Presentacion final carrito

Cesar Andrade Ramirez

May 2025

1 Introduction

Para este avance lo que intente fue implementar un modulo bluetooth para que todo pudiese funcionar sin necesidad de Wifi, sin exito, pues no fui capaz de lograrlo, en su lugar quise implementar una funcion de la libreria Pygame, la cual me permitio controlar todo el carro con un gamepad bluetooth (Mando Pro de Nintendo Switch, en versiones anteriores use un Dualsense de PS5, pero opte por cambiarlo). Enfrente algunas dificultades, sobre todo porque para el 13 de mayo yo ya tenia listo mi proyecto, pero en algun momento desde ese dia hasta ayer 19 de mayo, intuyo que en el momento de transporte, se desoldo el sensor ultrasonico, mismo que me permitia detener todo el carrito cuando tuviese enfrente un cuerpo a menos de 15cm de distancia, para mi mala fortuna despues de volver a soldarlo y hacer algunas pruebas mas, el sensor dejo de funcionar completamente, aun desconozco el porque, pero quiero hacer mencion de esto, pues en el video subido a YouTube se puede ver claramente que la distancia indicada por el sensor no cambia, y pues esta es la razon.

2 Codigo Arduino

```
#include <WiFi.h>
#include <WebServer.h>
#include <Servo.h>

// WiFi
const char* ssid = "INFINITUM29D7_2.4";
const char* password = "CrX76FTJST";
WebServer server(80);

// Pines del puente H
const int ENA = 7;
const int IN1 = 6;
const int IN2 = 5;
const int ENB = 2;
const int IN3 = 4;
```

```
const int IN4 = 3;

// Servo
Servo servo;
const int servoPin = 17;
int angulo = 90;
const int pulsoMin = 500;
const int pulsoMax = 2500;

// Sensor KY-015
const int DHpin = 0;
byte dat[5];

// Sensor ultrasónico HC-SR04
const int trigPin = 16;
const int echoPin = 15;

// Temporizador
unsigned long ultimoComando = 0;
const unsigned long tiempoLimite = 3000;

void detenerMotores() {
  analogWrite(ENA, 0);
  analogWrite(ENB, 0);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}

// Funciones KY-015
void start_test() {
  digitalWrite(DHpin, LOW);
  delay(30);
  digitalWrite(DHpin, HIGH);
  delayMicroseconds(40);
  pinMode(DHpin, INPUT);
  while (digitalRead(DHpin) == HIGH);
  delayMicroseconds(80);
  if (digitalRead(DHpin) == LOW) delayMicroseconds(80);
  for (int i = 0; i < 5; i++) dat[i] = read_data();
  pinMode(DHpin, OUTPUT);
  digitalWrite(DHpin, HIGH);
}

byte read_data() {
```

```
  byte data = 0;
  for (int i = 0; i < 8; i++) {
    while (digitalRead(DHpin) == LOW);
    delayMicroseconds(30);
    if (digitalRead(DHpin) == HIGH) data |= (1 << (7 - i));
    while (digitalRead(DHpin) == HIGH);
  }
  return data;
}


void handleSensor() {
  start_test();
  long distancia = medirDistancia();

  String response = "{";
  response += "\"temperature\":" + String(dat[2]) + "." + String(dat[3]) + ",";
  response += "\"humidity\":" + String(dat[0]) + "." + String(dat[1]) + ",";
  response += "\"distance\":" + String(distancia);
  response += "}";

  server.send(200, "application/json", response);
}

// Sensor ultrasónico
long medirDistancia() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duracion = pulseIn(echoPin, HIGH, 30000);  // 30 ms timeout
  long distancia = duracion * 0.034 / 2;
  return distancia;
}

void handleUltrasonico() {
  long distancia = medirDistancia();
  String response = "{";
  response += "\"distancia\":" + String(distancia);
  response += "}";
  server.send(200, "application/json", response);
}

void handleMover() {
```

```
  if (server.hasArg("angulo")) {
    angulo = constrain(server.arg("angulo").toInt(), 0, 180);
    servo.write(angulo);
    Serial.println("Servo a: " + String(angulo));
    server.send(200, "text/plain", "Servo movido a " + String(angulo));
    ultimoComando = millis();
  } else {
    server.send(400, "text/plain", "Falta el parámetro 'angulo'");
  }
}

void setupMotores() {
  server.on("/adelante", []() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
    digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
    analogWrite(ENA, 200); analogWrite(ENB, 200);
    ultimoComando = millis();
    server.send(200, "text/plain", "Motores avanzando");
  });

  server.on("/atras", []() {
    digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
    digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
    analogWrite(ENA, 200); analogWrite(ENB, 200);
    ultimoComando = millis();
    server.send(200, "text/plain", "Motores retrocediendo");
  });

  server.on("/girar_derecha", []() {
    digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
    analogWrite(ENA, 200); analogWrite(ENB, 200);
    ultimoComando = millis();
    server.send(200, "text/plain", "Giro a la derecha");
  });

  server.on("/girar_izquierda", []() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
    digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
    analogWrite(ENA, 200); analogWrite(ENB, 200);
    ultimoComando = millis();
    server.send(200, "text/plain", "Giro a la izquierda");
  });

  server.on("/detener", []() {
    detenerMotores();
```

```
    server.send(200, "text/plain", "Motores detenidos");
  });
}

void setup() {
  Serial.begin(9600);

  // Pines motores
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);

  // Pines sensor temp/hum
  pinMode(DHpin, OUTPUT);
  digitalWrite(DHpin, HIGH);

  // Pines ultrasónico
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  // WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi conectado: " + WiFi.localIP().toString());

  // Servo
  servo.attach(servoPin, pulsoMin, pulsoMax);
  servo.write(angulo);

  // Endpoints
  setupMotores();
  server.on("/mover", handleMover);
  server.on("/sensor", handleSensor);
  server.on("/ultrasonico", handleUltrasonico);

  server.begin();
  Serial.println("Servidor iniciado");
}

void loop() {
```

```
  server.handleClient();

  if (millis() - ultimoComando > tiempoLimite) {
    detenerMotores();
  }
}
```

