Hochschule Ravensburg-Weingarten

University of Applied Sciences

Task 4: Cooperation between two programs

Sr No	Student Names	Matriculation Numbers
1	Arjun Ramwshbhai Beladiya	28744163

Guided by:
Prof. Markus Pfeil

Task) Your Task is to create four programs (Two python scripts, a C-Program and theWebserver) that will read sensors and control the robot movement. You will reuse the webserver from the last task to control the robot remotely.
For this task the robot should be situated about 1 m in front of a straight wall, directly facing it.
You can reuse the Node.JS program and the python script from task 3 to read the encoders and communicate with the website.

Task D) A second Python script should read the line-following sensors (use the same strategy to count all the sensor events) and communicate the results to the C-Program every 50 ms

C-soket Code:

```c
#include <stdio.h>        // For standard input/output functions
#include <stdlib.h>       // For exit(), malloc(), etc.
#include <string.h>       // For memset()
#include <unistd.h>       // For close(), read(), write(), etc.
#include <arpa/inet.h>    // For sockaddr_in, inet_ntoa(), etc.

#define PORT 65432        // Server port number
#define BUFFER_SIZE 1024  // Size of buffer for receiving messages

int main() {
  int server_fd, new_socket;
  struct sockaddr_in address;
  int addrlen = sizeof(address);
  char buffer[BUFFER_SIZE] = {0};  // Initialize buffer to zeros

  // Create TCP socket (IPv4, stream socket, protocol = default)
  if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
    perror("Socket failed");  // Print error if socket creation fails
    exit(EXIT_FAILURE);
  }

  // Configure address structure
  address.sin_family = AF_INET;           // IPv4
  address.sin_addr.s_addr = INADDR_ANY;   // Listen on all interfaces
  address.sin_port = htons(PORT);         // Convert port to network
     byte order

  // Bind socket to specified IP and port
  if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) <
     0) {
    perror("Bind failed");
    close(server_fd);  // Clean up socket before exiting
    exit(EXIT_FAILURE);
  }

  // Listen for incoming connections (queue up to 3 pending connections
     )
  if (listen(server_fd, 3) < 0) {
    perror("Listen failed");
    close(server_fd);
    exit(EXIT_FAILURE);
  }

  // Accept a new connection from a client
  if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (
     socklen_t*)&addrlen)) < 0) {
    perror("Accept failed");
    close(server_fd);
```

```
    exit(EXIT_FAILURE);
  }

  // Continuously read messages from the client
  while (1) {
    int valread = read(new_socket, buffer, BUFFER_SIZE);
    if (valread > 0) {
      buffer[valread] = '\0';  // Null-terminate received data
      printf("Received event count: %s\n", buffer);  // Display message
      memset(buffer, 0, BUFFER_SIZE);  // Clear buffer for next read
    }
    // Note: You may want to add a break condition to exit this loop
       gracefully
  }

  // Close sockets when done (never reached in current infinite loop)
  close(new_socket);
  close(server_fd);

  return 0;
}
```

