**INSTITUTO LEONARDO MURIALDO**



**Reciclaje y Recolección**

**6° INFORMÁTICA B**

**Ferrando, Jaco**

**Iannone, Mia**

**Manfredo, Juan Ignacio**

**Montenegro, Franco**

**Siciliano, Thiago**

**Noviembre, 2024**

**Noviembre, 2024**

**CONTENIDO**

[Estructura del proyecto 3](#_bookmark0)

[Descripción del proyecto 3](#_bookmark1)

[Objetivos 3](#_bookmark2)

[Generales 3](#_bookmark3)

[Específicos 3](#_bookmark4)

[Meta del proyecto 3](#_bookmark5)

[Beneficiarios 4](#_bookmark6)

[Especificación operacional de las actividades/tareas a realizar 4](#_bookmark7)

[Calendario de actividades (carta Gantt) 4](#_bookmark9)

[Recursos 5](#_bookmark10)

[Humanos 5](#_bookmark11)

[Materiales 5](#_bookmark12)

[Tecnológicos 5](#_bookmark13)

[Costos operacionales 5](#_bookmark14)

[Referencias bibliográficas 6](#_bookmark16)

CONTENIDO GRÁFICO

[Ilustración 1: Ejemplo de imagen con epígrafe 4](#_bookmark8)

[Ilustración 2: Plataforma Scribbr 7](#_bookmark17)

[*Tabla 1: Costos por recursos empleados* *c*](#_bookmark15)

# Estructura del proyecto

## Descripción del proyecto

El proyecto "Reciclaje y Recolección Inteligente" busca solucionar problemas comunes en el tema de reciclaje, pensado en un entorno público (sea una plaza/parque), utilizando 3 tachos distintos que se mueven automáticamente dependiendo del tipo de basura que la persona arroje. Pensamos este concepto como una solución al verdadero problema dentro del reciclaje en nuestro país, el que la gente no se toma el tiempo de reciclar. En las plazas, aunque haya tachos distintos específicos, nadie se toma el tiempo de tirar su basura al tacho correspondiente.

Para solucionar esto tuvimos que pasar por múltiples ideas, algunas más plausible que otras, terminando en tres tachos que se posicionan debajo de un tacho superior (el que contiene la basura y los principales sensores). Debido a las limitaciones de sensores, pudimos separar únicamente los metales (utilizando un sensor inductivo) y los orgánicos (utilizando un sensor de temperatura y humedad), dejando un tercer tacho para el caso de que se detecte basura, pero esta no se pueda diferenciar. Se detecta la presencia de basura dentro del tacho superior, mediante el uso de un sensor infrarrojo. Cada tacho cuenta también con un sensor de peso, el cual se utiliza para calcular el nivel de llenado de cada tacho. De esta manera se facilita el limpiado de los tachos, ya que es fácilmente accesible, mediante la web, la información completa de los tres contenedores, pudiéndose saber cuando es recomendable ir a vaciar uno.

Este sistema puede cambiar el reciclaje en gran medida, con nuestro tacho inteligente la gente ya no tiene manera de no reciclar. Si tiran la basura en el tacho, siempre terminará en el espacio correcto, sin importar donde sea tirado.

## Objetivos

### Generales

* Mejorar el trabajo en el equipo, pudiéndonos comunicar y debatir opiniones o simplemente división de trabajo equitativa.
* Resolver problemas reales, que pueda ser aplicable a gran escala en el país.
* Entender lo máximo posible, buscando adquirir conocimientos en electrónica e informática.
* Aprender a administrar el trabajo de manera pareja y funcional, sin perder tiempo, pero administrándolo de manera lógica (sabiendo en que hay que utilizar más tiempo).
* Mejorar la comunicación, no solo con compañeros, si no también con los profesores, entendiendo como explicar nuestro proyecto y sus ideas de una manera entendible.
* Mejorar la búsqueda de información, tratando de filtrar y utilizar siempre fuentes confiables, sin dejarse engañar por cualquier resultado.
* Mejorar la creatividad en apartados fuera de nuestros conocimientos, buscando información y soluciones sobre cosas de las cuales no contábamos con muchos conocimientos previos (reciclaje).

### Específicos

* Utilizar por lo menos tres sensores.
* Entender el funcionamiento de los sensores.
* Entender el funcionamiento del motor paso a paso, utilizándolo de manera correcta.
* Medir el peso por tacho con celdas de 5 KG, derivando para sacar un porcentaje de llenado.
* Que el código funcione, y quede el tacho correspondiente debajo del de la basura en el momento correcto.
* Que funcione a una velocidad razonable, sin tardar tanto en mover un tacho a otro.
* Que la página se conecte bien a la maqueta, mostrando de manera correcta el porcentaje de llenado de cada tacho.

## Meta del proyecto

Se busca desarrollar una maqueta funcional de un tacho superior, que mediante sensores diferencie entre metales y orgánicos, dejando caer la basura en un tacho correspondiente, que será posicionado mediante el motor paso a paso de impresora. Este motor movería una tabla con los tres tachos de manera horizontal, uno al lado de otro.

El sistema detecta cuando alguien deja un objeto, reconoce de qué material es, mueve la base para ubicar el tacho correcto, abre una compuerta para que caiga adentro y mide cuánto se llenó cada tacho. Toda la información (porcentaje de llenado, peso y eventos) se guarda y se muestra en una página web sencilla, con alertas cuando un tacho está por llenarse. La idea es evitar desbordes, ahorrar tiempo en las recorridas y tomar decisiones con datos reales.

## Beneficiarios

Este proyecto beneficia, ante todo, a los lugares donde se genera basura todos los días: escuelas, universidades, oficinas, clubes, centros comunitarios y comercios. En estos espacios ayuda a mantener los tachos ordenados, evita desbordes y reduce recorridas “a ciegas”, porque avisa cuándo realmente hace falta vaciar. También favorece al personal de limpieza, que recibe alertas a tiempo y trabaja con menos urgencias, y a los gestores o responsables del lugar, que pueden decidir mejor con datos claros sobre qué tacho se llena más, cuándo y dónde. A nivel ciudad, los municipios y cooperativas de recolección optimizan rutas y reducen costos al priorizar puntos con mayor uso. Las ONG y programas de reciclaje ganan métricas reales para medir impacto y ajustar campañas. En lo cotidiano, los vecinos y usuarios encuentran espacios más limpios y agradables, con menos malos olores y menos residuos fuera de lugar. Y, por encima de todo, el ambiente se beneficia porque hay más material correctamente separado y menos viajes innecesarios, lo que recorta desperdicio y huella ambiental.

## Especificación operacional de las actividades/tareas a realizar

### Pinout (ESP32 WROOM / NodeMCU ESP32S)

**I²C (3.3 V lógica)**

SDA → GPIO 21

SCL → GPIO 22

Dispositivos en el bus: SHT31 (0x44) y PCA9685 (0x40)

**Sensores digitales (entradas)**

IR obstáculo (hay objeto) → GPIO 26 (entrada digital)

Inductivo NPN LJ12A3-4-Z/BX (detectar metal) → GPIO 27 (entrada digital, ver adaptación a 3.3 V abajo)

Endstop X\_min (NC, activo-bajo) → GPIO 16 (entrada con INPUT\_PULLUP)

**Señalización / actuadores lógicos**

NeoPixel DIN (vía 74HCT14 + R serie 330–470 Ω) → GPIO 23 (salida)

Buzzer (a través de NPN + R base) → GPIO 14 (salida)

**Balanzas / llenado (3 tachos)**

HX711 SCK (común a los 3 módulos) → GPIO 25 (salida)

HX711 DOUT Tacho 1 (Metal) → GPIO 34 (solo entrada)

HX711 DOUT Tacho 2 (Orgánico) → GPIO 35 (solo entrada)

HX711 DOUT Tacho 3 (Resto) → GPIO 36 (solo entrada)

**Eje lineal (ULN2003 + paso a paso 6 hilos)**

ULN2003 IN1 → GPIO 18 (salida)

ULN2003 IN2 → GPIO 19 (salida)

ULN2003 IN3 → GPIO 5 (salida)

ULN2003 IN4 → GPIO 17 (salida)

*(En firmware se usa AccelStepper FULL4WIRE con orden IN1, IN3, IN2, IN4.)*

Servo “gate” (abre compuerta)

PCA9685 canal CH0 → microservo de compuerta (PWM 50 Hz)

### Alimentación y dominios

3.3 V (lógica, desde el ESP32): ESP32, SHT31, PCA9685 (VCC, no V+), HX711 (recomendado a 3.3 V para compatibilidad de niveles).

5 V (potencia liviana): NeoPixel, V+ de PCA9685 (servo gate).

9–12 V (motor paso a paso): alimentación del motor al ULN2003 (recomendado 9 V según tu motor).

12 V (sensor inductivo NPN): el LJ12A3 suele requerir 6–36 V. 12 V es típico.

Masa común entre todas las fuentes (3.3 V, 5 V, 9/12 V) y retorno corto.

Desacople: 100 nF por IC; 1000 µF entre 5 V y GND cerca de la tira NeoPixel; 100–220 µF cerca del ULN2003.

Protección: fusible/PTC en 5 V (y en 9/12 V si es posible).

### Paso a paso de conexión

**1) Preparar fuentes y masas**

Colocá tus fuentes: 3.3 V (ESP32), 5 V ≥ 3 A (NeoPixel/servo), 9 V (motor), 12 V (sensor inductivo).

Uní todas las GND en un punto común (estrella) para evitar lazos.

Agregar 1000 µF/10 V entre 5 V y GND junto a la tira NeoPixel; 100 nF cerca de cada PCA9685, SHT31 y cada HX711.

Verificar polaridades y continuidad.

**2) Bus I²C (SHT31 + PCA9685)**

Conectar SDA (GPIO21) y SCL (GPIO22) del ESP32 a SDA/SCL de SHT31 y PCA9685 en paralelo.

VCC (3.3 V) a VCC de ambos; GND común.

En el PCA9685:

a. VCC = 3.3 V (lógica I²C).

b. V+ (servo power) = 5 V (no lo mezcles con VCC).

c. GND de V+ y GND de lógica deben estar en común.

Direcciones por defecto: SHT31 0x44, PCA9685 0x40

**3) Servo “gate” (PCA9685 CH0)**

Conectar el microservo (señal al pin CH0 del PCA9685, rojo a 5 V+, negro a GND).

En el firmware ya están mapeados los pulsos (≈500–2400 µs mapeados a ticks del PCA).

**4) NeoPixel con 74HCT14 (3.3→5 V)**

• ESP32 GPIO23 → R serie 47 Ω → DIN de la tira - OUT→DIN de la otra tira de LED Neopixel.

(Uso de 2 puertas en serie para no invertir la señal.)

• Tira NeoPixel: 5 V y GND robustos; 1000 µF cerca; no alimentarla desde el ESP32.

• GND de la tira común con el ESP32.

**5) Buzzer 5 V con NPN**

ESP32 GPIO2 → R base 1 kΩ → base de 2N2222/SS8050.

Emisor a GND; colector al terminal negativo del buzzer; terminal positivo a 5 V.

Si el buzzer es inductivo (no piezo activo), agregar diodo 1N4148/1N5819 en antiparalelo (cátodo a +5 V).

**6) Sensores de presencia/clasificación**

IR de obstáculo (trigger de ventana de muestreo)

VCC: 3.3 V (si el módulo lo soporta) para que la salida sea 3.3 V segura.

GND común.

OUT → GPIO26.

Inductivo NPN LJ12A3-4-Z/BX (detectar metal)

Marrón a +12 V, Azul a GND.

Negro (OUT): es colector abierto. Dos opciones seguras:

a. Adaptación simple: OUT al GPIO27 con pull-up de 10 kΩ a 3.3 V (y una R serie 1 kΩ al GPIO). GND en común con 12 V y 3.3 V.

b. Aislamiento (recomendado): OUT → optoacoplador → ESP32 (3.3 V).

Lógica en firmware: HIGH = metal (active-high tras el pull-up a 3.3 V).

Endstop X\_min (NC, activo-bajo)

Común del switch a GND; contacto NC al GPIO16.

Configurar el pin con INPUT\_PULLUP (ya está en firmware).

Cuando “toca”, el pin lee LOW

**7) Balanzas (3× HX711 + celdas de carga)**

Usar 1 HX711 por tacho. Aquí se asume 3 módulos.

Alimentación HX711: 3.3 V + GND (para que DOUT/SCK estén en 3.3 V).

Señales:

a. SCK (común) → GPIO25.

b. DOUT Tacho 1 (Metal) → GPIO34.

c. DOUT Tacho 2 (Orgánico) → GPIO35.

d. DOUT Tacho 3 (Resto) → GPIO36.

Celdas: cable corto, fijación rígida, sin cargas laterales. A+ / A- a la galga; E+ / E- al puente.

Calibración: offset con tacho vacío; escala con masa patrón; guardar en NVS o en firmware.

**8) Motor paso a paso 6 hilos + ULN2003 (eje horizontal)**

Motor 6 hilos unipolar (dos comunes al +V, 4 bobinas a colectores). En la placa ULN2003:

a. Conecta las 4 bobinas del motor a las salidas del ULN2003 (OUT1..OUT4).

b. Uní los dos comunes del motor al +9 V (o el valor nominal de tu motor).

Entradas ULN2003 desde el ESP32:

a. IN1 ← GPIO18

b. IN2 ← GPIO19

c. IN3 ← GPIO5

d. IN4 ← GPIO17

GND del ULN2003 común con el ESP32 y con la fuente del motor.

Alimentación del motor: +9 V al pin VM del ULN2003 / común del motor.

Condensador de 100–220 µF cerca de la bornera del ULN2003.

En firmware se usa AccelStepper (FULL4WIRE, IN1, IN3, IN2, IN4) para el orden correcto y movimiento no bloqueante.

**9) Distribución física / posiciones**

Distancia entre centros de tachos: 12 cm (120 mm).

Anchura de cada tacho: ~10 cm.

