



Estimación de posición con una sola imagen

Taller de Proyecto II

**LAVALLE Lautaro, PEINETTI MARTIN Sofía,
RIVERA FERNÁNDEZ Cristian**

Informe final

Universidad Nacional de La Plata
2do cuatrimestre 2023
4/12/2023

Índice

1. Descripción general del proyecto	2
2. Esquema gráfico del proyecto	3
2.1. Procesos y funciones	3
2.2. Hardware	9
2.2.1. Arduino IDE	9
2.2.2. Protocolo MQTT	10
2.2.3. Decisión de utilizar TCP en capa de transporte y HTTP en capa de aplicación en lugar de RTSP: Un análisis detallado	10
2.2.4. Pruebas empíricas	11
3. Documentación del software	12
3.1. Código Principal (Main)	12
3.2. Código del modelo YOLOv8	13
3.3. Código para calibración de la cámara	14
3.4. Código para detección de ArUco	14
3.5. Código para el funcionamiento del auto robótico	15
3.6. Código para Control Remoto de Robot con MQTT	16
3.7. Código para el manejo de la ESP32	17
4. Documentación relacionada	18
5. Conclusiones finales	20
6. Ápendice A: Materiales y Presupuestos	21

1. Descripción general del proyecto

El proyecto tiene como objetivo principal la creación de un sistema integral de detección de objetos y realidad aumentada para guiar un vehículo robótico, con la utilización únicamente de una cámara y de las librerías provistas por OpenCV de Python. A lo largo de su evolución, se han abordado distintos aspectos desde la instalación de herramientas clave hasta la implementación de estrategias avanzadas de detección y seguimiento.

La propuesta inicial contemplaba el desarrollo de un sistema que posibilitara al vehículo detectar y navegar de manera autónoma evitando obstáculos a partir de emplear una matriz originada desde un ArUco que permitiera obtener la distancia del objeto más cercano.

La adopción de la tecnología ArUco como marcadores visuales para la realidad aumentada fue otra decisión crucial. Esta tecnología permitió no solo la superposición de información virtual sobre el mundo real, sino también el trazado efectivo de líneas desde los marcadores a los objetos detectados, mejorando así la comprensión del entorno.

Como cambio radical, en una segunda instancia, se abandonó la idea que involucraba el trazado de una línea desde el centro del marcador ArUco al centro del objeto, por una línea desde el centro del ArUco hacia la esquina de un objeto. Esta modificación, impulsada por la necesidad de mejorar la precisión y reducir errores, se convirtió en un punto de inflexión en la estrategia de reconocimiento de objetos.

La etapa final de este proyecto hizo especial énfasis en la integración del software con el hardware. La implementación del protocolo MQTT ha permitido establecer una comunicación eficiente y bidireccional entre el sistema central y el robot, facilitando la transmisión de indicaciones y comandos vitales para su desplazamiento autónomo. Por otro lado, la transmisión de video mediante RTSP ha añadido una capa adicional de información, permitiendo al operador y al sistema central obtener una vista en tiempo real del entorno donde se encuentre el robot. Este enfoque integrado de software y hardware ha potenciado la autonomía y eficiencia del vehículo robótico.

Por su parte, la catédra proveyó de sustanciales aportes a la hora de la redistribución del hardware necesario para el proyecto, como también indicaciones de colocación de la cámara para una mejor captura de la imagen.

Como objetivo subsidiario del trabajo se encuentra la documentación y evaluación del rendimiento del sistema en diversas condiciones. La documentación y la evaluación exhaustiva han sido esenciales para comprender cómo se comporta nuestro sistema en una variedad de situaciones y escenarios. Realizamos pruebas y experimentos en diferentes condiciones de iluminación, terreno y escala de objetos, midiendo los tiempos de transmisión del video, concluyendo en el impacto que esto causa dentro de la visual del proyecto.

2. Esquema gráfico del proyecto

2.1. Procesos y funciones

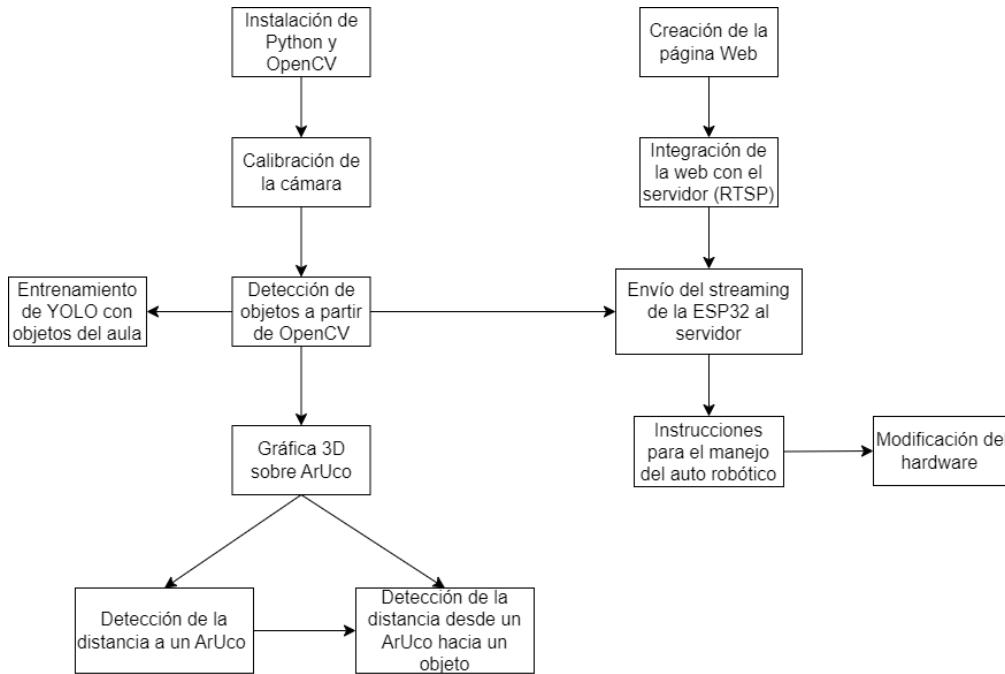


Figura 1: Esquema de procesos y funciones del proyecto

- **Instalación de OpenCV y Python en el entorno de desarrollo:** Hemos configurado el entorno de desarrollo instalando las bibliotecas esenciales de OpenCV y Python. Esto asegura que tengamos acceso a las herramientas necesarias para procesar imágenes y datos visuales.

Para empezar, elegimos PyCharm como nuestro entorno de desarrollo integrado (IDE) principal para el proyecto debido a su amplia compatibilidad con Python y su conjunto de herramientas para el desarrollo de aplicaciones de visión por computadora.

Allí, procedimos a instalar Python. Creamos un entorno virtual en PyCharm para mantener nuestro proyecto aislado y asegurarnos de que las bibliotecas y las dependencias se gestionen de manera eficiente. Esto se logró mediante la configuración de un entorno virtual específico para el proyecto, que evita conflictos entre diferentes versiones de bibliotecas.

Dentro del entorno virtual, instalamos OpenCV utilizando el administrador de paquetes pip de Python. Ejecutamos el siguiente comando para instalar OpenCV: `pip install OpenCV`.