Listing 1: C Example

Python Code:

```
import socket        # For TCP communication
import time          # For timing and delays
import RPi.GPIO as GPIO  # To read sensor inputs from Raspberry Pi GPIO
    pins

# Server details
HOST = 'localhost'   # Replace with server IP if not running on same
   machine
PORT = 65432         # Port number to connect to
INTERVAL = 0.05      # Time interval (in seconds) to send event counts
   (50 ms)

# GPIO pins for the left and right sensors
LEFT_SENSOR_PIN = 20
RIGHT_SENSOR_PIN = 21

def setup_sensors():
    """Initialize GPIO settings for sensor input pins."""
    GPIO.setmode(GPIO.BCM)  # Use BCM pin numbering
    GPIO.setup(LEFT_SENSOR_PIN, GPIO.IN)   # Set left sensor pin as
       input
    GPIO.setup(RIGHT_SENSOR_PIN, GPIO.IN)  # Set right sensor pin as
       input

def read_sensor_data():
    """Read and return current state of both sensors."""
    left_sensor = GPIO.input(LEFT_SENSOR_PIN)
    right_sensor = GPIO.input(RIGHT_SENSOR_PIN)
    return left_sensor, right_sensor

def main():
```

```
    """Main function to setup sensors, connect to server, and send
       sensor event counts."""
    setup_sensors()  # Initialize sensor GPIO pins
    s = None         # Initialize socket variable

    try:
        # Create a TCP socket and connect to the server
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print(f"Connecting to {HOST}:{PORT}")
        s.connect((HOST, PORT))
        print("Connected successfully")

        # Initialize event counters for left and right sensors
        event_count_left = 0
        event_count_right = 0
        start_time = time.time()  # Record the start time

        while True:
            # Read current sensor values
            left_sensor, right_sensor = read_sensor_data()

            # Count an event if sensor is triggered (i.e., logic HIGH)
            if left_sensor == 1:
                event_count_left += 1
            if right_sensor == 1:
                event_count_right += 1

            # Send event counts every INTERVAL seconds
            if time.time() - start_time >= INTERVAL:
                message = f"{event_count_left},{event_count_right}".
                   encode()
                print(f"Sending event counts - Left: {event_count_left
                   }, Right: {event_count_right}")
                s.sendall(message)  # Send data to the server
                start_time = time.time()  # Reset timer
                event_count_left = 0      # Reset event counts
                event_count_right = 0

    except ConnectionRefusedError:
        # Handle case where server is not running or not reachable
        print(f"Connection to {HOST}:{PORT} refused. Make sure the
           server is running.")
    except Exception as e:
        # Catch-all for any other errors
        print(f"An error occurred: {e}")
    finally:
        # Always close socket and clean up GPIO pins on exit
        if s is not None:
            s.close()
        GPIO.cleanup()

if __name__ == "__main__":
    main()
\
```

Listing 2: Python Example

```
Received event count: 4532,4532
Received event count: 6494,6494
Received event count: 6041,6041
Received event count: 4696,4696
Received event count: 4493,4493
Received event count: 4877,4877
Received event count: 3857,3857
Received event count: 2920,2920
Received event count: 2759,2759
Received event count: 2856,2856
Received event count: 2783,2783
Received event count: 2324,2324
Received event count: 5435,5435
Received event count: 5831,5831
Received event count: 5694,5694
Received event count: 2487,2487
Received event count: 3922,3922
Received event count: 4356,4356
Received event count: 4387,4387
Received event count: 5761,5761
Received event count: 4899,4899
Received event count: 5146,5146
Received event count: 5663,5663
```

Figure 1: C-soket code outpunt

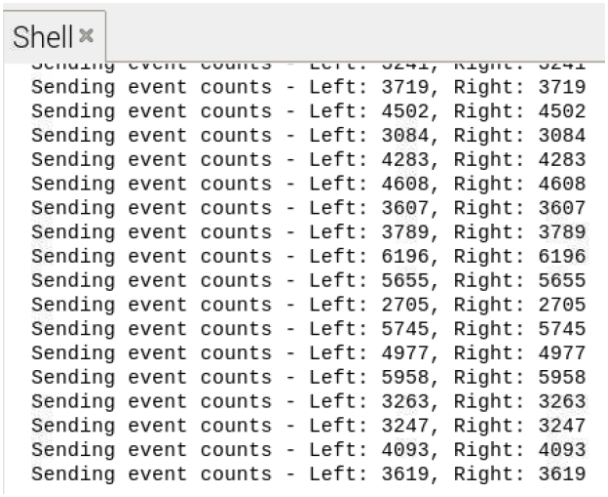

Figure 2: Python code output

Task E) A C script should read the SR04 Ultrasonic sensor and communicate the results to the original script every 50 ms