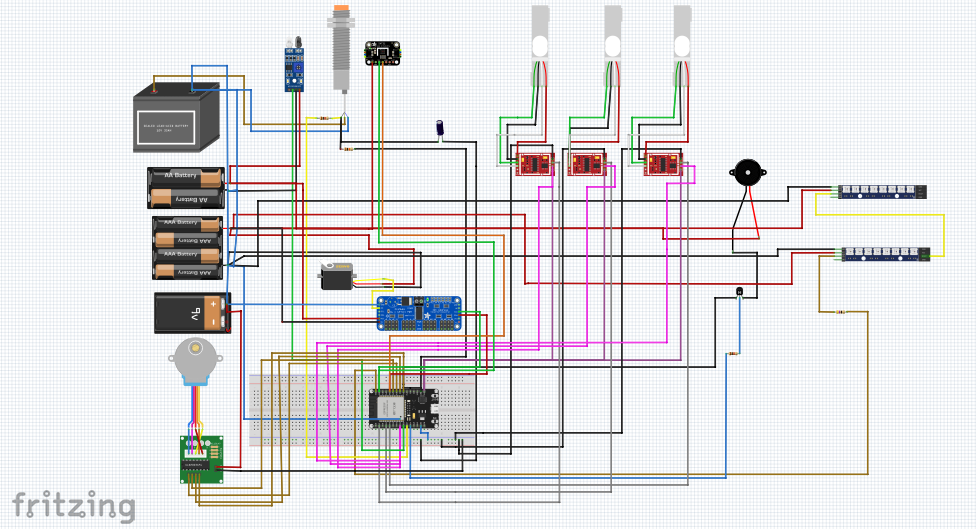
pos\_mm\_bins (firmware / API): {0, 120, 240} mm, medidos desde el cero hecho por homing (X\_min).

En homing: el eje baja a X\_min, libera (bounce), “kiss” lento, fija posición 0 y se posiciona al centro del Tacho 1.

### Chequeo final

* Fuentes separadas: 3.3 V (lógica), 5 V (NeoPixel/servo), 9 V motor, 12 V inductivo. GND común.
* NeoPixel con R 47 Ω en DIN - OUT→DIN del otro Neopixel.
* Inductivo con pull-up a 3.3 V + R serie 1 kΩ u opto (recomendado).
* HX711 a 3.3 V; SCK común (GPIO25); DOUT en GPIO34/35/36.
* I²C correcto: SDA21/SCL22 → SHT31 (0x44) y PCA9685 (0x40).
* ULN2003 IN1..IN4 en 18,19,5,17; motor a +9 V; GND común.
* Endstop X\_min NC a GPIO16 con INPUT\_PULLUP.
* Servo gate en PCA9685 CH0; V+ 5 V para potencia de servo.
* Verificar pos\_mm\_bins = [0,120,240] en dashboard/API/firmware.

### Diagrama



**Aclaración:** No se pudo encontrar un componente similar al final de carrera para añadir al diagrama, pero dejamos como es el pinout para demostrar cómo es su conexión

Endstop (o Final de Carrera) X\_min (NC, activo-bajo)

a) Común del switch a GND; contacto NC al GPIO16.

b) Configura el pin con INPUT\_PULLUP (ya está en firmware).

c) Cuando “toca”, el pin lee LOW.

### Bitácora

**1era semana (4 de agosto a 13 de agosto) – solo se trabajaba el proyecto en las horas de Werner**

Vimos muchas limitaciones y nos costaba pensar ideas aplicables a una Smart City (que mejore o aporte beneficios).

¿De qué trata la recolección? ¿Y el reciclaje? ¿Qué hace que sean métodos inteligentes en el proyecto? ¿Cómo lo incorporamos de esta forma?

¿Qué diferencia hace que deba aplicarse a una “Smart City”?

De esta forma podemos definir el alcance de nuestro proyecto (hasta donde podemos llegar)

Como hacer algo llamativo y funcional con las limitaciones que tenemos (componentes y temática compleja).

Repartimos roles y tareas:

Programación: Manfredo, Ferrando,

Informe: Manfredo, Ferrando y Iannone.

Hardware: todos (dependiendo del día nos turnamos para trabajar en la maqueta).

Investigación: Montenegro y Siciliano.

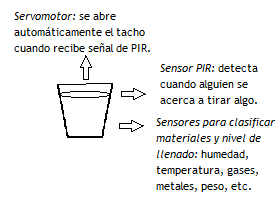
**2da semana y 3ra semana (13 de agosto a 31 de agosto) – solo se trabajaba el proyecto en las horas de Werner**

Tuvimos la **primera idea del proyecto.**

Un tacho, con distintos sensores:

* Detectar tipos de materiales.
* Nivel de llenado del tacho.
* Clasificar los materiales.

Se vería reflejado en la web



Lista de materiales:

* ESP32 más adaptador.
* Protoboard.
* 40 cables macho/hembra.
* 10 LEDs.
* 10 Resistores.
* Fuente 5V.
* Multimetro.
* Cooler 5V.
* Sensor capacitivo o inductivo.
* Sensor de gases.
* Sensor ultrasónico.
* Sensor PIR.
* Servomotor.
* Sensor IR.

Comprar materiales:

* Yurkov.
* Carluz.

Martes o jueves (salida del colegio).

Comenzar GANTT, opciones:

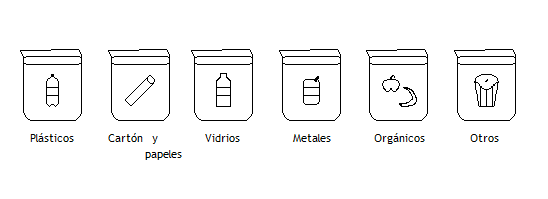
Clickup, Planner, Excel, Ganttpro, Wrike, Openproject.

Sensores:

* Capacitivo o Inductivo: detecta metales.
* Humedad y temperatura: orgánicos.
* Infrarrojo: vidrios o plásticos.
* Gases: orgánicos.
* PIR (movimiento): cuando alguien se acerca a tirar algo.
* Ultrasónico: distancia, nivel de llenado.
* Servomotor: abre la tapa de los tachos.

No convencidos con esta, tuvimos la **segunda idea del proyecto.**

Decidimos hacer 6 tachos (1 por material). Para cumplir el “reciclaje”.



Indicador de llenado: LEDs de colores (verde 0%, amarillo 50% y rojo 100%), Buzzer

Sistema de selección de residuos (botones para cada tacho)

Sensores y Actuadores:

* Ultrasonido: uno por contenedor, mide el nivel de llenado.
* Servomotor: uno por contenedor, abre el contenedor seleccionado desde la web.
* Humedad: papel y cartón, si está mojado/húmedo no puede ser reciclado.
* Temperatura y humedad: orgánicos, para el compostaje.

Página web: mapa de la “ciudad” con sector de Reciclaje y Recolección. Implementar Optimización de rutas del camión al ir a la zona para vaciar los contenedores.

Botón de apertura de cada contenedor.

Nivel de llenado (x cantidad), ultrasonido.

Indicador de “riesgos”, cartón y papel húmedo genera problemas (debe cambiarse al tacho de basura).

Compostaje en orgánicos.

Sistema de ruta más corta para la recolección inteligente: muestra en el mapa el recorrido y enciende LEDs blancos de la maqueta para indicar el orden.

Otros riesgos: se ignora la alerta del nivel de llenado y el tacho no cierra correctamente (permanece abierto). Activa LED rojo de ese contenedor.

Botón para iniciar ”proceso de recolección” una vez mostrada la ruta más corta. Abre contenedores en el orden correspondiente.

Esp32 a Proto: en medio (división de la proto).

Servomotor a Esp32:

C. rojo (VCC), pin 5v de Esp32 o Fuente de 5v.

C. negro (GND), pin GND Esp32.

C. Amarillo (señal), pin GPIO Esp32.

Sensor Ultrasónico a Esp32:

VCC, 5v o 3,3v, GND a GNDEsp32. Trig-pin dig. Salida

Echo, pin dig. Entrada

Sensor Temperatura y Humedad a Esp32:

VCC, 3,3v o 5v, GND a GNDEsp32.

Precios:

3 sensores ultrasónicos: 2.650 ARS.

3 servomotores: 3.227 ARS.

Esp32 + cable y adaptador: 13.940 ARS.

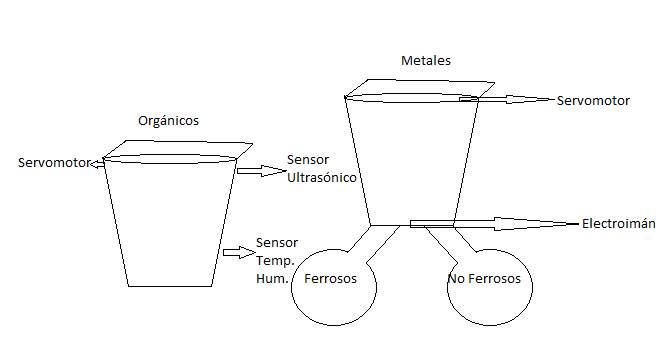
Protoboard + Fuente + 65 cables macho: 7.207 ARS.

**Viernes (22 de agosto y 29 de agosto) – Santisi**

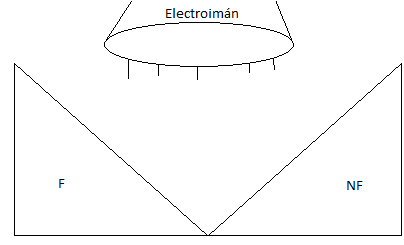
La idea anterior no nos convencía, pensamos en una **tercera idea**, que se centrase en dos tachos.

Un tacho para metales, que los separase en ferrosos y no ferrosos (mediante un electroimán). Un segundo tacho, serviría para realizar compostaje y utilizaría un censor de temperatura y humedad.

Servo y ultrasónico: control de tachos y nivel de llenado (“Recolección”).



No entendemos el funcionamiento del electroimán (descartada esa idea).



Al detectar el metal se abre la puerta correspondiente.

Materiales:

2 protos chicas.

1 electroimán.

1 sensor de peso.

2 servomotores.

2 sensores ultrasónicos.

1 sensor de temperatura Y humedad.

1 sensor de gases.

1 Arduino uno.

1 Buzzer.

1 sensor de metales. Resistencia de 10kohms, bobina de cobre, sensor hall y condensador 10nF.

**4ta semana (1 de septiembre a 4 de septiembre)**

1/9 de materiales comprados:

Una protoboard.

Un Esp32.

Cables (macho/macho y hembra/hembra) 30c/u.

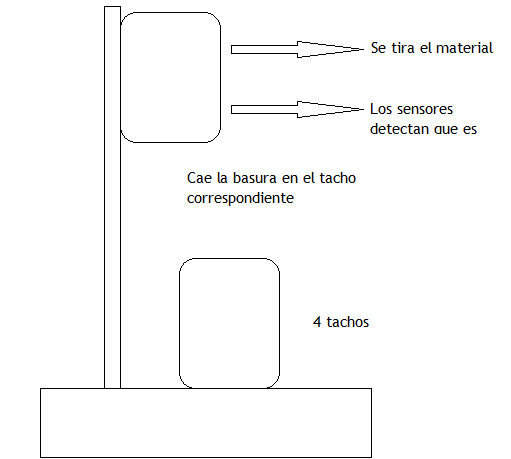
Tabla de madera de 50x50cm.

Leds, 3 de cada uno (rojo, amarillo, verde y blanco).

**4 de septiembre a 16 de septiembre**

Compra del resto de componentes y prueba de cada uno.

Surge una **nueva idea del proyecto** (gracias a la profesora Pedaci).



Inspirado en el video: [“How to make Wet Dry Metal Waste segregation project – Smart dustbin project using Arduino”](https://www.youtube.com/watch?v=4XedfXtPxLQ) – SkyNet Robotics.

Dashboard y página de presentación empezadas.

Repensamos la idea visita en el video, y decidimos complejizarlo más con la idea del Sistema Golberg (sistema que busca llevar a cabo una tarea simple de la manera más compleja y elaborada posible).

Debido a eso seguimos viendo y cambiando la programación.

Dashboard.

Código del Esp32.

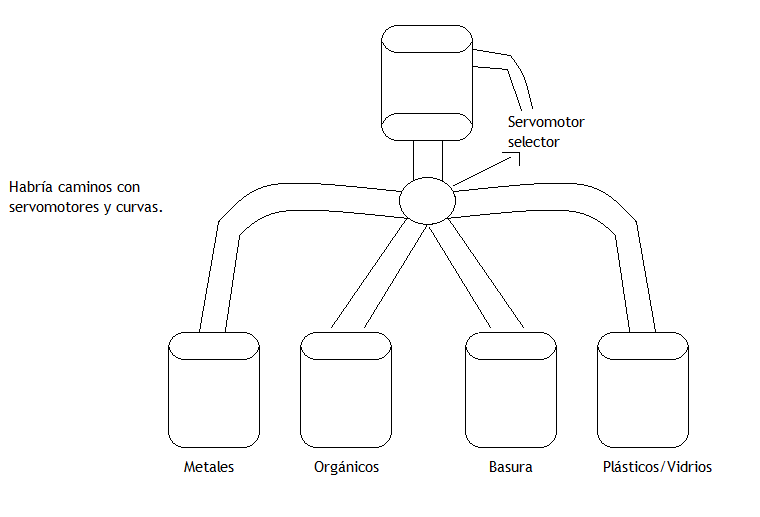
Api.

Presentación.

Empezamos con la documentación:

Excel con los precios, documento del proyecto, presentación y poster.

Máquina Sistema Golberg:



Además, sumariamos la idea de poner sensores de proximidad para abrir los tachos.

Nuestra lista de componentes quedó en:

* Esp 32 1x.
* Protoboard 1x.
* Cables (macho/macho, hembra/hembra y macho/hembra) 30 c/u.
* Neopixels de 8 LEDs 2x.
* Buzzers 4x.
* Servomotores 5x.
* Sensores de:

Peso 5Kg + amplificador 4x.

Proximidad 4x.

LDR 1x.

Inductivo 1x.

Temp/Hum. 1x.

Infrarrojo 1x.

* Fuente de poder 1x.
* Resistencias/transistores/capacitores.
* Adaptadores/Extensores/Convertidores 1x c/u.

En la página habría un registro de los datos de cada tacho (nivel de llenado, cuanto tardaban, a que hora caía más, cual era el tacho más propenso a llenarse, etc.).

**22-29 de septiembre**

Cambio de ideas por complejización y tiempo:

Idea de la Máquina de Goldberg descartada.

Consigo sacamos 3 servomotores y los sensores de proximidad de la lista.

Consigo hubo un cambio en los códigos y documentación.