Para asegurarnos de que OpenCV se haya instalado correctamente, creamos un proyecto de prueba en PyCharm e importamos la biblioteca cv2 (que es la interfaz de OpenCV en Python). Luego, ejecutamos un script simple para cargar una imagen y verificar que OpenCV esté funcionando correctamente.

Todas las bibliotecas utilizadas en el proyecto se obtuvieron de fuentes confiables y de código abierto. OpenCV se descargó de su sitio web oficial (<https://opencv.org/>). Por su parte, Python se descargó de python.org, que es la fuente oficial del lenguaje de programación.

- **Proceso de calibración de la cámara:** Realizamos un proceso de calibración de la cámara para me-

jorar la precisión de la detección. Esto implica ajustar la cámara para corregir distorsiones y garantizar mediciones precisas.

Para llevar a cabo la calibración de la cámara, adquirimos un código fuente específico llamado “calibracion.py”. Este código fue obtenido de una fuente confiable, que proporcionó un conjunto de scripts y herramientas dedicados a la calibración de cámaras.

Reunimos un conjunto de imágenes de calibración que se utilizaron para ajustar la cámara. Estas imágenes incluyeron patrones de ajedrez, que son comúnmente utilizados en la calibración de cámaras debido a sus características distintivas.

Una vez que el código de calibración finalizó su ejecución, obtuvimos una matriz de la cámara y coeficientes de distorsión. Estos parámetros se utilizaron en nuestro proyecto para corregir las distorsiones en las imágenes capturadas por la cámara.

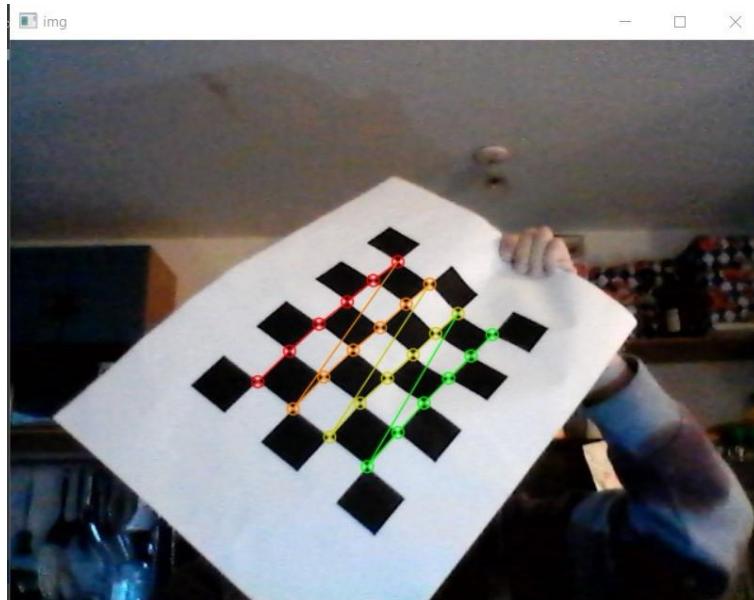


Figura 2: Calibración de la cámara

- **Implementación exitosa de la detección de objetos a partir de OpenCV:** Logramos implementar con éxito la detección de objetos utilizando librerías de OpenCV. Esto nos permitió identificar y rastrear objetos en imágenes capturadas por la cámara.

Utilizando Python y OpenCV, desarrollamos funciones específicas para la detección de objetos. Estas funciones incluyeron la preparación de imágenes, la configuración de parámetros de detección y la implementación de algoritmos de detección de objetos. Utilizamos diferentes escenarios y conjuntos de datos para evaluar la capacidad del sistema para identificar y rastrear objetos.

Primero usamos NumPy, una biblioteca fundamental en Python para realizar operaciones numéricas y matriciales. En nuestro proyecto, fue utilizada para: representar y manipular imágenes como matrices NumPy, realizar cálculos matemáticos en los datos de imagen, como normalización de píxeles, almacenar y procesar datos de coordenadas de objetos detectados y para crear matrices para representar colores y máscaras en la detección de objetos.

Luego, con YOLO, un modelo de detección de objetos en tiempo real cargamos un modelo preentrenado,

como YOLOv8, que es capaz de detectar una amplia variedad de objetos en imágenes. Configuramos los archivos de configuración del modelo y los pesos preentrenados, realizamos la detección de objetos en los fotogramas de la cámara en tiempo real, y recuperar información importante sobre objeto detectados, como las coordenadas de la caja delimitadora, la etiqueta de clase y la confianza de detección.

Los resultados de la detección, que incluyen la ubicación y la etiqueta de los objetos, se almacenan y se utilizan para resaltar objetos en los fotogramas y realizar acciones adicionales, como estimar distancias o tomar decisiones de control como se ve a continuación:

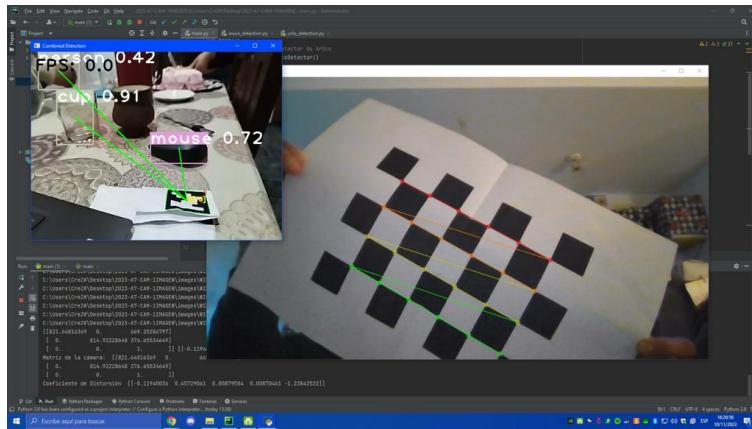


Figura 3: Detección de objetos junto con sus distancias

- **Creación de página web para la visualización de datos y video:** Hemos creado una página web que nos permite visualizar en tiempo real el video capturado por la cámara y los datos del microcontrolador. Para lograrlo, hemos diseñado una interfaz con HTML y CSS, incluyendo un título centrado y una imagen de fondo personalizada que se ajusta a la ventana del navegador. Hemos creado un contenedor principal que alinea tanto el video como los datos, manteniendo el contenido centrado tanto vertical como horizontalmente. Esto crea una presentación ordenada y agradable a la vista.

El video se reproduce en un elemento de video, con un mensaje de respaldo en caso de que el navegador no admita la reproducción. La sección de datos muestra las distancias registradas por el microcontrolador, actualizándose en tiempo real. Esta página web es un avance importante que nos permitirá supervisar el proyecto y avanzar en la integración de la detección de objetos y la navegación autónoma en el vehículo robótico.

- **Gráfica en 3D sobre los ArUcos:** Para lograr esto, primero detectamos y seguimos los marcadores ArUco en la imagen capturada por la cámara. Los marcadores ArUco son códigos bidimensionales que contienen información sobre su identidad y posición relativa en el espacio. Cuando los marcadores se detectan en la imagen, podemos calcular sus poses tridimensionales, es decir, sus posiciones y orientaciones en el mundo real.

