# UNIVERSIDAD MAYOR, REAL Y PONTIFICIA DE SAN FRANCISCO XAVIER DE CHUQUISACA

FACULTAD DE CIENCIAS Y TECNOLOGÍA


## SISTEMA DE RECICLAJE INTELIGENTE

INFORME DE TRABAJO REALIZADO

**Universitarios:**

| **Nombre:** | **Carrera:** | **C.U:** |
|---|---|---|
| ▪Calderón Flores Enrique Antonio | Ing. En Ciencias de la Computación | 111-397 |
| ▪Lopez Chavez Pablo | Ing. En Ciencias de la Computación | 111-386 |

**Materia**: Sistemas de tiempo real (COM460)
**Docente:** Ing. Juan Marcelo Arancibia Rodriguez

**Sucre-Bolivia**
**2024**

# Tabla de Contenidos

# Tabla de Figuras

# Índice de Tablas

# a. Introducción

El reciclaje se ha convertido en una prioridad global para mitigar el impacto ambiental de los desechos. En este contexto, se desarrolló un sistema de reciclaje inteligente que integra tecnologías de visión artificial e IoT, diseñado para optimizar la clasificación y manejo de residuos. Este sistema, implementado en forma de un basurero automatizado, emplea un modelo preentrenado y aplicando Transfer Learning Faster R-CNN para identificar y clasificar desechos reciclables, permitiendo únicamente el ingreso de botellas reciclables mientras descarta otros tipos de basura.

Además de albergar el modelo de reconocimiento, el sistema incluye un módulo de monitoreo basado en websockets, lo que amplía su funcionalidad para supervisar en tiempo real los datos de los sensores integrados. Este monitoreo permite, por ejemplo, visualizar el nivel de llenado del basurero, detectar el ingreso de nuevos objetos y mostrar la clasificación que se generó para los objetos.

Asimismo, se implementó un sistema de graficación histórica que almacena y analiza datos recopilados en las detecciones. Esto proporciona información sobre patrones de uso y clasificación de residuos, lo que facilita la toma de decisiones para optimizar su operación y mantenimiento.

Con esta solución, no solo se logra una clasificación automatizada de residuos, sino también un sistema integral de gestión y monitoreo que promueve prácticas sostenibles y mejora la eficiencia en el manejo de desechos reciclables.

# b. Desarrollo

## 1. Adaptaciones realizadas al Basurero

Como primer paso, se realizo las adaptaciones requeridas para que el basurero pueda funcionar con el diseño propuesto para lograr el objetivo de reciclaje. Las adaptaciones que se realizaron fueron:

### 1.1 Plataforma de decisión

Esta plataforma fue integrada al basurero para que sea controlada por un servomotor. Cuando una persona deposite una basura, esta caerá en dicha plataforma, el servo cumple la función de reflejar la decisión tomada por el sistema, girando en un sentido permitiendo el deposito en el basurero, o por el contrario, descartándolo girando en otro sentido hacia el basurero secundario de no reciclables.


*Figura 1: Servomotor MG995 (Fuente: https://tienda.sawers.com.bo)*

Para lograr esto se necesita un servomotor que tenga la fuerza suficiente para soportar la interacción con las personas y el ingreso de los desechos.

Se eligió el servomotor **MG995** porque tiene un torque de 15KG/cm lo cual es suficiente para este caso, y además funciona con 5 voltios, que es el voltaje en general utilizado por el Arduino y el circuito.


*Figura 2: Servomotor integrado al basurero de reciclaje (Fuente: Elaboración Propia)*


*Figura 3: Plataforma de Decisión Controlada por Servomotor (Fuente: Elaboración Propia)*

### 1.2 Implementación de sensores de distancia como sensores de nivel

Como siguiente paso se procedió a implementar el subsistema que se encargará de medir el nivel de desechos reciclables que se encuentran en el basurero.

*Figura 4: Sensor Ultrasónico HC-SR04 (Fuente: https://tienda.sawers.com.bo)*

Para lograr esto se utilizó 2 sensores ultrasónicos:

- Sensor de nivel medio: Que indica que el basurero llegó a su mediana capacidad
- Sensor de nivel alto: Que indica que el basurero alcanzó a su máxima capacidad

Se colocó este par de sensores de forma estratégica al interior del basurero para asegurar una correcta lectura de ambos, logrando así que no perjudique el ingreso de botellas al basurero y obteniendo lecturas correctas de ambos sensores.

Se eligió los sensores ultrasónicos HC-SR04 para este caso, debido a que a pesar de ser económicos, funcionan correctamente en este caso, ya que al mantenerlos estáticos en una posición estratégica, al onda de vuelta que requieren para medir la distancia llega siempre de forma constante y correcta.


*Figura 5: Sensores Ultrasónicos Vista lateral (Fuente: Elaboración Propia)*


*Figura 6: Sensores Ultrasónico Vista Frontal (Fuente: Elaboración Propia)*

### 1.3 Exclusa de descarte

Como tercer paso en el desarrollo del sistema, se implementó la exclusa de descarte. Esta componente tiene como propósito principal servir de conducto entre el basurero principal de reciclaje y un contenedor dedicado a los desechos no reciclables. De esta forma, cuando el servomotor gire en la dirección asignada para descartar un desecho, este será canalizado hacia la exclusa, asegurando su correcta disposición en el contenedor correspondiente.

Para su implementación, se realizó una apertura en uno de los laterales del basurero, diseñada como una "ventana". Además, se añadió una rampa que, en conjunto con el movimiento del servomotor, guía los desechos de manera eficiente hacia el contenedor de desechos no reciclables. Esta integración asegura un proceso ordenado y preciso para la separación de materiales no reciclables, optimizando la funcionalidad del sistema.

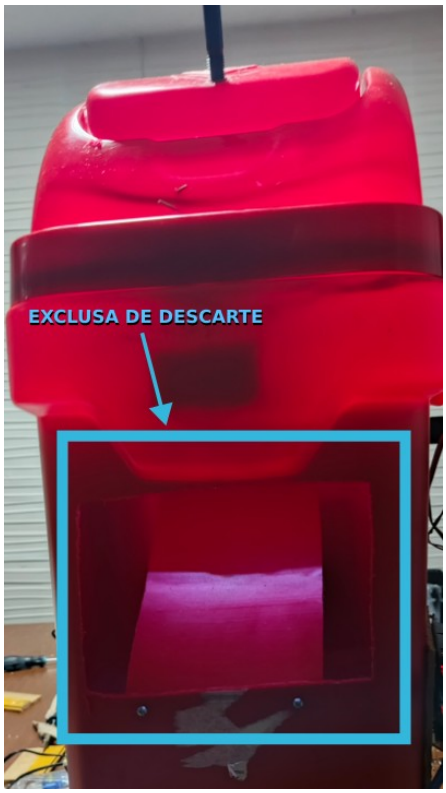


*Figura 7: Exclusa de Descarte (Fuente: Elaboración Propia)*


*Figura 8: Exclusa de descarte acoplada a contenedor de desechos no reciclables (Fuente: Elaboración propia)*

### 1.4 Posicionamiento de cámara ESP32 AI Thinker CAM

Como cuarto paso, se realizó el posicionamiento de la cámara ESP32 AI Thinker CAM en una ubicación estratégica que garantizara una visión completa de los objetos ingresados a la plataforma de decisión. La correcta ubicación de la cámara era crítica, ya que permanecerá estática y, si no se posicionaba adecuadamente, no podría capturar las imágenes necesarias para la detección de objetos de forma precisa.

Para determinar el lugar óptimo, se llevaron a cabo múltiples pruebas en diferentes posiciones, descartando aquellas que no abarcaban completamente el área de reconocimiento. Finalmente, se eligió una posición en un ángulo bajo, en uno de los laterales de la plataforma de decisión. Desde este ángulo, la cámara captura imágenes desde una perspectiva inclinada, lo que, aunque no sea frontal, asegura una cobertura total del área de reconocimiento.

Aunque inicialmente se consideró colocar la cámara en la parte superior del basurero para obtener una vista más directa, esta posición no permitía cubrir todo el espacio de reconocimiento. Sin embargo, esto no representa un inconveniente, ya que el modelo Faster R-CNN fue entrenado utilizando imágenes tomadas desde la posición final de la cámara. Esto garantiza que el modelo sea capaz de identificar tanto botellas reciclables como desechos no reciclables desde el ángulo específico en el que se encuentra la cámara.

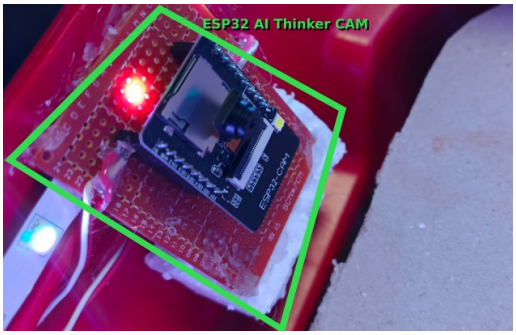


*Figura 9: ESP32 AI Thinker CAM posicionada dentro del contenedor principalt (Fuente: Elaboración Propia)*

*Figura 10: Ejemplo de Captura de Cámara (Fuente: Elaboración Propia)*


*Figura 11: ESP32 AI Thinker CAM mostrando ángulo de captura (posicionamiento) (Fuente: Elaboración Propia)*

### 1.5 Iluminación Interna

Dado que el basurero está diseñado para funcionar tanto de día como de noche, mantener una iluminación equilibrada y constante en su interior es esencial para garantizar una detección precisa de los objetos. Esto asegura que, sin importar las condiciones de iluminación del entorno, el modelo de IA pueda identificar con mayor facilidad los objetos ingresados.

Para lograrlo, se instaló una tira de LEDs en el contorno de la plataforma de decisión, el área donde se realiza la detección de objetos. Esta solución proporciona una iluminación uniforme que mejora significativamente la visibilidad y optimiza el rendimiento del modelo de reconocimiento.


*Figura 12: Tira de LEDs de iluminación interna y constante para la detección de objetos (Fuente: Elaboración Propia)*

### 1.6 Detector de ingreso de desechos

Para detectar cuando una persona introduce desechos en el basurero inteligente, se implementó un subsistema específico utilizando un sensor ultrasónico adicional (independiente de los sensores utilizados para medir el nivel del contenedor). Este sensor ultrasónico cuenta con una pequeña tablilla que bloquea su visión y mantiene su distancia de detección en un valor fijo de 2 "unidades".


