja dos servidores HTTP independientes:

- **Servidor de control (puerto 80):** Atiende peticiones no-streaming y expone `/`, `/status`, `/control`, `/capture` y el endpoint de movimiento

```
/move (acepta application/json, p.ej.,
{"direction":"forward","speed":150}).
```

- **Servidor de streaming (puerto 81):** Sirve /stream como MJPEG usando multipart/x-mixed-replace.

Este diseño aísla el plano de control del camino de video de alto ancho de banda, minimizando la latencia de comandos. En el host, la retención del último frame y los envíos con limitación de tasa y deduplicación reducen el retardo extremo a extremo y evitan saturar el microcontrolador.

*Nota:* La arquitectura está intencionalmente desacoplada para que detectores adicionales (p. ej., goal, opponent) puedan cablearse en la capa de percepción y fusionarse en la FSM sin modificar el firmware.

#### IV. IMPLEMENTACIÓN DE HARDWARE

El robot usa piezas comerciales de bajo costo. Un ESP32-CAM maneja sensado y red, mientras un L298N (doble puente H) proporciona potencia a motores y adaptación de niveles. La Tabla I lista los componentes principales.

TABLE I  
COMPONENTES CLAVE DE HARDWARE

Componente	Función principal
ESP32-CAM (AI-Thinker)	Cámara + Wi-Fi; aloja servidores HTTP; emite señales de control al driver de motor
L298N Doble puente H	Acciona dos motores DC; distribuye potencia (raíl lógico de 5 V)
Motores DC con caja (2x)	Locomoción por tracción diferencial
Batería (2S, 2 × 3.3 V)	Alimentación del sistema (VIN del L298N), interruptor maestro en serie
Chasis del robot	Estructura

##### A. Controlador principal y driver de motor

**ESP32-CAM:** El módulo AI-Thinker ESP32-CAM se alimenta del raíl de 5 V y expone dos servidores HTTP (control y streaming). Sus GPIO son de *lógica 3.3 V*, suficiente para excitar entradas del L298N (umbral alto típico  $\approx 2.3$  V). El módulo captura video, sirve endpoints y emite PWM/señales lógicas para la actuación vía el driver.

**Driver L298N:** El L298N recibe la batería en serie en VIN (a menudo “+12V”) para alimentar los puentes H. En nuestro montaje *mantenemos* el regulador de 5 V a bordo (jumper 5V-EN instalado) y usamos el pin +5V para alimentar tanto la lógica del L298N como el pin de 5 V del ESP32-CAM (ver Fig. 2). Con un pack 2S de dos celdas de 3.3 V ( $\approx 6.6$  V nominal) este regulador opera cerca de su dropout; funcionó de forma fiable en pruebas, aunque con margen limitado ante transitorios de motor.

*Alternativa robusta (opcional):* Si se observan brownouts o reinicios de la cámara, retire 5V-EN y alimente tanto el pin +5V del L298N como el ESP32-CAM desde un módulo DC-DC de 5 V ( $\geq 1$  A), manteniendo tierras comunes.

##### B. Alimentación y actuación

La plataforma usa tracción diferencial con dos motores DC.

- **Motores DC:** Dos motores de 6 V se accionan desde el L298N. La dirección se fija con IN1--IN4; la velocidad usa PWM en ENA/ENB (LEDC del ESP32). Un PWM en rango kHz bajo-medio evita ruido audible y respeta límites del L298N.
- **Sistema de potencia:** Dos celdas de 3.3 V en serie (2S,  $\approx 6.6$  V) alimentan VIN del L298N a través de un interruptor maestro (SW1). Con 5V-EN instalado, el regulador a bordo del L298N provee el raíl +5V usado por la lógica del L298N y el pin de 5 V del ESP32-CAM. Todas las tierras (batería, L298N, ESP32-CAM) son comunes.

Un diagrama de cableado consistente con esta configuración se muestra en la Fig. 2.