C code:

```c
#include <stdio.h>       // For input/output operations
#include <stdlib.h>      // For exit()
#include <unistd.h>      // For usleep()
#include <pigpio.h>      // pigpio library for GPIO control on
   Raspberry Pi

// Define GPIO pin numbers
#define TRIG 25
#define ECHO 27
#define MOTOR_LEFT_PWM 18
#define MOTOR_LEFT_IN1 17
#define MOTOR_LEFT_IN2 22
#define MOTOR_RIGHT_PWM 19
#define MOTOR_RIGHT_IN1 24
#define MOTOR_RIGHT_IN2 4

#define OBSTACLE_THRESHOLD 20.0  // Obstacle threshold distance in
   centimeters

// Function to initialize GPIO pins and motor settings
void setup_gpio() {
  if (gpioInitialise() < 0) {
    fprintf(stderr, "GPIO initialization failed\n");
    exit(1);  // Exit if initialization fails
  }

  // Set pin modes
  gpioSetMode(TRIG, PI_OUTPUT);
  gpioSetMode(ECHO, PI_INPUT);
  gpioSetMode(MOTOR_LEFT_PWM, PI_OUTPUT);
  gpioSetMode(MOTOR_LEFT_IN1, PI_OUTPUT);
  gpioSetMode(MOTOR_LEFT_IN2, PI_OUTPUT);
  gpioSetMode(MOTOR_RIGHT_PWM, PI_OUTPUT);
  gpioSetMode(MOTOR_RIGHT_IN1, PI_OUTPUT);
  gpioSetMode(MOTOR_RIGHT_IN2, PI_OUTPUT);

  // Initialize motor and TRIG states to LOW
  gpioWrite(TRIG, 0);
  gpioWrite(MOTOR_LEFT_IN1, 0);
  gpioWrite(MOTOR_LEFT_IN2, 0);
  gpioWrite(MOTOR_RIGHT_IN1, 0);
  gpioWrite(MOTOR_RIGHT_IN2, 0);

  // Set PWM frequency and range for motors
  gpioSetPWMfrequency(MOTOR_LEFT_PWM, 100);   // 100 Hz PWM
  gpioSetPWMfrequency(MOTOR_RIGHT_PWM, 100);
  gpioSetPWMrange(MOTOR_LEFT_PWM, 255);       // Max value for PWM
  gpioSetPWMrange(MOTOR_RIGHT_PWM, 255);

  // Start with motors stopped (0% duty cycle)
```

```
  gpioPWM(MOTOR_LEFT_PWM, 0);
  gpioPWM(MOTOR_RIGHT_PWM, 0);

  usleep(2000);  // Wait for 2 ms to settle
}

// Function to measure distance using the ultrasonic sensor
float measure_distance() {
  // Trigger the ultrasonic pulse
  gpioWrite(TRIG, 0);
  usleep(2000);           // Wait 2 ms
  gpioWrite(TRIG, 1);
  usleep(10);             // Pulse for 10  s
  gpioWrite(TRIG, 0);

  // Wait for ECHO pin to go HIGH (start timing)
  while (gpioRead(ECHO) == 0);
  uint32_t start_time = gpioTick();  // Timestamp in microseconds

  // Wait for ECHO pin to go LOW (end timing)
  while (gpioRead(ECHO) == 1);
  uint32_t end_time = gpioTick();

  // Calculate pulse duration and convert to distance
  uint32_t pulse_duration = end_time - start_time;
  return (pulse_duration / 100.0) * 0.0344 / 2;  // Convert to cm
}

// Function to move both motors forward
void move_forward() {
  gpioWrite(MOTOR_LEFT_IN1, 1);
  gpioWrite(MOTOR_LEFT_IN2, 0);
  gpioWrite(MOTOR_RIGHT_IN1, 1);
  gpioWrite(MOTOR_RIGHT_IN2, 0);
  gpioPWM(MOTOR_LEFT_PWM, 128);   // 50% duty cycle
  gpioPWM(MOTOR_RIGHT_PWM, 128);
}

// Function to stop both motors
void stop_motors() {
  gpioWrite(MOTOR_LEFT_IN1, 0);
  gpioWrite(MOTOR_LEFT_IN2, 0);
  gpioWrite(MOTOR_RIGHT_IN1, 0);
  gpioWrite(MOTOR_RIGHT_IN2, 0);
  gpioPWM(MOTOR_LEFT_PWM, 0);
  gpioPWM(MOTOR_RIGHT_PWM, 0);
}

// Main function
int main() {
  setup_gpio();  // Initialize all GPIO settings

  // Open file to save distance sensor data
  FILE *file = fopen("sensor_data.txt", "w");
  if (file == NULL) {
    fprintf(stderr, "Error opening file\n");
    gpioTerminate();  // Clean up pigpio
    exit(1);
```

```
  }

  while (1) {
    // Measure distance to obstacle
    float distance = measure_distance();

    // Save the distance to the file
    fprintf(file, "%.2f\n", distance);
    fflush(file);  // Ensure data is written immediately

    // Obstacle avoidance logic
    if (distance < OBSTACLE_THRESHOLD) {
      // If too close to an obstacle, stop the motors
      printf("Obstacle detected at %.2f cm. Stopping.\n", distance);
      stop_motors();
    } else {
      // Otherwise, move forward
      printf("Moving forward. Distance: %.2f cm\n", distance);
      move_forward();
    }

    usleep(50000);  // Delay 50 ms before next loop iteration
  }

  // Clean-up (not reached in infinite loop, but good practice)
  fclose(file);
  gpioTerminate();
  return 0;
}
```