*Figura 13: Detector de ingreso de desechos, vista cercana (Fuente: Elaboración Propia)*


*Figura 14: Detector de Ingreso de desechos, vista completa (Fuente: Elaboración Propia)*

Cuando se abre la tapa del basurero, la tablilla se desbloquea, permitiendo al sensor medir una nueva distancia. Si esta nueva medición difiere de la distancia bloqueada, el sistema interpreta que se está ingresando un desecho.

Esta solución fue clave para optimizar el funcionamiento del sistema, ya que evita que la cámara esté activa constantemente. De esta manera, se reduce el consumo eléctrico y se prolonga la vida útil de la cámara al activarla únicamente cuando es necesaria. Esta estrategia aumenta significativamente la eficiencia general del sistema.

### 1.7 Contrapeso

Para garantizar el correcto funcionamiento del detector de ingreso de desechos, era fundamental lograr un cierre rápido y efectivo de la tapa del basurero. Para ello, se implementó un contrapeso cuidadosamente colocado, lo suficientemente pesado como para asegurar un cierre preciso y veloz tras el ingreso de un desecho, pero sin ser tan pesado que dificulte o incomode a las personas al abrir la tapa.


*Figura 15: Contrapeso (Fuente: Elaboración Propia)*

Este equilibrio en el peso del contrapeso es clave para mantener tanto la eficiencia del sistema como la comodidad del usuario, asegurando una experiencia fluida y práctica en el uso del basurero inteligente.

### 1.8 Ubicación del circuito

Se seleccionó cuidadosamente la ubicación del circuito, teniendo en cuenta dos factores principales: que no interfiera con la interacción del usuario y que permita el fácil acceso de los cables provenientes del interior del basurero hacia el circuito. Para ello, se eligió uno de los laterales del basurero, a media altura, como el lugar más adecuado.

Todos los cables provenientes de los sensores, la cámara y la tira de LEDs ubicados en el interior del basurero se organizaron y conectaron mediante una placa perforada de circuitos electrónicos. Esta placa actúa como intermediaria, facilitando el cableado y las conexiones hacia el Arduino.

El uso de esta placa perforada resultó esencial, ya que el Arduino no tiene conexiones fijas (los componentes no se sueldan directamente a él), lo que podía generar problemas de estabilidad en las conexiones. Al soldar los cables a la placa, se logró una estructura firme y ordenada, permitiendo una conexión más segura y eficiente entre los sensores, los actuadores y el Arduino.

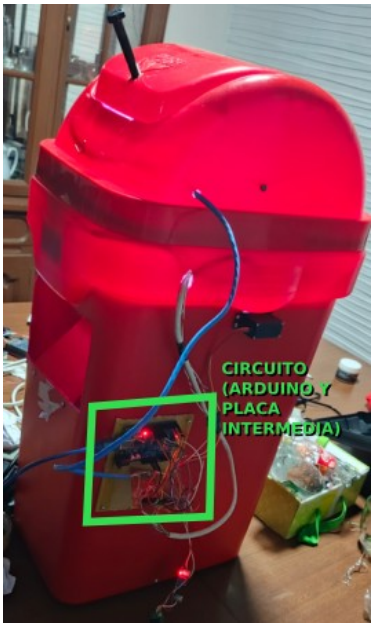


*Figura 16: Ubicación y acople de los sensores mediante placa perforada intermedia hacia Arduino UNO (Fuente: Elaboración Propia)*

### 1.9 Fuente de alimentación

Finalmente, para la alimentación del basurero inteligente, se diseñó un sistema modular y sencillo que pudiera satisfacer las necesidades de los diferentes componentes del circuito, los cuales requieren dos niveles de voltaje:

- 12 voltios para alimentar la tira de LEDs.
- 5 voltios para el resto de los componentes (servomotores, sensores, Arduino y ESP32).

*Figura 17: Regulador LM2596 (Fuente: https://tienda.sawers.com.bo/)*

Se optó por utilizar una fuente de alimentación de 12 voltios y 2 amperios, con formato de cargador, debido a su capacidad para proporcionar la potencia necesaria de forma estable. A la salida de esta fuente se agregó un adaptador con borneras, que facilita la conexión tanto a la tira de LEDs como a un regulador LM2596 tipo step-down. Este regulador convierte los 12 voltios en los 5 voltios requeridos por los demás componentes del circuito y además proporciona como máximo 2 Amperios.

Gracias a este diseño, fue posible utilizar una sola fuente de alimentación para generar ambos niveles de voltaje necesarios, simplificando la instalación y asegurando un suministro confiable y eficiente para todo el sistema.


*Figura 18: Sistema de alimentación del basurero inteligente*

## 2. Detección de Objetos (Bounding Boxes)


*Figura 19: LabelStudio (Fuente: Aplicación Label Studio corriendo en Docker)*

Para la detección de objetos, se utilizó la herramienta Label Studio, que facilita la creación de bounding boxes en las imágenes que se emplearán para entrenar el modelo. Esta herramienta también permite exportar los datos en formato XML VOC, lo que facilita su integración con el modelo Faster R-CNN.

En este proyecto, se etiquetaron 100 imágenes de botellas y 100 imágenes de papeles. Las botellas fueron clasificadas como reciclables, mientras que los papeles representaron los desechos no reciclables que el sistema debe descartar. Aunque el enfoque actual se centra en estos dos tipos de desechos, se dejó la posibilidad abierta de incorporar más categorías de basura en el futuro, aunque esto queda fuera del alcance del proyecto actual.

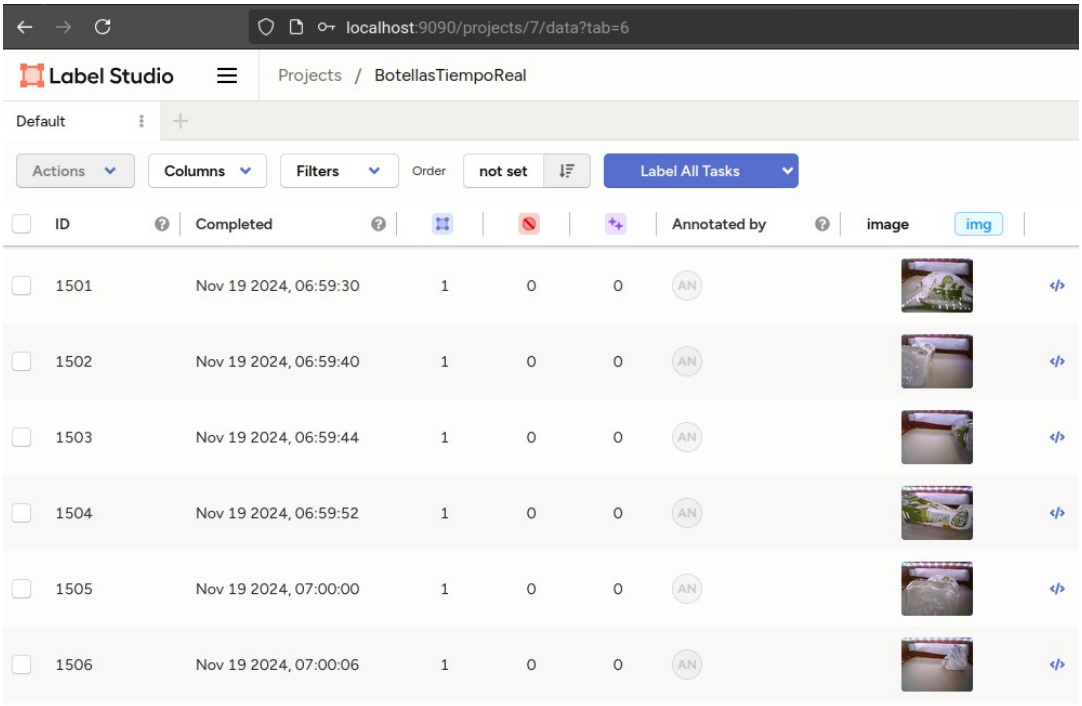


*Figura 20: Interfaz de Proyecto de Label Studio (Fuente: Elaboarción Propia)*

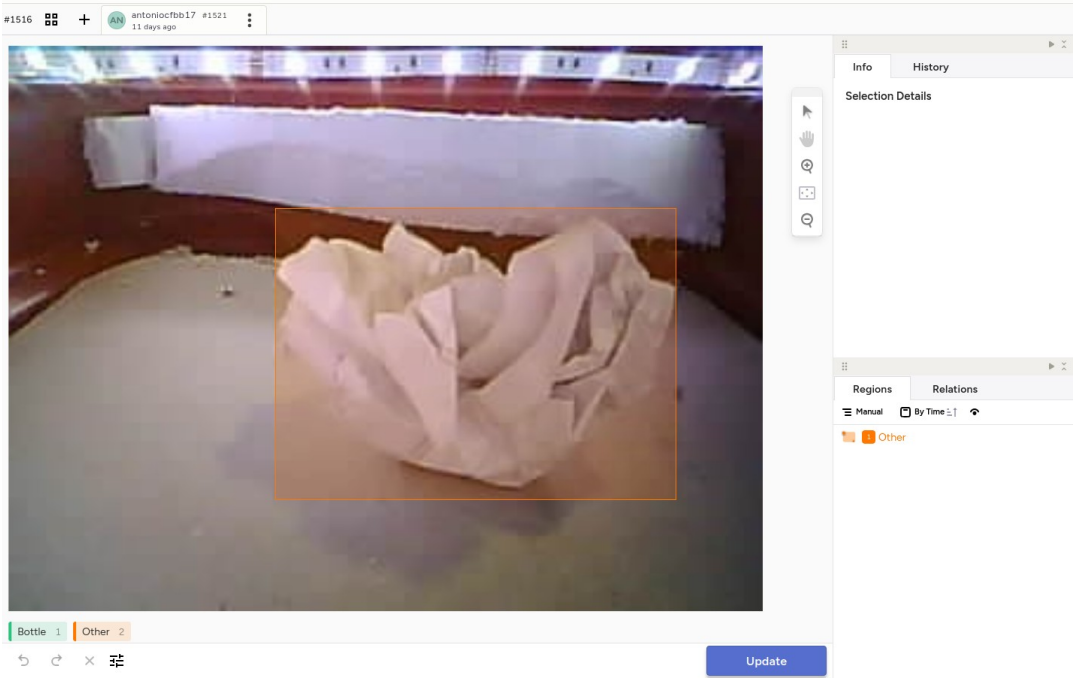


*Figura 21: Ejemplo de segmentacion de desechos descartables en Label Studio (Fuente: Elaboración Propia)*

