**CSE461 : Introduction to Robotics**

****

**Documented Code**

**Project Title :** 3-in-One Obstacle detection, Line following RC Robot

| **Group No : 05, CSE461 Lab Section : 09, Spring 2024** | |
| --- | --- |
| **ID** | **Name** |
| 21101348 | Ajmain Islam |
| 21301410 | Nafis Ashraf Roudra |
| 21301198 | Rafi Shahriyar |
| 21301181 | Neelavro Saha |
|  |  |

# Documented Code

# 

## Raspberry Pi

The raspberry pi runs the python script “robot.py”.

import asyncio

import socketio

import RPi.GPIO as GPIO

import time

# IP address of server

SERVER\_IP = "172.20.10.6"

sio = socketio.AsyncClient()

# Initialize current mode to joystick

current\_mode = "joystick"

pause = False

# Set GPIO mode

GPIO.setmode(GPIO.BCM)

# Define motor and sensor pins

motor1\_clockwise = 9

motor1\_anticlockwise = 11

motor2\_clockwise = 5

motor2\_anticlockwise = 6

TRIG = 21

ECHO = 20

IR\_SENSOR1 = 2

IR\_SENSOR2 = 3

#Servo

GPIO.setup(13,GPIO.OUT)

pwm=GPIO.PWM(13,50)

pwm.start(7)

# Setup motor pins as outputs

GPIO.setup(motor1\_clockwise, GPIO.OUT)

GPIO.setup(motor1\_anticlockwise, GPIO.OUT)

GPIO.setup(motor2\_clockwise, GPIO.OUT)

GPIO.setup(motor2\_anticlockwise, GPIO.OUT)

# Motor functions

# We utilize functions such as forward, backward, left, stop, and right to direct

# the movement of the robot in the respective directions. This is achieved by

# configuring GPIO pins to either a high or low state. For instance, in the case

# of moving forward, the clockwise pin of one motor (motor1) is set to high while

# the anticlockwise pin is set to low. Simultaneously, the clockwise pin of the

# other motor (motor2) is set to low, and the anticlockwise pin is set to high.

# Consequently, both motors rotate in a manner that propels the vehicle forward.

# The same procedure is followed for backward, left, stop, and right functions,

# wherein the motors are adjusted to either a low or high state.

def forward():

GPIO.output(motor1\_clockwise, GPIO.HIGH)

GPIO.output(motor1\_anticlockwise, GPIO.LOW)

GPIO.output(motor2\_clockwise, GPIO.LOW)

GPIO.output(motor2\_anticlockwise, GPIO.HIGH)

def backward():

GPIO.output(motor1\_clockwise, GPIO.LOW)

GPIO.output(motor1\_anticlockwise, GPIO.HIGH)

GPIO.output(motor2\_clockwise, GPIO.HIGH)

GPIO.output(motor2\_anticlockwise, GPIO.LOW)

def left():

GPIO.output(motor1\_clockwise, GPIO.LOW)

GPIO.output(motor1\_anticlockwise, GPIO.LOW)

GPIO.output(motor2\_clockwise, GPIO.LOW)

GPIO.output(motor2\_anticlockwise, GPIO.HIGH)

def right():

GPIO.output(motor1\_clockwise, GPIO.HIGH)

GPIO.output(motor1\_anticlockwise, GPIO.LOW)

GPIO.output(motor2\_clockwise, GPIO.HIGH)

GPIO.output(motor2\_anticlockwise, GPIO.HIGH)

def stop():

GPIO.output(motor1\_clockwise, GPIO.LOW)

GPIO.output(motor1\_anticlockwise, GPIO.LOW)

GPIO.output(motor2\_clockwise, GPIO.LOW)

GPIO.output(motor2\_anticlockwise, GPIO.LOW)

# Setup ultrasonic sensor

GPIO.setup(TRIG, GPIO.OUT)

GPIO.setup(ECHO, GPIO.IN)

# Setup IR sensors

GPIO.setup(IR\_SENSOR1, GPIO.IN)

GPIO.setup(IR\_SENSOR2, GPIO.IN)

# Connect to the server

async def connect\_to\_server():

await sio.connect(f"http://{SERVER\_IP}:3000", auth={"token": "rpiClientToken"})

print("Connected to server")

# Disconnect from server

@sio.event

async def disconnect():

print("Disconnected from server")

# Receive joystick commands from server

@sio.on("joystick")

async def on\_joystick\_input(data):

if data == "FORWARD":

forward()

elif data == "BACKWARD":

backward()

elif data == "LEFT":

left()

elif data == "RIGHT":

right()

elif data == "stop":

stop()

print("Joystick Data:", data)