Esto involucra la proyección de modelos 3D en la imagen capturada por la cámara, utilizando la información de pose de los marcadores ArUco para determinar la posición y orientación correctas de los objetos 3D en el espacio 2D de la imagen. Esto se hace mediante transformaciones geométricas y cálculos matemáticos que permiten la representación precisa de los objetos 3D en relación con la escena capturada. Esto se realiza mediante la función *estimatePoseSingleMarkers*, que toma las esquinas detectadas, la matriz de la cámara y los coeficientes de distorsión para calcular los valores de rotación (rvec) y traslación (tvec) del marcador.

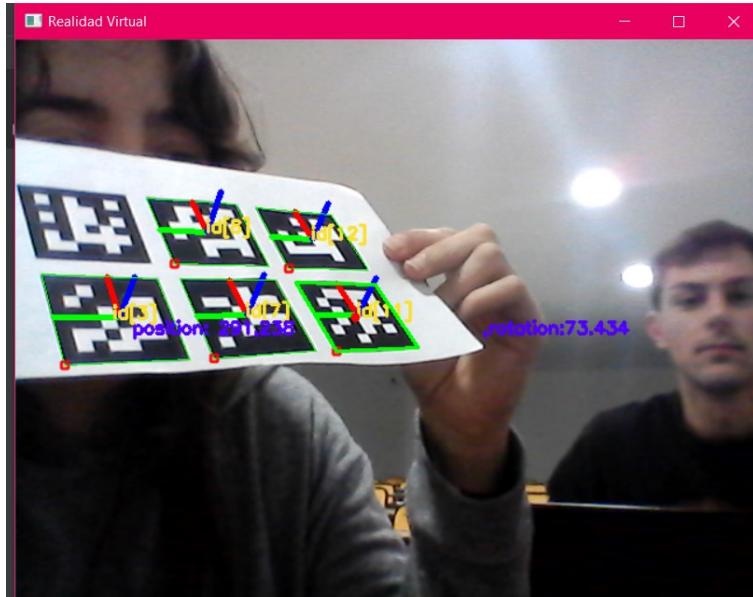


Figura 4: Gráfica sobre ArUco

- **Detección de la distancia a un ArUco:** Inicialmente, consideramos la utilización de una matriz para calcular la distancia entre el marcador ArUco y el objeto detectado. Luego pensamos en utilizar una traza que vaya desde el centro del ArUco hasta el centro del objeto. Sin embargo, tras una cuidadosa reflexión y evaluación de alternativas, optamos por una estrategia más precisa y versátil. En lugar de depender exclusivamente de la matriz o únicamente de una traza, decidimos marcar una línea desde el centro del ArUco hasta una esquina del objeto detectado. Esta elección se vio respaldada por varias razones fundamentales.

Primero, la traza de una línea desde el centro del ArUco hasta una esquina del objeto nos permite considerar la geometría específica de la detección. Al conectar el centro del marcador con una esquina del objeto, estamos tomando en cuenta la relación espacial real entre el ArUco y el objeto, lo cual es esencial para una estimación precisa de la distancia.

Además, la inclusión de una matriz de perspectiva en este proceso agrega un nivel adicional de precisión. Esta matriz se ha utilizado previamente en el proceso de calibración de la cámara, lo que significa que se tiene en cuenta la corrección de la distorsión y la proyección perspectiva en la imagen. Al combinar la información de la línea trazada con la contribución de la matriz de perspectiva, obtenemos una medida más precisa y contextualizada de la distancia entre el marcador ArUco y el objeto.

Al integrar estas consideraciones geométricas y matriciales, hemos mejorado la capacidad del sistema para interpretar de manera más precisa la disposición espacial de los elementos detectados, lo que, en última instancia, contribuye a un funcionamiento más confiable del vehículo en su entorno.

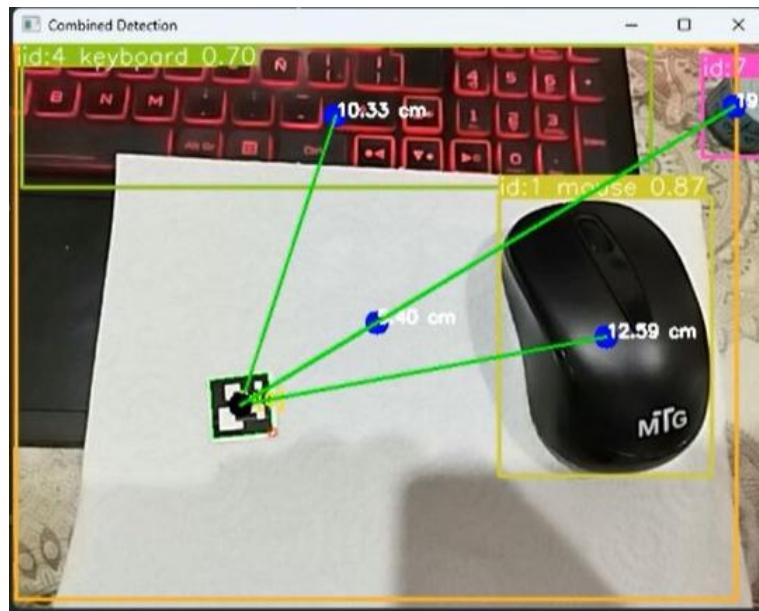


Figura 5: Medición desde un ArUco

■ **Enviar el streaming de la ESP32 al servidor:**
Acciones realizadas:

1. Utilizamos un código que incluye librerías de RTSP
2. Nos conectamos a un access point creado por nosotros.
3. Buscamos la IP local de la ESP32 asignada por el access point, utilizando el programa *Angry IP Scanner*.
4. Abrimos el video desde Python utilizando Open CV, con la IP que econtramos, a través de RTSP.