Volvemos a la idea vieja la cual es más similar a la del video, solo que sumando un nuevo tacho, el de plásticos/vidrios (a través de un LDR).

Avanzamos la documentación en general.

Cambio en todos los códigos por cambio de idea nuevamente (quedan algunas ideas y definimos el diseño y color del proyecto).

Prueba del código y los componentes en conjunto.

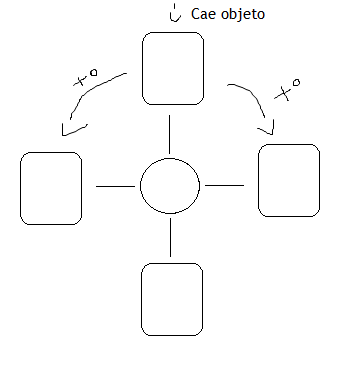
**29-3 de septiembre**

Nos percatamos de un problema a la hora de hacer la maqueta.

Los sensores de peso dificultarían el movimiento del servomotor selector de los tachos.

El sensor de peso se encuentra al fondo del tacho y los cables saldrían por un agujero, pero el movimiento giratorio los enroscaría y, al final, se terminarían rompiendo.

Se busco una solución: el servomotor no debe realizar un giro completo, sino un ángulo de movimiento para un lado y para el otro.



Depende el objeto daría un giro hacía ese lado, y volvería para que no se termine el giro.

La idea del selector de tachos no termina de convencer, nuevas ideas:

Utilizar un motor más fuerte (ej: de Alza cristal de 12v), una varilla metálica, dos roscas (1 para 2 tachos) y una polea.

El movimiento que debería hacer es en horizontal.

La rosca viajaría de izquierda a derecha o viceversa dependiendo del giro del motor que movería la varilla.

El motor se encontraría al costado de la madera, la polea en el otro lado, la varilla estaría soldada al motor, la rosca enroscada a la varilla y los tachos estarían sobre una base atornillada a las roscas.

El proyecto se vería más variado al sumarle mecánica, pero por complejizaciones y precios, la idea sería descartada.

La idea de que los tachos se muevan en horizontal.

Se pone en mente otra idea, la cual es discutida y otorgada por Martini.

Se utilizaría un motor paso a paso de impresora de 9v.

9v.

0,42ª.

Unipolar de 6 hilos.

50rpm a 300rpm.

Cables:

+A = Naranja.

-A = Azul.

+B = Blanco.

-B = Rojo.

Comunes = Marrón.

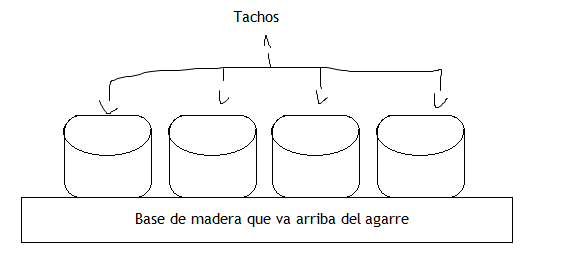
También una correa con dientes, un agarre y una polea (todo sacado de una impresora) y una base.

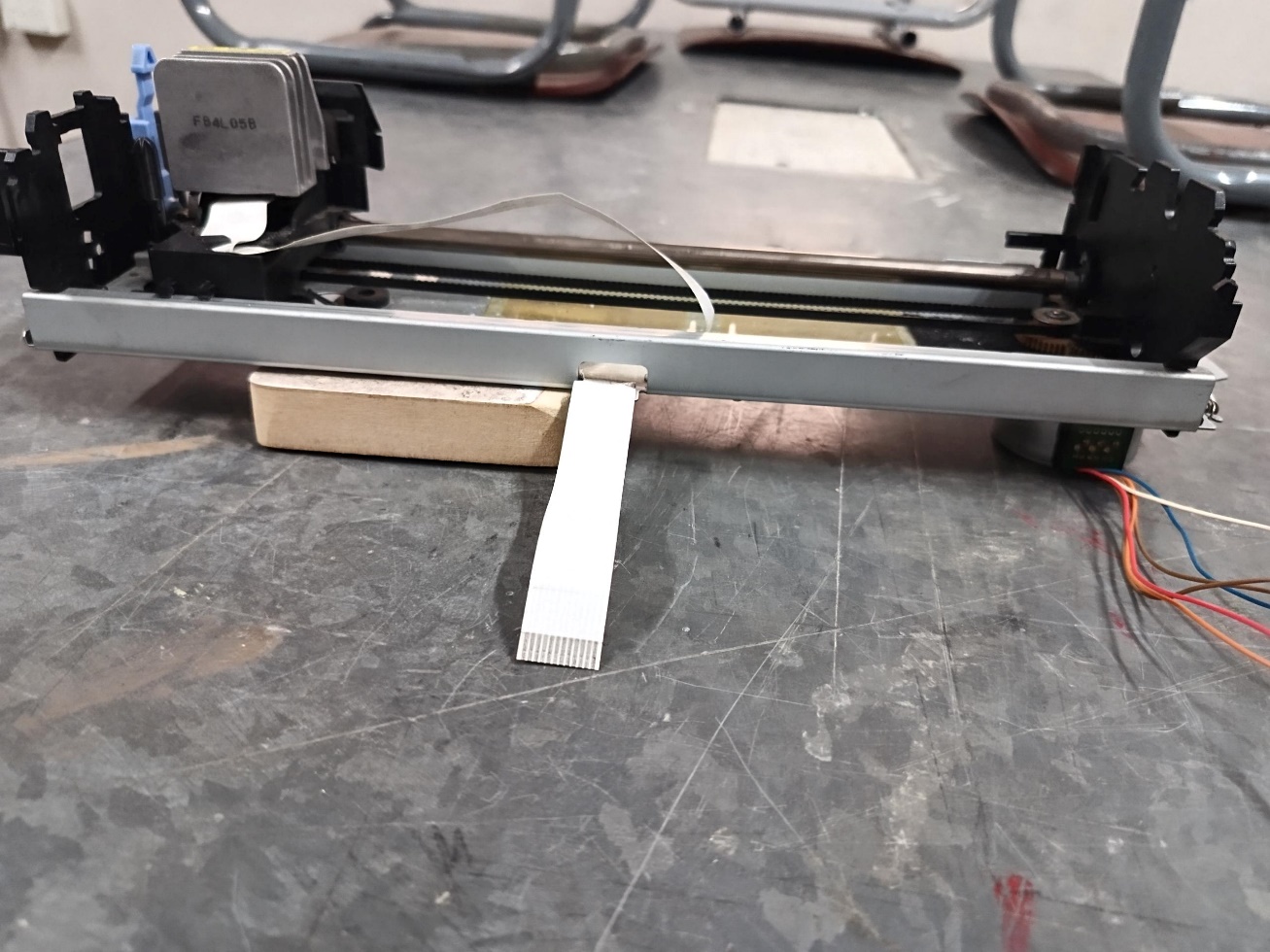
Funcionaría prácticamente de la misma forma, se movería el agarre y la base por la correa, dependiendo el movimiento que haga el motor y la polea, todo según el objeto desechado y detectado.

Todo se encontraría bajo la base de una impresora, el motor y la polea están distanciados y enfrentados, conectados por la correa que posee el agarre y la base con los tachos.

El proyecto mantendría la mecánica, pero de una forma más simplificada.

Idea Actual:





Descartamos todo lo que no sirve de la impresora, limpiamos lo que nos sirve y la lubricamos.

Por cambios de idea, tuvimos que modificar los códigos y documentación.

Planeamos la lista de materiales de la maqueta:

Caja Exterior 1x.

Madera de 50cm x 50cm.

Tornillos, tuercas, arandelas, etc.

Caño PCV de 3” (1m) 1x.

Caño PCV de ½” (7cm) 1x.

Caño PVC de ½” (40cm) 1x.

Codo de 90º PVC de ½” 1x.

Tapas ciegas para caño PVC de ½”.

Correa con dientes ≈ 50cm + Polea + Agarre.

Completamos la lista de componentes y materiales, la definimos en un Excel. Precio total de componentes: 144.159 ARS.

Precio total de materiales: 23.500 ARS.

Precio total: 67.559 ARS

**6-10 de octubre**

Avance de Documentación:

Poster: falta imagen del proyecto terminado.

Presentación: falta profundizar en algunas cosas y añadir imágenes.

Documento Word: falta tabla de costos, bitácora digital y el código comentado.

Tabla de costos: terminado.

Desarrollo y avance de los códigos, hubo retrasos por la integración del Driver Uln2003 (3000 ARS):

Controlador para el motor paso a paso que permite el control sobre aquellos que no requieren más de 500mA de corriente y funcionan a un voltaje de 5v-12v.

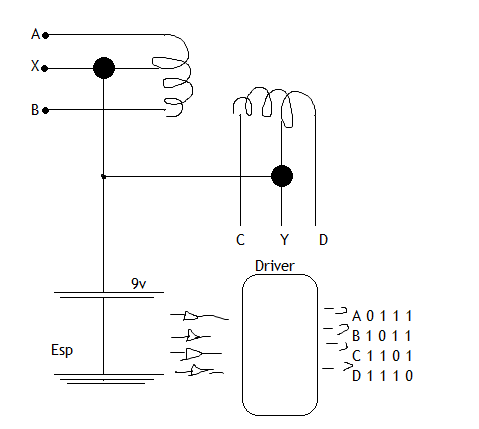
Llegamos a la integración del Driver gracias a Barral, que nos explicó su funcionamiento, y nos dio diferentes opciones de drivers, entre ellos:

El Drv8825 (para motores bipolares).

El A4988 (para motores bipolares).

El Uln2003 (para motores unipolares).

Después de hablar con Mogni, nos detalló cual era el mejor driver, como era su conexión, y nos dejó un video para ver de “Bitwise” (donde se explica cómo conectar un motor paso a paso a un Uln2003 en Arduino, y cual es su código de funcionamiento).



Sumamos la idea de agregarle el “final de carrear” con el que viene el carro de impresora.

1er cable: entrada de corriente (común o GND)

2do cable: salida que se activa (GPIO16)

Se mide la continuidad del motor paso a paso y el final de carrera, probamos que funcioe. Definimos el peso a cargar (3-4kg).

Definimos algunas medidas en la madera.

Pinout hecho, diagrama en proceso.

Motor paso a paso va al canal 0 (PCA9685).

Vcc/GND/SDA/SCL-BusI2C.

Desarrollamos y avanzamos más con la maqueta.

Avances con la maqueta:

Soldamos los sensores de peso, el convertidor y el expansor.

Ajustamos donde irían cada material a la madera.

Cortamos el tubo de 1m en 4 tachos iguales (6cm). Posteriormente los lijamos.

Ajustamos el alto del carro de impresora con una madera (la cual lijamos).

Nos llevamos los tachos para agrandarlos y el resto del tubo para hacer las tapas y bases.

Compramos el Uln2003.

Terminamos el pinout y queda así:

Lógica 3,3v: Esp32, SHT31 (sensor temp/hum.), ADS1115 (convertor), líneas de control.

Potencia 5v: PCA9685 (adaptador) + servomotor, buzzer, Neopixel (con 74hCT14 como level shifter), algunos sensores.

Potencia 9v: motor paso a paso (comunes del unipolar).

Todas las tierras en común: GND (3,3v, 5v, 9v) unidos en un punto para evitar lazos.

**13-19 de octubre**

Revisión de errores en el código (resuelto).

Confección de diagrama: faltan algunas conexiones.

Revisión del motor con el Uln2003: Funciona.

Vimos el código para el motor paso a paso: se trató de conectar a la batería, se terminó conectando a la fuente regulable (batería) (consejo de Werner).

Cambiamos el código del Esp32, el api y el dashboard por sacar un tacho (plásticos y vidrios).

Continuamos la documentación y se cambió la idea el poster.

Se definió el logotipo oficial del proyecto.

Mejoramos el diseño de la presentación.

Terminamos el diagrama.

Consideramos que los tachos eran muy chicos en diámetro, así que conseguimos un nuevo caño de 10cm.

Cortamos 4 tachos de 9cm, debido a que sacamos uno.

Terminamos de soldar los pines de los sensores.

Comenzamos a pintar parte de la maqueta.

Unimos las partes para tener el sector principal, donde se tiran los residuos

Sensor inductivo.

Sensor de proximidad infrarrojo.

**20-26 de octubre**

Terminamos el diseño del poster.

Continuamos mejorando la página web y completando la presentación.

Revisamos la documentación (detalles de lenguaje técnico, tabla de costos y gramática).

Completamos el diseño del dashboard y la página web.

Realizamos la mayoría de conexiones.

Conectamos en el contenedor los sensores inductivos, temp/hum e infrarrojo, los neopixel, el buzzer y el PCA9685).

Recortamos las tapas de los tachos, buscamos los agarres para la base recortada (de acrílico).

Verificamos que los códigos estén bien (Esp32, dashboard y la API).

Realizamos algunas soldaduras (ej: Neopixels, sensores de peso).

Ajustamos el contenedor y el carro de impresora al a base.

Modificamos la documentación (agregando la bitácora).

Pintamos la base de blanco.