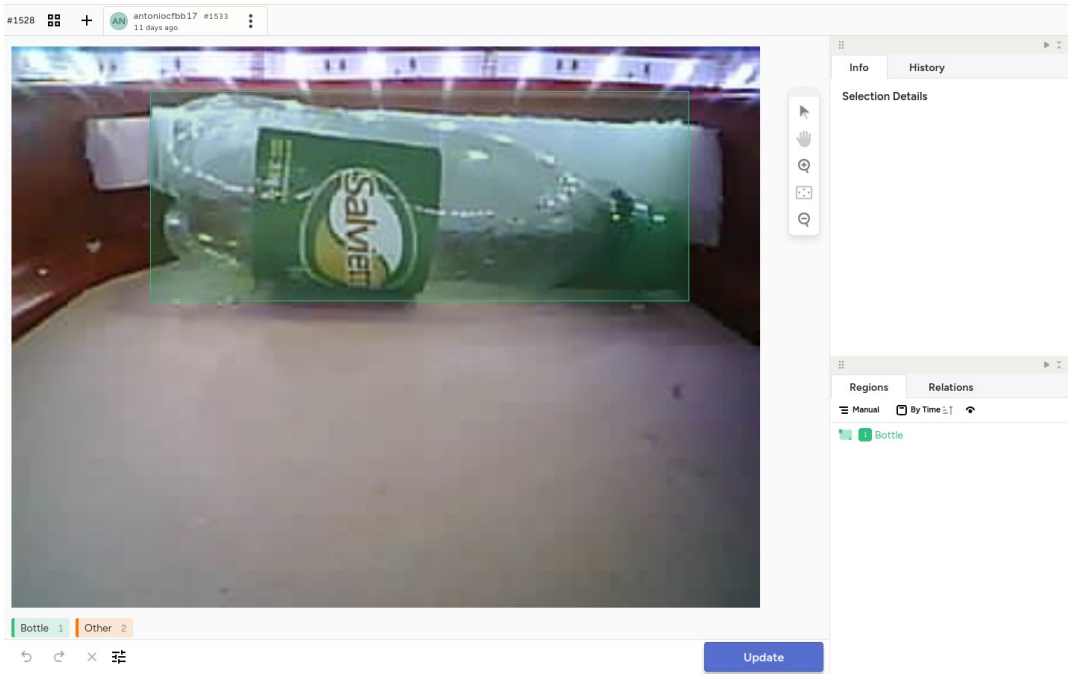


*Figura 22: Ejemplo de segmentacion de desechos reciclables (botellas) en Label Studio (Fuente: Elaboración Propia)*

## 3. Entrenamiento de Modelo con FasterRCNN

Una vez que los bounding boxes fueron colocados en todas las imágenes, se procedió al entrenamiento del modelo. El cuadernillo utilizado para este proceso será compartido en un repositorio de GitHub para su exploración. Los pasos seguidos en el entrenamiento del modelo fueron los siguientes:

1. **Configuración del entorno:** Se creó un entorno de Python con las dependencias necesarias, incluyendo bibliotecas como PyTorch, sklearn, Faster R-CNN, matplotlib, entre otras.

2. **Conversión de los datos:** Los datos exportados por Label Studio fueron convertidos a tensores y diccionarios. Para trabajar con Faster R-CNN y PyTorch, es necesario que todos los datos estén en formato tensor, y la salida debe ser un tensor dentro de un diccionario.

3. **Definición de las clases de salida:** Se definieron las nuevas clases que reemplazarían a las predeterminadas del modelo preentrenado:

    - 1: Bottle (reciclables)
    - 2: Paper (no reciclables, para ser descartados)

4. **Creación del dataset personalizado:** Se creó un dataset personalizado que incluyó nuestras imágenes y clases. Este dataset fue separado en batches utilizando data loaders, y se dividió en entrenamiento y validación. No se utilizó un conjunto de prueba debido a la limitación de imágenes, y la validación se realizó directamente con pruebas en vivo usando la aplicación del modelo.

5. **Definición del modelo:** Se creó una clase que define el modelo y su utilización en el proceso de entrenamiento. Esto incluyó la carga del modelo preentrenado, la sustitución de las clases de salida por las nuevas, la configuración de los epochs, el optimizador, entre otros.

6. **Entrenamiento del modelo:** El modelo fue entrenado durante 50 epochs, verificando la pérdida (loss) a lo largo del proceso. Al alcanzar los 50 epochs, se obtuvo una baja pérdida y una alta precisión en el modelo.

```python
# Crear el modelo Faster R-CNN con 3 clases (perro, gato y fondo)
model = get_object_detection_model(num_classes = 3, feature_extraction=True)  # Usar extracción de características

# Usar el optimizador de gradiente estocástico
params = [p for p in model.parameters() if p.requires_grad]
# optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9, weight_decay=0.0005)
optimizer = torch.optim.Adam(params, lr=0.001)


# Entrenar el modelo durante 10 épocas
log = train_fasterrcnn(model=model,
                       optimizer=optimizer,
                       n_epochs=5,  # Aumentar el número de épocas
                       train_loader=train_dl,
                       test_loader=val_dl,
                       log=None,
                       keys=None,
                       device=device)
```

*Figura 23: Fragmento para entrenamiento del modelo usando Transfer Learning con Faster RCNN (Fuente: Elaboración propia)*

7. **Exportación del modelo:** Una vez completado el entrenamiento, el modelo fue exportado para su uso posterior. En este caso, el modelo entrenado fue integrado en el servidor Flask para la detección de objetos en el basurero inteligente.


*Figura 24: Modelo Exportado (Fuente: Elaboración Propia)*

## 4. Diseño del circuito

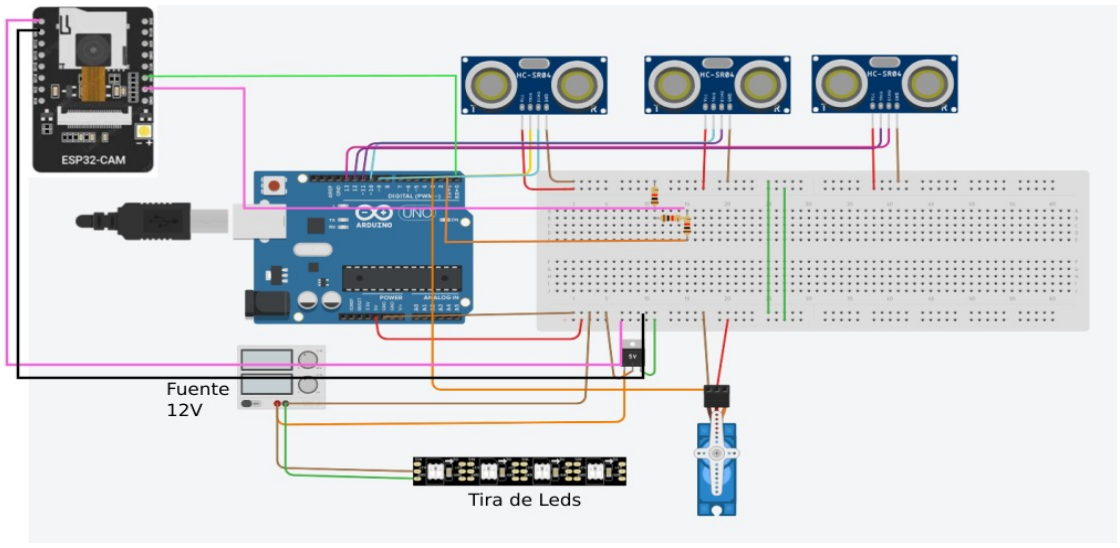


*Figura 25: Vista - Circuito (Fuente: Elaboración Propia)*

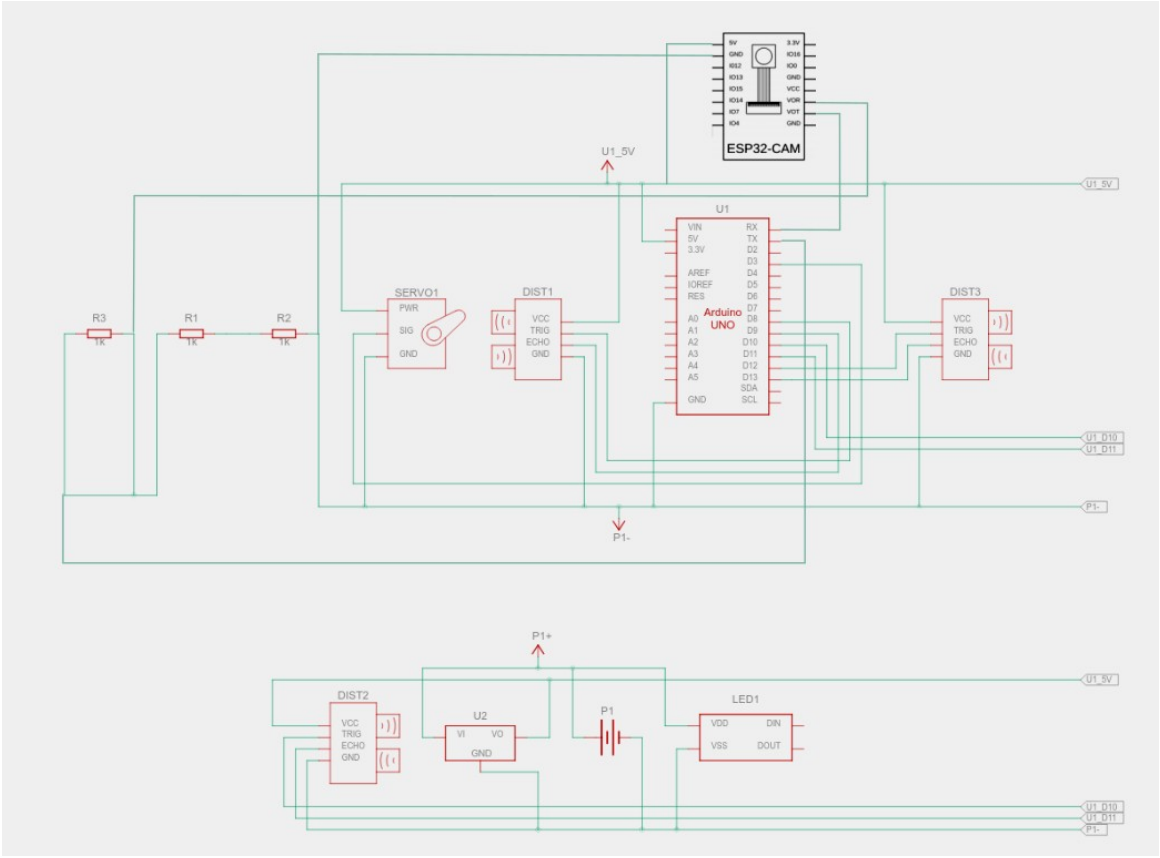


*Figura 26: Vista - Esquemático (Fuente: Elaboración Propia)*

### 4.1 Componentes

En las secciones anteriores ya se explicó el rol de cada uno de estos componentes. En esta sección, se presentan los pines utilizados para las conexiones y un resumen consolidado de todos los componentes.

