PROTORP

Carpeta técnica

Año de desarrollo: 2025

E.E.T. Nº7 de Quilmes

Sitio web: https://protorp2025.netlify.app/

Trello: https://trello.com/b/dudjmzhY/kanban

GitHub: https://github.com/impatrq/Protorp

Escuela de Educación Técnica Nº7 “Taller Regional Quilmes”

Especialidad: Avionica

---

1. Presentación del equipo

1.1 Integrantes del proyecto PROTORP

Joaquín González Nogueira

Santiago Nicolás Vila

1.2 Tiempo del proyecto

Cada integrante trabajó un promedio de 12 horas semanales en el proyecto.

---

2. Introducción

2.1 Objetivo del proyecto

PROTORP consiste en el desarrollo de los sistemas **ATS (Automatic Train Stop)** y **ATP (Automatic Train Protection)** a escala. Este proyecto tiene dos objetivos principales:

- Controlar las formaciones que circulan sobre la vía y la forma en la que se respetan los espacios para cada una, mejorando la seguridad del personal y reduciendo la posibilidad de errores humanos.
- Servir como entrenamiento para aspirantes a maquinistas, recreando distintas situaciones ferroviarias en un entorno seguro.

2.2 Descripción y Funcionamiento

La solución consiste en una maqueta ferroviaria que integra los principios de los sistemas ATS y ATP. El circuito reproduce un entorno ferroviario a escala, permitiendo controlar la circulación de los trenes y garantizar la seguridad de la operación.

2.4 Segmento destino y alcance

El proyecto está orientado principalmente a personas que deseen comprender el funcionamiento de los sistemas ATS y ATP en un entorno educativo y controlado. En cuanto al alcance geográfico, el sistema se proyecta a nivel nacional, dado que estos sistemas aún no se implementan ampliamente en Argentina.

---

3. Sistemas embebidos

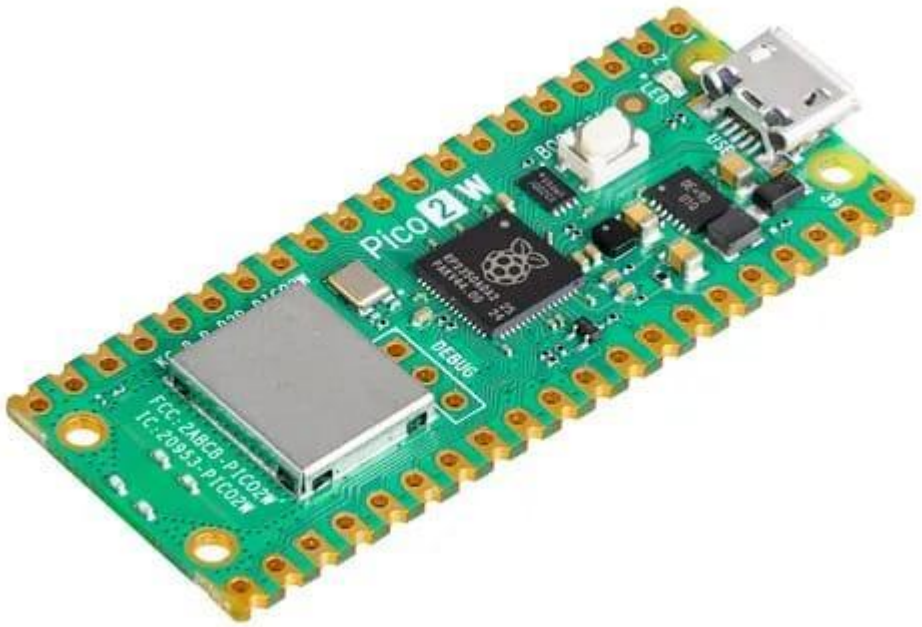
3.1 Microcontroladores y microprocesadores utilizados

ESP32 (MicroPython)

Función: Controla la locomotora, gestiona el movimiento mediante un driver PWM, recibe comandos por BLE (NRF Connect) y transmite datos de velocidad y estado hacia las Raspberry Pi Pico mediante UART. Además, recibe señales RF433 MHz desde la vía para frenar o reducir la velocidad según la condición del tramo.

Raspberry Pi Pico 2W (MicroPython)

Función: Supervisa sensores de ocupación, controla semáforos (LEDs), comunica estados por Wi-Fi y envía datos a la locomotora vía RF433 MHz.



3.2 Placas de desarrollo

- ESP32 (módulo con BLE y Wi-Fi integrado).
- Raspberry Pi Pico W (RP2040) con conectividad inalámbrica 2.4 GHz.

3.3 Software utilizados para el desarrollo

- **Thonny:** IDE para MicroPython, usado para cargar librerías y scripts.
- **MicroPython:** Firmware liviano de Python para microcontroladores, utilizado en las placas ESP32 y Raspberry Pi Pico 2W.