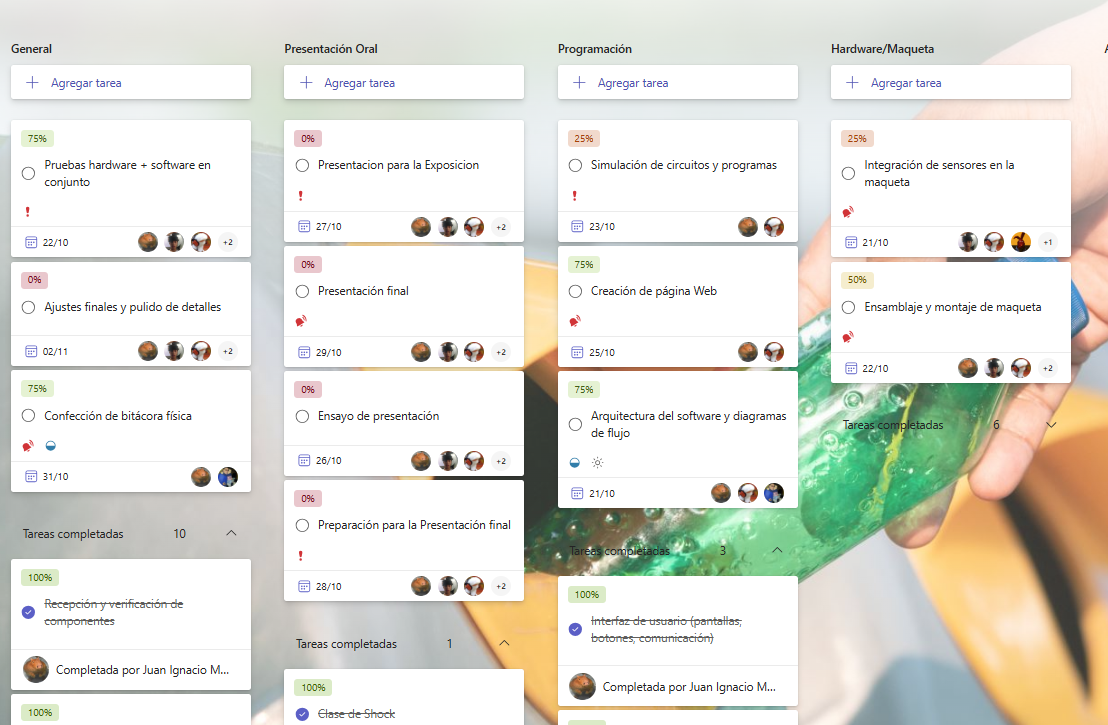
Cambiamos de Esp32, ahroa usando el Esp32 Devkit. Por esto:

Modificamos el documento, código de la página, firmware y diagrama.

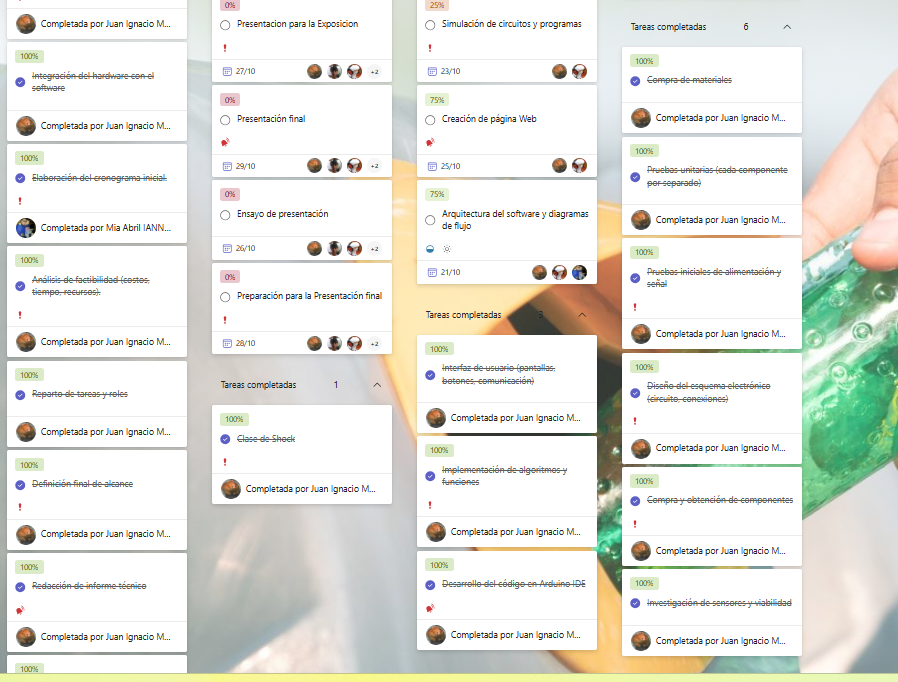
Realizamos las calibraciones y medidas para ajustar el código.

Calendario de actividades (carta Gantt)

[Planner – Smart Cities – Reciclaje y Recolección Inteligente.](https://teams.microsoft.com/l/entity/com.microsoft.teamspace.tab.planner/planner.v1.95d94ae7-1356-499a-a5bd-aacd9f060443_p_aTHcOEjtNUClzzs-qVqSUmUAByHe?tenantId=bf9504f3-4385-4e7e-b7ec-5997e0e6b0a5&webUrl=https%3A%2F%2Ftasks.teams.microsoft.com%2Fteamsui%2FpersonalApp%2Falltasklists&context=%7B%22subEntityId%22%3A%22%2Fv1%2Fplan%2FaTHcOEjtNUClzzs-qVqSUmUAByHe%22%2C%22channelId%22%3A%2219%3AsVyLkJ47Jx3qQNTDjhzqiGfBYQGz2j-OkzUPJgocy5w1%40thread.tacv2%22%7D)

[](https://teams.microsoft.com/l/entity/com.microsoft.teamspace.tab.planner/planner.v1.95d94ae7-1356-499a-a5bd-aacd9f060443_p_aTHcOEjtNUClzzs-qVqSUmUAByHe?tenantId=bf9504f3-4385-4e7e-b7ec-5997e0e6b0a5&webUrl=https%3A%2F%2Ftasks.teams.microsoft.com%2Fteamsui%2FpersonalApp%2Falltasklists&context=%7B%22subEntityId%22%3A%22%2Fv1%2Fplan%2FaTHcOEjtNUClzzs-qVqSUmUAByHe%22%2C%22channelId%22%3A%2219%3AsVyLkJ47Jx3qQNTDjhzqiGfBYQGz2j-OkzUPJgocy5w1%40thread.tacv2%22%7D)

[](https://teams.microsoft.com/l/entity/com.microsoft.teamspace.tab.planner/planner.v1.95d94ae7-1356-499a-a5bd-aacd9f060443_p_aTHcOEjtNUClzzs-qVqSUmUAByHe?tenantId=bf9504f3-4385-4e7e-b7ec-5997e0e6b0a5&webUrl=https%3A%2F%2Ftasks.teams.microsoft.com%2Fteamsui%2FpersonalApp%2Falltasklists&context=%7B%22subEntityId%22%3A%22%2Fv1%2Fplan%2FaTHcOEjtNUClzzs-qVqSUmUAByHe%22%2C%22channelId%22%3A%2219%3AsVyLkJ47Jx3qQNTDjhzqiGfBYQGz2j-OkzUPJgocy5w1%40thread.tacv2%22%7D)

[](https://teams.microsoft.com/l/entity/com.microsoft.teamspace.tab.planner/planner.v1.95d94ae7-1356-499a-a5bd-aacd9f060443_p_aTHcOEjtNUClzzs-qVqSUmUAByHe?tenantId=bf9504f3-4385-4e7e-b7ec-5997e0e6b0a5&webUrl=https%3A%2F%2Ftasks.teams.microsoft.com%2Fteamsui%2FpersonalApp%2Falltasklists&context=%7B%22subEntityId%22%3A%22%2Fv1%2Fplan%2FaTHcOEjtNUClzzs-qVqSUmUAByHe%22%2C%22channelId%22%3A%2219%3AsVyLkJ47Jx3qQNTDjhzqiGfBYQGz2j-OkzUPJgocy5w1%40thread.tacv2%22%7D)

## Recursos

### Humanos

En el proyecto trabajamos: Ferrando, Iannone, Manfredo, Montenegro y Siciliano. No hubo roles definidos realmente, fue más un trabajo en conjunto para algunas actividades. Igualmente, si tuviéramos que separar por quienes se especializaron más en algo podríamos decir que:

En el código: Manfredo fue el que más se enfocó en este apartado, pero Ferrando tuvo implicancias importantes. El profesor Barral fue muy importante para ayudar a que el código funcione.

En el hardware, con la conexión de componentes: este es el más variado, todos trabajaron en este apartado. Si tuviéramos que destacar a uno, podría ser Iannone, pero Manfredo y Ferrando tuvieron implicancias importantes. El profesor Werner fue importante en este apartado, aunque también tuvimos ayuda de Martini (con el motor de impresora) y de Mogni (que nos ayudó a entender el motor paso a paso).

En la maqueta, fuera de la conexión de componentes: como la anterior, hubo un trabajo muy mixto. No podríamos destacar a nadie por encima de otro. Hubo importantes ayudas de Werner.

En los documentos: Ferrando fue el que más trabajo en este apartado, aunque Manfredo tuvo implicancias importantes. Pedaci fue la profesora más importante para el desarrollo de la documentación, junto con Martini y Santisi para el desarrollo de la la lista de precios y la Carta Gantt.

### Materiales

**Materiales**

Electrónica principal (capa física)

*MCU y comunicaciones*

1× ESP32 DevKit (WROOM), lógica 3,3 V, Wi-Fi/BLE integrado.

Placa de desarrollo para proyectos de IoT (Internet de las Cosas) que integra conectividad Wi-Fi y Bluetooth en una sola plataforma, con un microcontrolador potente y fácil de usar.

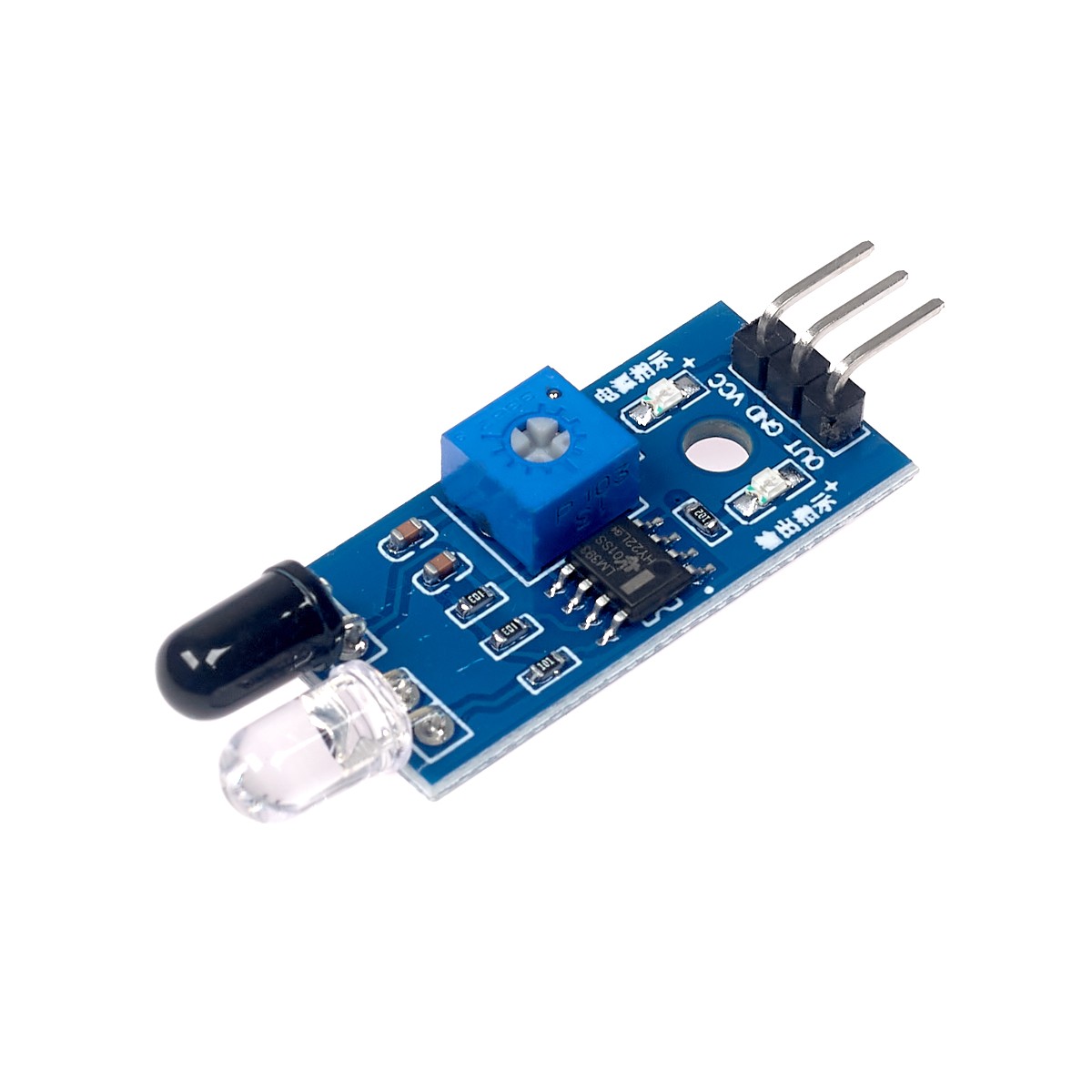


Cable USB-C/Micro-USB para programación y alimentación durante pruebas.

Clasificación (sensado en boca)

1× Sensor IR de obstáculos (emisor/receptor infrarrojo) para detectar si hay basura y apertura de ventana de clasificación.

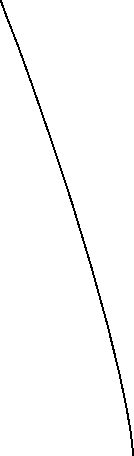
Dispositivo que utiliza un transmisor infrarrojo para emitir luz y un receptor para detectar la luz reflejada por un obstáculo cercano.



1× Sensor inductivo NPN LJ12A3-4-Z/BX (detector de metal). Salida a 12 V → adaptación a 3,3 V vía divisor resistivo o NPN/opto. Se usa activo-alto hacia el ESP32 tras la adaptación.