El circuito consta de los siguientes componentes:

- 3 Sensores ultrasonido HC-SE04
- 1 servomotor MG995
- ESP32 AI Thinker Cam
- Arduino Uno
- Regulador de Voltaje  LM2596
- Tira de Leds de 12V
- Fuente de alimentación de 12v 2 A.
- 3 Resistencias

## 5. Comunicación entre Componentes del Sistema: Arduino, ESP32 y Servidor Flask

Este proyecto consta de tres partes principales que deben comunicarse entre sí: dos microcontroladores (MCU) y una computadora que ejecuta el servidor para monitoreo y detección de objetos. Los microcontroladores son:

- Arduino UNO
- ESP32 AI Thinker CAM

La computadora ejecuta el servidor Flask, que se conecta con el frontend mediante WebSockets.

Para la comunicación entre el Arduino y el ESP32, se utilizó comunicación serial. En este caso, el pin RX del Arduino recibe señales del pin TX del ESP32, y el pin RX del ESP32 recibe señales del pin TX del Arduino. Sin embargo, debido a que el ESP32 funciona con 3.3V, a pesar de alimentarse con 5V, se implementó un divisor de tensión con tres resistencias de 1kΩ para reducir la tensión de la señal que llega al ESP32, de modo que reciba correctamente la señal del Arduino sin dañarse.

El ESP32 AI Thinker CAM, además de capturar las imágenes, actúa como intermediario entre el Arduino y el servidor de monitoreo. Recibe mensajes en serie del Arduino para tomar fotografías y actualiza el sistema de monitoreo con el estado de los sensores de nivel medio, nivel alto y detección de ingreso de desechos. Gracias a su módulo WiFi integrado, el ESP32 envía las fotografías capturadas al servidor Flask mediante consultas HTTP POST, mientras que cada vez que un sensor cambia de estado, el ESP32 envía consultas HTTP GET al servidor para actualizar el estado de los sensores en tiempo real mediante WebSockets.

De esta manera, dependiendo del mensaje recibido a través de la comunicación serial desde el Arduino, el ESP32 decide si debe tomar una fotografía o enviar una actualización del estado de los sensores al servidor.

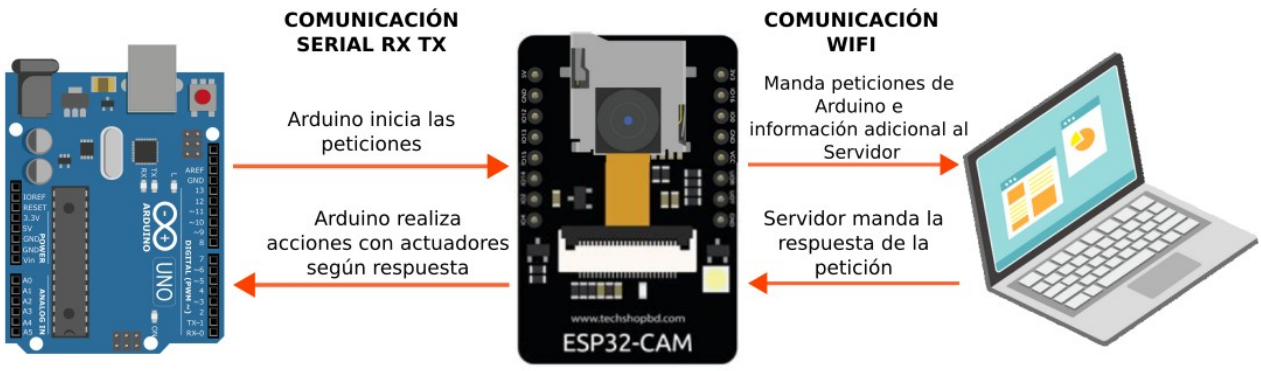


*Figura 27: Resumen del flujo de comunicación del proyecto (Fuente: Elaboración Propia)*

## 6. Servidor Flask Para detección y Monitoreo

Para el sistema de monitoreo se eligió **Flask**, ya que es una herramienta muy versátil y adecuada para proyectos pequeños y medianos. Además, al estar basado en Python, facilita la integración de modelos de inteligencia artificial, así como el manejo de WebSockets para el sistema de monitoreo.
El servidor cumple dos funciones principales:

### 6.1 Recibir fotografías y enviar predicciones

La función principal del servidor es realizar las predicciones de las fotografías que toma el ESP32. Estas imágenes se reciben mediante una solicitud HTTP POST en formato RAW, y el servidor devuelve la predicción al ESP32, que luego la envía al Arduino a través de comunicación serial. Esta parte es crítica para el funcionamiento del proyecto, ya que la clasificación de las imágenes con la cámara y el modelo de IA es el componente central del sistema.

### 6.2 Sistema de monitoreo

La segunda función del servidor es mostrar el sistema de monitoreo, que permite conocer en todo momento el estado del sistema sin necesidad de recargar las páginas, gracias a la integración de WebSockets. El sistema de monitoreo consta de dos páginas:

1. **Monitoreo principal**: Esta página permite saber cuándo una persona ha ingresado un objeto, si el sistema está en proceso de detección, el nivel del tanque (lleno, medio o vacío) y visualizar las fotografías de las predicciones realizadas.

2. **Gráficas**: Todas las fotografías y detecciones se guardan en una base de datos, lo que permite llevar un registro de todo lo que ha sido ingresado al basurero. Este registro es muy importante para la toma de decisiones en el mantenimiento del sistema de monitoreo. Además, nos proporciona información útil sobre qué días y en qué horarios se utilizan más el basurero, lo cual nos permite planificar con anticipación los vacíos y la recolección cuando el basurero se llene.


*Figura 28: Sistema de monitoreo, nivel de contenedor bajo – realizando detección (Fuente: Elaboración Propia)*


*Figura 29: Sistema de monitoreo, nivel de contenedor medio – detección realizada (Fuente: Elaboración Propia)*


*Figura 30: Sistema de monitoreo, nivel de contenedor alto – detección realizada (Fuente: Elaboración Propia)*

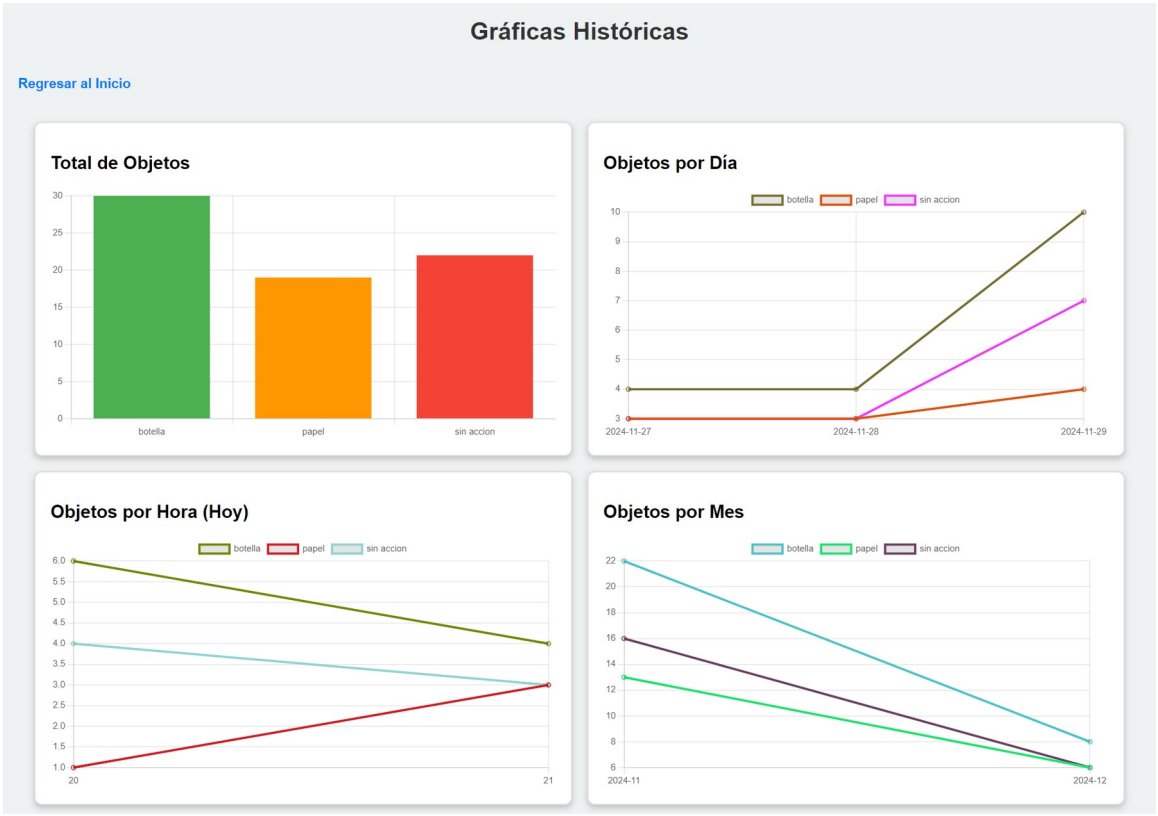


*Figura 31: Sistema de monitoreo - Gráficas históricas (Fuente: Elaboración Propia)*

## 7. Rutina de trabajo del proyecto

Una vez explicados todos los componentes y las funciones que cumple cada subsistema, a continuación se presenta la rutina de trabajo en conjunto de todas las partes:

1. **Verificación del nivel del contenedor**
   Se verifica el nivel del contenedor mediante los dos sensores ultrasónicos correspondientes. Si el contenedor está lleno, la rutina se detiene y no permite realizar ninguna acción hasta que el contenedor sea vaciado.