3.4 Diagrama en bloque de la solución

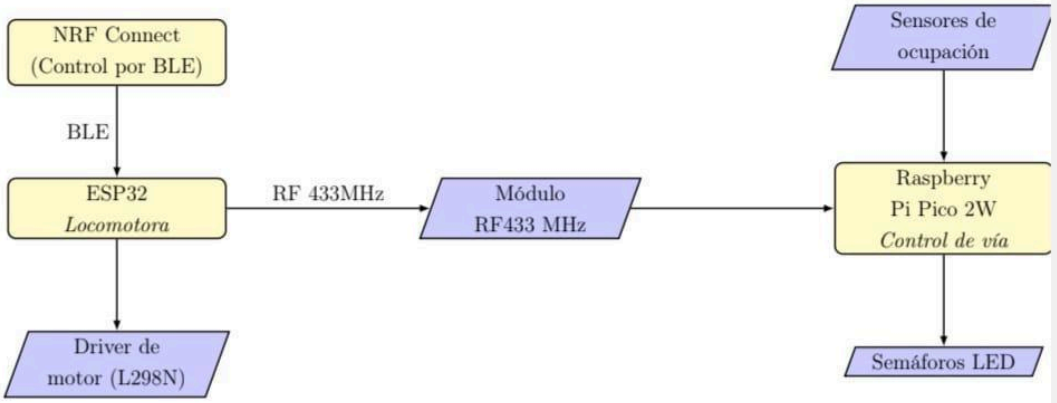


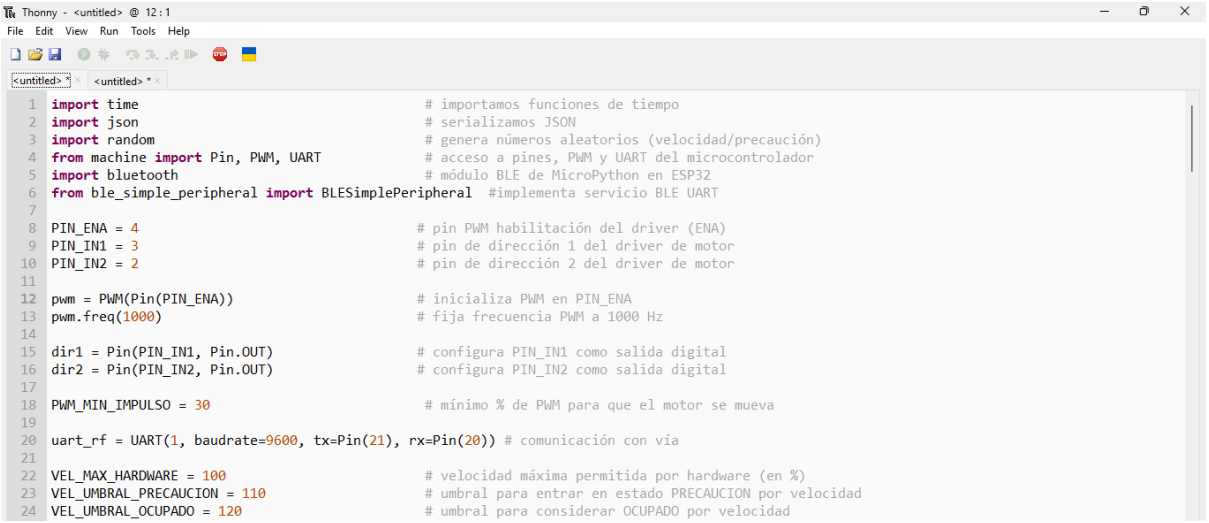
Figura 1: Diagrama en bloques del sistema embebido PROTORP.

3.5 Lenguajes de programación usados

- **MicroPython:** utilizado en ESP32 y Raspberry Pi Pico 2W para control de periféricos, BLE, UART, RF433 MHz y Wi-Fi.
- **Python:** empleado para depuración y pruebas desde PC.

3.6 Capturas del código

ESP32




```python
import time                                 # importamos funciones de tiempo
import json                                 # serializamos JSON
import random                               # genera números aleatorios (velocidad/precaución)
from machine import Pin, PWM, UART          # acceso a pines, PWM y UART del microcontrolador
import bluetooth                            # módulo BLE de MicroPython en ESP32
from ble_simple_peripheral import BLESimplePeripheral  #implementa servicio BLE UART

PIN_ENA = 4                                 # pin PWM habilitación del driver (ENA)
PIN_IN1 = 3                                 # pin de dirección 1 del driver de motor
PIN_IN2 = 2                                 # pin de dirección 2 del driver de motor

pwm = PWM(Pin(PIN_ENA))                     # inicializa PWM en PIN_ENA
pwm.freq(1000)                              # fija frecuencia PWM a 1000 Hz

dir1 = Pin(PIN_IN1, Pin.OUT)                # configura PIN_IN1 como salida digital
dir2 = Pin(PIN_IN2, Pin.OUT)                # configura PIN_IN2 como salida digital

PWM_MIN_IMPULSO = 30                         # mínimo % de PWM para que el motor se mueva

uart_rf = UART(1, baudrate=9600, tx=Pin(21), rx=Pin(20)) # comunicación con vía

VEL_MAX_HARDWARE = 100                      # velocidad máxima permitida por hardware (en %)
VEL_UMBRAL_PRECAUCION = 110                 # umbral para entrar en estado PRECAUCION por velocidad
VEL_UMBRAL_OCUPADO = 120                    # umbral para considerar OCUPADO por velocidad
```

```python
ID_LOCO = 7                                  # ID único de la locomotora
NOMBRE_LOCO = "PROTORP"                      # nombre que anuncia por BLE

ble = bluetooth.BLE()
sp = BLESimplePeripheral(ble, name=NOMBRE_LOCO) # crea el peripheral BLE con servicio UART

vel_bt = 0                                   # velocidad deseada recibida vía BLE (inicialmente 0)
estado_seguridad = "LIBRE"                   # estado interno de seguridad inicial
vel_precaucion_fija = 0                      # valor fijado aleatorio cuando se aplica PRECAUCIÓN por vel

def set_motor_speed(vel):
    vel_abs = abs(vel)                       # valor absoluto de la velocidad solicitada

    if vel > 0:
        dir1.value(1)                        # dirección hacia adelante
        dir2.value(0)
    elif vel < 0:
        dir1.value(0)                        # dirección hacia atrás
        dir2.value(1)
    else:
        dir1.value(0)                        # detener motor (no hay dirección)
        dir2.value(0)
        pwm.duty(0)                          # ciclo de trabajo 0 -> motor parado
        return
```

```python
    if vel_abs > 0 and vel_abs < PWM_MIN_IMPULSO:
        vel_a_usar = PWM_MIN_IMPULSO         # mínimo pulso si vel > 0 pero pequeña
    else:
        vel_a_usar = vel_abs

    duty = int(min(vel_a_usar, VEL_MAX_HARDWARE) * 1023 / 100)  # convierte % a duty (0-1023)
    pwm.duty(duty)                           # aplica duty al PWM

def on_rx(data):
    global vel_bt, estado_seguridad, vel_precaucion_fija
    try:
        new_vel = int(data.decode().strip())   # decodifica bytes BLE -> string -> int

        if -150 <= new_vel <= 150:             # valida rango de velocidad aceptable
            vel_bt = new_vel

            vel_abs_bt = abs(vel_bt)

            if vel_abs_bt < VEL_UMBRAL_PRECAUCION or vel_abs_bt > VEL_UMBRAL_OCUPADO:
                vel_precaucion_fija = 0      # limpia la vel fija si BLE fuera de rango precaucion

            if vel_abs_bt > VEL_UMBRAL_OCUPADO:
                estado_seguridad = "OCUPADO"  # estado interno por velocidad demasiado alta
            elif vel_abs_bt >= VEL_UMBRAL_PRECAUCION:
```

```
                estado_seguridad = "OCUPADO"  # estado interno por velocidad demasiado alta
            elif vel_abs_bt >= VEL_UMBRAL_PRECAUCION:
                estado_seguridad = "PRECAUCION"
            else:
                estado_seguridad = "LIBRE"

            print(f"Velocidad deseada (BLE): {vel_bt}")      # debug
            print(f"Estado de Seguridad Interna (VELOCIDAD): {estado_seguridad}")

    except Exception as e:
        print(f"Error al procesar datos BLE: {e}")

sp.on_write(on_rx)                                  # registra callback para escrituras BLE

def receive_security_status():
    global senal_via_uart
    senal_via_uart = "VERDE"                        # valor por defecto si no hay dato UART
    if uart_rf.any():                               # si hay datos en el buffer UART
        try:
            line = uart_rf.readline()               # lee una línea terminada en '\n'
            if line:
                data_str = line.decode('utf-8').strip()
                data = json.loads(data_str)
                new_status = data.get("signal", "VERDE") # extrae el campo "signal"
```

```

                if new_status in ["ROJO", "AMARILLO", "VERDE"]:
                    senal_via_uart = new_status
                    print(f"Estado de bloqueo REAL (VÍA - UART): {senal_via_uart}")

        except Exception as e:
            print(f"Error al decodificar JSON UART: {e}. Limpiando buffer.")
            while uart_rf.any():                    # limpia buffer si hubo un error
                uart_rf.read(1)
            time.sleep_ms(10)

def send_loco_data(vel_aplicada):
    payload = {
        "id": ID_LOCO,                              # agrega identificador de la locomotora
        "vel": vel_aplicada                         # velocidad aplicada reportada
    }
    mensaje_tx = json.dumps(payload) + "\n"         # serializa a JSON + newline

    try:
        uart_rf.write(mensaje_tx.encode('utf-8')) # envía por UART a la vía
    except Exception as e:
        pass                                        # si falla, no rompe el loop principal

print(f"--- Locomotora {NOMBRE_LOCO} ID {ID_LOCO} Iniciada ---")  # mensaje de arranque
```

```

while True:
    receive_security_status()                       # lee si la vía envió estado ROJO/AMARILLO/VERDE

    vel_aplicada = vel_bt                           # por defecto aplicar velocidad por BLE
    estado_final = "LIBRE"
    vel_aplicada_print = str(vel_bt)

    if senal_via_uart == "ROJO":                    # si la vía esta bloqueada -> parada inmediata
        vel_aplicada = 0
        vel_aplicada_print = "0"
        estado_final = "BLOQUEADA (VÍA ROJA)"

    elif senal_via_uart == "AMARILLO":              # si la vía esta en precaución -> se limita la velocidad
        vel_limit = 30
        if abs(vel_bt) > vel_limit:
            vel_aplicada = int((vel_bt / abs(vel_bt)) * vel_limit)  # mantiene signo, limita mag
        else:
             vel_aplicada = vel_bt
        vel_aplicada_print = str(vel_aplicada)
        estado_final = "PRECAUCION (VÍA AMARILLA)"

    elif senal_via_uart == "VERDE":                 # vía libre -> aplicar reglas internas
        if estado_seguridad == "OCUPADO":           # si por velocidad interna es OCUPADO -> parar
```

```
            vel_aplicada = 0
            vel_aplicada_print = "0"
            estado_final = "BLOQUEADA (VEL > 120)"

        elif estado_seguridad == "PRECAUCION":    # si PRECAUCIÓN por velocidad -> fija valor
            if vel_precaucion_fija == 0:
                vel_precaucion_fija = random.randint(50, 60)  # fija valor aleatorio entre 50-60

            vel_aplicada = vel_precaucion_fija
            if vel_bt < 0:
                vel_aplicada *= -1                  # conserva sentido si velocidad BLE negativa

            vel_aplicada_print = str(vel_aplicada)
            estado_final = "PRECAUCION (VEL) APLICADA"

        else:
            vel_aplicada = vel_bt
            vel_aplicada_print = "---"
            estado_final = "LIBRE"

    else:
        vel_aplicada = 0
        vel_aplicada_print = "0"
        estado_final = "DESCONOCIDO"
```

```
            estado_final = "PRECAUCION (VEL) APLICADA"

        else:
            vel_aplicada = vel_bt
            vel_aplicada_print = "---"
            estado_final = "LIBRE"

    else:
        vel_aplicada = 0
        vel_aplicada_print = "0"
        estado_final = "DESCONOCIDO"

    if estado_final == "LIBRE":
        set_motor_speed(vel_bt)                     # aplica directamente velocidad BLE
        vel_aplicada_send = vel_bt
    else:
        set_motor_speed(vel_aplicada)               # aplica velocidad determinada por reglas
        vel_aplicada_send = vel_aplicada

    send_loco_data(vel_aplicada_send)               # reporta por UART la velocidad aplicada

    print(f"Motor | Estado Final: {estado_final} | Deseada (BLE): {vel_bt} | Aplicada: {vel_aplicada_print}")
    time.sleep_ms(50)                               # corta espera -> loop ~20 Hz