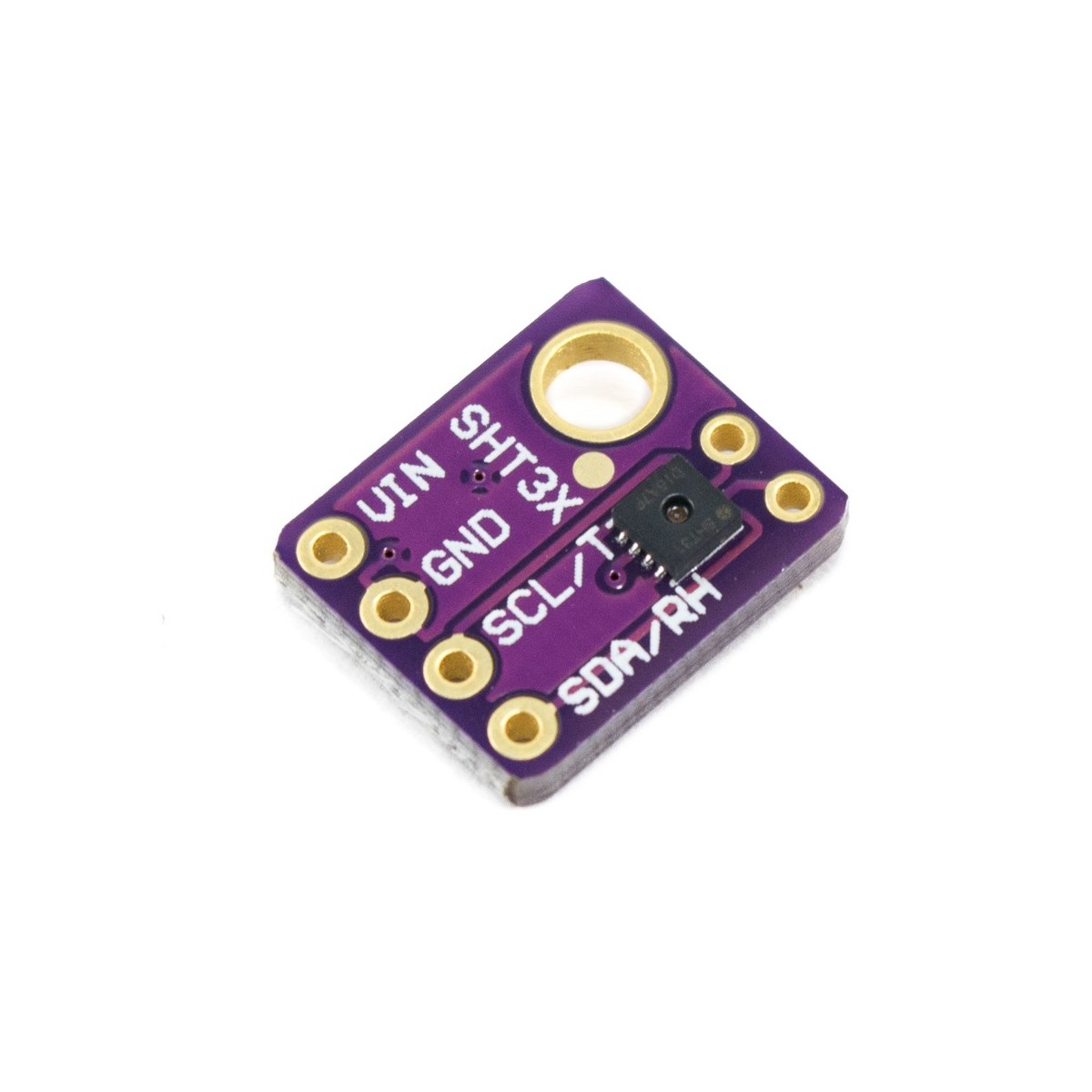
Sensor sin contacto para detectar objetos metálicos a una distancia de hasta 4 mm.





1× SHT31 (I²C) para temperatura y humedad, con baseline exponencial (EMA) en firmware.

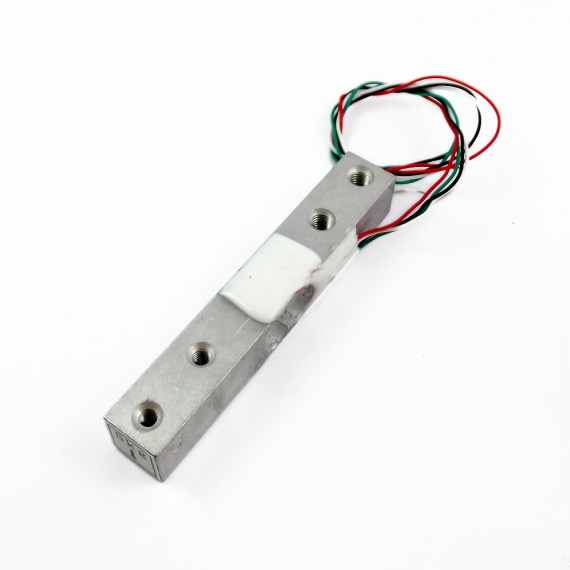
Sensor digital de temperatura y humedad de alta precisión que se comunica a través de la interfaz I²C.



Medición de llenado (por tacho)

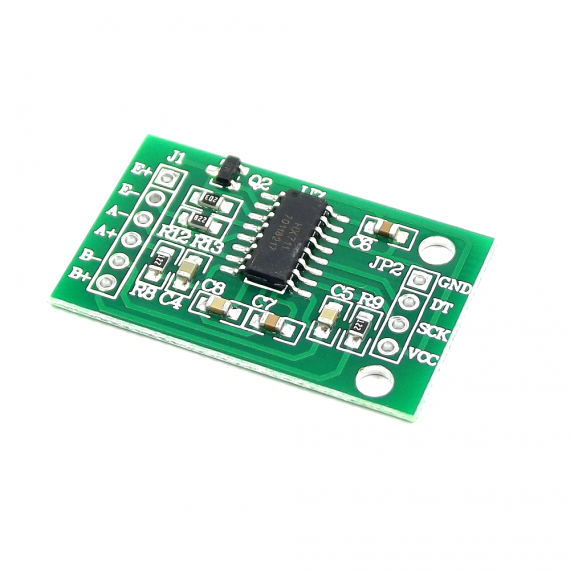
3× Celdas de carga 5 kg (una por tacho) montadas rígidas y sin fricción lateral.

Sensor que mide el peso y lo convierte en una señal eléctrica.



3× Módulos HX711 (ganancia 128), SCK común y DOUT a GPIO sólo-entrada (34/35/36).

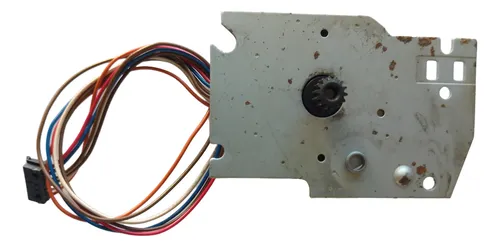
Conversores analógico-digitales de 24 bits diseñados para leer celdas de carga en básculas electrónicas y sistemas de pesaje.



Actuación

1× Motor paso a paso unipolar de 6 hilos (ej. 28BYJ-48/9 V equivalente del taller) para mover la base deslizante que posiciona los 3 tachos bajo la compuerta.

 Tiene dos bobinas por fase, donde cada bobina tiene un extremo conectado a un cable y un punto central compartido por un tercer cable (común).



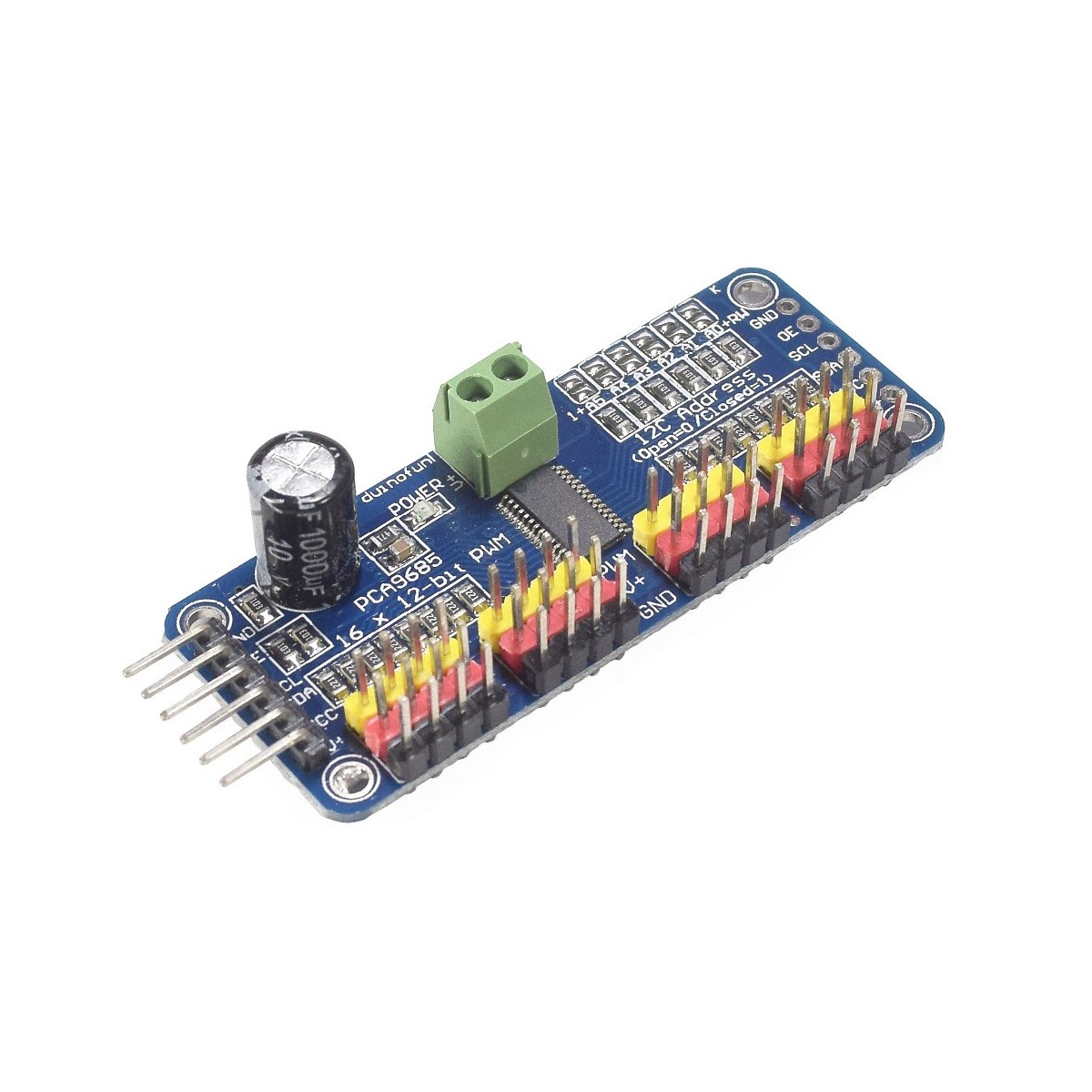
1× Driver ULN2003 (4 canales Darlington) para el paso a paso.

Circuito integrado con siete transistores Darlington que amplifican las señales de bajo voltaje de un microcontrolador para controlar cargas de mayor corriente.



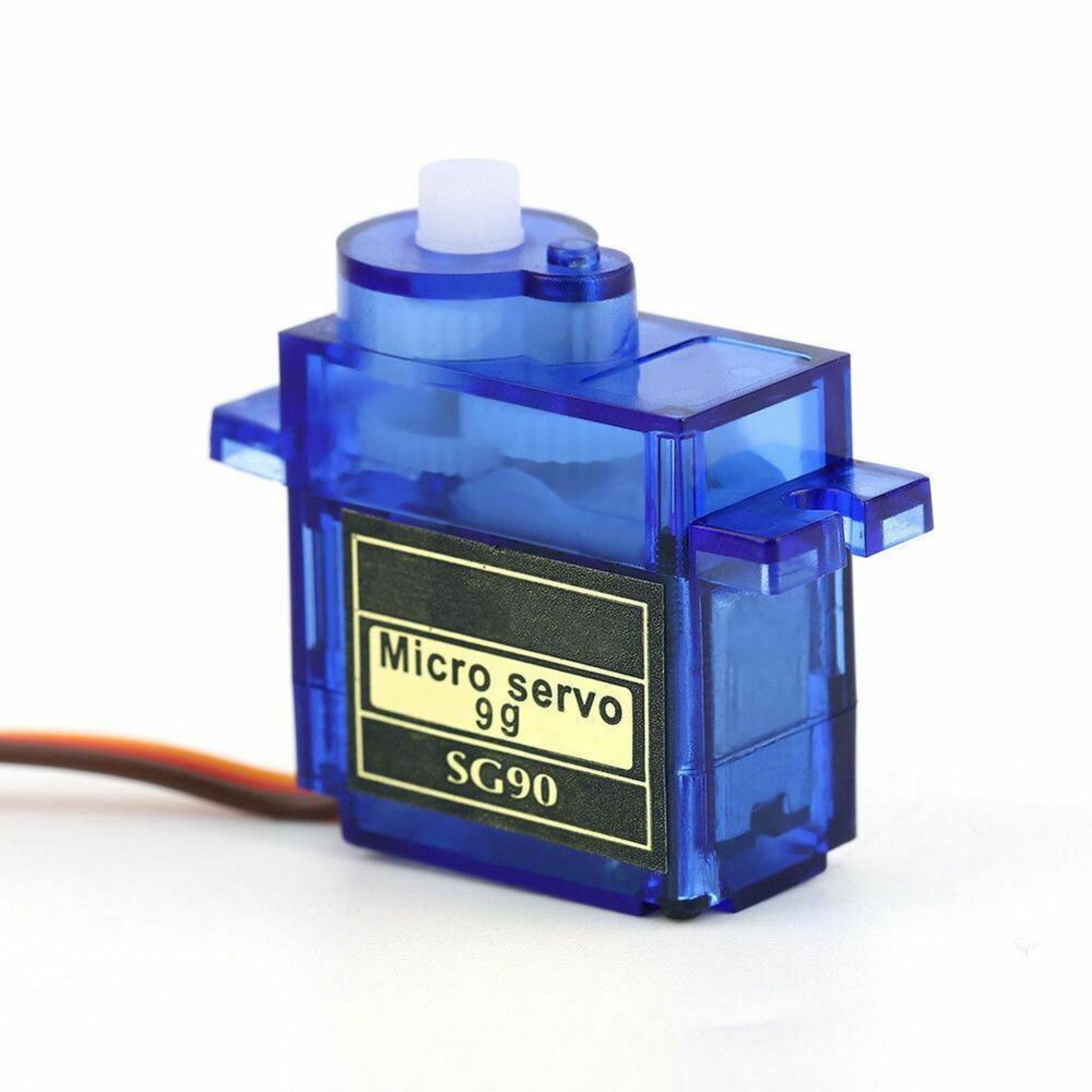
1× PCA9685 (I²C, 16 PWM) para servo de compuerta (gate), 50 Hz estable.

Chip controlador de 16 canales y 12 bits que genera señales de Pulse Width Modulation (PWM) a través de la interfaz I2C, permitiendo controlar hasta 16 servos o LEDs con solo dos pines de un microcontrolador.



1× Servo 9 g para compuerta (apertura/cierre controlada).

Actuador rotatorio o lineal que controla con precisión la posición, la velocidad y la aceleración de su eje. En este caso, de pequeño tamaño.