   Si el contenedor no está lleno, se procede al siguiente paso.

2. **Detección de ingreso de desechos**
   El detector de ingreso de desechos identifica cuando se introduce un nuevo objeto en el basurero. Una vez detectado el ingreso, espera a que la tapa, impulsada por el contrapeso, se cierre completamente antes de avanzar al siguiente paso.

3. **Captura de fotografía**
   La cámara toma una fotografía tras recibir la señal correspondiente desde el Arduino. Esta imagen es enviada al servidor mediante una consulta HTTP POST.

4. **Procesamiento en el servidor**
   El servidor recibe la fotografía, procesa la imagen con el modelo de IA y devuelve el resultado de la detección al ESP32.

5. **Recepción y envío de predicción**
   El ESP32 recibe la predicción del servidor y la envía al Arduino mediante comunicación serial.

6. **Acción del actuador**
   El Arduino interpreta la respuesta del ESP32 y, según el mensaje, acciona la plataforma de decisión con el servomotor:

   - Si el objeto detectado es una **botella** (reciclable), la plataforma gira para dejarlo ingresar al contenedor principal del basurero.
   - Si el objeto detectado es un **papel** (u otro desecho no reciclable), la plataforma gira en sentido contrario, descartando el objeto a través de la esclusa de desecho, donde cae en el contenedor de no reciclables.

7. **Reinicio del ciclo**
   La rutina vuelve al estado inicial, quedando en espera para detectar el ingreso de un nuevo objeto.

## 8. Tablas

A continuación se presentan las tablas de tiempos realizadas según el funcionamiento del proyectos

*Tabla 1: Flujo de funcionamiento y Tiempos de ejecución*

| CODE | TAREA | PERIODO ACTIVACION | TIEMPO MAXIMO DE EJECUCION | PLAZO MAXIMO DE TERMINACION | TIEMPO DE RESPUESTA MAXIMO | PRIORIDAD DE TAREA | DESFASE RESPECTO AL MOMENTO INICIAL |
|---|---|---|---|---|---|---|---|
| 1 | Solicitar lectura del sensor de distancia 1 STR-Sensor | 1s | 300ms | 500ms | 200ms | 2 | - |
| 2 | Respuesta de lectura del sensor de contenedor 1 STR - Sensor | 1s | 1ms | 200ms | 300ms | 2 | - |
| 3 | Solicitar lectura del sensor de distancia 2 STR-Sensor | 1s | 300ms | 200ms | 200ms | 1 | - |
| 4 | Respuesta de lectura del sensor de contenedor STR - Sensor | 1s | 300ms | 200ms | 300ms | 1 | - |
| 5 | Solicitar lectura del sensor de distancia compuerta STR-Sensor | 1s | 300ms | 300ms | 400ms | 1 | - |
| 6 | Respuesta de lectura del sensor de compuerta STR - Sensor | 1s | 600ms | 400ms | 200ms | 1 | - |
| 7 | Activar cámara STR - Cámara | - | - | 100ms | 200ms | 0 | - |
| 8 | Mandar Respuesta de lectura de cámara 1 STR - Cámara | - | - | 100ms | 200ms | 1 | - |
| 9 | Mandar Respuesta de lectura de cámara 2 STR - Cámara | - | - | 100ms | 200ms | 1 | - |
| 10 | Respuesta de Servidor Flask | - | - | 500ms | 1s | 1 | - |
| 11 | Activar servomotor para abrir esclusa STR - Servomotor esclusa | - | - | 600ms | 1s | 0 | - |

*Tabla 2: Rangos de Funcionamiento*

| No | Nombre | Máximo | Mínimo |
|---|---|---|---|
| A1 | Sensor Compuerta | 4 u. | 0 u. |
| B1 | Sensor Capacidad | 21 u. | 16 u. |
| B2 | Sensor Capacidad | 21 u. | 16 u. |
| C1 | Servomotor | 120 ° | 0 ° |

*Tabla 3: Pruebas de funcionamiento (tomas de tiempo)*

| Nro | CODIGO | NOMBRE | FECHA | HORA | TIEMPO DE EJECUCIÓN | DIFERENCIA DE TIEMPO | LECTURA | ALERTA |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Solicitar lectura Sensor Capacidad 1 | 19/11/2024 | 0.93 | 600ms | "0ms" | 2 | 0 |
| 2 | 2 | Solicitar lectura Sensor Capacidad 1 | 19/11/2024 | 0.89 | 600ms | "0ms" | 4 | 1 |
| 3 | 3 | Solicitar lectura Sensor Capacidad 1 | 19/11/2024 | 0.93 | 600ms | "200ms" | 22 | 1 |
| 4 | 4 | Solicitar lectura Sensor Capacidad 1 | 19/11/2024 | 0.89 | 400ms | "100ms" | 26 | 1 |
| 5 | 5 | Solicitar lectura Sensor Capacidad 2 | 19/11/2024 | 0.93 | 600ms | "0ms" | 21 | 0 |
| 6 | 6 | Solicitar lectura Sensor Capacidad 2 | 19/11/2024 | 0.89 | 400ms | "-200ms" | 22 | 0 |
| 7 | 7 | Activar Servomotor 1 | 9/11/2024 | 0.93 | 200ms | "100ms" | 50 | 1 |
| 8 | 8 | Activar Servomotor 1 | 9/11/2024 | 0.89 | 100ms | "0ms" | 120 | 0 |

## 9. Diagramas de Petri

Se modeló el funcionamiento del proyecto en 2 redes de Petri presentadas en la siguiente imagen.

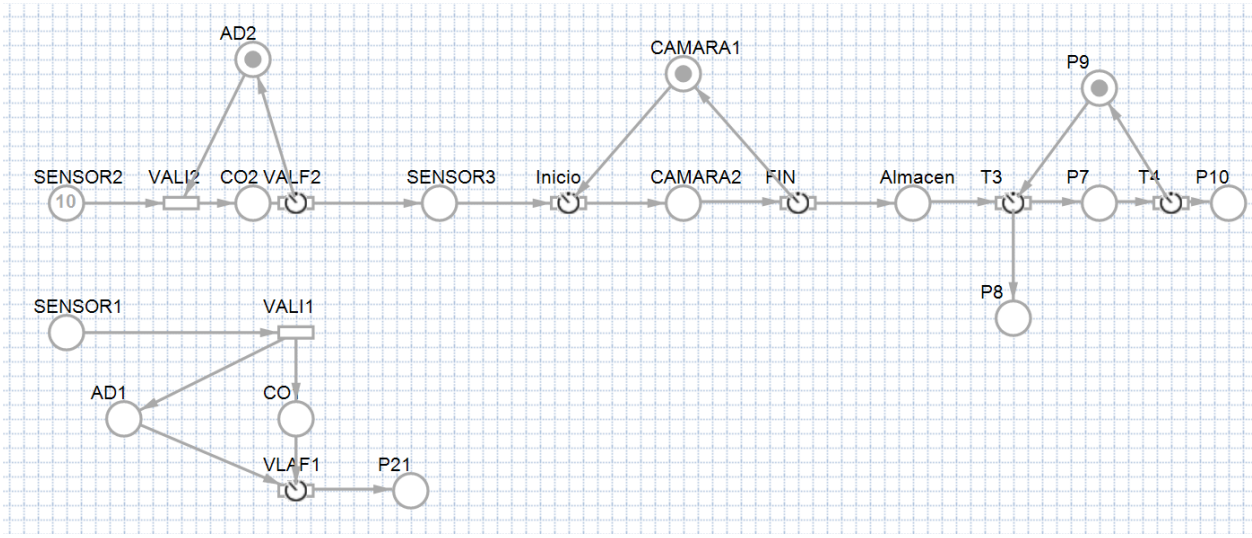


*Figura 32: Redes de Petri representando el funcoinamiento del proyecto*

## c. Conclusiones

### 1. Sistema modular y bien integrado

El proyecto se basa en una arquitectura modular en la que cada componente cumple una función específica (detección, procesamiento y monitoreo). Esta modularidad facilita la comprensión, el mantenimiento y la escalabilidad futura del sistema.

### 2. Uso eficiente de recursos

La elección de una única fuente de alimentación de 12 V con un regulador para obtener los 5 V necesarios muestra un diseño práctico y eficiente.

La integración de comunicación serial entre el Arduino y el ESP32 reduce la complejidad de interconexión.

El uso de websockets para monitoreo en tiempo real minimiza la necesidad de recargar páginas, lo que mejora la experiencia del usuario y optimiza el uso de recursos de red.

### 3. Automatización inteligente y precisa

El sistema no permite operar si el contenedor está lleno, evitando errores y asegurando que las acciones solo se realicen bajo condiciones adecuadas.

La toma de decisiones basadas en predicciones del modelo de IA (clasificación entre reciclables y no reciclables) optimiza la separación de desechos y mejora la utilidad del sistema.

### 4. Uso eficaz de la inteligencia artificial

La implementación del modelo Faster R-CNN, entrenado específicamente con datos etiquetados del proyecto, proporciona una solución personalizada y eficaz para la clasificación de desechos.

Aunque limitado a botellas y papeles en esta fase, el sistema puede extenderse a otros tipos de desechos, demostrando un diseño escalable.

### 5. Enfoque en la sostenibilidad y automatización de procesos

El uso de un sistema que separa automáticamente los desechos reciclables de los no reciclables reduce la intervención humana y fomenta prácticas de reciclaje.
La integración de un sistema de monitoreo y registro de datos en tiempo real asegura que el sistema pueda ser evaluado y mejorado con base en estadísticas, como horarios de mayor uso.

### 6. Innovación en diseño y comunicación

La implementación del ESP32 AI Thinker Cam para tareas de captura y transmisión de imágenes combina funcionalidad avanzada con simplicidad de comunicación mediante HTTP POST.