Listing 3: C Example

```
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
Obstacle detected at 0.19 cm. Stopping.
```

Figure 3: C code output

F) The original program should receive the sensor values every 50 ms and calculate an estimated speed and orientation value from the data. This can be achieved by calculating an average forward movement from the wheel encoders as well as the turning rate (you can use the simple solution, just rotating around one wheel). This can be compared with the change in distance from the ultrasonic sensor that depends on the speed forward and the change of angle with respect to the wall (this can be done geometrically, but is optional).

Python code:

```
import time      # For timekeeping
import os        # For file system interaction

# Constants
UPDATE_INTERVAL = 0.05        # Time interval between updates (50 ms)
OBSTACLE_THRESHOLD = 20.0     # Threshold distance in cm to detect
   obstacles
WHEEL_RADIUS = 0.05           # Wheel radius in meters (5 cm)
WHEEL_BASE = 0.15             # Distance between left and right wheels
   (in meters)

def read_distance():
    """
    Reads the latest distance measurement from the file 'sensor_data.
       txt'.
    Returns the distance in cm if available and valid, otherwise
       returns None.
    """
    try:
        with open("sensor_data.txt", "r") as file:
            line = file.readline().strip()
            if line:
                return float(line)  # Convert to float and return
    except FileNotFoundError:
        print("Sensor data file not found.")
    except ValueError:
        print("Invalid data in sensor file.")
    return None

def calculate_speed(distance, previous_distance, time_elapsed):
    """
    Calculates the speed (m/s) based on change in distance over time.
    If this is the first reading, returns 0.
    """
    if previous_distance is None:
        return 0
    distance_change = distance - previous_distance
```

```
    return distance_change / time_elapsed  # Simple velocity formula

def calculate_orientation(left_encoder, right_encoder, time_elapsed):
    """
    Estimates distance traveled and orientation change (radians) based
       on encoder ticks.
    Uses differential drive kinematics.
    """
    # Calculate linear distance moved by each wheel
    left_distance = 2 * 3.14159 * WHEEL_RADIUS * left_encoder
    right_distance = 2 * 3.14159 * WHEEL_RADIUS * right_encoder

    # Average of left and right distance = total forward distance
    distance_traveled = (left_distance + right_distance) / 2

    # Orientation change (angular displacement) using difference in
       wheel travel
    orientation_change = (right_distance - left_distance) / WHEEL_BASE

    return distance_traveled, orientation_change

def main():
    """
    Main loop:
    - Reads sensor distance
    - Detects obstacle
    - Calculates speed and orientation
    - Simulates encoder values
    """
    previous_distance = None
    last_time = time.time()

    # Simulated encoder tick counters (can be replaced by real encoders
       )
    left_encoder = 0
    right_encoder = 0

    try:
        while True:
            current_time = time.time()
            time_elapsed = current_time - last_time  # Time since last
               update

            # Read distance from sensor_data.txt
            distance = read_distance()
            if distance is not None:
                # Obstacle avoidance check
                if distance < OBSTACLE_THRESHOLD:
                    print(f"Obstacle detected at {distance:.2f} cm.
                       Stopping.")
                    speed = 0
                else:
                    # Calculate speed using distance change
                    speed = calculate_speed(distance, previous_distance
                       , time_elapsed)
                    print(f"Distance: {distance:.2f} cm, Speed: {speed
                       :.2f} m/s")
```

```
                previous_distance = distance

                # Simulate encoder ticks (assume constant small
                    movement)
                left_encoder += 0.01
                right_encoder += 0.01

                # Calculate estimated travel and heading
                distance_traveled, orientation_change =
                    calculate_orientation(left_encoder, right_encoder,
                    time_elapsed)
                print(f"Distance Traveled: {distance_traveled:.2f} m,
                    Orientation Change: {orientation_change:.2f} radians
                    ")

                last_time = current_time  # Update timestamp

            time.sleep(UPDATE_INTERVAL)  # Wait for next cycle
    except KeyboardInterrupt:
        # Handle Ctrl+C termination gracefully
        print("Program terminated by user")

if __name__ == "__main__":
    main()
```