1× Buzzer 5 V con transistor NPN (2N2222/BC239C) + resistencia de base (1 k–4.7 k) y diodo flyback si corresponde.

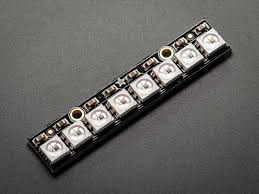
Transductor electroacústico que emite un sonido o zumbido constante al ser alimentado con 5 voltios, gracias a un oscilador interno integrado.



Señalización

NeoPixel (WS2812): 2× barras de 8 LED (total 16) en serie o 1× tira de 16 LED.

LED RGB inteligente que incluye un chip controlador en el mismo encapsulado, lo que permite controlar cada LED individualmente a través de una sola línea de datos



Seguridad y referencia

1× Final de carrera (endstop) NC para homing del eje (X\_min), cableado a INPUT\_PULLUP (LOW = activado).

Sensor electromecánico que detecta el final del recorrido de un objeto móvil mediante un contacto físico.



Potencia, pasivos y cableado

Fuente 5 V / ≥3 A (NeoPixel + servo + ULN2003).

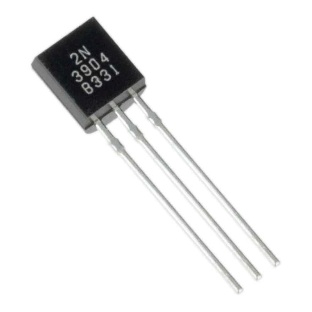
Fuente (ajustable) 9 V / ≥1 A para el paso a paso.



GND común entre lógicas y potencia.

Transistores NPN (2n2222/BC239C), utilizado para conmutar las señales del Buzzer activo.

componente electrónico fundamental que se utiliza principalmente para amplificar y conmutar señales eléctricas.

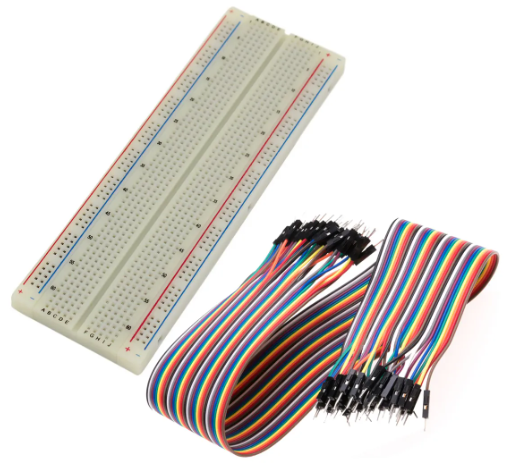


Resistencias varias (33–47 Ω DIN NeoPixel, 1 k–4.7 k bases NPN, divisores para inductivo si se usa).

Funciona como un opositor al flujo de corriente eléctrica a través de un conductor, bajando los voltajes o amperios según como lo ubiquemos.



Cables (AWG dupont / multifilar), borneras, headers, protoboard o PCB/perfboard.



Mecánica:

Correa GT2 + poleas (motor y loca), eje/rodamientos o guías lineales.

Bastidor/placa para las celdas de carga (rigidez y linealidad), topes mecánicos y alojamientos de tachos (ancho útil ~10 cm cada uno; centros separados ~120 mm).

**Herramientas**

Multímetro. Utilizada para medir la corriente y voltajes de las fuentes (de alimentación y regulable) y asegurar que las conexiones se correctas y seguras.

Sierra. Para cortar los tubos y poder hacer los 4 tachos (el contenedor, el de orgánicos, el de metales y el de la basura).

Lija/Lijadora. Para arreglar imperfecciones de los tachos, tapas, maderas, etc. que se cortaron, agujerearon o quemaron.

Agujereadora. Utilizada en la base (tanto de madera como acrílica) y en el soporte de la base y de la impesora.

Soldador (de lápiz) y Estaño. Para realizar conecciones especificas como las del motor paso a paso, el final de carrera, los sensores de peso, etc. Tambien utilizado para conectar los mismos pines a algunos sensores y componentes (sensor de Temp/hum, el actuador PCA8695, los amplificadores HX711, etc.)

Destornillador, pinzas, alicates, etc. Diferentes herramientas que ayudaron para realizar labores más generales como cortar, soldar y conectar.

### Tecnológicos

**Firmware (ESP32)**

Entorno: Arduino IDE 2.x o PlatformIO.

Librerías:

AccelStepper (control no bloqueante del paso a paso/ULN2003, homing).

Adafruit\_SHT31 (lectura T/HR).

HX711 (lectura celdas de carga, 3 canales).

Adafruit\_PWMServoDriver (PCA9685, servo de compuerta a 50 Hz).

Adafruit\_NeoPixel (WS2812, 800 kHz).

ArduinoJson (parseo robusto de configuración remota).

WiFi.h, HTTPClient.h (telemetría y registros hacia API).

Arquitectura de control: FSM (IDLE → ENTRY\_SAMPLE → AIM → RELEASE → WEIGHING → RESET), EMA para baseline T/HR, antirrebote temporal, homing con endstop NC, publicación periódica de estado de eje y registro de depósitos con reintentos/backoff.

Cinemática ajustable vía API: steps\_per\_mm, v\_max\_mm\_s, a\_max\_mm\_s2, bin\_positions\_mm=[0,120,240].

**Backend (API)**

Stack: Python 3.10+, Flask 3.x, Flask-CORS, Flask-SocketIO + eventlet, SQLite.

Endpoints clave:

GET /api/bins (estado actual por tacho: gramos, %)

GET /api/config / POST /api/config (parámetros persistentes)

POST /api/deposits (evento: bin, material, Δg, %), GET /api/deposits/recent

GET /api/deposits/history?days=7 (series para gráfico histórico)

GET /api/axis (config mínima para firmware: pasos/mm, vmax, acc, posiciones)

GET|POST /api/axis/state (telemetría del eje; también por WebSocket hacia el dashboard)

Tiempo real: emisión Socket.IO de deposit y axis a los clientes.

**Dashboard (PWA)**

Frontend: HTML5/CSS3/JS puro, PWA (manifest + service worker), modo claro/oscuro.

Interacción con API:

Fetch periódico a /api/bins, /api/deposits/recent, /api/config.

WebSocket (Socket.IO) para estado del eje y nuevos depósitos en tiempo real.

UI/UX: tarjetas por tacho (3), barras de progreso con umbrales (80 %), KPIs básicos (peso total, eventos, material más frecuente), gráfico histórico (Chart.js o equivalente), panel de Ajustes para guardar cinemática/posiciones.

Instalable: manifest.webmanifest con icono, sw.js con cache estático y fallback de demo si la API no responde.

**Utilidades y DevOps**

Git (control de versiones), venv/poetry para aislar dependencias Python, Postman/Insomnia para probar endpoints, logger (nivel INFO/ERROR) en backend, y scripts de inicialización de base de datos (migraciones simples).

### Código

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* EcoSmart – Sistema de clasificación automática de residuos

\* ESP32 + ULN2003 + 28BYJ-48 + HOMING (X\_min)

\* 3 tachos: Metal / Orgánico / Resto

\*

\* COMPONENTES PRINCIPALES:

\* - Sensor SHT31 (Temperatura/Humedad) para detectar orgánicos

\* - 3×HX711 (Básculas) para pesar cada tacho

\* - Servo PCA9685 para compuerta de liberación

\* - NeoPixel para indicación visual

\* - Motor paso a paso para movimiento del eje

\* - Sensores: IR (detección objeto), inductivo (metal), endstop (límite)

\*

\* FUNCIONAMIENTO:

\* 1. Detecta objeto con sensor IR

\* 2. Clasifica material (metal/orgánico/resto)

\* 3. Mueve eje al tacho correspondiente

\* 4. Abre compuerta y libera residuo

\* 5. Pesa el tacho y reporta a la API

\* 6. Vuelve a posición inicial

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// ==================== BIBLIOTECAS NECESARIAS ====================

#include <WiFi.h> // Conexión WiFi

#include <HTTPClient.h> // Cliente HTTP para API

#include <Wire.h> // Comunicación I2C

#include <Adafruit\_SHT31.h> // Sensor temperatura/humedad

#include <Adafruit\_NeoPixel.h> // LEDs direccionables

#include <Adafruit\_PWMServoDriver.h> // Control servo

#include <HX711.h> // Celda de carga

#include <AccelStepper.h> // Control motor paso a paso

#include <ArduinoJson.h> // Manejo JSON

#include <math.h> // Funciones matemáticas

// ==================== CONFIGURACIÓN RED / API ====================

const char\* WIFI\_SSID = "LABO"; // Nombre de la red WiFi

const char\* WIFI\_PASS = ""; // Contraseña WiFi (vacía en este caso)

String API\_BASE = "http://192.168.1.100:5000"; // Dirección del servidor backend

// ==================== CONFIGURACIÓN CAPACIDAD Y ALERTAS ====================

const float CAPACITY\_KG = 1.0f; // Capacidad máxima de cada tacho (1kg)

uint8\_t ALERT\_THRESHOLD = 50; // Porcentaje de llenado para alerta (50% = 500g)

// ==================== CONFIGURACIÓN DE PINES ====================

// I2C (comunicación con sensores)

const int PIN\_SDA = 21; // Datos I2C

const int PIN\_SCL = 22; // Reloj I2C

// Sensores digitales

const int PIN\_IR\_OBS = 26; // Sensor infrarrojo - HIGH cuando detecta objeto

const int PIN\_IND\_MET = 27; // Sensor inductivo - HIGH cuando detecta metal

// NeoPixel (tira de LEDs para indicación visual)

const int PIN\_NEOPIX = 23; // Pin de datos NeoPixel

const int NEOPIX\_N = 16; // Número de LEDs en la tira

Adafruit\_NeoPixel pixels(NEOPIX\_N, PIN\_NEOPIX, NEO\_GRB + NEO\_KHZ800);

// Buzzer (altavoz para alertas audibles)

const int PIN\_BUZZER = 14;

// HX711 (básculas para los 3 tachos)

const int PIN\_HX\_SCK = 25; // Pin común de reloj para las 3 básculas

const int PIN\_HX\_DT[3]= {34, 35, 36}; // Pines de datos para cada báscula

// PCA9685 (controlador de servos para la compuerta)

Adafruit\_PWMServoDriver pwm = Adafruit\_PWMServoDriver(0x40); // Dirección I2C

const int SERVO\_CH = 0; // Canal del servo

const int SERVO\_MIN\_US = 500; // Pulso mínimo en microsegundos

const int SERVO\_MAX\_US = 2400; // Pulso máximo en microsegundos

const int SERVO\_CLOSED = 0; // Posición cerrado (grados)

const int SERVO\_OPEN = 60; // Posición abierto (grados)

// Motor paso a paso ULN2003 + 28BYJ-48

const int PIN\_STP\_IN1 = 18; // Entrada 1 del driver

const int PIN\_STP\_IN2 = 19; // Entrada 2 del driver

const int PIN\_STP\_IN3 = 5; // Entrada 3 del driver

const int PIN\_STP\_IN4 = 17; // Entrada 4 del driver

// Endstop (sensor de fin de carrera)

const int PIN\_ENDSTOP = 16; // Pin del endstop - LOW cuando está presionado

const int HOME\_BOUNCE\_MM = 3; // Distancia de retroceso después de homing (mm)

const float HOME\_FEED\_MM\_S = 40.0; // Velocidad rápida para homing (mm/s)

const float HOME\_KISS\_MM\_S = 20.0; // Velocidad lenta para homing (mm/s)

// ==================== CONFIGURACIÓN CINEMÁTICA DEL EJE ====================

float stepsPerMm = 2.5f; // Pasos por milímetro (CALIBRAR)

float vmax\_mm\_s = 120.0f; // Velocidad máxima (mm/s)

float acc\_mm\_s2 = 400.0f; // Aceleración (mm/s²)

int pos\_mm\_bins[3] = { 0, 120, 240 }; // Posiciones de los tachos (mm)

// ==================== OBJETOS Y CONTROLADORES ====================

Adafruit\_SHT31 sht31 = Adafruit\_SHT31(); // Sensor de temperatura/humedad

HX711 HX[3]; // Array de 3 básculas

AccelStepper stepper(AccelStepper::FULL4WIRE, PIN\_STP\_IN1, PIN\_STP\_IN3, PIN\_STP\_IN2, PIN\_STP\_IN4);

// ==================== MÁQUINA DE ESTADOS ====================

// Estados posibles del sistema

enum State {

IDLE, // Esperando objeto

ENTRY\_SAMPLE, // Muestreo y clasificación

AIM, // Moviendo al tacho destino

RELEASE, // Liberando residuo

WEIGHING, // Pesando después de liberar

RESET // Volviendo a posición inicial

};

State state = IDLE; // Estado inicial

// Variables para referencia de temperatura/humedad

float baseT = NAN, baseH = NAN; // Valores base para detección de orgánicos

// Calibración de las básculas HX711

float hxOffset[3] = { 0, 0, 0 }; // Offset de calibración

float hxScale [3] = { 1, 1, 1 }; // Escala de calibración

// Umbrales para clasificación de orgánicos

float T\_DH = 1.5f; // Umbral de cambio de humedad (%)

float T\_DT = 0.5f; // Umbral de cambio de temperatura (°C)

int T\_MIN\_G = 30; // Peso mínimo para confirmar depósito (gramos)

// Comportamiento después de pesar

constexpr bool RETURN\_TO\_HOME = true; // Volver al tacho 1 después de pesar

// ==================== PROTOTIPOS DE FUNCIONES ====================

void performHoming(); // Rutina de búsqueda de cero

void axisMoveToMM(float mmTarget); // Mover eje a posición absoluta

void axisApplyKinematics(); // Aplicar parámetros cinemáticos

// ==================== SISTEMA DE CLASIFICACIÓN ====================

// Tipos de materiales

enum Material { MAT\_RESTO, MAT\_METAL, MAT\_ORG };

/\*\*

\* Obtiene el nombre del material como string

\*/

const char\* matName(Material m){

switch(m){

case MAT\_METAL: return "Metal";

case MAT\_ORG: return "Orgánico";

default: return "Resto";