La elección de Flask como servidor demuestra una buena alineación entre la simplicidad de implementación y la necesidad de integrar IA y monitoreo web en un solo lugar.

### 7. Desafíos resueltos con ingenio

La adaptación de los niveles de voltaje para evitar dañar los componentes muestra atención al detalle en el diseño electrónico.

La necesidad de optimizar recursos, como evitar que la cámara detecte continuamente, fue abordada de manera efectiva, mejorando el rendimiento y la durabilidad del sistema.

### 8. Potencial para mejoras futuras

El diseño del sistema permite la adición de más clases al modelo de IA, ampliando las capacidades de clasificación.

El sistema de monitoreo podría complementarse con notificaciones automáticas o alertas para usuarios en caso de llenado del contenedor o fallos técnicos.

## d. Bibliografía


- **Label Studio - Detección de Objetos:**
  https://labelstud.io/templates/image_bbox.html

- **Tutorial de Flask:**
  https://www.geeksforgeeks.org/flask-tutorial/

- **Comunicación de AI Thinker CAM ESP32 con Arduino:**
  https://robotzero.one/esp32-cam-arduino-ide/

- **Faster R-CNN en PyTorch:**
  https://pytorch.org/vision/main/models/faster_rcnn.html

- **Cuadernillo sobre Detección de Objetos:**
  https://colab.research.google.com/drive/16vatrNB8ZRnm_uAGSr6awWkv6xETJPCR

- **Flask y Websockets:**
  https://pythonexamples.org/python-flask-and-websocket-example/

## e. Anexos

### 1. Fotografías de la presentación en aula

## 2. Código de Arduino

```
#include <Servo.h>

Servo miServo;
int posicionCentro = 50; // Posición neutra o central
int posicionIzquierda = 0; // Posición de 0 grados
int posicionDerecha = 120;

int servoPin = 3;
// Pines primer sensor
int TRIG = 8;
int ECHO = 9;
// Pines del segundo sensor ultrasónico
int segundoTRIG = 10;
int segundoECHO = 11;
// Tercer Sensor
int tercerTRIG = 12;
int tercerECHO = 13;

int distanciaPrimerSensor;
int distanciaSegundoSensor;
int distanciaTercerSensor;

bool flack1 = false;
bool flack2 = false;
bool flack3 = false;
bool flackAlto = false;
bool flackBajo = false;

bool LLenado1 = false;
bool Llenado2 = false;

int contador = 0;
int contador3 = 0;
int contadorAlto = 0;
int contadorBajo = 0;
// Contadores para validación
int contadorPrimerSensor = 0;
int contadorSegundoSensor = 0;

int intervalo = 1000; // Tiempo en milisegundos para regresar al centro

void setup() {
miServo.attach(servoPin); // Conectar el servo al pin definido
miServo.write(posicionCentro); // Posicionar el servo en el centro
pinMode(TRIG, OUTPUT);
pinMode(ECHO, INPUT);
pinMode(segundoTRIG, OUTPUT); // Configuración del segundo TRIG
pinMode(segundoECHO, INPUT); // Configuración del segundo ECHO
pinMode(tercerTRIG, OUTPUT); // Configuración del TRIG del tercer sensor
pinMode(tercerECHO, INPUT);
Llenado2 = false;

Serial.begin(115200); // Inicia la comunicación serial con el ESP32
Serial.println("Arduino listo para comunicarse.");
}

void loop() {

// Leer el segundo sensor de distancia

digitalWrite(TRIG, LOW);
delayMicroseconds(2);
digitalWrite(TRIG, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG, LOW);
long duracionPrimerSensor = pulseIn(ECHO, HIGH);
distanciaPrimerSensor = duracionPrimerSensor / 58.2;
Serial.print("PrimerSensor:");
Serial.println(distanciaPrimerSensor);
delay(100);

digitalWrite(segundoTRIG, LOW);
delayMicroseconds(2);
digitalWrite(segundoTRIG, HIGH);
delayMicroseconds(10);
digitalWrite(segundoTRIG, LOW);
long duracionSegundoSensor = pulseIn(segundoECHO, HIGH);
distanciaSegundoSensor = duracionSegundoSensor / 58.2;
Serial.println(distanciaSegundoSensor);

// Validar el rango del segundo sensor
if (distanciaSegundoSensor > 19 && distanciaSegundoSensor < 24) {
```

```
digitalWrite(tercerTRIG, LOW);
delayMicroseconds(2);
digitalWrite(tercerTRIG, HIGH);
delayMicroseconds(10);
digitalWrite(tercerTRIG, LOW);
long duracionTercerSensor = pulseIn(tercerECHO, HIGH);
distanciaTercerSensor = duracionTercerSensor / 58.2;
Serial.print("Distancia tercer sensor: ");
Serial.println(distanciaTercerSensor);

if (LLenado1) {
if (distanciaPrimerSensor > 18 && distanciaPrimerSensor < 24) {
flack2 = true;
flack3 = false;
}
}
if (flack2) {
contador++;
if (contador > 6) {
if (distanciaPrimerSensor > 18 && distanciaPrimerSensor < 24) {
LLenado1 = false;
contador = 0;
Serial.println("mitadfalse");
flack2 = false;
}
}
}

if (!LLenado1) {
if (distanciaPrimerSensor < 18 || distanciaPrimerSensor > 24) {
flack3 = true;
flack2 = false;
}
}
if (flack3) {
contador3++;
if (contador3 > 6) {
if (distanciaPrimerSensor < 18 || distanciaPrimerSensor > 24) {
Serial.println("mitadtrue");
contador3 = 0;
LLenado1 = true;
flack3 = false;
}
}
}


// Evaluar condiciones para enviar el mensaje al ESP32
if (distanciaTercerSensor > 6 || distanciaTercerSensor < 0) {
flack1 = true;
delay(300);
}
if (flack1) {
if (distanciaTercerSensor <= 5) {
delay(300);
Serial.println("true"); // Enviar el mensaje al ESP32
flack1 = false;
}
}

// Verificar si hay datos del ESP32 en el puerto serie
if (Serial.available()) {
String mensajeESP32 = Serial.readStringUntil('\n'); // Leer el mensaje del ESP32
mensajeESP32.trim(); // Limpiar caracteres adicionales
if (mensajeESP32.length() > 0) { // Validar que el mensaje no esté vacío
Serial.print("Mensaje del ESP32: ");
Serial.println(mensajeESP32);

// Control del servo según el mensaje recibido
if (mensajeESP32 == "botella") {
miServo.write(posicionDerecha);
} else if (mensajeESP32 == "papel") {
miServo.write(posicionIzquierda);
} else if (mensajeESP32 == "sin accion") {
miServo.write(posicionCentro);
}
delay(intervalo); // Tiempo para regresar al centro
miServo.write(posicionCentro); // Volver el servo a la posición central
}
}
}
if (Llenado2) {
if (distanciaSegundoSensor > 18 && distanciaSegundoSensor < 24) {
flackAlto = true;
```

```
flackBajo = false;
}
}
if (flackAlto) {
contadorAlto++;
if (contadorAlto > 6) {
if (distanciaSegundoSensor > 18 && distanciaSegundoSensor < 24) {
Llenado2 = false;
contadorAlto = 0;
Serial.println("llenofalse");
flackAlto = false;
}
}
}

if (!Llenado2) {
if (distanciaSegundoSensor < 18 || distanciaSegundoSensor > 24) {
flackBajo = true;
flackAlto = false;
}
}
if (flackBajo) {
contadorBajo++;
if (contadorBajo > 6) {
if (distanciaSegundoSensor < 18 || distanciaSegundoSensor > 24) {
Serial.println("llenotrue");
contadorBajo = 0;
Llenado2 = true;
flackBajo = false;
}
}
}

delay(300); // Esperar antes de la siguiente iteración del loop
}
```

## 3. Código de ESP32 AI Thinker CAM

```
#include "esp_camera.h"
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

// WiFi Credentials
const char* ssid = "Mpa";
const char* password = "pablo123";
const char* serverUrl = "http://192.168.43.57:4001/predict_esp32";
const char* serverUrl2 = "http://192.168.43.57:4001/Nivel/";
const char* serverUrl3 = "http://192.168.43.57:4001/Estado/";

// ESP32-CAM Pin Configuration
#define PWDN_GPIO_NUM 32
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM 0
#define SIOD_GPIO_NUM 26
#define SIOC_GPIO_NUM 27
#define Y9_GPIO_NUM 35
#define Y8_GPIO_NUM 34
#define Y7_GPIO_NUM 39
#define Y6_GPIO_NUM 36
#define Y5_GPIO_NUM 21
#define Y4_GPIO_NUM 19
#define Y3_GPIO_NUM 18
#define Y2_GPIO_NUM 5
#define VSYNC_GPIO_NUM 25
#define HREF_GPIO_NUM 23
#define PCLK_GPIO_NUM 22

void setup() {
Serial.begin(115200);

// Camera configuration
camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
config.pin_d1 = Y3_GPIO_NUM;
config.pin_d2 = Y4_GPIO_NUM;
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
```

```
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.pixel_format = PIXFORMAT_JPEG;

config.frame_size = FRAMESIZE_VGA; // 640x480
config.jpeg_quality = 10; // 0-63 lower number means higher quality
config.fb_count = 1;

// Camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
Serial.println("ERROR"); // Send error to Arduino
return;
}

// Connect to WiFi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
}
Serial.println("READY"); // Tell Arduino we're ready
}

void sendImageToPrediction() {
HTTPClient http;
String fullUrl = String(serverUrl3) + String(1);
http.begin(fullUrl);

// Envía la solicitud GET
int httpResponseCodePreliminar = http.GET();

// Procesa la respuesta
if (httpResponseCodePreliminar > 0) {
String response = http.getString();
}

http.end();
delay(100);
// Primer buffer que se descarta
camera_fb_t* fb = nullptr;
while (fb == nullptr) {
fb = esp_camera_fb_get();
delay(100); // Ajusta si es necesario
}
if (!fb) {
Serial.println("ERROR"); // Envía error al Arduino
return;
}
esp_camera_fb_return(fb); // Libera el primer buffer

// Segundo buffer que se envía
fb = nullptr;
while (fb == nullptr) {
fb = esp_camera_fb_get();
delay(100); // Ajusta si es necesario
}
if (!fb) {
Serial.println("ERROR"); // Envía error al Arduino
return;
}

// Debugging frame size
Serial.print("Frame size: ");
Serial.println(fb->len);

// Prepara la solicitud HTTP
// HTTP http;
http.begin(serverUrl);
// Establece el tipo de contenido para la carga de archivos
http.addHeader("Content-Type", "image/jpeg");

// Envía la solicitud POST con la imagen
int httpResponseCode = http.POST(fb->buf, fb->len);

// Libera el buffer de frame
esp_camera_fb_return(fb);

// Procesa la respuesta
if (httpResponseCode > 0) {
String response = http.getString();
```

```
// Analiza la respuesta JSON
StaticJsonDocument<200> doc;
DeserializationError error = deserializeJson(doc, response);
if (!error) {
const char* prediction = doc["prediction"];
Serial.println(prediction); // Envía la predicción al Arduino
} else {
Serial.println("JSON Error");
}
} else {
Serial.println("HTTP Error");
}
http.end();

// Retardo para asegurar frames nuevos en la siguiente captura
delay(500); // Ajusta según tu aplicación
}

void sendLevelToPrediction(int level) {
// Prepara la solicitud HTTP
HTTPClient http;
String fullUrl = String(serverUrl2) + String(level);
http.begin(fullUrl);

// Envía la solicitud GET
int httpResponseCode = http.GET();

// Procesa la respuesta
if (httpResponseCode > 0) {
String response = http.getString();
Serial.println("Nivel Actualizado en Monitoreo"); // Envía la respuesta al Arduino
} else {
Serial.println("HTTP Error");
}
http.end();

// Retardo para evitar solicitudes frecuentes
delay(500);
}

void loop() {
if (Serial.available()) {
String command = Serial.readStringUntil('\n');
command.trim();

if (command == "true") {
sendImageToPrediction();
} else if (command == "llenotrue") {
sendLevelToPrediction(3);
} else if (command == "llenofalse") {
sendLevelToPrediction(2);
} else if (command == "mitadtrue") {
sendLevelToPrediction(2);
} else if (command == "mitadfalse") {
sendLevelToPrediction(1);
}
}
}
```