```

Raspberry Pi Pico 2W

```
import urequests as requests                    # cliente HTTP (usado para POST al servidor)
import time
import machine                                  # funciones de máquina: reset etc.
from machine import Pin, UART                   # control de pines y UART hardware
import network                                  # gestión de Wi-Fi en Pico W
import json

WIFI_SSID = "Cooperadora Profesores"            # SSID de la red Wi-Fi
WIFI_PASSWORD = "Profes_IMPA_2022"              # contraseña Wi-Fi

PICO_ID = 3                                     # identificador de la Pico (para el servidor)
TRACK_SEGMENT = "V1_3"                          # nombre del tramo en la maqueta

SERVER_IP = "192.168.111.54"                    # IP del servidor Flask
SERVER_PORT = 5000                              # puerto del servidor

BASE_URL = f"http://{SERVER_IP}:{SERVER_PORT}" # URL base del servidor
SERVER_UPDATE_URL = f"{BASE_URL}/update_pico/{PICO_ID}"  # endpoint para actualizar estado

PIN_SENSOR = 15                                 # pin donde está el sensor de ocupación
sensor_ocupacion = Pin(PIN_SENSOR, Pin.IN, Pin.PULL_UP)  # sensor como entrada con PULL-UP

SEMAFORO_1 = {                                  # mapa de pines para semáforo 1
    'ROJO': Pin(21, Pin.OUT),
```

```
    'AMARILLO': Pin(20, Pin.OUT),
    'VERDE': Pin(19, Pin.OUT)
}
SEMAFORO_2 = {                                  # mapa de pines para semáforo 2
    'ROJO': Pin(18, Pin.OUT),
    'AMARILLO': Pin(17, Pin.OUT),
    'VERDE': Pin(16, Pin.OUT)
}

uart0 = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))   # UART0: enlace entre loco y vía
uart1 = UART(1, baudrate=9600, tx=Pin(4), rx=Pin(5))   # UART1: segundo puerto UART
UARTS = [uart0, uart1]                                 # lista de UARTs para leer
wlan = network.WLAN(network.STA_IF)                    # objeto WLAN (station mode)

def connect_wifi():
    """Conecta la Pico W a la red Wi-Fi."""
    wlan.active(True)                                   # activa interfaz Wi-Fi
    if not wlan.isconnected():
        print(f"Intentando conectar a {WIFI_SSID}...")
        wlan.connect(WIFI_SSID, WIFI_PASSWORD)          # inicia conexión
        max_wait = 10
        while max_wait > 0 and not wlan.isconnected(): # espera hasta conectarse (timeout simple)
            time.sleep(1)
            print('.', end='')
```

```
            max_wait -= 1

    if not wlan.isconnected():
        print("\n Conexión fallida. Reiniciando en 5s.")
        time.sleep(5)
        machine.reset()                                 # reinicia si no pudo conectar
    else:
        status = wlan.ifconfig()
        print("\n Conexión exitosa.")
        print('IP local:', status[0])                   # imprime IP obtenida

def actualizar_semaforos(nombre_estado):
    """Enciende el color correspondiente en ambos semáforos."""
    print(f"Estado de Semáforo: {nombre_estado}")

    for semaforo in [SEMAFORO_1, SEMAFORO_2]:
        semaforo['ROJO'].value(0)                       # apaga rojo
        semaforo['AMARILLO'].value(0)                   # apaga amarillo
        semaforo['VERDE'].value(0)                      # apaga verde

        if nombre_estado == 'VERDE':
            semaforo['VERDE'].value(1)                  # enciende verde
        elif nombre_estado == 'AMARILLO':
            semaforo['AMARILLO'].value(1)               # enciende amarillo
```

```
        elif nombre_estado == 'ROJO':
            semaforo['ROJO'].value(1)                   # enciende rojo

def determine_signal_state(is_occupied):
    """Devuelve 'ROJO' si el tramo está ocupado, 'VERDE' si está libre."""
    if is_occupied:
        return 'ROJO'
    else:
        return 'VERDE'

def get_loco_data():
    """Lee datos JSON de la locomotora desde los UART configurados."""
    for uart in UARTS:
        if uart.any():                                  # si hay datos disponibles
            try:
                line = uart.readline()                  # lee hasta '\n'
                if line:
                    data_str = line.decode('utf-8').strip()
                    data = json.loads(data_str)
                    loco_id = data.get('id', 0)         # extrae id
                    loco_speed = data.get('vel', 0)     # extrae velocidad
                    print(f"  [UART Recibido] ID: {loco_id}, Vel: {loco_speed}")
                    return loco_id, loco_speed
            except Exception as e:
```

```
                    loco_id = data.get('id', 0)         # extrae id
                    loco_speed = data.get('vel', 0)     # extrae velocidad
                    print(f"  [UART Recibido] ID: {loco_id}, Vel: {loco_speed}")
                    return loco_id, loco_speed
            except Exception as e:
                print(f" Error al decodificar UART/JSON: {e}. Limpiando buffer.")
                while uart.any():                       # limpia buffer en caso de error
                    uart.read(1)
                time.sleep_ms(10)
    return 0, 0                                         # si no hubo datos, devuelve 0,0

def send_signal_to_loco(loco_id, signal_state):
    """Envía por UART el estado de la vía a la locomotora en formato JSON."""
    if loco_id != 0:
        signal_to_loco = json.dumps({"signal": signal_state}) + "\n"
        encoded_signal = signal_to_loco.encode('utf-8')
        for uart in UARTS:
            uart.write(encoded_signal)                  # escribe en cada UART disponible
        print(f"  [UART Enviado] Señal a Locomotora {loco_id}: {signal_state}")

def send_update_to_server(is_occupied):
    """Publica el estado del tramo al servidor central mediante HTTP POST."""
    data = {
        "track_segment": TRACK_SEGMENT,
```

```
        "occupied": is_occupied,
        "speed_kmh": 0
    }
    headers = {'Content-Type': 'application/json'}
    print(f"  [HTTP Enviando] Segmento {TRACK_SEGMENT}, Ocupado: {is_occupied}")
    try:
        response = requests.post(SERVER_UPDATE_URL, json=data, headers=headers)
        if response.status_code == 200:
            print(f"  [HTTP OK] {response.json().get('message')}")
        else:
            print(f"  [HTTP Error {response.status_code}] {response.text}")
        response.close()
    except Exception as e:
        print(f"  [HTTP Error de red] No se pudo comunicar con el servidor: {e}")

def main_loop():
    print(f"--- Pico W Tramo Fijo (Segmento: {TRACK_SEGMENT}) INICIADO ---")
    connect_wifi()                                      # establece conexión Wi-Fi al arrancar

    last_occupied_state = not sensor_ocupacion.value()  # lectura inicial del sensor (activo LOW)
    send_update_to_server(last_occupied_state)          # informa al servidor el estado inicial
    signal_state = determine_signal_state(last_occupied_state)
    actualizar_semaforos(signal_state)                  # actualiza semáforos al estado inicial

```

```

        if current_occupied_state != last_occupied_state:
            time.sleep_ms(50)
            if (not sensor_ocupacion.value()) == current_occupied_state:
                signal_state = determine_signal_state(current_occupied_state)
                actualizar_semaforos(signal_state)
                send_update_to_server(current_occupied_state)
                send_signal_to_loco(loco_id, signal_state)
                last_occupied_state = current_occupied_state

        elif loco_id != 0:
            # si no cambió la ocupación pero se recibió info de la loco, se reenvia el estado
            signal_state = determine_signal_state(current_occupied_state)
            send_signal_to_loco(loco_id, signal_state)

        time.sleep(0.5)                                  # espera 0.5 s entre iteraciones

try:
    main_loop()                                         # ejecuta bucle principal
except Exception as e:
    print(f"FATAL ERROR en main_loop: {e}")
    time.sleep(5)
    machine.reset()                                     # reinicia placa en caso de fallo crítico

```

3.7 Periféricos utilizados

- **Driver de motor (L298N):** controla el movimiento del tren mediante PWM y pines digitales IN1/IN2.
- **Módulos RF433 MHz:** comunicación inalámbrica entre vía y locomotora (transmisión de estado de señal).
- **Semáforos LED:** tres LEDs (rojo, amarillo, verde) controlados por GPIO en las Raspberrys.

3.8 Estructuras de datos

La comunicación entre dispositivos utiliza el formato **JSON**, que facilita la lectura y compatibilidad entre módulos.

Mensaje de la locomotora hacia la vía:

```
{
   "id": 7,
   "vel": 85
}
```

- Identifica la locomotora y su velocidad actual.

Mensaje de la vía hacia la locomotora:

```
{
   "signal": "AMARILLO"
}
```

- Indica el estado del semáforo del tramo siguiente.
- **Variables principales:**
  - `vel_bt:` velocidad recibida por BLE.
  - `estado_seguridad:` modo actual de seguridad.
  - `sensor_ocupacion:` determina si el tramo está ocupado.
  - `led_rojo`, `led_amarillo`, `led_verde:` control de semáforos.