3 Codigo Spyder

```
import tkinter as tk
import requests
import math
import threading
import time
import pygame
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

# === CONFIGURACIÓN ===
pico_ip = "192.168.1.83"
zona_muerta = 0.5
umbral_mov = 0.7
ultimo_comando = None
mover_activo = False
distancia_actual = 100
datos_temperatura = []
datos_humedad = []
timestamplist = []



# === FUNCIONES CONTROL CARRO ===
def enviar_comando(comando):
    global ultimo_comando
    if comando != ultimo_comando:
        try:
            url = f"http://{pico_ip}/{comando}"
            response = requests.get(url)
            if response.status_code == 200:
                print(f"Comando {comando} enviado con éxito")
            else:
                print(f"Error al enviar {comando}: {response.status_code}")
        except Exception as e:
```

```
            print(f"Error de conexión: {e}")
        ultimo_comando = comando

def manejar_tecla(event):

    tecla = event.keysym.lower()
    if tecla == "w":
        enviar_comando("adelante")
    elif tecla == "s":
        enviar_comando("atras")
    elif tecla == "a":
        enviar_comando("girar_izquierda")
    elif tecla == "d":
        enviar_comando("girar_derecha")
    elif tecla == "space":
        enviar_comando("detener")

# === FUNCIONES SERVO ===
def mover_servo(angulo):
    try:
        url = f"http://{pico_ip}/mover?angulo={int(angulo)}"
        requests.get(url)
    except Exception as e:
        print(f"Error al mover el servo: {e}")

def barrido_servo():
    global mover_activo
    mover_activo = True
    while mover_activo:
        for angulo in range(0, 181, 10):
            mover_servo(angulo)
            actualizar_slider_visual(angulo)
            time.sleep(0.5)
            if not mover_activo: return
        for angulo in range(180, -1, -10):
            mover_servo(angulo)
            actualizar_slider_visual(angulo)
            time.sleep(0.5)
            if not mover_activo: return

def iniciar_barrido():
    global mover_activo
    if not mover_activo:  # Evitar crear el hilo si ya está activo
        hilo = threading.Thread(target=barrido_servo)
        hilo.daemon = True
        hilo.start()
```

```
def detener_barrido():
    global mover_activo
    mover_activo = False

def resetear_angulo():
    mover_servo(90)
    actualizar_slider_visual(90)
    label_angulo.config(text="Ángulo: 90°")

# === GAMEPAD ===
def leer_gamepad():
    pygame.init()
    pygame.joystick.init()
    if pygame.joystick.get_count() == 0:
        print("No se detectó ningún gamepad.")
        return
    joystick = pygame.joystick.Joystick(0)
    joystick.init()
    angulo_servo = 90

    time.sleep(1)  # Esperar a que se estabilice el estado de botones
    barrido_activo = False  # Bandera para evitar reinicios múltiples

    while True:
        pygame.event.pump()
        x_axis = joystick.get_axis(0)
        y_axis = joystick.get_axis(1)
        servo_axis = joystick.get_axis(3)
        comando = "ninguno"


        if abs(x_axis) < zona_muerta and abs(y_axis) < zona_muerta:
            comando = "detener"
        elif abs(y_axis) > abs(x_axis):
            comando = "adelante" if y_axis < -umbral_mov else "atras"
        else:
            comando = "girar_derecha" if x_axis > umbral_mov else "girar_izquierda"

        if joystick.get_button(0):  # Cuadrado → detener
            comando = "detener"
        enviar_comando(comando)

        # Control del servo con joystick derecho
        if abs(servo_axis) > 0.1:
            angulo_servo -= servo_axis * 5
```

```
            angulo_servo = max(0, min(180, angulo_servo))