Listing 4: Python Example

Shell

```
Obstacle detected at 11.30 cm. Stopping.
Distance Traveled: 1.76 m, Orientation Change: 0.00 radians
Obstacle detected at 11.30 cm. Stopping.
Distance Traveled: 1.76 m, Orientation Change: 0.00 radians
Obstacle detected at 11.30 cm. Stopping.
Distance Traveled: 1.76 m, Orientation Change: 0.00 radians
Obstacle detected at 11.30 cm. Stopping.
Distance Traveled: 1.77 m, Orientation Change: 0.00 radians
Obstacle detected at 11.30 cm. Stopping.
Distance Traveled: 1.77 m, Orientation Change: 0.00 radians
Obstacle detected at 11.30 cm. Stopping.
Distance Traveled: 1.77 m, Orientation Change: 0.00 radians
Obstacle detected at 11.30 cm. Stopping.
Distance Traveled: 1.77 m, Orientation Change: 0.00 radians
Obstacle detected at 11.30 cm. Stopping.
Distance Traveled: 1.78 m, Orientation Change: 0.00 radians
Obstacle detected at 11.30 cm. Stopping.
Distance Traveled: 1.78 m, Orientation Change: 0.00 radians
```

Figure 4: Python code Output

G) The robot should move in a straight line towards the wall slowly for 30 to 40 cm and then use the sensor input to turn right 30 degrees and move forward another 20 cm. It should then move back to the original position and repeat the action to the left side.

```
# Import required libraries
import RPi.GPIO as GPIO
import time
import json

# Define GPIO pins for motor control
MOTOR_LEFT_PWM = 18
MOTOR_LEFT_IN1 = 17
MOTOR_LEFT_IN2 = 22
MOTOR_RIGHT_PWM = 19
MOTOR_RIGHT_IN1 = 24
MOTOR_RIGHT_IN2 = 4

# Define GPIO pins for ultrasonic sensor
TRIGGER = 25
ECHO = 27

# GPIO setup
GPIO.setmode(GPIO.BCM)
GPIO.setup([MOTOR_LEFT_PWM, MOTOR_LEFT_IN1, MOTOR_LEFT_IN2,
            MOTOR_RIGHT_PWM, MOTOR_RIGHT_IN1, MOTOR_RIGHT_IN2], GPIO.
               OUT)
GPIO.setup(TRIGGER, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

# Initialize PWM on motor pins at 100 Hz
left_pwm = GPIO.PWM(MOTOR_LEFT_PWM, 100)
right_pwm = GPIO.PWM(MOTOR_RIGHT_PWM, 100)
left_pwm.start(0)
right_pwm.start(0)

# Function to measure distance using ultrasonic sensor
def distance_measurement():
    GPIO.output(TRIGGER, True)
    time.sleep(0.00001)
    GPIO.output(TRIGGER, False)

    start_time = time.time()
    stop_time = time.time()

    # Wait for echo to go high
    while GPIO.input(ECHO) == 0:
        start_time = time.time()

    # Wait for echo to go low
    while GPIO.input(ECHO) == 1:
        stop_time = time.time()
```

```
    # Calculate time difference and convert to distance
    time_elapsed = stop_time - start_time
    distance = (time_elapsed * 34300) / 2  # in cm
    return distance

# Move forward for a given duration in seconds
def move_forward(duration):
    GPIO.output(MOTOR_LEFT_IN1, True)
    GPIO.output(MOTOR_LEFT_IN2, False)
    GPIO.output(MOTOR_RIGHT_IN1, True)
    GPIO.output(MOTOR_RIGHT_IN2, False)
    left_pwm.ChangeDutyCycle(50)
    right_pwm.ChangeDutyCycle(50)
    time.sleep(duration)
    left_pwm.ChangeDutyCycle(0)
    right_pwm.ChangeDutyCycle(0)

# Turn right by a given angle in degrees
def turn_right(degrees):
    GPIO.output(MOTOR_LEFT_IN1, True)
    GPIO.output(MOTOR_LEFT_IN2, False)
    GPIO.output(MOTOR_RIGHT_IN1, False)
    GPIO.output(MOTOR_RIGHT_IN2, True)
    left_pwm.ChangeDutyCycle(50)
    right_pwm.ChangeDutyCycle(50)
    time.sleep(abs(degrees) / 360 * 2)  # Simple timing-based turning
    left_pwm.ChangeDutyCycle(0)
    right_pwm.ChangeDutyCycle(0)

# Turn left by a given angle in degrees
def turn_left(degrees):
    GPIO.output(MOTOR_LEFT_IN1, False)
    GPIO.output(MOTOR_LEFT_IN2, True)
    GPIO.output(MOTOR_RIGHT_IN1, True)
    GPIO.output(MOTOR_RIGHT_IN2, False)
    left_pwm.ChangeDutyCycle(50)
    right_pwm.ChangeDutyCycle(50)
    time.sleep(abs(degrees) / 360 * 2)
    left_pwm.ChangeDutyCycle(0)
    right_pwm.ChangeDutyCycle(0)

# Move backward for a given duration in seconds
def move_backward(duration):
    GPIO.output(MOTOR_LEFT_IN1, False)
    GPIO.output(MOTOR_LEFT_IN2, True)
    GPIO.output(MOTOR_RIGHT_IN1, False)
    GPIO.output(MOTOR_RIGHT_IN2, True)
    left_pwm.ChangeDutyCycle(50)
    right_pwm.ChangeDutyCycle(50)
    time.sleep(duration)
    left_pwm.ChangeDutyCycle(0)
    right_pwm.ChangeDutyCycle(0)

# Stop both motors immediately
def stop():
    GPIO.output(MOTOR_LEFT_IN1, False)
    GPIO.output(MOTOR_LEFT_IN2, False)
    GPIO.output(MOTOR_RIGHT_IN1, False)
```

```
    GPIO.output(MOTOR_RIGHT_IN2, False)
    left_pwm.ChangeDutyCycle(0)
    right_pwm.ChangeDutyCycle(0)

# Main routine to demonstrate basic movement and sensor reading
def main():
    move_forward(2)
    stop()
    turn_right(30)
    stop()
    move_forward(1)
    stop()
    move_backward(1)
    stop()
    turn_left(30)
    stop()
    move_backward(2)

    # Measure distance after movement
    distance = distance_measurement()

    # Prepare data to send back (e.g., over socket or serial)
    data = {
        'distance': distance,
        'speed': 5,  # Example fixed value
        'orientation': 30  # Example orientation (not calculated)
    }

    # Print data in JSON format
    print(json.dumps(data))

# Run main logic and clean up GPIO
if __name__ == '__main__':
    main()
    GPIO.cleanup()
```