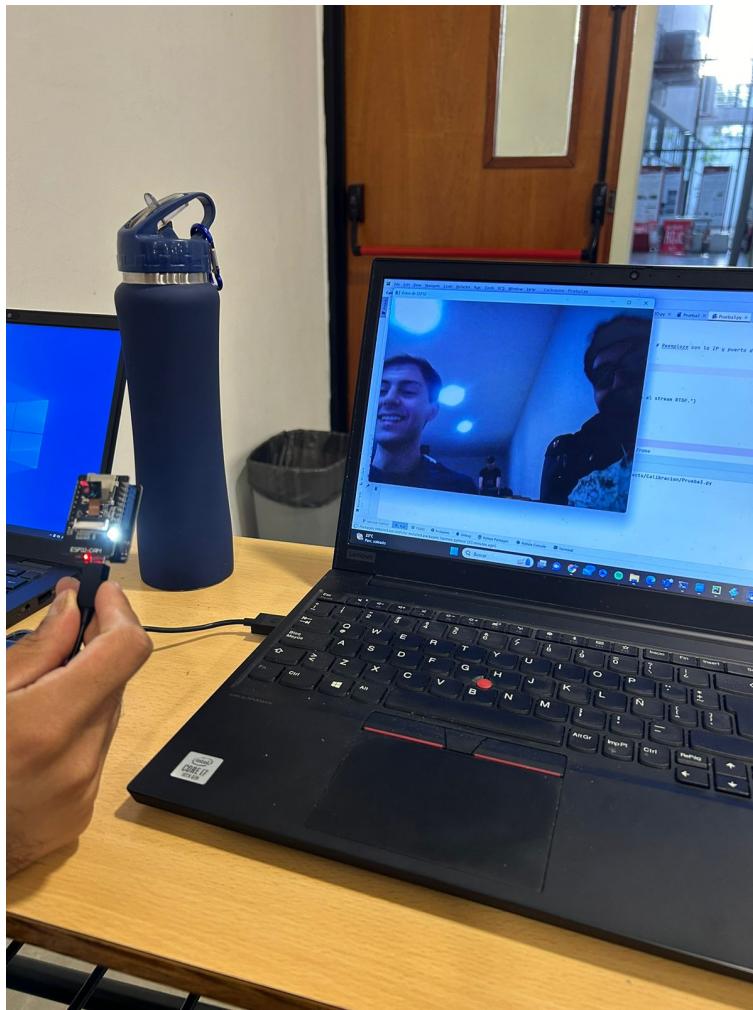


Figura 6: ESP32 en funcionamiento a través de Python

■ **Entrenamiento de YOLO con objetos del aula:**

Acciones realizadas:

1. Capturamos imágenes dentro del aula de los objetos que deseamos detectar. Asegurándonos de tener una variedad de ángulos, iluminación y condiciones.
2. Utilizamos una herramienta de etiquetado de objetos, LabelImg, para marcar los objetos en las imágenes. Asignamos etiquetas a cada objeto y generamos archivos de anotación en formato YOLO.
3. Ajustamos el archivo de configuración de YOLO según nuestras necesidades. Específicamente, configuramos el número de clases, las rutas de archivos de datos y clases, y otros parámetros según nuestro conjunto de datos.
4. Compilamos el código fuente de YOLO en el sistema.
5. Dividimos el conjunto de datos etiquetado en conjuntos de entrenamiento y prueba.
6. Iniciamos el proceso de entrenamiento utilizando el conjunto de datos de entrenamiento. Esto lo hicimos mediante el comando `./darknet detector train ...`, donde especificamos la ruta al archivo de configuración y al conjunto de datos.
7. Monitorizamos el proceso de entrenamiento y ajustamos los parámetros necesarios. Los parámetros clave incluyen la tasa de aprendizaje, el número de iteraciones y la arquitectura del modelo.
8. Evaluamos el modelo utilizando el conjunto de datos de prueba. Analizamos las métricas de rendimiento, como precisión y recall.

2.2. Hardware

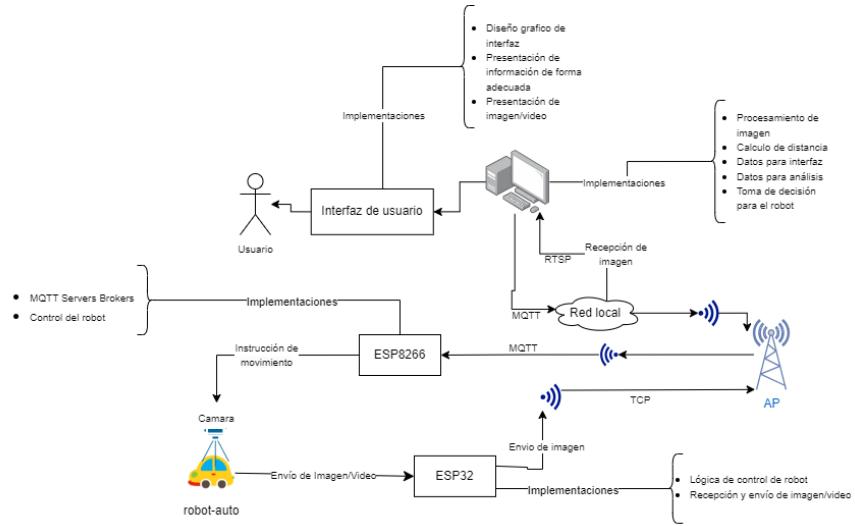


Figura 7: Esquema general del hardware del proyecto

Como se observa en el gráfico, hemos establecidos una comunicación fluida y eficiente entre la ESP32, la ESP8266 en el robot y la PC utilizando una red local configurada a través de un access point. Este access point se benefició de la zona de cobertura de Windows 11, aprovechando las capacidades de una red de 2,4 GHz. Esta elección estratégica de utilizar la banda de 2,4 GHz es fundamental para facilitar la conectividad de dispositivos específicos, como las ESP32 y la ESP8266, que operan en esta frecuencia.

La configuración de una red local proporciona una conexión estable y de baja latencia entre los componentes del sistema. Al utilizar un access point, creamos un entorno de red dedicado que permite una comunicación eficiente entre la PC y las plataformas embebidas en el robot. Esta red local resulta esencial para la transmisión de datos, comandos y la recepción de información, contribuyendo directamente a la coordinación y funcionamiento integral del vehículo autónomo.

Asimismo, la elección de la banda de 2,4 GHz se alinea con la capacidad de las ESP32 y la ESP8266 para operar en este rango de frecuencia. Este enfoque asegura una compatibilidad óptima y una conexión estable, ya que ambos dispositivos están diseñados para trabajar eficientemente en la banda de 2,4 GHz.

2.2.1. Arduino IDE

En el contexto del desarrollo del proyecto, hemos llevado a cabo una identificación y configuración de los manejadores de placas en el entorno de desarrollo integrado (IDE) de Arduino. Estos manejadores son esenciales para la correcta compilación y carga de código en las placas microcontroladoras que utilizamos, en este caso, la ESP32 y la ESP8266.

Para la ESP32, seleccionamos la placa “AI Thinker ESP32-CAM” en el IDE de Arduino. Además, utilizamos la librería de placas “esp32 by Espressif” Systems en su versión 2.0.11. Esta configuración específica garantiza que el código desarrollado sea compatible y optimizado para la ESP32. Como referencia, basamos nuestro trabajo en el ejemplo proporcionado por la librería, accesible a través de la ruta *Examples -> esp32-cam -> CameraWebServer*.

En el caso de la ESP8266, optamos por la placa “Generic ESP8266 Module” con el manejador de placas en su versión 3.1.2. La librería utilizada es “esp8266 by ESP8266 Community”, proporcionando un entorno de desarrollo robusto y actualizado para la ESP8266. Para ilustrar la configuración, nos basamos en el ejemplo

proporcionado por la librería, ubicado en *Examples ->ESP8266WIFI ->mqtt_esp8266*.

Es importante destacar que, además de la configuración de la placa, hemos incorporado librerías adicionales para el desarrollo de funciones específicas. Para la ESP8266, hemos utilizado las librerías *ESP8266WiFi.h*, *ESPPubSubClientWrapper.h*, y “*Servo.h*” para el control del auto y la implementación del servidor MQTT.

2.2.2. Protocolo MQTT

El protocolo **MQTT** (**M**essage **Q**ueuing **T**elemetry **T**ransport) ha desempeñado un papel esencial en la implementación y operación del vehículo robótico, actuando como el medio principal para la transmisión de indicaciones y comandos a través de una red local. En este proyecto, MQTT se ha adoptado como el protocolo de comunicación por sus características eficientes y su capacidad para facilitar la transferencia de mensajes en tiempo real entre el sistema central y el vehículo autónomo.