# 4. Código de Servidor Flask

## 4.1 App.py (principal)

```
import torch
import torchvision.transforms.functional as F
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection import fasterrcnn_resnet50_fpn, FasterRCNN_ResNet50_FPN_Weights
from PIL import Image
import io
import base64
from flask import Flask, request, jsonify, render_template
from flask_socketio import SocketIO
import numpy as np
import cv2
import gc  # Garbage collection for clearing memory
import time
from threading import Thread
import sqlite3
from datetime import datetime, timedelta



class ObjectDetector:
    def __init__(self, model_path, num_classes=3):
        self.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        print(f"Using device: {self.device}")

        if torch.cuda.is_available():
            print(f"CUDA is available: {torch.version.cuda}")
            print(f"Device Name: {torch.cuda.get_device_name(0)}")
        else:
            print("CUDA is not available. Using CPU.")

        # Load model architecture
        self.model = self.get_object_detection_model(num_classes)

        # Load saved weights
        checkpoint = torch.load(model_path, map_location=self.device)
        self.model.load_state_dict(checkpoint['model_state_dict'])

        self.model.to(self.device)
        self.model.eval()

    def get_object_detection_model(self, num_classes):
        model = fasterrcnn_resnet50_fpn(weights=FasterRCNN_ResNet50_FPN_Weights.COCO_V1)

        # Replace the classifier head
        in_features = model.roi_heads.box_predictor.cls_score.in_features
        model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

        return model

    def predict(self, image):
        # Prepare image
        image_tensor = F.to_tensor(image).to(self.device)

        # Prediction
        with torch.no_grad():
            predictions = self.model([image_tensor])

        # Process predictions
        best_prediction = None
        highest_score = 0

        for idx, box in enumerate(predictions[0]['boxes']):
            score = predictions[0]['scores'][idx]
            label = predictions[0]['labels'][idx]

            if score > highest_score:
                highest_score = score
                best_prediction = {
                    "label": label.item(),
                    "score": score.item(),
                    "box": box.tolist()
                }

        # Classify result
        if best_prediction is None or highest_score < 0.5:
            prediccion = "sin accion"
        else:
            if best_prediction["label"] == 1:
```

```
                prediccion = "botella"
            elif best_prediction["label"] == 2:
                prediccion = "papel"
            else:
                prediccion = "sin accion"

        # Clear Torch cache to free memory
        torch.cuda.empty_cache()
        gc.collect()  # Run garbage collection

        return {
            "prediction": prediccion,
            "details": best_prediction
        }

DATABASE = 'db_basurero.db'

# Función para obtener la conexión
def get_db_connection():
    conn = sqlite3.connect(DATABASE)
    conn.row_factory = sqlite3.Row
    return conn

# Crear la tabla si no existe (solo una vez)
def create_table():
    conn = get_db_connection()
    conn.execute('''
        CREATE TABLE IF NOT EXISTS items (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            objeto TEXT NOT NULL,
            datetime TEXT NOT NULL,
            url_imagen TEXT NOT NULL
        )
    ''')
    conn.commit()
    conn.close()
# Flask Application
app = Flask(__name__)
create_table()

socketio = SocketIO(app)

# Variable para enviar a los clientes

@app.route('/Nivel/<int:valor>', methods=['GET'])

def send_nivel(valor):
        socketio.emit('nivel', {'value': valor})
        return f"Evento 'nivel' emitido con valor: {valor}", 200


@app.route('/Estado/<int:valor>', methods=['GET'])
def set_estado(valor):
    # Convertir valor a booleano (aceptar "true" o "false", insensible a mayúsculas)
    socketio.emit('estado', {'value': valor})
    # Confirmar al cliente que el estado fue procesado
    return f"Estado recibido: {valor}", 200

# Initialize model
detector = ObjectDetector('fasterrcnn_model1.pth')

# Web Interface Route
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/graficas')
def graficas():
    conn = get_db_connection()

    # Total de cada objeto
    total_objetos = conn.execute('''
        SELECT objeto, COUNT(*) as total
        FROM items
        GROUP BY objeto
    ''').fetchall()
    total_objetos = [{'objeto': row['objeto'], 'total': row['total']} for row in total_objetos]

    # Última semana
    hoy = datetime.now().date()
    hace_una_semana = hoy - timedelta(days=6)
    por_dia = conn.execute('''
        SELECT objeto,
               substr(datetime, 1, 4) || '-' || substr(datetime, 5, 2) || '-' || substr(datetime, 7, 2) as dia,
               COUNT(*) as total
        FROM items
```

```
        WHERE dia BETWEEN ? AND ?
        GROUP BY dia, objeto
    ''', (hace_una_semana, hoy)).fetchall()
    por_dia = [{'objeto': row['objeto'], 'dia': row['dia'], 'total': row['total']} for row in por_dia]

    # Día actual
    por_hora = conn.execute('''
        SELECT objeto,
               substr(datetime, 10, 2) as hora,
               COUNT(*) as total
        FROM items
        WHERE substr(datetime, 1, 8) = ?
        GROUP BY hora, objeto
    ''', (hoy.strftime('%Y%m%d'),)).fetchall()
    por_hora = [{'objeto': row['objeto'], 'hora': row['hora'], 'total': row['total']} for row in por_hora]

    # Por mes
    por_mes = conn.execute('''
        SELECT objeto,
               substr(datetime, 1, 4) || '-' || substr(datetime, 5, 2) as mes,
               COUNT(*) as total
        FROM items
        GROUP BY mes, objeto
    ''').fetchall()
    por_mes = [{'objeto': row['objeto'], 'mes': row['mes'], 'total': row['total']} for row in por_mes]

    conn.close()

    return render_template('graficas.html',
                           total_objetos=total_objetos,
                           por_dia=por_dia,
                           por_hora=por_hora,
                           por_mes=por_mes)

# ESP32-CAM Route
@app.route('/predict_esp32', methods=['POST'])
def predict_from_esp32():
    if request.data is None or len(request.data) == 0:
        return jsonify({"error": "No image data"}), 400

    try:
        # Convertir imagen a formato PIL
        image = Image.open(io.BytesIO(request.data))

        # Guardar la imagen localmente
        from datetime import datetime
        import os
        save_directory = "static/esp32_images"
        os.makedirs(save_directory, exist_ok=True)
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        file_path = os.path.join(save_directory, f"image_{timestamp}.jpg")
        image.save(file_path)

        # Detectar objetos
        result = detector.predict(image)
        objeto = result['prediction']
        url_imagen = f"/static/esp32_images/image_{timestamp}.jpg"

        # Insertar en la base de datos
        conn = get_db_connection()
        conn.execute(
            'INSERT INTO items (objeto, datetime, url_imagen) VALUES (?, ?, ?)',
            (objeto, timestamp, url_imagen)
        )
        conn.commit()
        conn.close()

        # Emitir eventos a los sockets
        socketio.emit('estado', {'value': 2})
        socketio.emit('objeto', {'value': objeto})
        socketio.emit('imagen', {'value': url_imagen})

        # Responder al ESP32
        esp32_response = {
            "prediction": result['prediction'],
            "confidence": result['details']['score'] if result['details'] else 0,
            "class": result['details']['label'] if result['details'] else 0
        }
        return jsonify(esp32_response)
    except Exception as e:
        return jsonify({"error": str(e)}), 500

if __name__ == '__main__':
    # app.run(host='0.0.0.0', port=4001, debug=True)
    # Inicia un hilo para actualizar la variable periódicamente
    socketio.run(app, host='0.0.0.0', port=4001, debug=True)
```

## 4.2 Templates

### *4.2.1 Index.html (Pagina Principal de monitoreo)*

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SCADA - Basurero Inteligente</title>
    <style>
        body {
            font-family: 'Arial', sans-serif;
            background-color: #eef2f3;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: row;
            height: 100vh;
            /* Pantalla completa */
        }

        #left-panel,
        #right-panel {
            flex: 1;
            display: flex;
            flex-direction: column;
            justify-content: center;
            align-items: center;
        }