# Receive mode change commands from server

@sio.on("change\_mode")

async def on\_change\_mode(data):

global current\_mode

current\_mode = data

print(f"Mode changed to {current\_mode}")

# Read IR sensor data

async def read\_ir():

ir1\_state = GPIO.input(IR\_SENSOR1)

ir2\_state = GPIO.input(IR\_SENSOR2)

await sio.emit("ir\_sensor\_data", {"left": ir1\_state, "right": ir2\_state})

print("IR Data:", ir1\_state, ir2\_state)

return ir1\_state, ir2\_state

# Read ultrasonic sensor data

async def read\_ultrasonic():

GPIO.output(TRIG, False)

await asyncio.sleep(0.1)

GPIO.output(TRIG, True)

await asyncio.sleep(0.00001)

GPIO.output(TRIG, False)

pulse\_start = time.time()

while GPIO.input(ECHO) == 0:

pulse\_start = time.time()

pulse\_end = time.time()

while GPIO.input(ECHO) == 1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150

distance = round(distance, 2)

await sio.emit("ultrasonic\_sensor\_data", {"distance": distance})

print("Ultrasonic Data:", distance)

return distance

# Function to control servo angle

def set\_servo\_angle(angle):

duty = angle / 18 + 2

pwm.ChangeDutyCycle(duty)

time.sleep(0.5)

pwm.ChangeDutyCycle(0)

# Function to measure distance at a specific servo angle

async def measure\_distance\_at\_angle(angle):

set\_servo\_angle(angle)

time.sleep(1)

distance = await read\_ultrasonic()

time.sleep(1)

return round(distance, 2)

async def control\_loop():

global pause

while True:

if not pause:

IR\_left, IR\_right = await read\_ir()

ultrasonic\_distance = await read\_ultrasonic()

######################################

#### Logic For Line Following Mode ###

######################################

if current\_mode == "line\_follow":

# If left sensor detects the line, turn left

if IR\_left == 1 and IR\_right == 0:

left()

print("Turning left")

# If right sensor detects the line, turn right

elif IR\_right == 1 and IR\_left == 0:

right()

print("Turning right")

# Else move forward

else:

forward()

print("Line Following Logic", IR\_left, IR\_right)

######################################

#### Logic For Obstacle Avoidance ###

######################################

elif current\_mode == "obstacle\_avoid":

if ultrasonic\_distance < 10: # distance threshold in cm

pause = True

stop()

# wait 2 seconds before measuring distances

await asyncio.sleep(2)

print("Obstacle detected - Stopping")

# Left and right distances are measured at 0 and 180 degrees

distance\_0\_deg = await measure\_distance\_at\_angle(0)

distance\_180\_deg = await measure\_distance\_at\_angle(180)

print(f"Measured distances: 0 degrees = {distance\_0\_deg} cm, 180 degrees = {distance\_180\_deg} cm")

set\_servo\_angle(90)

# If more space is available on one side, turn towards that side

if distance\_0\_deg > distance\_180\_deg:

right()

print("Turning right - More space detected")

else:

left()

print("Turning left - More space detected")

await asyncio.sleep(2)

pause = False

# If no obstacle detected, move forward

else:

forward()

print("Path clear - Moving forward")

print("Obstacle Avoidance Logic", ultrasonic\_distance)

await asyncio.sleep(0.1)

async def main():

await connect\_to\_server()

try:

await control\_loop()

except KeyboardInterrupt:

print("Shutting down...")

finally:

GPIO.cleanup()

await sio.disconnect()

if \_\_name\_\_ == "\_\_main\_\_":

try:

asyncio.run(main())

except KeyboardInterrupt:

print("Shutting down...")

## Server

The server runs on the laptop (PORT 3000) and it uses “server.js”.