Su función principal radica en enviar indicaciones al robot, como instrucciones de movimiento, velocidad y cambios en la trayectoria, permitiendo así la operación remota y autónoma del vehículo.

La elección de MQTT sobre otros protocolos se fundamenta en varias características clave. En primer lugar, MQTT es extremadamente liviano y eficiente, lo que reduce la carga de la red y asegura una comunicación rápida y efectiva, aspecto crucial en aplicaciones en tiempo real como la conducción autónoma. Además, su arquitectura basada en el modelo de publicación-suscripción permite una fácil expansión y escalabilidad, ya que múltiples entidades pueden suscribirse y publicar información de manera simultánea sin afectar la eficiencia de la red.

La implementación de MQTT en el proyecto implica la configuración de un servidor MQTT, que actúa como intermediario entre el sistema central y el vehículo robótico. El servidor gestiona los mensajes entrantes y salientes, asegurando una comunicación confiable y oportuna. El vehículo, por su parte, está equipado con un cliente MQTT que se suscribe a los temas relevantes y responde a las indicaciones recibidas. Esta infraestructura proporciona un canal de comunicación robusto y flexible, permitiendo ajustes dinámicos en la operación del vehículo en tiempo real.

2.2.3. Decisión de utilizar TCP en capa de transporte y HTTP en capa de aplicación en lugar de RTSP: Un análisis detallado

En el proceso de desarrollo de nuestro proyecto, llegamos a una decisión clave relacionada con la selección de protocolos de comunicación. En lugar de utilizar el Protocolo de Transmisión en Tiempo Real (RTSP), optamos por la combinación de TCP en la capa de transporte y HTTP en la capa de aplicación. Esta elección se basó en varios factores que consideramos críticos para el rendimiento, la eficiencia y la robustez del sistema.

1. Estabilidad y Confiabilidad:

TCP (Transmission Control Protocol): Ofrece una comunicación confiable y sin pérdida de datos. En situaciones donde la integridad de los datos es crucial, como la transmisión de imágenes y comandos para el control de un robot, la confiabilidad de TCP es esencial.

2. Orden de Entrega:

TCP: Garantiza que los paquetes se entreguen en el orden correcto. Esta característica es vital cuando se transmiten datos que están sincronizados, como el video de una cámara en tiempo real. Mantener la secuencia correcta de los datos es esencial para una visualización coherente.

3. Manejo de Errores:

TCP: Proporciona mecanismos efectivos para la detección y corrección de errores. La detección temprana de posibles problemas de transmisión es fundamental para evitar resultados incorrectos en la aplicación, lo que podría ser crítico en el contexto de un vehículo autónomo.

4. Condiciones de Red Variables:

TCP: Funciona eficientemente en redes con variabilidad en la latencia y la pérdida de paquetes. Dada la naturaleza móvil del robot y las posibles fluctuaciones en la calidad de la señal, es esencial contar con un protocolo que se desempeñe bien en condiciones variables.

5. Compatibilidad y Accesibilidad:

HTTP (Hypertext Transfer Protocol): Es un protocolo ampliamente compatible y bien soportado por una variedad de dispositivos y entornos. Utilizar HTTP facilita la interacción con otros sistemas y tecnologías, y permite el acceso fácil a través de navegadores web estándar.

6. Facilidad de Implementación:

La implementación de TCP y HTTP es más accesible y simplificada en comparación con la complejidad asociada con RTSP, lo que resulta en un desarrollo más rápido y una mayor facilidad para el mantenimiento futuro.

Al tomar en consideración estos factores, decidimos conscientemente utilizar TCP en la capa de transporte y HTTP en la capa de aplicación para nuestro proyecto. Esta elección está respaldada por un análisis cuidadoso y la consideración de las necesidades específicas de nuestro sistema. Creemos firmemente que esta combinación proporcionará la estabilidad y el rendimiento requeridos para el flujo de datos críticos en nuestro entorno de aplicación.

2.2.4. Pruebas empíricas

En esta sección, se presenta un análisis detallado del rendimiento del robot al utilizar el protocolo de comunicación TCP para la transmisión de datos. Se llevaron a cabo pruebas exhaustivas, y los resultados demuestran la eficiencia y rapidez de la implementación con este protocolo.

Se registró el tiempo transcurrido desde que el robot ingresó al bucle principal "while true" hasta que devolvió la distancia de un objeto detectado. Este intervalo de tiempo se consideró como el tiempo total de procesamiento.

Con el protocolo TCP, el robot logró completar el procesamiento y la transmisión de datos en un tiempo promedio de 0,09 segundos. Este tiempo de ejecución rápido demuestra la eficacia de dicho protocolo en el contexto de nuestro proyecto. La baja latencia y la capacidad de TCP para manejar la transmisión de datos de manera eficiente contribuyeron significativamente a esta rapidez en el rendimiento.

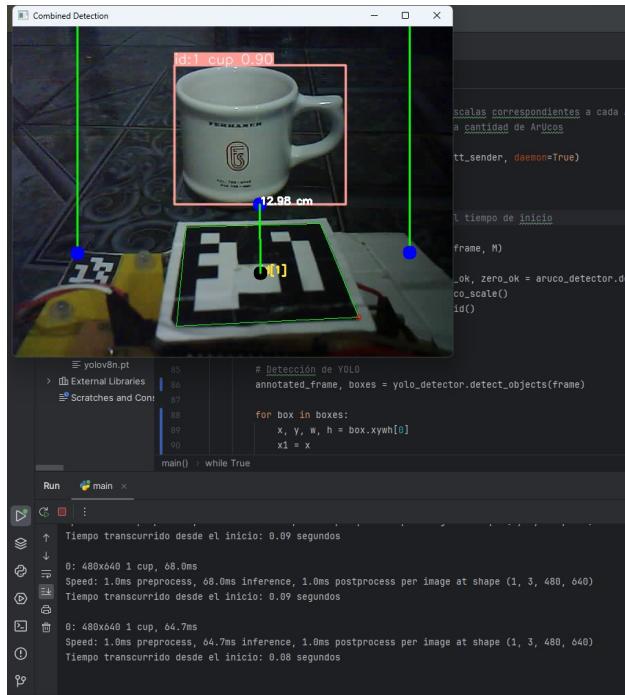


Figura 8: Tiempo de captura

3. Documentación del software

3.1. Código Principal (Main)

El código en `main.py` constituye el núcleo del sistema, integrando la detección de ArUco, YOLO y la lógica de control para evitar colisiones. A continuación, se presenta una explicación detallada:

1. Inicialización de Detectores:

Se inicializan dos detectores clave: `aruco_detector` para la detección de marcadores ArUco y `yolo_detector` para la detección de objetos mediante el modelo YOLO. La calibración de la cámara se realiza con la clase `Calibracion`.

2. Calibración de la Cámara:

Se realiza la calibración de la cámara mediante la matriz `matrix`, los coeficientes de distorsión `dist` y la matriz de perspectiva `M`. Estos parámetros son esenciales para la corrección de distorsiones y la transformación de perspectiva.