        #container {
            background-image: url('/static/contenedor.png');
            background-size: cover;
            background-repeat: no-repeat;
            background-position: center;
            width: 450px;
            height: 600px;
            border-radius: 10px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
            position: relative;
            align-items: center;
        }
        a {
            display: inline-block;
            margin: 20px;
            text-decoration: none;
            color: #007bff;
            font-size: 18px;
            font-weight: bold;
        }

        a:hover {
            text-decoration: underline;
        }

        #fill-container {
            padding-bottom: 20px;
            padding-left: 28px;
            display: flex;
            flex-direction: column;
            justify-content: flex-end;
            align-items: center;
            height: 80%;
            width: 90%;
            position: absolute;
            bottom: 0;
        }

        .rectangle {
            width: 70%;
            height: 30%;
            margin: 3px 0;
            border-radius: 5px;
            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
        }

        #right-panel {
            display: flex;
            flex-direction: column;
```

```
            gap: 20px;
            /* Espaciado entre secciones */
            padding: 20px;
            background-color: #fff;
            border-radius: 10px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
            width: 90%;
            max-width: 400px;
            margin: 20px;
        }

        .section {
            width: 100%;
            padding: 15px;
            border: 2px solid #ddd;
            border-radius: 10px;
            text-align: center;
            background-color: #f9f9f9;
            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
            transition: transform 0.2s;
        }

        .section:hover {
            transform: scale(1.03);
        }
        h1 {
            font-size: 24px;
            margin: 5px 0 2px 0; /* Reduce espacio entre h1 y h2 */
            color: #444;
        }

        h2 {
            font-size: 20px;
            margin: 2px 0 10px 0; /* Menor espacio bajo h2 */
            color: #666;
        }

        .title {
            font-size: 20px;
            font-weight: bold;
            margin-bottom: 10px;
            color: #333;
        }

        .value {
            font-size: 22px;
            font-weight: bold;
            color: #444;
        }

        .subsection {
            margin-top: 15px;
        }

        .subsection-title {
            font-size: 18px;
            font-weight: bold;
            color: #555;
        }

        .subsection-value {
            font-size: 20px;
            font-weight: bold;
            color: #777;
        }

        img {
            width: 200px;
            height: 200px;
        }
    </style>
</head>

<body>
    <div id="left-panel">
        <div id="container">
            <div id="fill-container"></div>
        </div>
    </div>
    <div id="right-panel">
        <h1>Sistema de monitoreo</h1>
        <h2>Basurero Inteligente</h2>

    <a href="{{ url_for('graficas') }}">Ir a Gráficas</a>
        <div class="section" id="capacidad-section">
            <div class="title">Capacidad</div>
```

```
            <div class="value" id="capacidad-value">Bajo</div>
        </div>
        <div class="section" id="status-section">
            <div class="title">Estado</div>
            <div class="value" id="status-value">En espera</div>
        </div>
        <div class="section" id="last-object-section">
            <div class="title">Último objeto detectado</div>
            <div class="value" id="last-object-value">Nada</div>
        </div>
        <div class="section" id="last-object-section" style="width: 200px; height: 200px;">
            <img src="" alt="" id="imagen-detectada">
        </div>


    </div>

    <script src="https://cdn.socket.io/4.0.0/socket.io.min.js"></script>
    <script>
        const socket = io();

        const fillContainer = document.getElementById('fill-container');
        const capacidadvalue = document.getElementById('capacidad-value');
        const lastobjectvalue = document.getElementById('last-object-value');
        const imagendetectada = document.getElementById('imagen-detectada');
        window.onload = () => {
            fetch('/Nivel/1') // Enviar el valor inicial 1 al servidor
                .then(response => console.log('Nivel inicial enviado'))
                .catch(error => console.error('Error al enviar nivel inicial:', error));

        };
        socket.on('imagen', (data) => {
            const value = data.value; // Ruta relativa enviada desde el servidor
            console.log(`Imagen completa: ${value}`);
            imagendetectada.src = value;

        });
        socket.on('objeto', (data) => {
            const value = data.value;
            console.log(`objeto: ${value}`);
            lastobjectvalue.textContent = value;
        });
        socket.on('estado', (data) => {
            const value = data.value;
            console.log(`estado: ${value}`);
            const statusvalue = document.getElementById('status-value');
            if (value == 1) {
                statusvalue.textContent = "Detectando"
            } else {
                statusvalue.textContent = "En espera"
            }
        });
        socket.on('nivel', (data) => {
            const value = data.value;
            console.log(`Valor recibido: ${value}`);
            fillContainer.innerHTML = '';

            let color = '';

            if (value === 1) {
                color = 'green';
                capacidadvalue.textContent = 'Bajo';
            } else if (value === 2) {
                color = 'yellow';
                capacidadvalue.textContent = 'Medio';
            } else if (value === 3) {
                color = 'red';
                capacidadvalue.textContent = 'Alto';
            }


            // Crear rectángulos según el valor
            for (let i = 0; i < value; i++) {
                const rectangle = document.createElement('div');
                rectangle.classList.add('rectangle');
                rectangle.style.backgroundColor = color;
                fillContainer.appendChild(rectangle);
            }
        });
        // Escuchar el evento 'update' para recibir datos del servidor

    </script>
</body>

</html>
```

### 4.2.2 graficas.html (Pagina de Graficas historicas)

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Gráficas - Basurero Inteligente</title>
    <style>
        body {
            font-family: 'Arial', sans-serif;
            background-color: #eef2f3;
            margin: 0;
            padding: 0;
        }

        h1 {
            text-align: center;
            margin-top: 20px;
            color: #333;
        }

        #charts-container {
            display: flex;
            flex-wrap: wrap;
            justify-content: center;
            gap: 20px;
            margin: 20px;
        }

        .chart-section {
            width: 45%;
            background-color: #fff;
            border: 2px solid #ddd;
            border-radius: 10px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
            padding: 20px;
        }

        canvas {
            max-width: 100%;
            height: auto;
        }

        a {
            display: inline-block;
            margin: 20px;
            text-decoration: none;
            color: #007bff;
            font-size: 18px;
            font-weight: bold;
        }

        a:hover {
            text-decoration: underline;
        }
    </style>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>

<body>
    <h1>Gráficas Históricas</h1>
    <a href="{{ url_for('index') }}">Regresar al Inicio</a>
    <div id="charts-container">
        <div class="chart-section">
            <h2>Total de Objetos</h2>
            <canvas id="totalObjetosChart"></canvas>
        </div>
        <div class="chart-section">
            <h2>Objetos por Día</h2>
            <canvas id="objetosPorDiaChart"></canvas>
        </div>
        <div class="chart-section">
            <h2>Objetos por Hora (Hoy)</h2>
            <canvas id="objetosPorHoraChart"></canvas>
        </div>
        <div class="chart-section">
            <h2>Objetos por Mes</h2>
            <canvas id="objetosPorMesChart"></canvas>
        </div>
    </div>
```

```
<script>
    // Datos inyectados desde Flask
    const totalObjetosData = {{ total_objetos | tojson }};
    const porDiaData = {{ por_dia | tojson }};
    const porHoraData = {{ por_hora | tojson }};
    const porMesData = {{ por_mes | tojson }};

    // Gráfica de Total de Objetos
    const totalObjetosLabels = totalObjetosData.map(item => item.objeto);
    const totalObjetosCounts = totalObjetosData.map(item => item.total);

    new Chart(document.getElementById('totalObjetosChart'), {
        type: 'bar',
        data: {
            labels: totalObjetosLabels,
            datasets: [{
                label: 'Total',
                data: totalObjetosCounts,
                backgroundColor: ['#4caf50', '#ff9800', '#f44336'],
            }]
        },
        options: {
            responsive: true,
            plugins: {
                legend: { display: false },
            }
        }
    });

    // Gráfica de Objetos por Día
    const porDiaLabels = [...new Set(porDiaData.map(item => item.dia))];
    const objetosPorDia = [...new Set(porDiaData.map(item => item.objeto))];

    const porDiaDatasets = objetosPorDia.map(objeto => ({
        label: objeto,
        data: porDiaLabels.map(dia => {
            const entry = porDiaData.find(item => item.dia === dia && item.objeto === objeto);
            return entry ? entry.total : 0;
        }),
        borderColor: '#' + Math.floor(Math.random() * 16777215).toString(16),
        fill: false,
    }));

    new Chart(document.getElementById('objetosPorDiaChart'), {
        type: 'line',
        data: {
            labels: porDiaLabels,
            datasets: porDiaDatasets
        },
        options: {
            responsive: true,
            plugins: {
                legend: { position: 'top' },
            }
        }
    });

    // Gráfica de Objetos por Hora
    const porHoraLabels = [...new Set(porHoraData.map(item => item.hora))];
    const objetosPorHora = [...new Set(porHoraData.map(item => item.objeto))];

    const porHoraDatasets = objetosPorHora.map(objeto => ({
        label: objeto,
        data: porHoraLabels.map(hora => {
            const entry = porHoraData.find(item => item.hora === hora && item.objeto === objeto);
            return entry ? entry.total : 0;
        }),
        borderColor: '#' + Math.floor(Math.random() * 16777215).toString(16),
        fill: false,
    }));

    new Chart(document.getElementById('objetosPorHoraChart'), {
        type: 'line',
        data: {
            labels: porHoraLabels,
            datasets: porHoraDatasets
        },
        options: {
            responsive: true,
            plugins: {
                legend: { position: 'top' },
            }
        }
    });
```

```
        // Gráfica de Objetos por Mes
        const porMesLabels = [...new Set(porMesData.map(item => item.mes))];
        const objetosPorMes = [...new Set(porMesData.map(item => item.objeto))];

        const porMesDatasets = objetosPorMes.map(objeto => ({
            label: objeto,
            data: porMesLabels.map(mes => {
                const entry = porMesData.find(item => item.mes === mes && item.objeto === objeto);
                return entry ? entry.total : 0;
            }),
            borderColor: '#' + Math.floor(Math.random() * 16777215).toString(16),
            fill: false,
        }));

        new Chart(document.getElementById('objetosPorMesChart'), {
            type: 'line',
            data: {
                labels: porMesLabels,
                datasets: porMesDatasets
            },
            options: {
                responsive: true,
                plugins: {
                    legend: { position: 'top' },
                }
            }
        });
    </script>
</body>

</html>
```

## 5. Repositorio en Github

Todos los codigos y el cuadernillo de entrenamiento pueden encontrarse en el siguiente repositorio: https://github.com/antoniocfetngnu/reciclajeInteligente