            mover_servo(angulo_servo)
            actualizar_slider_visual(angulo_servo)

        # Control del barrido (evita reiniciar si ya está activo)
        if joystick.get_button(3) and not barrido_activo:  # Triángulo → iniciar
            iniciar_barrido()
            barrido_activo = True
        if joystick.get_button(2):  # Cuadrado → detener
            detener_barrido()
            barrido_activo = False
        if joystick.get_button(1):  # Círculo → resetear
            resetear_angulo()

        time.sleep(0.1)


# === SENSOR Y GRAFICA ===
def actualizar_sensores():
    global distancia_actual
    try:
        # Leer sensor DHT (Temperatura y Humedad)
        r1 = requests.get(f"http://{pico_ip}/sensor", timeout=3)
        if r1.ok:
            datos = r1.json()
            temperatura = datos['temperature']
            humedad = max(0, datos["humidity"] - 5)

            # Limitar la humedad al rango [0, 100]
            humedad = max(0, min(100, humedad))  # Limitar humedad entre 0 y 100
            print(f"Humedad recibida: {humedad}")  # Agregar log para ver el valor real

            label_temp.config(text=f"Temperatura: {temperatura} °C")
            label_hum.config(text=f"Humedad: {humedad} %")

            # Guardar los datos en las listas
            datos_temperatura.append(temperatura)
            datos_humedad.append(humedad)
            timestamplist.append(time.strftime("%H:%M:%S"))

            # Actualizar la gráfica
            ax.clear()
            ax.plot(timestamplist[-20:], datos_temperatura[-20:], label="Temp (°C)")
            ax.plot(timestamplist[-20:], datos_humedad[-20:], label="Humedad (%)")
            ax.set_title("Sensor DHT")
            ax.set_xlabel("Hora")
```

```
            ax.set_ylabel("Valor")
            ax.legend()
            ax.grid(True)
            canvas_grafica.draw()

        # Leer sensor ultrasónico
        r2 = requests.get(f"http://{pico_ip}/ultrasonico", timeout=5)
        if r2.ok:
            datos = r2.json()
            distancia_actual = datos['distancia']
            label_distancia.config(text=f"Distancia: {distancia_actual} cm")

        if distancia_actual is not None and 1 < distancia_actual < 15:
            if ultimo_comando != "detener":
                enviar_comando("detener")
            label_alerta.config(text="¡Obstáculo detectado!")
            time.sleep(0.1)
            root.after(2000, actualizar_sensores)
            return

        label_alerta.config(text="Sin Obstáculos")  # Limpia alerta si no hay obstáculo
        root.after(2000, actualizar_sensores)

    except Exception as e:
        print(f"Error al leer sensores: {e}")
        root.after(2000, actualizar_sensores)  # Reintentar después de 2 segundos


# === INTERFAZ TKINTER ===
root = tk.Tk()
root.title("Control del Carrito con Sensores")
root.geometry("1400x900")
root.configure(bg="#242323")

# Estructura
frame_izq = tk.Frame(root, bg="#242323")
frame_izq.pack(side="left", fill="y", padx=10, pady=10)
frame_der = tk.Frame(root, bg="#242323")
frame_der.pack(side="right", fill="both", expand=True, padx=10, pady=10)

# Botones
tk.Button(frame_izq, text="Iniciar", bg="#0B5ED7", fg="white",
          font=("Minecraft Dungeons", 14), width=20, command=lambda: print("Iniciado")).pack
tk.Button(frame_izq, text="Detener", bg="#DC3545", fg="white",
          font=("Minecraft Dungeons", 14), width=20, command=lambda: print("Detenido")).pack
tk.Button(frame_izq, text="Exportar a Excel", bg="#28A745", fg="white",
```

```
          font=("Minecraft Dungeons", 14), width=20, command=lambda: pd.DataFrame({