// First, we initialize the server using socket io

const http = require("http");

const { Server } = require("socket.io");

// We set the port to 3000

const PORT = 3000;

// This object will keep track of the connections

const connections = { rpiClient: "Disconnected", reactClient: "Disconnected" };

const sockets = {};

console.log("Server Started");

const httpServer = http.createServer();

// This part ensures that front end can connect to the server

const io = new Server(httpServer, {

cors: {

origin: ["http://localhost:5173"],

credentials: true,

},

});

httpServer.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

// When a connection is established

io.on("connection", (socket) => {

const auth = socket.handshake.auth;

console.log(auth.token);

// Raspberry pi is trying to connect

if (auth.token === "rpiClientToken") {

// Create room for raspberry pi

socket.join("rpiClient");

// Saves the socket id and the room it is in

sockets[socket.id] = "rpiClient";

// Update the connections object

connections.rpiClient = "Connected";

// Send the connections object to the front-end

io.to("reactClient").emit("check\_connections", connections);

}

// Web Dashboard is trying to connect

else if (auth.token === "reactClientToken") {

// Create room for react client

socket.join("reactClient");

// Saves the socket id and the room it is in

sockets[socket.id] = "reactClient";

// Update the connections object

connections.reactClient = "Connected";

// Send the connections object to the front-end

io.to("reactClient").emit("check\_connections", connections);

}

// If the token is invalid, disconnect the socket

else {

socket.disconnect();

console.log("YOU HAVE BEEN DISCONNECTED!!!");

}

console.log(`Someone connected with the ID: ${socket.id}`);

// When the raspberry pi sends the IR sensor data

socket.on("ir\_sensor\_data", (data) => {

io.to("reactClient").emit("ir\_sensor\_data", data);

console.log(data);

});

// When the raspberry pi sends the ultrasonic sensor data

socket.on("ultrasonic\_sensor\_data", (data) => {

io.to("reactClient").emit("ultrasonic\_sensor\_data", data);

console.log(data);

});

socket.on("check\_connections", () => {

io.to("reactClient").emit("check\_connections", connections);

console.log("Sent", connections);

});

socket.on("joystick", (data) => {

io.to("rpiClient").emit("joystick", data);

console.log(data);

});

// When the react client sends the mode change

socket.on("change\_mode", (data) => {

io.to("rpiClient").emit("change\_mode", data);

console.log(data);

});

socket.on("disconnect", (reason) => {

// If Raspberry Pi disconnects

if (sockets[socket.id] === "rpiClient") {

// Update the connections object

connections.rpiClient = "Disconnected";

// Send the connections object to the front-end

io.to("reactClient").emit("check\_connections", connections);

}

// If React Client disconnects

else if (sockets[socket.id] === "reactClient") {

connections.reactClient = "Disconnected";

}

console.log(connections);

console.log(`Someone disconnected with the ID: ${socket.id}`);

});

});

## Web Dashboard (Frontend)

We used React for the frontend, and it is running on the laptop (PORT 5173)

import React, { useState, useEffect } from "react";

import "./App.css";

import { Joystick } from "react-joystick-component";

import socket from "./socketConnection";

import { IoIosCheckmarkCircle } from "react-icons/io";

import { IoMdCloseCircle } from "react-icons/io";

function App() {

const [connections, setConnections] = useState({

rpiClient: "Disconnected",

reactClient: "Disconnected",

});

const [currentMode, setCurrentMode] = useState("joystick");

const [IRsensorData, setIRsensorData] = useState(null);

const [ultrasonicData, setUltrasonicData] = useState(null);

useEffect(() => {

socket.emit("check\_connections");

socket.on("check\_connections", (data) => {

setConnections({

...connections,

rpiClient: data.rpiClient,

reactClient: data.reactClient,

});

console.log(connections);

});

socket.on("ir\_sensor\_data", (data) => {

setIRsensorData(data);

console.log(data);

});

socket.on("ultrasonic\_sensor\_data", (data) => {

setUltrasonicData(data);

console.log(data);

});

const handleDisconnect = () => {

setConnections({

...connections,

reactClient: "Disconnected",

rpiClient: "Disconnected",

});

setIRsensorData(null);

setUltrasonicData(null);

};

socket.on("disconnect", handleDisconnect);

return () => {

socket.off("disconnect", handleDisconnect);

};

}, []);

const handleMove = (movement) => {

console.log(movement.direction);

socket.emit("joystick", movement.direction);

};

const handleStop = () => {

console.log("stop");

socket.emit("joystick", "stop");

};

const handleModeChange = (mode) => {

setCurrentMode(mode);

socket.emit("change\_mode", mode);

};

return (

<>

<div className="container p-5 text-white">

<div className="bg-dark rounded rounded-4 px-5 py-4 me-md-5 d-flex align-items-baseline mb-4">