3. Inicialización de la Cámara:

Se inicia la captura de video desde la cámara especificada mediante `cv2.VideoCapture`. El streaming de video se obtiene desde una dirección IP específica (`"http://192.168.137.129:81/stream"`), que puede variar según la configuración de la red.

4. Bucle Principal del Programa:

Se inicia un bucle principal que captura y procesa cada frame de la transmisión de video en tiempo real.

5. Detección de ArUco:

Se detectan y se siguen los marcadores ArUco en el frame transformado aplicando la matriz de perspectiva. Se obtienen las coordenadas y estados de dos marcadores ArUco específicos (con IDs “ocho” y “zero”).

6. Detección de Objetos con YOLO:

Se utiliza YOLO para detectar objetos en el frame. Se obtienen las coordenadas de los bounding boxes de los objetos detectados.

7. Cálculo de Distancias y Control del Vehículo:

Se calcula la distancia entre el centro del marcador ArUco seleccionado y los objetos detectados. Esta distancia se utiliza para tomar decisiones sobre el control del vehículo, evitando colisiones.

8. Interacción Visual y de Control:

Se visualizan las distancias en el frame y se dibujan líneas entre el centro del marcador ArUco y los objetos detectados. Además, se implementa la lógica de control del vehículo (`nc.no_chocar()`) basada en la distancia actual y la posición del objeto.

9. Visualización del Frame Resultante:

Se muestra el frame resultante que combina la información de detección de ArUco y YOLO. Se dibujan líneas y se añaden círculos en los marcadores detectados.

10. Finalización del Programa:

La ejecución del programa puede ser interrumpida presionando la tecla “q”. Esto libera los recursos y cierra todas las ventanas.

Este código refleja una integración efectiva de detección de objetos y marcadores, así como la implementación de lógica de control para la navegación segura del vehículo robótico en tiempo real.

3.2. Código del modelo YOLOv8

El código en `yolo_detection.py` define la clase `YoloDetector`, que se encarga de utilizar el modelo YOLOv8 para la detección de objetos en un frame de video. A continuación, se presenta una explicación detallada:

1. Inicialización de la Clase:

Se inicializa la clase `YoloDetector` con el modelo YOLOv8. Se utiliza el paquete `ultralytics` para cargar el modelo preentrenado, cuya ruta es especificada por el parámetro `model_path`. La lista `detected_objects` se utiliza para almacenar información sobre los objetos detectados.

2. Método `detect_objects`:

Este método toma un frame como entrada y ejecuta YOLOv8 sobre él para realizar la detección de objetos. La función `self.model.track` se utiliza para realizar un seguimiento entre frames y persistir las trayectorias. Los resultados incluyen la información sobre los objetos detectados y sus trayectorias.

El método devuelve el frame resultante después de la detección y una lista de bounding boxes (`boxes`) que representan las posiciones de los objetos detectados.

3. Método `get_object_centers`:

Este método devuelve una lista de las coordenadas (centros) de los objetos detectados. Utiliza la información almacenada en `self.detected_objects`.

4. Método `get_detected_objects`:

Este método devuelve la lista de objetos detectados. La información sobre los objetos detectados se almacena en `self.detected_objects` durante el proceso de detección.

5. Visualización del Resultado:

La función `results[0].plot()` se utiliza para visualizar los resultados en el frame. Esto incluye la representación gráfica de los bounding boxes y cualquier información adicional proporcionada por el modelo.

6. Uso de YOLOv8 con Ultralytics:

El modelo YOLOv8 se carga y se utiliza para la detección de objetos. La librería `ultralytics` proporciona una interfaz conveniente para trabajar con YOLO.

3.3. Código para calibración de la cámara

Este código pertenece a la clase `Calibracion`, que se encarga de calibrar la cámara para su uso en la detección y seguimiento de objetos. A continuación, se presenta una explicación detallada del código:

1. Inicialización de la Clase:

Se inicializa la clase con algunos parámetros iniciales, como el tamaño del tablero de ajedrez (`self.tablero`), el tamaño del frame (`self.tam_frame`), y un criterio de terminación para el proceso de refinamiento de esquinas (`self.criterio`).

2. Preparación de Puntos del Tablero:

Se prepara una cuadrícula de puntos en 3D correspondientes a las esquinas del tablero de ajedrez. Estos puntos se almacenan en `self.puntos_obj`. Además, se inicializan las listas `self.puntos_3d` y `self.puntos_img` para almacenar los puntos en 3D y 2D respectivamente.

3. Método `calibracion_cam`:

Este método realiza la calibración de la cámara utilizando imágenes de un tablero de ajedrez capturadas desde diferentes perspectivas.

- Se utiliza la biblioteca `glob` para obtener la lista de nombres de archivo de las imágenes del tablero de ajedrez.
- En el bucle `for`, se lee cada imagen, se convierte a escala de grises, y se buscan las esquinas del tablero de ajedrez utilizando `cv2.findChessboardCorners`.
- Si se encuentran las esquinas, se refinan mediante `cv2.cornerSubPix` y se almacenan los puntos en las listas correspondientes.
- Las esquinas refinadas se dibujan en la imagen para verificar la detección.
- Al final del bucle, se realiza la calibración de la cámara mediante `cv2.calibrateCamera`, obteniendo la matriz de la cámara (`cameraMatrix`), los coeficientes de distorsión (`dist`), y otros parámetros de transformación.
- Se imprime la matriz de la cámara original, los coeficientes de distorsión originales y una matriz de transformación de perspectiva (`M`) obtenida mediante `cv2.getPerspectiveTransform`.

4. Matriz de Transformación de Perspectiva (`M`):

Se crea una matriz de transformación de perspectiva (`M`) utilizando puntos de origen y destino. En este caso, los puntos de origen y destino son los cuatro vértices del frame original.

5. Retorno de Parámetros:

El método devuelve la matriz de la cámara (`cameraMatrix`), los coeficientes de distorsión (`dist`), y la matriz de transformación de perspectiva (`M`).

3.4. Código para detección de ArUco

Este código pertenece a la clase `ArucoDetector`, que se encarga de detectar y analizar marcadores ArUco en un frame de video. A continuación, se presenta una explicación detallada del código:

1. Inicialización de la Clase:

Se inicializa la clase con varios atributos, como el diccionario predefinido de marcadores ArUco (`self.diccionario`), parámetros del detector (`self.parametros`), y varias variables para almacenar información sobre la detección actual.

2. Método `detect_markers`:

Este método realiza la detección de marcadores ArUco en el frame de entrada.

- Se convierte el frame a escala de grises y se utiliza `cv2.aruco.detectMarkers` para encontrar los marcadores ArUco en la imagen.

- Se realiza la estimación de la pose (posición y orientación) de cada marcador detectado utilizando `cv2.aruco.estimatePoseSingleMarkers`.
- Se extraen las coordenadas del centro del marcador (`self.aruco_center`), su identificador (`self.aruco_id`), y se realiza el cálculo de escala para un marcador específico (identificado por su id) utilizando el método `calculate_scale`.
- Se gestionan también la detección de marcadores específicos con id 0 y 8, y se actualizan variables relacionadas con estas detecciones.

El método devuelve información relevante sobre la posición de los marcadores detectados.

3. Método `calculate_scale`:

Este método calcula la escala del marcador ArUco utilizando dimensiones conocidas del mismo y las dimensiones en píxeles del marcador detectado.