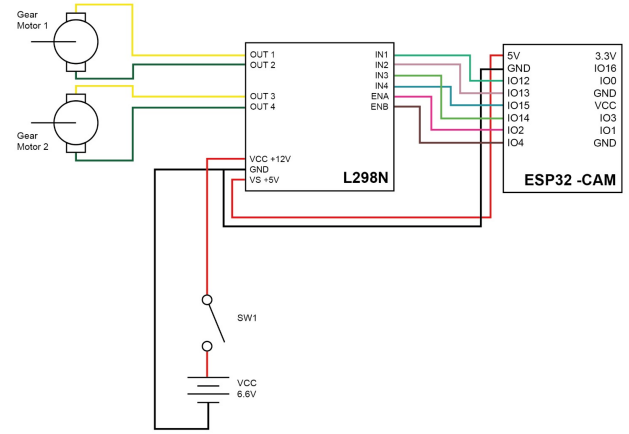


Fig. 2. Diagrama: batería 2S (2 × 3.3 V) a VIN del L298N, 5V-EN instalado y +5V alimentando el ESP32-CAM. Salidas de motor en OUT1/OUT2 y OUT3/OUT4; líneas de control conectan GPIO del ESP32-CAM a IN1--IN4 y ENA/ENB.

#### V. IMPLEMENTACIÓN DE SOFTWARE

La funcionalidad se divide entre el firmware en el robot (ESP32-CAM) y módulos Python en el host. El firmware provee E/S en tiempo real y una pequeña API HTTP; el host realiza percepción y toma de decisiones. Todos los módulos del host usan entornos virtuales aislados y archivos de configuración (manual\_control/config.py, auto\_soccer\_bot/config\_auto.py) para reproducibilidad.

##### A. Firmware del ESP32

El firmware está escrito en C++ sobre Arduino core para ESP32 y esp\_http\_server del ESP-IDF. Al arrancar, inicializa GPIO (enable/dirección de motores), la cámara (QVGA, calidad JPEG 30), se une a Wi-Fi (modo estación) y luego inicia dos servidores HTTP.

*Diseño de doble servidor:* (dentro de `esp32cam_robot`): `WebServerManager.h/.cpp` y `WebRequestHandlers.h/.cpp`:

- **Control (puerto 80):** rutas `/`, `/status`, `/control`, `/capture`, `/move`.
- **Streaming (puerto 81):** ruta `/stream` (JPEG, `multipart/x-mixed-replace`).

*Endpoints (esquema/semántica):*

- `GET /status` → JSON con heap, estimación de FPS, RSSI de Wi-Fi y estado de GPIO.
- `GET /control?var=name&val=int` → ajusta parámetros de cámara (`framesize`, `quality`, `brightness`, etc.).
- `GET /capture` → frame JPEG único.
- `GET /stream` → MJPEG continuo con límite explícito; cada parte carga un JPEG.
- `POST /move` (JSON) → comando de movimiento:

```
{
  "direction": "
    forward|
    backward|
    left|
    right|
    soft_left|
    soft_right|
    stop",
  "speed": 0..255,
  "turn_ratio": 0.0..1.0 (optional)
}
```

Responde 200 OK con un pequeño ack en JSON; cargas inválidas devuelven 400.

*Actuación:* `MotorControl.h/.cpp` abstrae el control diferencial usando PWM LEDC del ESP32 y pines de dirección del L298N. Llamadas de alto nivel (`moveForward()`, `turnLeft()`, `stopMotors()`) limitan internamente el duty, aplican *deadband* calibrado y traducen `turn_ratio` a velocidades asimétricas de ruedas para giros suaves.

## B. Aplicación en el host: Control Manual

El módulo de teleoperación (`manual_control/`) implementa control por gestos con una webcam del portátil.

*Flujo (sensar → interpretar → comandar):*

- 1) **Sensado** (`camera_manager.py`): captura frames vía OpenCV con apertura/cierre seguro y *throttling* de FPS.
- 2) **Percepción** (`hand_detector.py`): MediaPipe Hands devuelve 21 puntos y lateralidad; los frames se espejan (selfie view) y convierten a RGB.
- 3) **Clasificación** (`gesture_classifier.py`): la mano derecha codifica la dirección discreta (lógica fingertip-MCP); la distancia pulgar-índice de la mano izquierda (normalizada) se mapea a velocidad en  $[0, 255]$ .

- 4) **Actuación** (`robot_communicator.py`): construye JSON y publica a `/move` de forma asíncrona (HTTPX). Una capa de deduplicación suprime comandos idénticos consecutivos y un limitador de tasa evita inundar el plano de control.

*Configuración de ejecución:* `config.py` contiene IP/puertos del robot, índice de cámara, confidencias de MediaPipe y límites de mapeo de velocidad. Se ejecuta con `python -m manual_control.main` y muestra una ventana superpuesta (ESC para salir).