}

}

/\*\*

\* Devuelve el número de tacho para cada material

\*/

int binFor(Material m){

if(m==MAT\_METAL) return 1; // Tacho 1 para metal

if(m==MAT\_ORG) return 2; // Tacho 2 para orgánico

return 3; // Tacho 3 para resto

}

/\*\*

\* Clasifica el material una vez

\* Prioridad: Metal > Orgánico > Resto

\*/

Material classifyOnce(){

// 1) Primero verificar si es metal (más rápido y confiable)

if(digitalRead(PIN\_IND\_MET)==HIGH) return MAT\_METAL;

// 2) Luego verificar si es orgánico por cambios en T/HR

float t = sht31.readTemperature();

float h = sht31.readHumidity();

// Inicializar valores base si es la primera lectura

if(isnan(baseT)||isnan(baseH)){

baseT=t;

baseH=h;

}

// Calcular diferencias respecto a la base

float dT = t - baseT;

float dH = h - baseH;

// Si supera umbrales, es orgánico

if(dH >= T\_DH || dT >= T\_DT) return MAT\_ORG;

// 3) Por defecto, es resto

return MAT\_RESTO;

}

// ==================== CONTROL DE NEOpixel Y BUZZER ====================

/\*\*

\* Establece el color de todos los LEDs NeoPixel

\*/

void setPixels(uint8\_t r, uint8\_t g, uint8\_t b){

for(int i=0; i<NEOPIX\_N; i++) {

pixels.setPixelColor(i, pixels.Color(r, g, b));

}

pixels.show();

}

/\*\*

\* Emite un sonido con el buzzer

\*/

void beep(int ms=80){

digitalWrite(PIN\_BUZZER, HIGH);

delay(ms);

digitalWrite(PIN\_BUZZER, LOW);

}

// ==================== CONTROL DEL SERVO (COMPUERTA) ====================

/\*\*

\* Convierte microsegundos a ticks PWM para PCA9685

\*/

uint16\_t usToTicks(int us){

float tick = (us/20000.0f) \* 4096.0f; // 50 Hz = 20ms periodo

if(tick<0) tick=0;

if(tick>4095) tick=4095;

return (uint16\_t)tick;

}

/\*\*

\* Mueve el servo a un ángulo específico

\*/

void servoWriteDeg(int deg){

deg = constrain(deg, 0, 180); // Limitar ángulo entre 0-180°

int us = map(deg, 0, 180, SERVO\_MIN\_US, SERVO\_MAX\_US);

pwm.setPWM(SERVO\_CH, 0, usToTicks(us));

}

/\*\*

\* Abre la compuerta

\*/

void gateOpen(){

servoWriteDeg(SERVO\_OPEN);

}

/\*\*

\* Cierra la compuerta

\*/

void gateClose(){

servoWriteDeg(SERVO\_CLOSED);

}

// ==================== COMUNICACIÓN WiFi Y HTTP ====================

/\*\*

\* Asegura la conexión WiFi, reconecta si es necesario

\*/

bool wifiEnsure(){

if(WiFi.status() == WL\_CONNECTED) return true;

WiFi.begin(WIFI\_SSID, WIFI\_PASS);

unsigned long t0 = millis();

// Intentar conectar por máximo 7 segundos

while(WiFi.status() != WL\_CONNECTED && millis()-t0 < 7000) {

delay(120);

}

return WiFi.status() == WL\_CONNECTED;

}

/\*\*

\* Realiza una petición GET y devuelve el JSON

\*/

bool httpGetJSON(const String& path, String& out){

if(!wifiEnsure()) return false;

HTTPClient http;

http.begin(API\_BASE + path);

int code = http.GET();

if(code == 200){

out = http.getString();

http.end();

return true;

}

http.end();

return false;

}

/\*\*

\* Realiza una petición POST JSON con reintentos

\*/

bool httpPostJSON\_withRetry(const String& path, const String& json){

if(!wifiEnsure()) return false;

HTTPClient http;

int tries = 0;

int code = -1;

// Reintentar hasta 3 veces con backoff exponencial

while(tries < 3){

http.begin(API\_BASE + path);

http.addHeader("Content-Type", "application/json");

code = http.POST(json);

http.end();

if(code == 200 || code == 201 || code == 202) return true;

delay( (1 << tries) \* 250 ); // 250, 500, 1000 ms

tries++;

}

return false;

}

// Cola para eventos pendientes de enviar

String pendingEventJSON = "";

// ==================== TELEMETRÍA DEL EJE ====================

unsigned long lastAxisReportMs = 0; // Último reporte de telemetría

float lastAxisPosSent = -9999.0f; // Última posición reportada

const char\* lastAxisStateSent = "IDLE"; // Último estado reportado

bool axisHomed = false; // Si el eje ha hecho homing

/\*\*

\* Obtiene la posición actual en mm

\*/

inline float axisPosMm(){

return (float)stepper.currentPosition() / stepsPerMm;

}

/\*\*

\* Envía el estado del eje a la API

\*/

void pushAxisState(const char\* st, bool homed, float pos\_mm){

if (WiFi.status() != WL\_CONNECTED) return;

HTTPClient http;

http.begin(String(API\_BASE) + "/api/axis/state");

http.addHeader("Content-Type", "application/json");

String body = String("{\"state\":\"") + st + "\",\"homed\":" +

(homed ? "true" : "false") + ",\"pos\_mm\":" +

String(pos\_mm, 1) + "}";

http.POST(body);

http.end();

}

/\*\*

\* Reporta el estado del eje inmediatamente

\*/

void reportAxisNow(const char\* st){

lastAxisStateSent = st;

pushAxisState(st, axisHomed, axisPosMm());

}

/\*\*

\* Reporte periódico del estado (cada 1s o cuando cambia posición)

\*/

void reportAxisPeriodic(){

unsigned long now = millis();

float pos = axisPosMm();

if ((now - lastAxisReportMs) > 1000UL || fabs(pos - lastAxisPosSent) > 1.0f){

pushAxisState(lastAxisStateSent, axisHomed, pos);

lastAxisReportMs = now;

lastAxisPosSent = pos;

}

}

// ==================== COMANDOS DESDE EL DASHBOARD ====================

/\*\*

\* Consulta comandos pendientes del dashboard

\*/

void pollDashboardCommands() {

static unsigned long lastPoll = 0;

// Consultar cada 500ms

if (millis() - lastPoll < 500) return;

lastPoll = millis();

String payload;

if (httpGetJSON("/api/axis/pending\_commands", payload)) {

StaticJsonDocument<256> doc;

if (deserializeJson(doc, payload)) return;

// Procesar comando de movimiento (JOG)

if (doc.containsKey("jog\_mm")) {

float jogMm = doc["jog\_mm"];

if (jogMm != 0.0f) {

float targetPos = axisPosMm() + jogMm;

axisMoveToMM(targetPos);

lastAxisStateSent = "MOVING";

Serial.printf("Comando JOG recibido: %.1f mm\n", jogMm);

}

}

// Procesar comando de homing

if (doc.containsKey("home") && doc["home"] == true) {

performHoming();

Serial.println("Comando HOME recibido");

}

}

}

// ==================== CONFIGURACIÓN DESDE API ====================

unsigned long lastCfgPollMs = 0; // Última consulta de configuración

uint32\_t lastCfgCrc = 0; // CRC de la última configuración

/\*\*

\* Aplica los parámetros cinemáticos al motor

\*/

void axisApplyKinematics(){

stepper.setMaxSpeed(vmax\_mm\_s \* stepsPerMm); // Pasos por segundo

stepper.setAcceleration(acc\_mm\_s2 \* stepsPerMm); // Pasos por segundo²

}

/\*\*

\* Consulta cambios en la configuración desde la API

\*/

void pollConfigIfChanged(){

// Consultar cada 5 segundos

if (millis() - lastCfgPollMs < 5000) return;

lastCfgPollMs = millis();

String payload;

if (!httpGetJSON("/api/config", payload)) return;

// Calcular CRC para detectar cambios

uint32\_t crc = 2166136261u;

for (size\_t i=0; i<payload.length(); ++i){

crc ^= (uint8\_t)payload[i];

crc \*= 16777619u;

}

// Si no hay cambios, salir

if (crc == lastCfgCrc) return;

lastCfgCrc = crc;

StaticJsonDocument<1024> j;

if (deserializeJson(j, payload)) {

Serial.println("❌ Error parseando JSON de configuración");

return;

}

bool changedKin = false;

// Actualizar parámetros cinemáticos si han cambiado

if (j.containsKey("steps\_per\_mm")){

float v = j["steps\_per\_mm"].as<float>();

if (v>0 && v!=stepsPerMm){

stepsPerMm=v;

changedKin=true;

}

}

if (j.containsKey("v\_max\_mm\_s")){

float v = j["v\_max\_mm\_s"].as<float>();

if (v>0 && v!=vmax\_mm\_s){

vmax\_mm\_s=v;

changedKin=true;

}

}

if (j.containsKey("a\_max\_mm\_s2")){

float v = j["a\_max\_mm\_s2"].as<float>();

if (v>0 && v!=acc\_mm\_s2){

acc\_mm\_s2=v;

changedKin=true;

}

}

if (j.containsKey("bin\_positions\_mm") && j["bin\_positions\_mm"].is<JsonArray>()){

JsonArray a = j["bin\_positions\_mm"].as<JsonArray>();

int i=0;

for (JsonVariant v : a){

if (i<3) pos\_mm\_bins[i++] = v.as<int>();

}

}

// Actualizar parámetros de clasificación

if (j.containsKey("t\_dh")) T\_DH = j["t\_dh"].as<float>();

if (j.containsKey("t\_dt")) T\_DT = j["t\_dt"].as<float>();

if (j.containsKey("t\_min\_g")) T\_MIN\_G = j["t\_min\_g"].as<int>();

if (j.containsKey("alert\_threshold")) ALERT\_THRESHOLD = j["alert\_threshold"].as<uint8\_t>();

if (changedKin) {

axisApplyKinematics();

Serial.println("✅ Configuración cinemática actualizada");

}

}

// ==================== LECTURA DE SENSORES ====================

/\*\*

\* Lee el peso en gramos de una báscula

\*/

long hxReadG(int i, int samples=12){

if(!HX[i].is\_ready()) return 0;

// Promediar varias lecturas para mayor precisión

long sum=0;

for(int k=0; k<samples; k++){

sum += HX[i].read();

}

long raw = sum / samples;

float g = (raw - hxOffset[i]) / hxScale[i];

if(g<0) g=0;

return (long)g;

}

// ==================== CONTROL DEL EJE LINEAL ====================

/\*\*

\* Convierte milímetros a pasos del motor

\*/

inline long mmToSteps(float mm){

return lround(mm \* stepsPerMm);

}

/\*\*

\* Convierte pasos del motor a milímetros

\*/

inline float stepsToMm(long st){

return (float)st / stepsPerMm;

}

/\*\*

\* Verifica si el endstop está presionado

\*/

inline bool endstopHit(){

return digitalRead(PIN\_ENDSTOP) == LOW; // LOW = presionado (NC)

}

/\*\*

\* Mueve el eje a una posición absoluta en mm

\*/

void axisMoveToMM(float mmTarget){

stepper.moveTo(mmToSteps(mmTarget));

}

/\*\*

\* Rutina de homing (búsqueda del cero)

\*/

void performHoming(){

reportAxisNow("HOMING");

// Guardar configuración actual

float origMax = stepper.maxSpeed();

float origAcc = stepper.acceleration();

// (1) Fase rápida: buscar endstop a alta velocidad

stepper.setMaxSpeed(HOME\_FEED\_MM\_S \* stepsPerMm);

stepper.setAcceleration((HOME\_FEED\_MM\_S \* stepsPerMm) \* 2);

stepper.moveTo(stepper.currentPosition() - 100000); // Movimiento largo negativo

while(!endstopHit()){

stepper.run();

delay(1);

}

stepper.stop();

while(stepper.isRunning()) stepper.run();

// (2) Fase de retroceso: alejarse del endstop

stepper.setCurrentPosition(0);

stepper.moveTo(mmToSteps(HOME\_BOUNCE\_MM));

while(stepper.distanceToGo() != 0){

stepper.run();

delay(1);

}

// (3) Fase lenta: acercamiento preciso al endstop

stepper.setMaxSpeed(HOME\_KISS\_MM\_S \* stepsPerMm);

stepper.setAcceleration((HOME\_KISS\_MM\_S \* stepsPerMm) \* 2);

stepper.moveTo(-mmToSteps(HOME\_BOUNCE\_MM + 2));

while(!endstopHit()){

stepper.run();

delay(1);

}

stepper.stop();

while(stepper.isRunning()) stepper.run();

// (4) Establecer cero y restaurar configuración

stepper.setCurrentPosition(0);

stepper.setMaxSpeed(origMax);

stepper.setAcceleration(origAcc);

// Posicionar sobre el tacho 1

axisMoveToMM(pos\_mm\_bins[0]);

while(stepper.distanceToGo() != 0){

stepper.run();

delay(1);

}

axisHomed = true;

reportAxisNow("IDLE");

}

// ==================== CONFIGURACIÓN INICIAL ====================

/\*\*

\* Setup inicial - se ejecuta una vez al inicio

\*/

void setup(){

Serial.begin(115200); // Inicializar comunicación serie

// Configurar pines de entrada/salida

pinMode(PIN\_IR\_OBS, INPUT); // Sensor IR como entrada

pinMode(PIN\_IND\_MET, INPUT); // Sensor inductivo como entrada

pinMode(PIN\_BUZZER, OUTPUT); // Buzzer como salida

digitalWrite(PIN\_BUZZER, LOW); // Apagar buzzer

pinMode(PIN\_ENDSTOP, INPUT\_PULLUP); // Endstop con resistencia pull-up

// Inicializar NeoPixel

pixels.begin();

setPixels(0, 20, 0); // Color verde suave

// Inicializar I2C y sensor SHT31

Wire.begin(PIN\_SDA, PIN\_SCL);

sht31.begin(0x44);

// Inicializar servo y cerrar compuerta

pwm.begin();

pwm.setPWMFreq(50); // Frecuencia PWM estándar para servos

gateClose();

// Inicializar las 3 básculas HX711

for(int i=0; i<3; i++){

HX[i].begin(PIN\_HX\_DT[i], PIN\_HX\_SCK);

HX[i].set\_gain(128); // Ganancia estándar

delay(50); // Pequeña pausa entre inicializaciones

}

// Configurar motor paso a paso

axisApplyKinematics();