---

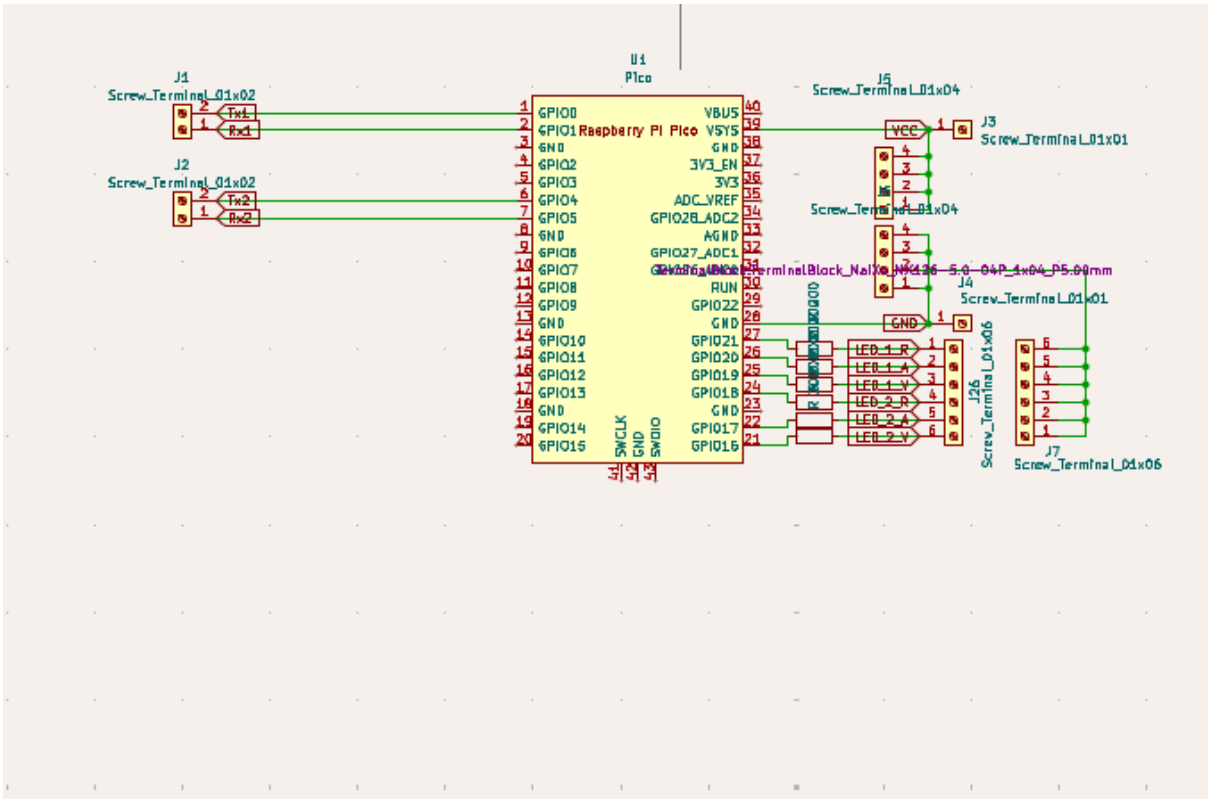
4. Electrónica

4.1 Software usado para esquemáticos y PCB

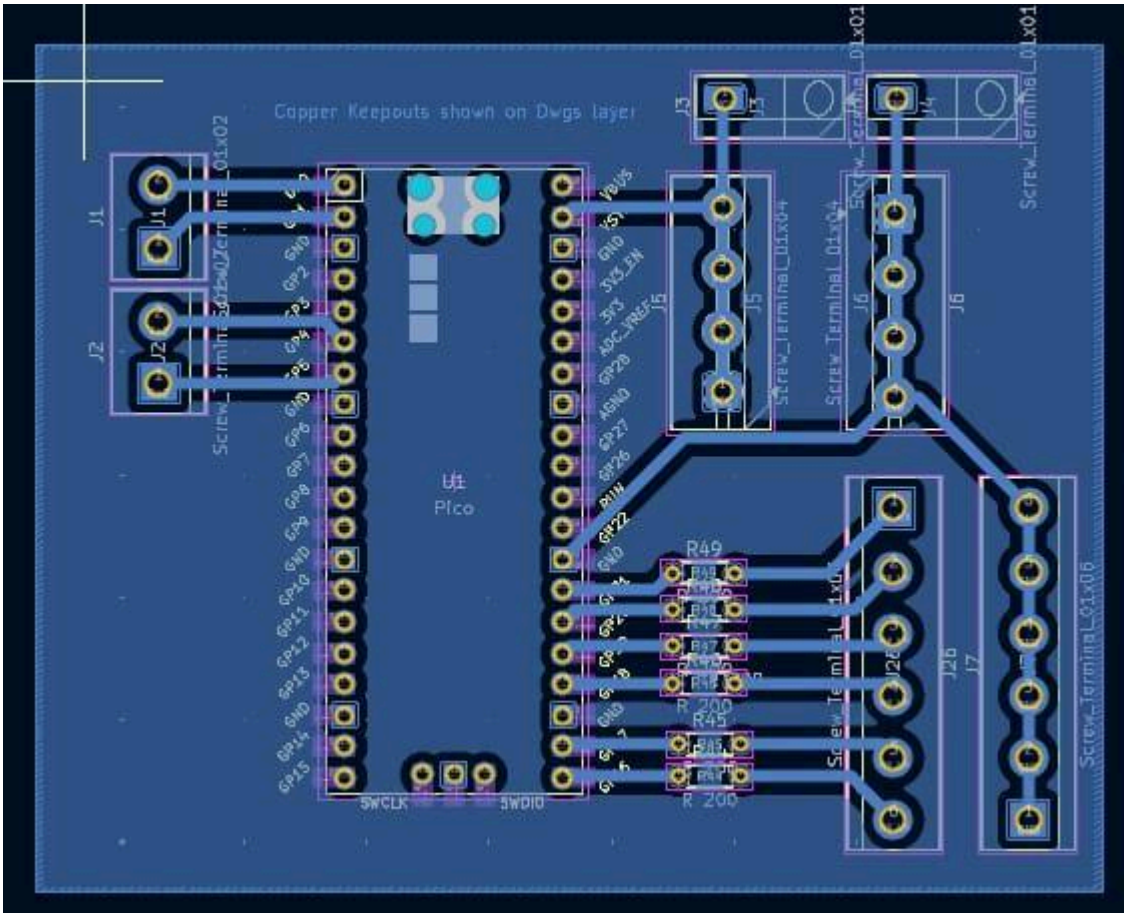
Para el diseño de los esquemáticos y PCB se utilizó el software **KiCad 9**. KiCad permitió desarrollar los diagramas eléctricos y los PCB, garantizando una adecuada organización del sistema electrónico del proyecto.

