Cognitive Robotics Lab
Lab 8: Create a map data structure with details

Srikar Ganesh 22BAI1080

29-09-2025

Aim:

To create a simple 2D mapping system using a servo-mounted ultrasonic sensor that scans the surrounding area. The system measures distances at various angles, sends the data via serial communication to a computer, and visualizes obstacles in real-time on a 2D map.

Components:

1. **Arduino Board** (e.g., Arduino Uno) – Controls the servo and reads sensor data.
2. **Servo Motor** – Rotates the ultrasonic sensor to scan angles from 0° to 180°.
3. **Ultrasonic Sensor (HC-SR04)** – Measures distances to obstacles in front of it.
4. **Jumper Wires** – Connect Arduino, servo, and ultrasonic sensor.
5. **Power Supply** – USB or external power for Arduino and servo (if high torque needed).
6. **Computer with Python** – Receives serial data and visualizes the 2D map.
7. **Optional: Breadboard** – For easy connections.

Output:

Source Code:

```
#include <Servo.h>

Servo myServo;

const int trigPin = 9;     // HC-SR04 TRIG pin
const int echoPin = 10;    // HC-SR04 ECHO pin
const int servoPin = 3;    // Servo control pin

int servoMin = 0;          // Minimum servo angle
int servoMax = 180;        // Maximum servo angle
int step = 1;              // Servo step

void setup() {
  Serial.begin(9600);
  myServo.attach(servoPin);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

long readDistanceCM() {
  // Send trigger pulse
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read echo pulse
  long duration = pulseIn(echoPin, HIGH, 30000); // timeout 30ms (~5m)
  if (duration == 0) return -1; // no reading

  long distance = duration * 0.034 / 2; // cm
  return distance;
}

void loop() {
  // Sweep servo from 0 to 180
  for (int angle = servoMin; angle <= servoMax; angle += step) {
    myServo.write(angle);
    delay(15); // allow servo to move

    long distance = readDistanceCM();
    Serial.print(angle);
    Serial.print(",");
    Serial.println(distance);
    delay(50); // small delay to avoid flooding
  }

  // Optional: sweep back from 180 to 0
  for (int angle = servoMax; angle >= servoMin; angle -= step) {
    myServo.write(angle);
    delay(15);

    long distance = readDistanceCM();
    Serial.print(angle);
    Serial.print(",");
    Serial.println(distance);
    delay(50);
  }
```

```
}
from PIL import Image, ImageDraw
import math
import serial
import time
import matplotlib.pyplot as plt
import numpy as np

class Map2D:
    def _init_(self, width=500, height=500, scale=10):
        self.width = width
        self.height = height
        self.scale = scale  # e.g. 1 pixel = 1 cm
        self.image = Image.new("RGB", (width, height), "white")
        self.draw = ImageDraw.Draw(self.image)
        self.origin = (width // 2, height // 2)  # Center as (0,0)

    def draw_robot(self, x, y, color="blue"):
        px = self.origin[0] + int(x * self.scale)
        py = self.origin[1] - int(y * self.scale)
        self.draw.ellipse((px-2, py-2, px+2, py+2), fill=color)

    def draw_obstacle(self, x, y, color="black"):
        px = self.origin[0] + int(x * self.scale)
        py = self.origin[1] - int(y * self.scale)
        self.draw.rectangle((px, py, px+1, py+1), fill=color)

    def get_image_array(self):
        return np.array(self.image)

    def save(self, filename="map.png"):
        self.image.save(filename)

map2d = Map2D(width=500, height=500, scale=1)  # 1cm per pixel
map2d.draw_robot(0,0)
# Suppose the robot is at (0,0), facing 0° (upward)
robot_x = 0
robot_y = 0

plt.ion()
fig, ax = plt.subplots()
img_display = ax.imshow(map2d.get_image_array())
plt.title("Live Map")
plt.axis('off')

SERIAL_PORT = 'COM13'      # or '/dev/ttyUSB0' or '/dev/ttyACM0'
```

```
BAUD_RATE = 9600

try:
    ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
    time.sleep(2)  # Wait for Arduino to reset
    print(f"Connected to {SERIAL_PORT} at {BAUD_RATE} baud.\n")

    while True:
        line = ser.readline().decode('utf-8').strip()
        if line:
            try:
                angle_str, dis_str = line.split(',')
                angle = int(angle_str.strip())
                distance = int(dis_str.strip())
                print(angle,distance)

                obs_x = robot_x + distance * math.cos(math.radians(angle))
                obs_y = robot_x + distance * math.sin(math.radians(angle))
                if distance != -1 and distance > 2 and distance < 100: #real distance within a range gets
mapped
                    map2d.draw_obstacle(obs_x,obs_y)

                img_display.set_data(map2d.get_image_array())
                plt.draw()
                plt.pause(0.001)
            except ValueError:
                print(f"Invalid line: {line}")

except serial.SerialException as e:
    print(f"Error opening serial port: {e}")
except KeyboardInterrupt:
    print("Interrupted by user. Exiting...")
finally:
    if 'ser' in locals() and ser.is_open:
        ser.close()
        print("Serial port closed.")


# Save the map
map2d.save("simple_map.png")``
```

Explanation:

The system integrates **face authentication** with **head gesture recognition** to control an Arduino. FaceNet is used to generate embeddings for three registered users—Srikar, Sukesh, and Abhinav. These embeddings are compared against pre-trained models, and if a match above the confidence threshold is found, the system grants access. This ensures that only authenticated users can control the hardware, adding a secure layer to the interaction.

Once authenticated, the system switches to gesture control using OpenCV’s Haar Cascade. It continuously tracks the vertical position of the detected face and compares it with the previous frame. If the face moves upward, the system interprets it as a **Forward** command, while a downward movement is interpreted as a **Backward** command. These commands are then sent via serial communication to the Arduino, which triggers corresponding actions (such as lighting LEDs or driving motors).

Result:

The setup successfully scanned a 180° field in front of the robot, detecting obstacles and plotting them in real-time on a 2D map. The generated map shows positions of walls or objects relative to the robot’s starting point, providing a clear spatial layout. This demonstrates a basic but effective method for environmental mapping using low-cost components and can serve as a foundation for autonomous navigation or obstacle avoidance applications.