## C. Aplicación en el host: Modo Automático

El controlador autónomo (`auto_soccer_bot/`) cierra el lazo percepción→actuación desde el flujo del ESP32-CAM. La orquestación en `application.py` comprende cuatro tareas asíncronas: ingesta, percepción, decisión y transporte de comandos.

### 1) Ingesta de flujo (`camera_manager.py`):

Un `httpx.AsyncClient` se conecta a `http://<ESP32_IP>:81/stream` con `timeout` de conexión y sin `timeout` de lectura. El límite MJPEG se analiza incrementalmente; sólo se conserva el *último* frame decodificado (descartando obsoletos) para minimizar latencia. Geometría por defecto: QVGA ( $320 \times 240$ ); una ruta de *resize* opcional está desactivada.

2) *Percepción híbrida* (`ball_detector.py`): Combinamos un detector HSV ligero con pasadas programadas de YOLO:

- **HSV:** umbral en HSV usando `LOWER_BALL_COLOR` / `UPPER_BALL_COLOR`, morfología suave y filtro de área mínima.
- **YOLO (Ultralytics):** ejecutado cada `DETECTION_INTERVAL` frames (por defecto 6); detecciones filtradas por `TARGET_CLASS_NAMES` y umbral de confianza. Los resultados se *cachean* con un TTL corto para cubrir frames entre pasadas.

Ambos detectores emiten ( $c_x, c_y, area$ ). Una regla de prioridad prefiere YOLO válido; en otro caso, el estimado HSV.

3) *Decisión* (`robot_controller.py`): Una FSM gobierna el comportamiento: `SEARCHING` → `BALL_DETECTED` → `APPROACHING` → `CAPTURED`. Un corredor objetivo  $[x_{min}, x_{max}]$  alrededor del centro reduce oscilación; fuera del corredor, el guiado usa giros suaves con `APPROACH_TURN_RATIO`. Ventanas de confirmación y *grace timers* (`BALL_CONFIRMATION_THRESHOLD`, `BALL_LOST_TIMEOUT_MS`) desbounced transiciones.

4) *Transporte de comandos* (`robot_communicator.py`): Los comandos se publican a `http://<ESP32_IP>:80/move` con JSON `{direction, speed, turn_ratio}`. El comunicador deduplica cargas consecutivas idénticas, aplica espaciado mínimo entre envíos y reintentos acotados con backoff ante errores transitorios. Se registra éxito/fallo.

*Configuración:* Umbrales, URLs, límites HSV y rutas YOLO viven en `config_auto.py`. Flujo por defecto QVGA con calidad JPEG moderada; la percepción

usa SATURATION=3.5 y BRIGHTNESS=1 (opcional) para separar color.

#### D. Entrenamiento del modelo de visión (*soccer\_vision/*)

El módulo *soccer\_vision/* proporciona entrenamiento/evaluación para modelos YOLOv11 con dos clases: goal y opponent.

**Datos y anotación.:** Las imágenes se etiquetaron en Label Studio y se exportaron en formato YOLO (images/labels), ubicándolas bajo *soccer\_vision/dataset/* (train/, opcional val/). Los nombres de clase se validan con *classes.txt*.

**Flujo de entrenamiento.:** Un cuaderno *notebooks/01\_retrain\_yolo.ipynb* invoca *notebooks/modules/train.py* para:

- 1) verificar estructura del dataset y mapa de clases;
- 2) crear partición de validación si falta;
- 3) generar *data.yaml* para Ultralytics;
- 4) lanzar entrenamiento YOLOv11 con semillas fijas y *checkpoints*.

Los artefactos (mejores pesos, matrices de confusión, curvas PR/F1, logs) se copian a *soccer\_vision/results/* para inclusión en el artículo. Un segundo cuaderno *02\_test\_and\_demo.ipynb* demuestra inferencia en medios nuevos.

**Reproducibilidad.:** Cada submódulo Python incluye su *requirements.txt*; los entornos se crean dentro de cada módulo.

## VI. EXPERIMENTOS Y RESULTADOS

Evaluamos (i) los modelos de visión entrenados en *soccer\_vision/* y (ii) el comportamiento del lazo autónomo integrado (streaming → percepción → decisión → actuación). Salvo mención, se usaron valores por defecto de Ultralytics y el ESP32-CAM transmitió MJPEG QVGA (320×240).

#### A. Desempeño del modelo de visión

Entrenamos un **YOLOv11s** ligero en un dataset de dos clases (goal, opponent). La canalización produjo un conjunto de validación retenido y artefactos estándar (matriz de confusión, curvas PR/F1). La Tabla II resume las métricas;<sup>1</sup> las Figuras 3–4 muestran la matriz de confusión y la curva PR.