4.2 Fotos de cada PCB

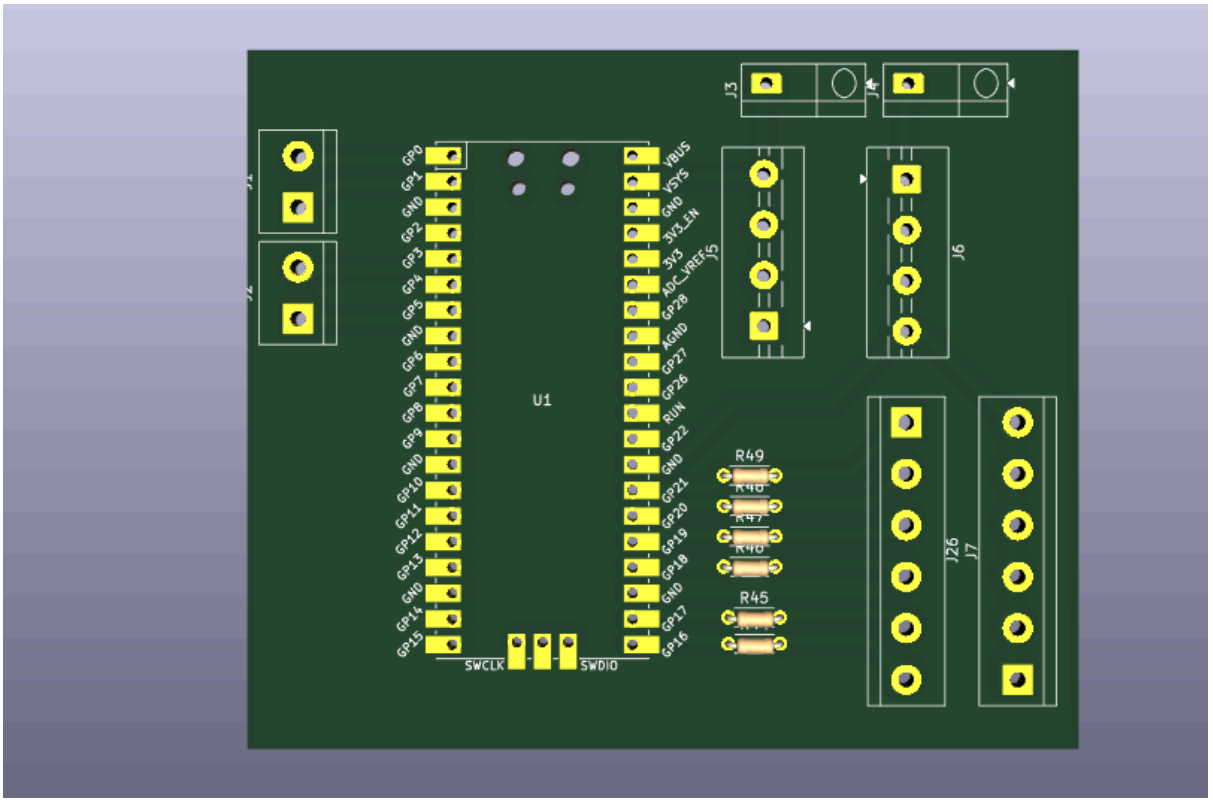
PCB de la placa:



PCB en el editor de placas:



Modelo 3D de la placa:



Placa en fisico:

4.3 Especificación de fuentes de alimentación y potencias

La alimentación de la placa es de 5V con 10 amperes y una potencia total de aproximadamente 0.5 W por componente de control.

4.4 Datasheets

ESP32 datasheet: https://documentation.espressif.com/esp32-c3

Raspberry Pi Pico 2W datasheet:
https://datasheets.raspberrypi.com/picow/pico-2-w-datasheet.pdf

Driver L298N datasheet:
https://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf

Módulos RF 433MHz datasheet:
https://tecmikro.com/modulos-shields/326-modulo-rf-radiofrecuencia-433-mhz.html