<p className="m-0 fs-4 me-4" style={{ fontWeight: "800" }}>

CSE461 Lab Project

</p>

<p className="m-0 fs-5" style={{ fontWeight: "700" }}>

All in One Robot

</p>

</div>

<div className="row mt-5 mt-md-3">

<div className="col-md-6">

<div className="text-white py-5 pe-md-5 me-md-5">

<div>

<p className="fs-3" style={{ fontWeight: "800" }}>

Connections

</p>

</div>

{/\* Websocket Connection with web dashboard and server \*/}

<div className="d-flex justify-content-between py-2 mb-1">

<div>

<p className="p-0 m-0">Websocket Connection Status</p>

<div

className={

connections.reactClient === "Connected"

? "text-success"

: "text-danger"

}

>

{connections.reactClient}

</div>

</div>

<div>

{connections.reactClient === "Connected" ? (

<IoIosCheckmarkCircle size={50} className="text-white" />

) : (

<IoMdCloseCircle size={50} className="text-danger" />

)}

</div>

</div>

{/\* Connection between raspberry pi and web dashboard \*/}

<div className="d-flex justify-content-between py-2">

<div>

<p className="p-0 m-0">Raspberry Pi Status</p>

<div

className={

connections.rpiClient === "Connected"

? "text-success"

: "text-danger"

}

>

{connections.rpiClient}

</div>

</div>

<div>

{connections.rpiClient === "Connected" ? (

<IoIosCheckmarkCircle size={50} className="text-white" />

) : (

<IoMdCloseCircle size={50} className="text-danger" />

)}

</div>

</div>

</div>

</div>

<div className="col-md-6 container py-5">

{/\* Sensor Data \*/}

<div className="text-white px-md-5">

<div>

<p className="fs-3" style={{ fontWeight: "800" }}>

Sensor Reading

</p>

</div>

{/\* Infra Red Sensor Data \*/}

<div className="d-flex justify-content-between">

<div>

<p className="p-0 m-0">IR Sensor (Left)</p>

</div>

<div>

{IRsensorData && connections.rpiClient === "Connected" ? (

<p className="text-primary">{IRsensorData.left}</p>

) : (

<p className="text-danger">null</p>

)}

</div>

</div>

<div className="d-flex justify-content-between">

<div>

<p className="p-0 m-0">IR Sensor (Right)</p>

</div>

<div>

{IRsensorData && connections.rpiClient === "Connected" ? (

<p className="text-primary">{IRsensorData.right}</p>

) : (

<p className="text-danger">null</p>

)}

</div>

</div>

{/\* Ultrasonic Sensor Data \*/}

<div className="d-flex justify-content-between">

<div>

<p className="p-0 m-0">Ultrasonic Sensor</p>

</div>

<div>

{ultrasonicData && connections.rpiClient === "Connected" ? (

<p className="text-primary">{ultrasonicData.distance}</p>

) : (

<p className="text-danger">null</p>

)}

</div>

</div>

</div>

</div>

</div>

<div className="row">

<div className="col-md-6">

{/\* Description on different modes \*/}

<div className="text-white py-5">

<div>

<p className="fs-3" style={{ fontWeight: "800" }}>

Mode Description

</p>

</div>

<div>

<p>

<strong>Joystick Control</strong>: Use the joystick to control

the robot.

</p>

<p>

<strong>Line Follower</strong>: The robot will follow a black

line on a white surface.

</p>

<p>

<strong>Obstacle Avoidance</strong>: The robot will avoid

obstacles in its path.

</p>

</div>

</div>

</div>

<div className="col-md-6">

{/\* Select Mode \*/}

<div className="d-flex justify-content-center py-5">

{/\* <p className="m-0 d-flex align-items-center">Select Mode</p> \*/}

<div

style={{ display: "inline-flex" }}

className="gap-3 bg-dark py-1 rounded rounded-2 px-2"

>

<button

className={`btn ${

currentMode === "joystick" ? "btn-light" : "btn-dark"

}`}

onClick={() => handleModeChange("joystick")}

>

Joystick Control

</button>

<button

className={`btn ${

currentMode === "line\_follow" ? "btn-light" : "btn-dark"

}`}

onClick={() => handleModeChange("line\_follow")}

>

Line Follower

</button>

<button

className={`btn ${

currentMode === "obstacle\_avoid" ? "btn-light" : "btn-dark"

}`}

onClick={() => handleModeChange("obstacle\_avoid")}

>

Obstacle Avoidance

</button>

</div>

</div>

{/\* Joystick \*/}

<div className="d-flex justify-content-center py-3">

<Joystick

size={120}

sticky={false}

baseColor="white"

stickColor="black"

move={handleMove}

stop={handleStop}

></Joystick>

</div>

</div>

</div>

</div>

</>

);

}

export default App;

.