TABLE II  
DESEMPEÑO EN VALIDACIÓN DE YOLOV11S (2 CLASES)

Métrica	Valor
mAP@0.5 (macro, todas)	0.991
F1 pico	0.86–0.90
AP (goal)	0.995
AP (opponent)	0.987

<sup>1</sup>mAP@0.5 es la media de precisión promedio a umbral IoU 0.5. El F1 pico es la media armónica máxima de precisión y recall al barrer umbrales de score.

El detector alcanzó AP casi perfecta en goal (0.995) y rendimiento robusto en opponent (0.987). La matriz de confusión normalizada (Fig. 3) indica 1.00 de acierto en goal y 0.95 en opponent (5% omitido como fondo). La curva PR (Fig. 4) muestra alta precisión consistente a lo largo del recall, lo que respalda su idoneidad para el control downstream.

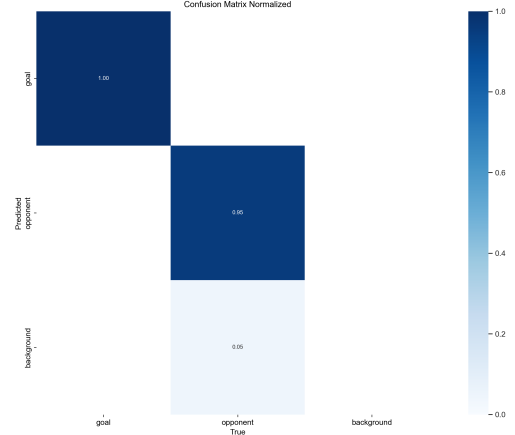


Fig. 3. Matriz de confusión normalizada para YOLOv11s.

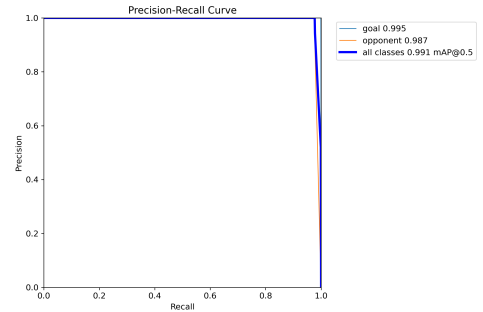


Fig. 4. Curva Precisión-Recall; mAP@0.5 global = 0.991.

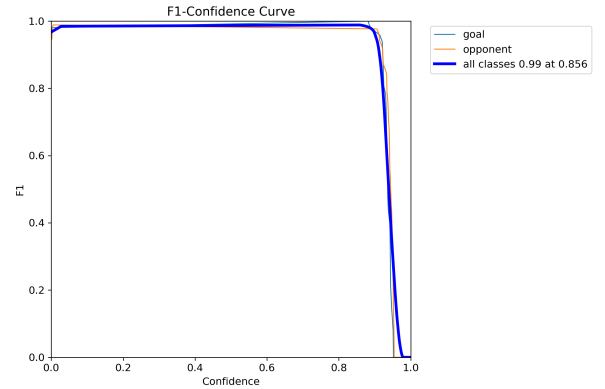


Fig. 5. Curva F1 a través de umbrales de score.

## B. Evaluación del sistema integrado

*Configuración e instrumentación.*: Evaluamos el lazo autónomo completo bajo el § V-C: QVGA (320×240) MJPEG, calidad JPEG 30, YOLO cada  $N=6$  frames, HSV cada frame y la FSM de `RobotController.py` con ventanas de confirmación y corredor horizontal. Registramos tiempos en cada etapa, el detector activo (cache de YOLO vs. HSV) y estadísticas de emisión de comandos (limitación y deduplicación).

*Evolución de la percepción: color  $\rightarrow$  YOLO  $\rightarrow$  híbrido.*

- 1) *Sólo color*: HSV puro entregó respuesta por frame pero fue frágil ante iluminación y fondos. El controlador reaccionaba al signo instantáneo del error lateral del balón, generando pivotes izquierda-derecha (“*thrashing*”) alrededor del centro. La actuación retrasada hacía que el robot “*persiguiera el pasado*”.
- 2) *Sólo YOLO*: Mayor robustez a iluminación/oclusiones, pero cómputo más pesado y ráfagas. Corriendo cada frame, la inferencia acumulaba y, con captura ingenua por URL, formaba colas; corriendo esporádico, las pérdidas transitorias causaban *flapping*. Decisiones con frames obsoletos producían sobrecorrecciones.
- 3) *Híbrido + ingesta fresca*: La versión final combina ambos y corrige la ingesta. `BallDetector` ejecuta YOLO cada `DETECTION_INTERVAL` y trata detecciones como válidas por `yolo_ttl_frames = max(N×2, 3)`; de otro modo retorna HSV. En paralelo, la ingesta migró a un parser MJPEG con *retención del último frame*, eliminando colas y anclando decisiones al frame más reciente.