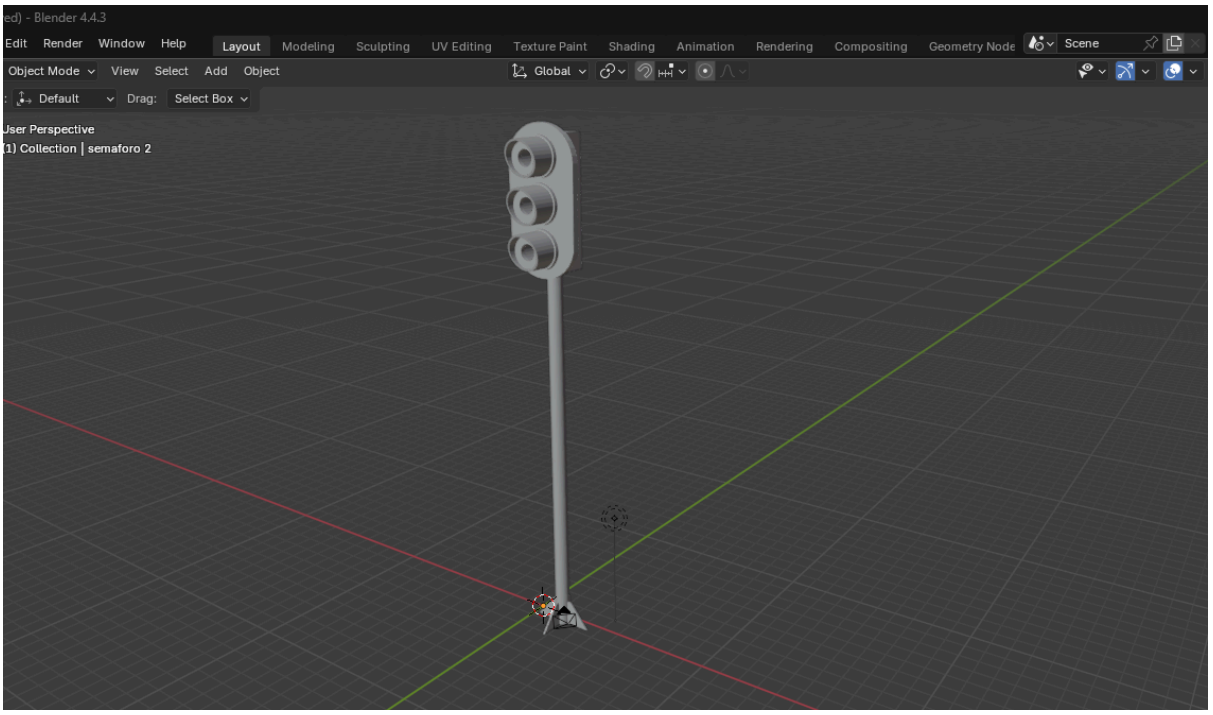
---

5. Estructura

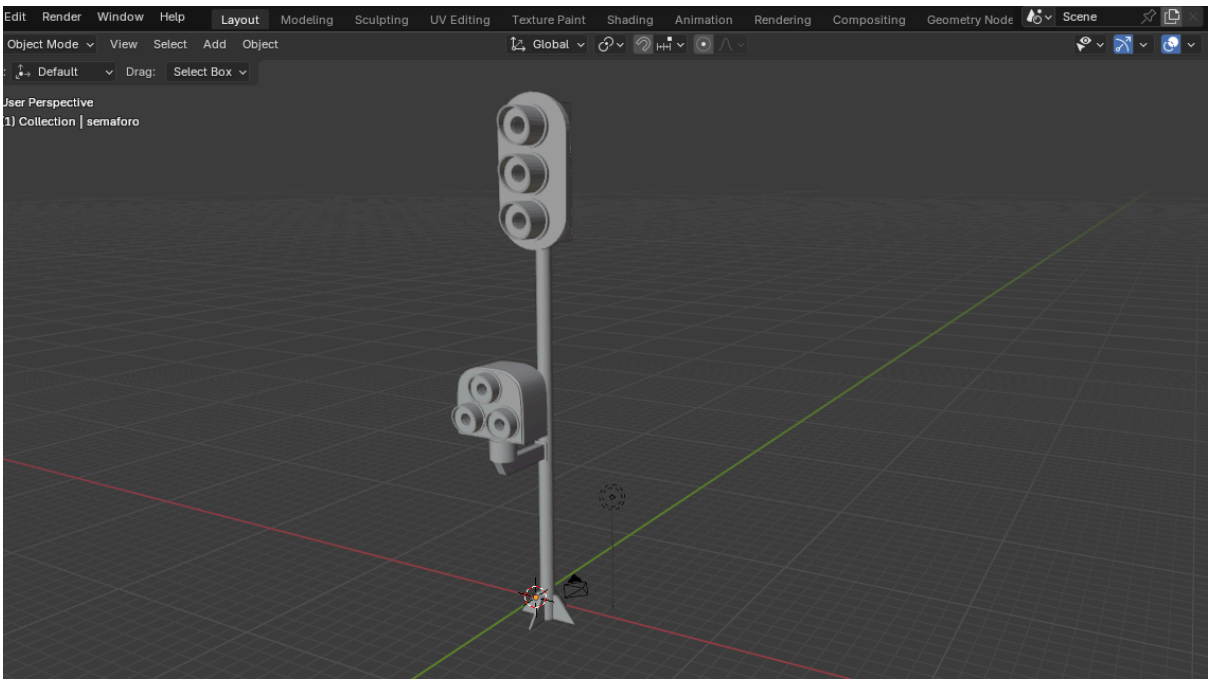
5.1 Software utilizados

Blender: Usamos **Blender** para hacer el modelo 3D de los semaforos, semaforos de cambio de via y el tender del motor.

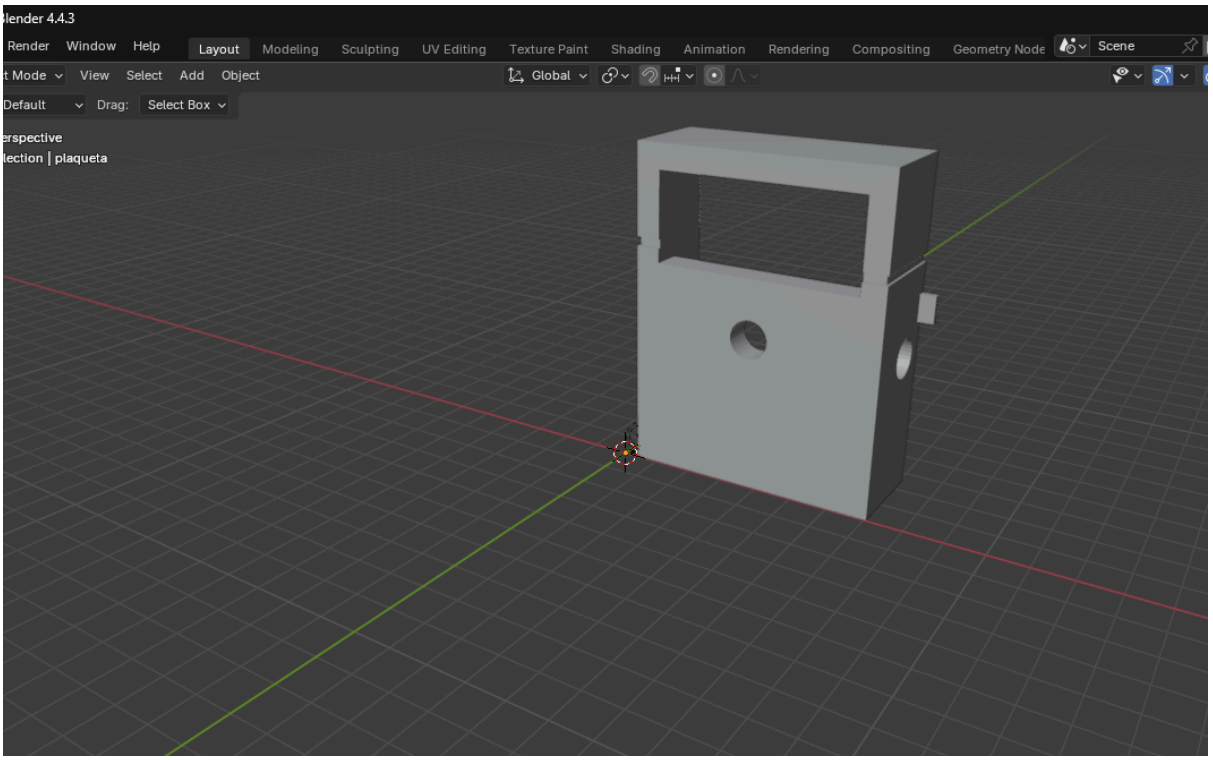
Semaforo normal



Semaforo del cambio de via



Tender del motor



Ultimaker Cura: Usamos **Ultimaker Cura** para imprimir los modelos en 3D.

5.2 Descripcion de cada parte de la estructura

Semáforos