Las dimensiones conocidas (`aruco_width_m`) y las dimensiones en píxeles se utilizan para calcular la escala (`self.aruco_scale`).

4. Método `calculate_distance`:

Este método calcula la distancia entre el centro de un objeto detectado y el centro del marcador ArUco utilizando la escala previamente calculada.

Se determina la longitud de una línea que conecta el centro del objeto con el centro del marcador en la imagen.

La distancia en centímetros se calcula multiplicando esta longitud por la escala.

5. Métodos Get:

Se proporcionan métodos (`get_aruco_center`, `get_aruco_id`, `get_aruco_scale`) para obtener la información almacenada sobre el marcador detectado.

3.5. Código para el funcionamiento del auto robótico

La clase `NoChocar` toma decisiones en el funcionamiento del vehículo autónomo en base a la distancia actual al objeto detectado y la coordenada *X* del centro del objeto en el marco de la imagen. A continuación, se presenta una explicación detallada:

Constantes y Atributos de Clase:

- `W_FRAME`: Ancho del frame de la imagen (480 píxeles en este caso).
- `D_MAX`: Distancia máxima permitida antes de tomar medidas para evitar colisiones.
- `dist_max`: Distancia máxima a la que el vehículo se acercará (inicializada a 5).

Método `no_chocar`:

- Este método toma decisiones en función de la distancia actual al objeto detectado y la posición horizontal del objeto en el marco de la imagen.
- Si `dist_actual` es menor o igual a la distancia máxima permitida (`D_MAX`), el vehículo toma medidas para evitar colisiones:
 - Frena y espera durante 3 segundos.
 - Determina si el objeto está a la izquierda o a la derecha de la mitad del marco.
 - Dobla a la izquierda si el objeto está a la izquierda.
 - Dobla a la derecha si el objeto está a la derecha.
- Si `dist_actual` es mayor que la distancia máxima permitida, el vehículo avanza.

3.6. Código para Control Remoto de Robot con MQTT

■ Inclusión de Librerías:

- Se importan las librerías necesarias para el proyecto.
- `ESP8266WiFi.h`: Permite al ESP8266 conectarse a una red WiFi.
- `ESPPubSubClientWrapper.h`: Proporciona una interfaz para la implementación de un cliente MQTT en el ESP8266.
- `Servo.h`: Librería para controlar servomotores.

■ Configuración de WiFi y Servidor MQTT:

- Se establecen las credenciales de la red WiFi (`ssid` y `password`) a la que se conectará el ESP8266.
- Se define la dirección del servidor MQTT (`mqtt_server`).

■ Instancia del Cliente MQTT y Variables:

- `ESPPubSubClientWrapper client(mqtt_server)`: Se crea una instancia del cliente MQTT con la dirección del servidor.
- `long lastMsg = 0`: Registra el tiempo de la última comunicación MQTT.
- `char msg[50]`: Buffer para almacenar mensajes MQTT.
- `int value = 0`: Variable de estado (potencialmente utilizada para algún propósito específico).
- `char ultimo`: Variable para realizar un seguimiento del último comando recibido.

■ Definición de Pines para Hardware:

- Se definen los pines utilizados para conectar hardware al ESP8266.
- `int servo_pin = 0`: Pin al que está conectado el servo.
- `int motor_pin_a = 5`: Primer pin de control del motor.
- `int motor_pin_b = 14`: Segundo pin de control del motor.

■ Instancia del Servo:

- Se crea una instancia del objeto Servo llamada `myservo`.
- `int angle = 0`: Variable para almacenar el ángulo actual del servo (aunque no se utiliza explícitamente en este código).

■ Funciones para Controlar el Robot:

- Funciones como `adelante()`, `atras()`, `frenar()`, `derecha()`, `izquierda()`, y `centro()` son definidas para controlar el movimiento del robot según comandos específicos recibidos a través de MQTT.

■ Configuración Inicial:

- `void setup_wifi()`: Configura la conexión WiFi del ESP8266. Se conecta a la red especificada y muestra la dirección IP asignada.

■ Manejo de Comandos MQTT:

- `void callback(char* topic, byte* payload, unsigned int length)`: Se ejecuta cuando llega un mensaje MQTT. Analiza el comando recibido y realiza la acción correspondiente, como mover el robot en una dirección específica.

■ Configuración Inicial del Programa:

- `void setup()`: En la configuración inicial, se realiza lo siguiente:

- Inicializa la comunicación serial.
- Configura la conexión WiFi.
- Asocia la función callback al cliente MQTT.
- Se suscribe a un tema MQTT llamado “inTopic”.
- Configura el servo y los pines de los motores.
- Establece el robot en un estado inicial, frenado y con el servo en posición central.

■ **Bucle Principal:**

- `void loop()`: Este método se ejecuta en un bucle infinito y es responsable de manejar eventos MQTT. Mantiene activa la conexión MQTT y gestiona los mensajes entrantes.

Este código permite controlar el robot de manera remota a través de comandos MQTT, ofreciendo una interfaz flexible y escalable para futuras ampliaciones del proyecto. La comunicación MQTT facilita la interacción y el control del robot desde cualquier dispositivo conectado a la red.

3.7. Código para el manejo de la ESP32

Este código es para el manejo de una cámara con el ESP32 para realizar streaming de video o capturar imágenes. Aquí hay una explicación detallada de las secciones más importantes:

1. Selección del Modelo de Cámara:

- Selecciona el modelo de la cámara que se está utilizando comentando/descomentando las líneas bajo “Select camera model”.

2. Configuración de Credenciales WiFi:

- Se especifican las credenciales de la red WiFi a la que se conectará el ESP32 (`ssid` y `password`).

3. Funciones de Inicialización:

- `startCameraServer()`: Inicia el servidor de la cámara para manejar las solicitudes de video e imágenes.
- `setupLedFlash(int pin)`: Configura un LED para que parpadee cuando la cámara esté activa (si se define el pin en `camera_pins.h`).

4. Configuración de la Cámara:

- Se configuran los pines de la cámara, frecuencia del reloj (`xclk_freq_hz`), tamaño del marco (`frame_size`), formato de píxel (`pixel_format`), calidad JPEG (`jpeg_quality`), entre otros.
- Se verifica la presencia de PSRAM (memoria extra) para ajustar la configuración en consecuencia.

5. Inicialización de la Cámara:

- `esp_camera_init(&config)`: Inicia la cámara con la configuración especificada.

6. Ajustes Adicionales de la Imagen:

- Se realiza un ajuste adicional a la imagen según el tipo de sensor (brillo y saturación).
- Para el formato JPEG, se limita el tamaño del marco cuando no hay PSRAM disponible.

7. Inicio del WiFi:

- Se inicia la conexión WiFi y se espera hasta que la conexión sea exitosa.

8. Inicio del Servidor de la Cámara:

- Llama a `startCameraServer()` para iniciar el servidor de la cámara y habilitar la transmisión de video.

9. Mensajes de Inicio:

- Se imprimen mensajes en la consola serial indicando que la cámara está lista y se proporciona la dirección IP para acceder a ella.