*Refinamientos del controlador que estabilizaron el rumbo.*

- 1) **Ventana de confirmación (`BALL_DETECTED`)**. Requiere `BALL_CONFIRMATION_THRESHOLD` detecciones consecutivas antes de pasar a `APPROACHING_BALL`, con giros correctivos a velocidad decreciente y `MAX_ADJUSTMENT_TIMEOUT_MS` que evita abortos prematuros.
- 2) **Corredor objetivo con giros suaves (`APPROACHING_BALL`)**. En vez de pivotar por error instantáneo, define umbrales de píxeles  $[x_{\min}, x_{\max}] = [\text{TARGET\_ZONE\_X\_MIN} \cdot W, \text{TARGET\_ZONE\_X\_MAX} \cdot W]$ . Si  $x$  cae fuera, aplica `soft_left/right` con `APPROACH_TURN_RATIO` acotado; de lo contrario avanza `forward`.
- 3) **Temporizadores de pérdida**. Brechas cortas activan `stop` manteniendo el estado; pérdidas prolongadas más allá de `BALL_LOST_TIMEOUT_MS` regresan a `SEARCHING_FOR_BALL`.

*Resultados observados.*

- **Latencia y frescura**. HTTPX + *último frame* removió retrasos por cola; el lazo se mantuvo responsivo.
- **Robustez**. El programa híbrido (YOLO cada  $N$  con TTL, HSV cada frame) sostuvo el seguimiento a través de fallos

breves de color y cambios de iluminación sin el costo de correr YOLO en cada frame.

- **Estabilidad**. La ventana de confirmación, el corredor y los giros suaves eliminaron oscilaciones al cruzar el centro y redujeron el *flapping*.
- **Plano de comandos**. Deduplicación e intervalo mínimo evitaron inundar `/move`, produciendo tiempos de actuación más predecibles.

*Comprobación del modo manual.*: El modo por gestos fue fluido con iluminación frontal; los fallos (oclusiones parciales, poses no frontales) se mitigaron al subir umbrales de confianza y mejorar la luz.

*Limitaciones.*: Los resultados reflejan una única geometría de cámara (QVGA) y nuestro dataset; la generalización a canchas/luces diversas requiere más pruebas. La FSM autónoma actualmente sólo fusiona el balón; integrar `goal/opponent` queda como trabajo futuro (Sec. VII). Finalmente, el control usa HTTP sin cifrar, adecuado para LAN de laboratorio pero no para redes no confiables.

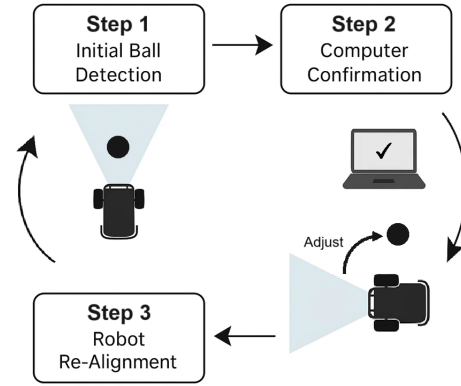


Fig. 6. Corridor-based heading realignment with soft turns and confirmation window.

## VII. CONCLUSIONES Y TRABAJO FUTURO

### A. Conclusiones

Este trabajo presentó el *Auto Soccer Bot*, un robot móvil de bajo costo y control híbrido que acopla una plataforma ESP32-CAM con una pila de percepción y decisión en portátil sobre una API HTTP. Al descargar la visión intensiva al host y mantener la actuación en tiempo real en el robot, el sistema logra comportamientos inviables sólo con el microcontrolador.

#### Resumen de contribuciones.

- Una canalización completa de **doble modo**: (i) teleoperación por gestos con `MediaPipe` y (ii) persecución autónoma del balón impulsada por percepción en host y FSM.
- Una **percepción híbrida** que combina YOLOv11s (programado cada  $N$  frames con cache de TTL corto) y detección por color HSV por frame, sobre ingesta MJPEG de baja latencia con retención del último frame.