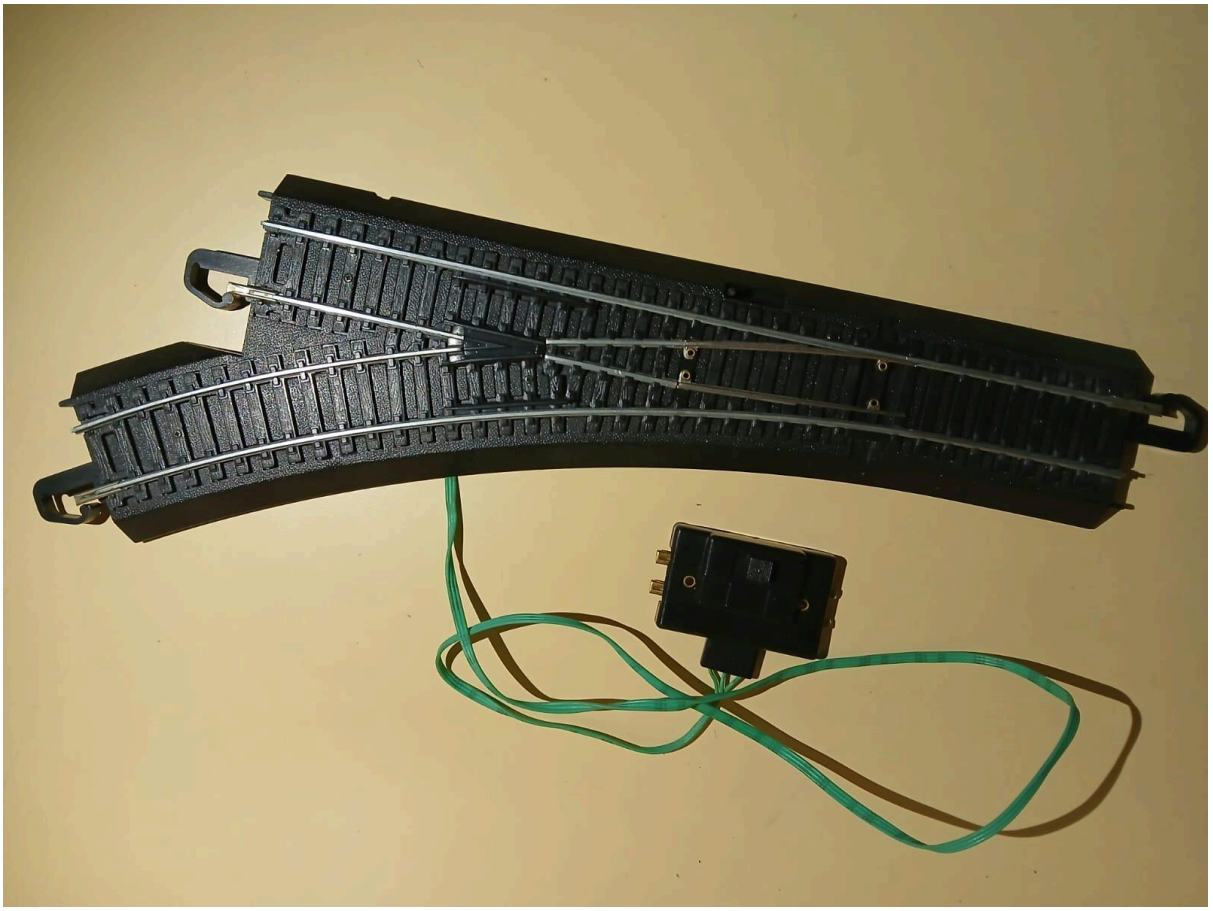
Existen dos tipos de semáforos. El primero es el semáforo regular, que cuenta con las luces roja, amarilla y verde. El segundo es el semáforo de cambio de vía, que es similar al anterior pero incluye una seccion de LED blancos. Estos LED indican la posición del cambio de vía, ya que están directamente conectados con los mecanismos eléctricos que controlan dicho cambio. De esta forma, el semáforo muestra en todo momento la posición real del desvío.