// Conectar a WiFi

WiFi.mode(WIFI\_STA);

wifiEnsure();

// Realizar homing inicial

performHoming();

setPixels(0, 80, 20); // Color verde indicando listo

reportAxisNow("IDLE"); // Reportar estado inicial

}

// ==================== BUCLE PRINCIPAL ====================

// Variables para la máquina de estados

unsigned long tWinStart = 0; // Tiempo de inicio de ventana de muestreo

Material lastMat = MAT\_RESTO; // Último material detectado

int targetBin = 3; // Tacho destino

long g0 = 0; // Peso inicial antes de liberar

/\*\*

\* Ejecuta el movimiento del motor (no bloqueante)

\*/

inline void pumpMotion(){

stepper.run();

}

/\*\*

\* Intenta enviar eventos pendientes a la API

\*/

void tryFlushPendingEvent(){

if(pendingEventJSON.length() == 0) return;

if(httpPostJSON\_withRetry("/api/deposits", pendingEventJSON)){

pendingEventJSON = ""; // Limpiar cola si se envió correctamente

}

}

/\*\*

\* Loop principal - se ejecuta continuamente

\*/

void loop(){

// ========== TAREAS DE FONDO (siempre activas) ==========

// 1. Ejecutar movimiento del motor si está pendiente

pumpMotion();

// 2. Reportar telemetría periódicamente

reportAxisPeriodic();

// 3. Consultar comandos desde el dashboard

pollDashboardCommands();

// 4. Verificar cambios de configuración

pollConfigIfChanged();

// ========== MÁQUINA DE ESTADOS PRINCIPAL ==========

switch(state){

// ========== ESTADO: REPOSO ==========

case IDLE: {

// Actualizar valores de referencia de temperatura/humedad

float t = sht31.readTemperature();

float h = sht31.readHumidity();

if(!isnan(t) && !isnan(h)){

if(isnan(baseT) || isnan(baseH)){

// Primera lectura - establecer valores base

baseT = t;

baseH = h;

} else {

// Actualizar con filtro EMA (suavizado)

baseT = baseT \* 0.995f + t \* 0.005f;

baseH = baseH \* 0.995f + h \* 0.005f;

}

}

// Indicación visual de estado reposo

setPixels(0, 40, 10);

// Intentar enviar eventos pendientes

tryFlushPendingEvent();

// Verificar si hay objeto en la entrada

if(digitalRead(PIN\_IR\_OBS) == HIGH){

tWinStart = millis(); // Iniciar temporizador

lastMat = MAT\_RESTO; // Resetear clasificación

state = ENTRY\_SAMPLE; // Cambiar a estado de muestreo

lastAxisStateSent = "SENSING";

setPixels(20, 60, 0); // Cambiar color indicando detección

}

} break;

// ========== ESTADO: MUESTREO Y CLASIFICACIÓN ==========

case ENTRY\_SAMPLE: {

// Ventana de muestreo de 500ms para clasificación estable

if(millis() - tWinStart < 500){

Material m = classifyOnce();

// Prioridad: Metal > Orgánico > Resto

if(m == MAT\_METAL)

lastMat = MAT\_METAL;

else if(m == MAT\_ORG && lastMat != MAT\_METAL)

lastMat = MAT\_ORG;

} else {

// Fin de ventana de muestreo - determinar tacho destino

targetBin = binFor(lastMat);

// Mover al tacho correspondiente

axisMoveToMM((float)pos\_mm\_bins[targetBin-1]);

lastAxisStateSent = "MOVING";

state = AIM; // Cambiar a estado de movimiento

}

} break;

// ========== ESTADO: MOVIMIENTO AL TACHO ==========

case AIM: {

setPixels(40, 80, 0); // Color indicando movimiento

// Esperar a que termine el movimiento

if(stepper.distanceToGo() == 0){

reportAxisNow("POSITIONED");

state = RELEASE; // Cambiar a estado de liberación

}

} break;

// ========== ESTADO: LIBERACIÓN DE RESIDUO ==========

case RELEASE: {

reportAxisNow("RELEASING");

// Abrir compuerta, esperar, cerrar

gateOpen();

delay(250);

gateClose();

// Leer peso inicial después de liberar

g0 = hxReadG(targetBin-1, 15);

state = WEIGHING;

lastAxisStateSent = "WEIGHING";

} break;

// ========== ESTADO: PESADO FINAL ==========

case WEIGHING: {

static unsigned long t0 = 0;

if(t0 == 0){

t0 = millis(); // Iniciar temporizador de asentamiento

}

if(millis() - t0 < 300) break; // Esperar 300ms para asentamiento

t0 = 0; // Resetear temporizador

// Leer peso final

long g1 = hxReadG(targetBin-1, 15);

long dg = g1 - g0;

if(dg < 0) dg = 0; // Evitar valores negativos

// Calcular porcentaje de llenado

float percent = (float)g1 / (CAPACITY\_KG \* 1000.0f) \* 100.0f;

int ipct = (int)percent;

// Preparar datos para enviar a la API

String json = String("{\"bin\":") + targetBin +

",\"material\":\"" + String(matName(lastMat)) + "\"," +

"\"delta\_g\":" + dg + "," +

"\"fill\_percent\":" + ipct + "}";

// Intentar enviar inmediatamente, guardar en cola si falla

if(!httpPostJSON\_withRetry("/api/deposits", json)){

pendingEventJSON = json;

}

// Activar alertas si es necesario

if(percent >= ALERT\_THRESHOLD){

setPixels(180, 80, 0); // Color naranja/alerta

beep(120); // Sonido de alerta

} else {

setPixels(0, 80, 20); // Color normal

}

state = RESET;

lastAxisStateSent = "MOVING";

} break;

// ========== ESTADO: RESET Y VUELTA A INICIO ==========

case RESET: {

// Volver a posición inicial (tacho 1) o mantener posición

float target = RETURN\_TO\_HOME ? (float)pos\_mm\_bins[0] : axisPosMm();

axisMoveToMM(target);

if(stepper.distanceToGo() == 0){

reportAxisNow("IDLE");

state = IDLE; // Volver al estado de reposo

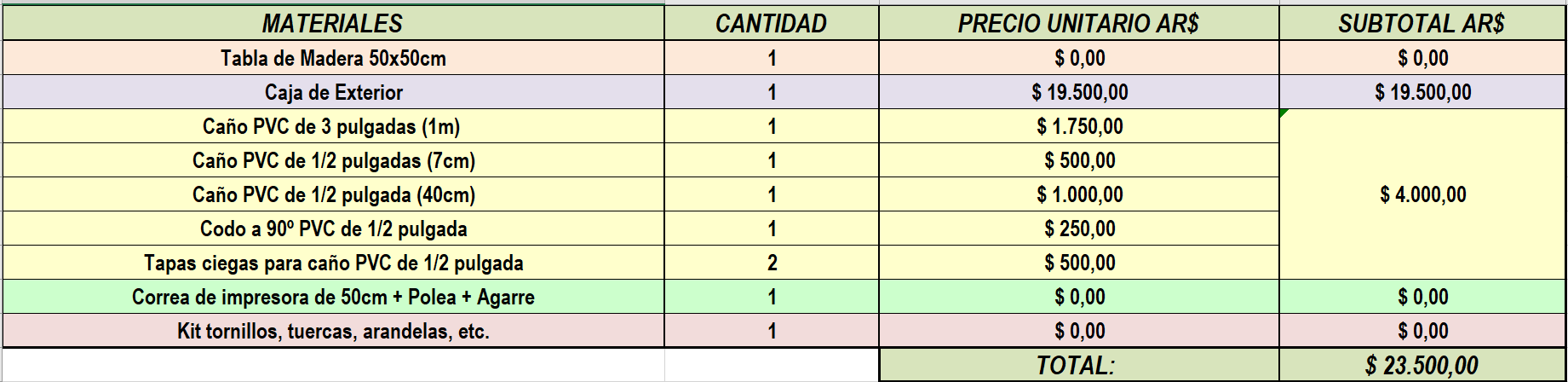
}

} break;

}

}

### Costos Operacionales

[](https://github.com/manfredojuanignacio08-prog/Proyecto-Final-Reciclaje-y-Recoleccion-Inteligente/blob/main/Documentaci%C3%B3n/Tabla%20de%20Precios%20-%20Manfredo%2C%20Ferrando%2C%20Iannone%2C%20Siciliano%20y%20Montenegro%203%20(1).xlsx)**[](https://github.com/manfredojuanignacio08-prog/Proyecto-Final-Reciclaje-y-Recoleccion-Inteligente/blob/main/Documentaci%C3%B3n/Tabla%20de%20Precios%20-%20Manfredo%2C%20Ferrando%2C%20Iannone%2C%20Siciliano%20y%20Montenegro%203%20(1).xlsx)**

## Conclusión

### Cuestiones logradas

Fue difícil, nos costó un montón poder encontrar un proyecto que, cumpla nuestras expectativas, respete la temática y que esté a nuestro alcance realizarlo. No queríamos que nuestro proyecto peque de simple, hubiera sido más sencillo hacer un tacho que solo midiera nivel de llenado y poco más, pero quisimos hacer algo que demuestre nuestras capacidades en el ámbito que llevamos estudiando tres años, la informática. Además, la idea era ver cómo resolver el verdadero problema del reciclaje: el *no* reciclaje. Poca gente es la que se toma el tiempo de revisar donde tira su basura, si es una botella de plástico, o una manzana. Por lo tanto, nuestro proyecto apuntó a resolver *ese* problema, el *verdadero* problema. Cosa que conseguimos, gracias también a los profesores, pudimos encontrar una solución con el sistema de tacos que se mueven con el motor paso a paso.

Al final, logramos demostrar que, a pesar del tiempo y la dificultad de la temática, trabajando bien y en equipo se puede llegar a realizar algo bien, y a tiempo.

### Cuestiones no logradas

Hubo montones de ideas, algunas que solo fueron soltadas al aire y no se exploraron lo suficiente como para ser dignas de anotarse. Pensamos enfocarnos en una página más interactiva, siendo lo físico algo más secundario, por ejemplo. Pero todo esto resultaba poco práctico de realizar, lo *más* importante era lo *físico*.

Ya decididos a enfocarnos en los tachos, surgió un problema importante. **No había forma de detectar plásticos.** En vez de eso, tuvimos que optar por detectar el nivel de transparencia/luminosidad, de esta manera, detectando si un objeto es transparente, podemos asumir que probablemente se trate de un plástico o un vidrio. Después vimos que por tiempo y dificultad no podríamos integrar el LDR, ya que lo único que haría ese tacho era ocupar espacio, no tenía un funcionamiento del todo útil y necesario. Así que optamos por descartarlo y dejar solamente tres tachos (orgánicos, metales y basura). Igualmente, tampoco podemos diferenciar el papel, cartón o tela.

### El porque no se logró

El problema radica, no en nuestra capacidad o tiempo, si no en los recursos. No contamos con un sensor que pueda detectar ninguno de esos tipos de materiales, no existe. Por lo tanto, hubo que optar por juntar los objetos transparentes, y dejar de lado el papel/cartón, y demás como otros. y basura). Igualmente, tampoco podemos diferenciar el papel, cartón o tela.

## Referencias bibliográficas

Espressif Systems. (2023). ESP32-WROOM-32 Datasheet (Rev. más reciente). Espressif Systems. <https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf>

NXP Semiconductors. (2015). PCA9685 – 16-channel, 12-bit PWM Fm+ I²C-bus LED controller (Rev. más reciente). NXP. https://www.nxp.com/docs/en/data-sheet/PCA9685.pdf

Worldsemi. (2020). WS2812B Intelligent control LED integrated light source (Datasheet). Worldsemi. <https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf>

Adafruit. (s. f.). Adafruit SHT31 Library (documentación y ejemplos). GitHub. <https://github.com/adafruit/Adafruit_SHT31>

Adafruit. (s. f.). Adafruit PWM/Servo Driver Library (PCA9685). GitHub. <https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library>

Avia Semiconductor. (2018). HX711 24-Bit ADC for Weigh Scales (Datasheet). Avia. <https://www.mouser.com/datasheet/2/813/hx711_english-1022875.pdf>

Greiman, B., & otros. (s. f.). HX711 Arduino Library (lectura de celdas de carga). GitHub. <https://github.com/bogde/HX711>

McCauley, M. (AirSpayce). (s. f.). AccelStepper Library (documentación). AirSpayce. <http://www.airspayce.com/mikem/arduino/AccelStepper/>

Arduino. (s. f.). HTTPClient.h — ESP32 Arduino core (referencia). Arduino-ESP32. <https://github.com/espressif/arduino-esp32/tree/master/libraries/HTTPClient>

bblanchon. (s. f.). ArduinoJson — C++ JSON library for IoT (documentación). <https://arduinojson.org/>

Pallets. (s. f.). Flask — Documentation. <https://flask.palletsprojects.com/>

Miguel Grinberg. (s. f.). Flask-SocketIO — Documentation. <https://flask-socketio.readthedocs.io/>

Socket.IO. (s. f.). Socket.IO — Documentation. <https://socket.io/docs/v4/>

Chart.js. (s. f.). Chart.js — Documentation. <https://www.chartjs.org/docs/latest/>

Fritzing. (s. f.). Fritzing — Herramienta de diseño de prototipos electrónicos. <https://fritzing.org/>

Bitwise. (s. f.). Curso Arduino – Capítulo 30: Motores paso a paso [Video]. YouTube. (Visto para comprender el control del motor paso a paso y secuencias de conmutación). <https://www.youtube.com/watch?v=2-nVV9S7leM&t=570s>

Bitwise. (s. f.). Curso Arduino – Capítulo 29: Neopixel LED RGB inteligente WS2812 (y tiras) [Video]. YouTube. (Visto para comprender el control del neopixel led y las diferentes formas de ejecutarlo). <https://www.youtube.com/watch?v=MeTmdw-FTPA>

ElectroDH / Variantes OEM. (s. f.). LJ12A3-4-Z/BX Inductive Proximity Sensor (hoja técnica/resumen comercial). (Consulta la ficha del proveedor que corresponda a tu modelo para la versión exacta del pinout y tensión de operación).

IA`s que ayudaron al desarrollo del proyecto: Chatgpt (<https://chatgpt.com/>), Gemini (<https://gemini.google.com/?hl=es>), Claude (<https://claude.ai/new>) y DeepSeck (<https://www.deepseek.com/>)