              "Tiempo": timestamplist,
              "Temperatura": datos_temperatura,
              "Humedad": datos_humedad
          }).to_excel("lecturas.xlsx", index=False)).pack(pady=5)
# Botones adicionales para el control del servomotor
tk.Button(frame_izq, text="Iniciar Barrido", bg="#FF9900", fg="white",
          font=("Minecraft Dungeons", 14), width=20, command=iniciar_barrido).pack(pady=5)
tk.Button(frame_izq, text="Detener Barrido", bg="#FF0000", fg="white",
          font=("Minecraft Dungeons", 14), width=20, command=detener_barrido).pack(pady=5)
tk.Button(frame_izq, text="Resetear Ángulo", bg="#28A745", fg="white",
          font=("Minecraft Dungeons", 14), width=20, command=resetear_angulo).pack(pady=5)
# Cerrar correctamente
def cerrar():
    global mover_activo
    mover_activo = False
    mover_servo(90)
    root.quit()
    root.destroy()

# Botón para cerrar la aplicación
tk.Button(frame_izq, text="Cerrar", bg="#DC3545", fg="white",
          font=("Minecraft Dungeons", 14), width=20, command = cerrar).pack(pady=5)


# Etiquetas
label_temp = tk.Label(frame_izq, text="Temperatura: -- °C", font=("Arial", 14), bg="#f0f0f0'
label_temp.pack(pady=5)
label_hum = tk.Label(frame_izq, text="Humedad: -- %", font=("Arial", 14), bg="#f0f0f0")
label_hum.pack(pady=5)
label_distancia = tk.Label(frame_izq, text="Distancia: -- cm", font=("Arial", 14))
label_distancia.pack(pady=5)
label_alerta = tk.Label(frame_izq, text="", font=("Arial", 12), fg="red")
label_alerta.pack()

# Gráfica
frame_grafica = tk.Frame(frame_der, bg="#242323")
frame_grafica.pack(fill="both", expand=True)
fig, ax = plt.subplots(figsize=(7, 4), dpi=100)
canvas_grafica = FigureCanvasTkAgg(fig, master=frame_grafica)
canvas_grafica.draw()
canvas_grafica.get_tk_widget().pack(fill="both", expand=True)

# Slider circular visual
canvas = tk.Canvas(frame_izq, width=250, height=250, bg="#dddddd")
canvas.pack(pady=10)
```

```
centro_x, centro_y, radio = 125, 125, 100
radio_marcador = 8
canvas.create_oval(centro_x - radio, centro_y - radio, centro_x + radio, centro_y + radio)
brazo = canvas.create_line(centro_x, centro_y, centro_x, centro_y - (radio - 50), width=3, f
marcador = canvas.create_oval(centro_x - radio_marcador, centro_y - radio,
                              centro_x + radio_marcador, centro_y - radio, fill="red")
label_angulo = tk.Label(frame_izq, text="Ángulo: 90°", font=("Arial", 14), bg="#dddddd")
label_angulo.pack()
canvas.bind("<B1-Motion>", lambda e: actualizar_angulo(e))

def actualizar_angulo(event):
    x, y = event.x - centro_x, centro_y - event.y
    angulo = math.degrees(math.atan2(y, x))
    if angulo < 0: angulo += 360
    if 0 <= angulo <= 180:
        actualizar_slider_visual(angulo)
        mover_servo(angulo)

def actualizar_slider_visual(angulo):
    marcador_x = centro_x + radio * math.cos(math.radians(angulo))
    marcador_y = centro_y - radio * math.sin(math.radians(angulo))
    canvas.coords(marcador, marcador_x - radio_marcador, marcador_y - radio_marcador,
                            marcador_x + radio_marcador, marcador_y + radio_marcador)
    brazo_x = centro_x + (radio - 50) * math.cos(math.radians(angulo))
    brazo_y = centro_y - (radio - 50) * math.sin(math.radians(angulo))
    canvas.coords(brazo, centro_x, centro_y, brazo_x, brazo_y)
    label_angulo.config(text=f"Ángulo: {int(angulo)}°")


root.protocol("WM_DELETE_WINDOW", cerrar)
root.bind("<KeyPress>", manejar_tecla)
root.focus_set()

# Lanzar hilos
threading.Thread(target=leer_gamepad, daemon=True).start()
actualizar_sensores()
root.mainloop()
```

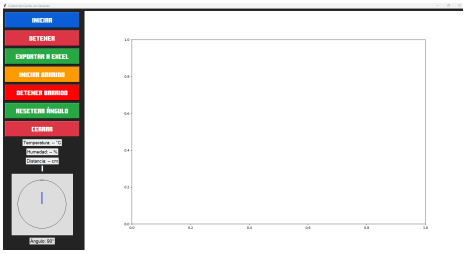

Figure 1: Interfaz