Vías

Las vias están son de escala H0. La maqueta está conformada principalmente por curvas y rectas, y cuenta con cuatro cambios de vía: dos hacia la derecha y dos hacia la izquierda. Estos cambios permiten realizar la rotación de la locomotora.Como se mencionó anteriormente, los cambios de vía están directamente asociados a los LED blancos de los semáforos de cambio de vía, lo que permite visualizar en todo momento la posición del desvío.

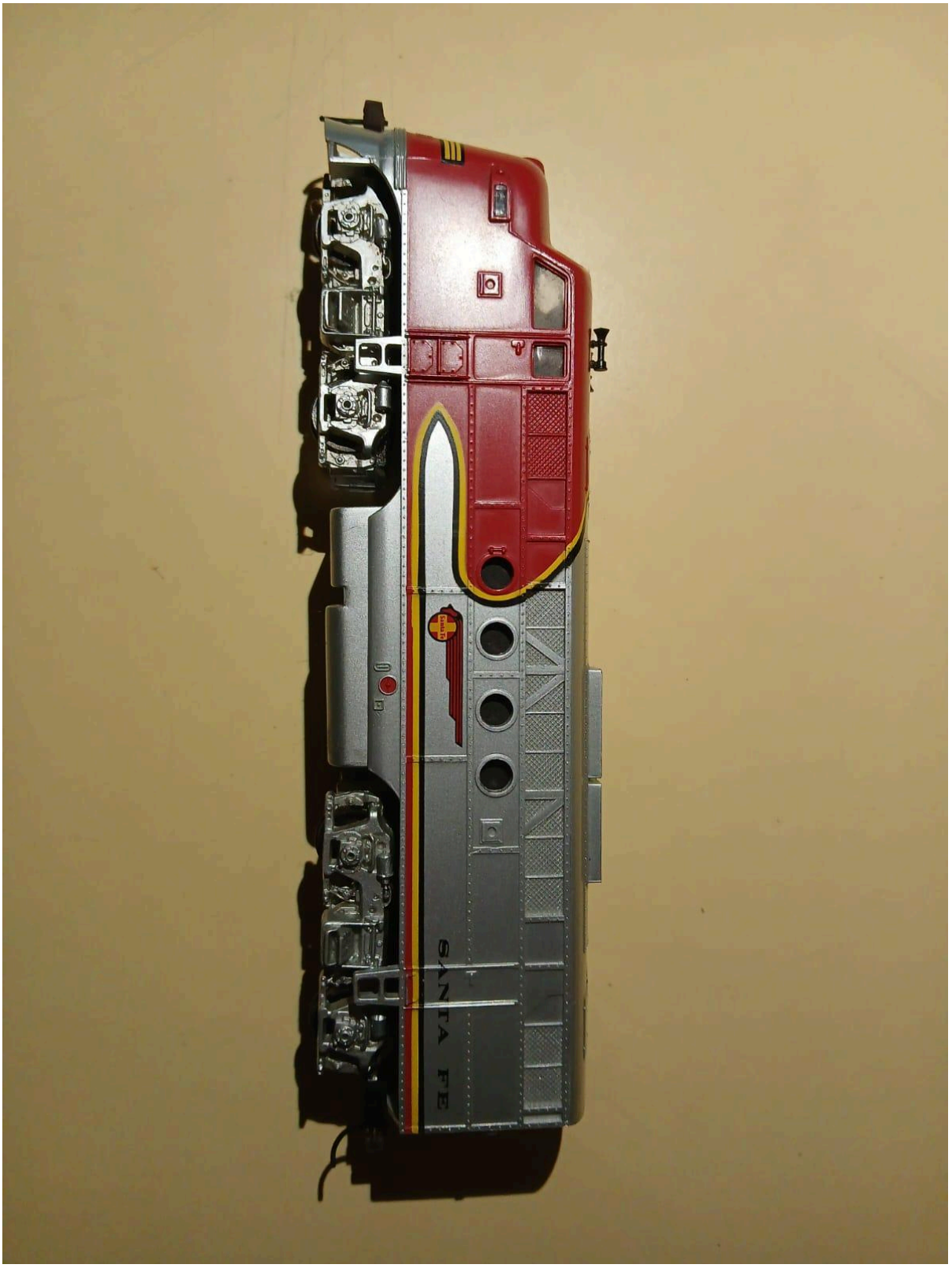
Vias rectas y curvas:

Cambio de via:



Locomotora

La locomotora circula sobre las vias H0. A traves de sus componentes, transmite su informacion y recibe el estado de la via, para determinar si puede seguir, si debe parar o disminuir su velocidad.



Potenciómetro

Esta maqueta se alimenta mediante un transformador de 16 V en corriente alterna (AC). Luego, la tensión se rectifica a corriente continua (DC), permitiendo regular el voltaje de la vía mediante un potenciómetro. El voltaje de salida puede variar entre 0 V y un máximo de

16 V en continua. Además, desde el potenciómetro, como se mencionó anteriormente, también se puede controlar la posición de los cambios de vía, ya que estos son eléctricos.

6. Bibliografia

Licitacion ADIFSE
https://plataforma.adifse.com.ar/uploads/archivo_adjunto/licitacion/20250401_094855-67ebe0b72a73b.pdf?v=2.1.70

ATS Linea roca: https://es.scribd.com/document/318056277/Ats-Linea-Roca