10. Bucle Principal:

- El bucle principal (`loop()`) no realiza ninguna acción y simplemente tiene un retraso de 10 segundos.
- La mayor parte de la lógica está en tareas asincrónicas manejadas por el servidor de la cámara.

Este código configura la cámara ESP32 para la transmisión de video o captura de imágenes, la conecta a una red WiFi y proporciona una interfaz para acceder a la cámara a través de un navegador web.

4. Documentación relacionada

Se muestra a continuación el vehículo robótico como fue provisto por la cátedra:

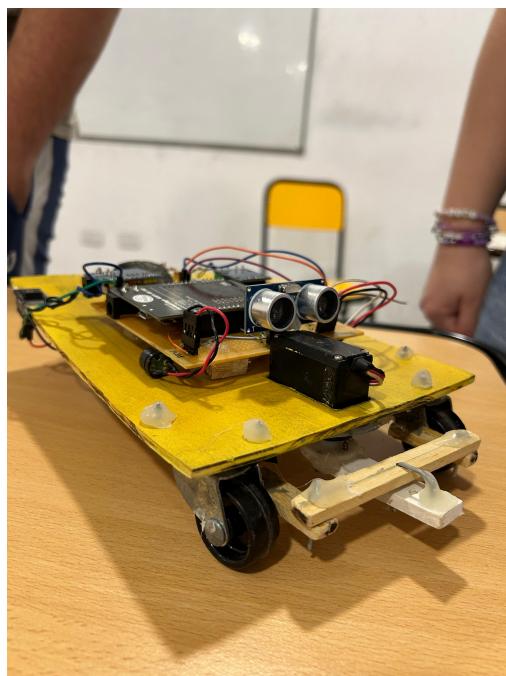


Figura 9: Vehículo robótico

Luego, se muestra a continuación el auto con las modificaciones pertinentes para que la cámara tenga una visión más precisa:

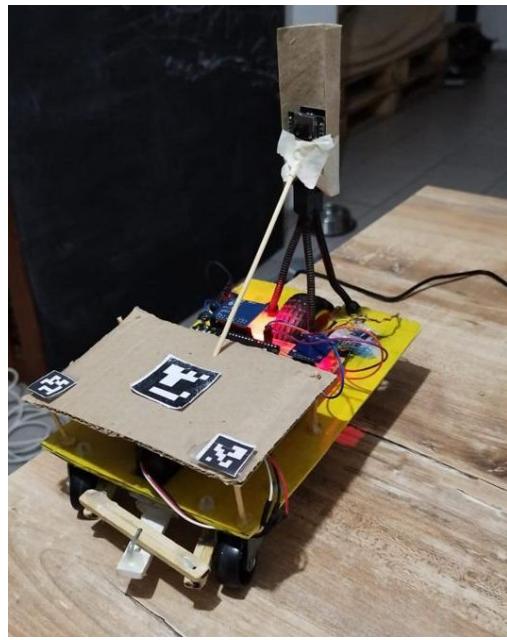


Figura 10: Vehículo robótico modificado

Se puede ver en la siguiente fotografía al robot en funcionamiento midiendo las distancias a los objetos próximos.

Aclaración: *Luego de varias pruebas empíricas realizadas se puede detectar que a partir de los 50 cm de proximidad la cámara tiene un error de 5 cm aproximadamente*

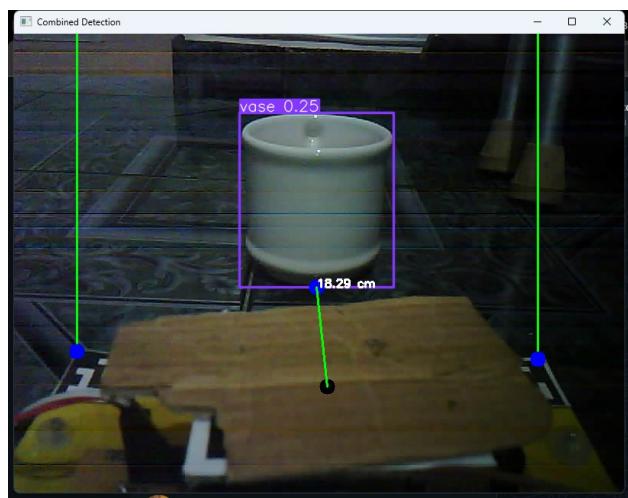


Figura 11: Vehículo robótico en funcionamiento

Por otro lado, se encuentra la interfaz web mencionada con anterioridad para la muestra de los datos capturados por la cámara:



Figura 12: Página web para visualización de datos

Se encuentra a disposición el link de los códigos fuente y de la bitácora de Github donde se ha documentado de manera cronológica los avances e investigaciones que se han llevado a cabo.

- Bitácora
- Códigos
- Video demostrativo del funcionamiento del robot

5. Conclusiones finales

El desarrollo de este proyecto de un vehículo autónomo ha sido una experiencia enriquecedora que nos ha permitido enfrentarnos a diversos desafíos y aprender de manera significativa. A lo largo del proceso, hemos consolidado nuestros conocimientos en áreas clave como visión por computadora, detección de objetos, realidad aumentada, y control de vehículos autónomos. A continuación, se destacan algunas conclusiones y reflexiones sobre nuestro trabajo:

Por el lado de los desafíos técnicos, desde la instalación de bibliotecas y entornos de desarrollo hasta la integración de diferentes módulos, enfrentamos desafíos técnicos que nos obligaron a investigar, experimentar y resolver problemas de manera creativa. La superación de estas dificultades no solo mejoró nuestras habilidades técnicas, sino que también fortaleció nuestra capacidad para abordar problemas complejos.

Más allá de la complejidad técnica, el proyecto tuvo un valor significativo para nuestro desarrollo profesional. Nos proporcionó la oportunidad de aplicar los conceptos teóricos aprendidos en un entorno práctico, lo que es fundamental para consolidar la comprensión y prepararnos para desafíos futuros en el campo de la ingeniería y la tecnología.

A su vez, el proyecto destacó la importancia de la colaboración y la comunicación efectiva en un equipo de desarrollo. La coordinación entre los miembros del equipo fue esencial para integrar de manera fluida los componentes de hardware y software, así como para compartir conocimientos y experiencias.

Reconocemos que el proyecto es un punto de partida y que hay espacio para futuras mejoras y expansiones. La transición de la cámara de la notebook a la ESP32, el entrenamiento del modelo YOLO con elementos específicos del entorno y la integración completa con la plataforma web son áreas que planeamos abordar en el futuro.

6. Ápendice A: Materiales y Presupuestos

Se incluye a continuación una lista de todos los materiales utilizados en el proyecto junto con sus costos y referencia a los sitios de donde se adquirieron:

Hardware utilizado					
Nombre	Código	Cantidad	Disponible en	Costo	DataSheet
Cámara ESP32	Ov2640	1	Link a Mercado Libre	\$14000	Link a datasheet
Servomotor		1	Link a Mercado Libre	\$3000	Link a datasheet
Ruedas		3	Link a Mercado Libre	\$1500	Link a datasheet
Motor	MT01	1	Link a Mercado Libre	\$3000	Link a datasheet
ESP8266WIFI	ESP6266EX	1	Link a Mercado Libre	\$2500	Link a datasheet

Tabla 1: Tabla de material utilizado