- Un **controlador estabilizado** con ventana de confirmación, corredor horizontal con giros suaves y temporizadores de pérdida, implementado en `RobotController.py`, que elimina oscilaciones y *flapping*.
- Un **detector entrenado de dos clases** (goal, opponent) con rendimiento fuerte ( $mAP@0.5 = 0.991$ ), reproducible vía `soccer_vision`.

A lo largo de los experimentos, la combinación de detección híbrida, ingesta de frames frescos y refinamientos de la FSM resolvió los problemas principales (latencia de flujo, decisiones obsoletas, oscilación de rumbo), resultando en un comportamiento autónomo responsivo y predecible.

## B. Trabajo Futuro

El sistema valida la arquitectura y demuestra seguimiento robusto; varias extensiones son naturales:

- **Fusión de decisiones multiobjeto.** Integrar `goal` y `opponent` en el lazo percepción→acción. Ampliar la FSM para *alinearse a portería, disparar, evitar oponente y re-adquirir*.
- **Mejoras del controlador.** Explorar control proporcional ligero del error lateral dentro del corredor (ganancias acotadas) y términos anticipatorios sencillos, preservando la *deadband*.
- **Evaluación a escala.** Añadir ensayos cuantitativos en diversa iluminación/canchas y reportar histogramas de latencia, tasas de re-adquisición, tiempo a centrar y estabilidad de aproximación.
- **Coordinación multirrobot.** Añadir comunicación inter-robot para roles (atacante/defensa) y evasión simple, manteniendo el cómputo distribuido.
- **Transporte y seguridad.** Reemplazar HTTP sin cifrar por autenticación mínima (token precompartido) o migrar comandos a UDP/WebSocket con números de secuencia; considerar TLS para redes no confiables.
- **Portabilidad de modelo.** Empaquetar pesos y configs YOLO vía artefactos/Git LFS y soportar selección dinámica (CPU/GPU) con chequeos al inicio.

## REFERENCES

- [1] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, *et al.*, “MediaPipe: A Framework for Building Perception Pipelines,” *arXiv preprint arXiv:1906.08172*, 2019.
- [2] G. R. Bradski, “The OpenCV Library,” *Dr. Dobbs’s Journal of Software Tools*, 2000.
- [3] Ultralytics, “YOLO by Ultralytics (v11),” GitHub repository, 2024. [Online]. Available: <https://github.com/ultralytics/ultralytics>
- [4] A. A. A. Sethaputri, A. F. P. A. P. Putra and I. K. E. Purnama, “Displacing computing operations to an operator station over a wireless link in autonomous mobile robots,” *International Journal of Computer and Communication Engineering*, vol. 1, no. 2, pp. 125–129, 2012.
- [5] S. Tatipamula, “Computer Vision with OpenCV on ESP32-CAM: Building Intelligent Vision Systems,” *ThinkRobotics*, 2023. [Online]. Available: <https://thinkrobotics.com/blogs/learn/computer-vision-with-opencv-on-esp32-cam-building-intelligent-vision-systems>
- [6] J. J. G. R., A. M. L. G. and J. S. A., “Decision-making system of soccer-playing robots using finite state machine based on skill hierarchy and path planning through Bezier polynomials,” in *2017 IEEE 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, 2017, pp. 1–6.
- [7] J. G. Guarnizo and M. Mellado Artech, “Robot Soccer Strategy Based on Hierarchical Finite State Machine to Centralized Architectures,” *IEEE Latin America Transactions*, vol. 14, no. 8, pp. 3586–3596, 2016.
- [8] Y.-C. Lin, C.-Y. Lin, C.-C. Chen and C.-Y. Chang, “A Hybrid YOLOv4 and Particle Filter Based Robotic Arm Grabbing System in Nonlinear and Non-Gaussian Environment,” *Sensors*, vol. 21, no. 10, p. 3430, 2021.
- [9] Y. Chen, Z. Wang, Z. Li, and S. Li, “Vision-based Gesture Tracking for Teleoperating Mobile Manipulators,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 8889–8895.
- [10] A. M. Al-Khafaji and H. T. S. Al-Rikabi, “Tracked Robot Control with Hand Gesture Based on MediaPipe,” *Journal of Engineering*, vol. 29, no. 6, pp. 123–136, 2023.