Listing 5: Python Example

Figure 5: Output

i) Modify the index.html page with the html tags for displaying the distance and the checkboxes for initiating the movement to reflect the new desired behaviour.

```
<!DOCTYPE html>
<html>
<head>
    <title>Robot Controller</title>
    <!-- Load Socket.IO client library -->
    <script src="/socket.io/socket.io.js"></script>
    <script>
        // Establish connection with server
        var socket = io();

        // Send control command based on checkbox state
        function controlRobot(action) {
            var checkbox = document.getElementById(action + 'Checkbox')
                ;
            socket.emit(action, checkbox.checked);
        }

        // Update robot status data received from server
        socket.on('robotData', function(data) {
            document.getElementById('distance').innerText = data.
                distance + ' cm';
            document.getElementById('speed').innerText = data.speed + '
                 cm/s';
            document.getElementById('orientation').innerText = data.
                orientation + ' degrees';
        });
    </script>
</head>
<body>
    <h1>Robot Controller</h1>

    <!-- Checkbox to start or stop robot movement -->
    <input type="checkbox" id="startMovementCheckbox" onclick="
       controlRobot('startMovement')"> Start Movement<br>

    <h2>Robot Status</h2>
    <p>Distance from obstacle: <span id="distance">0 cm</span></p>
    <p>Speed: <span id="speed">0 cm/s</span></p>
    <p>Orientation: <span id="orientation">0 degrees</span></p>

    <!-- Container to display additional logs if needed -->
    <div id="log"></div>
</body>
</html>
```

Listing 6: HTML Example

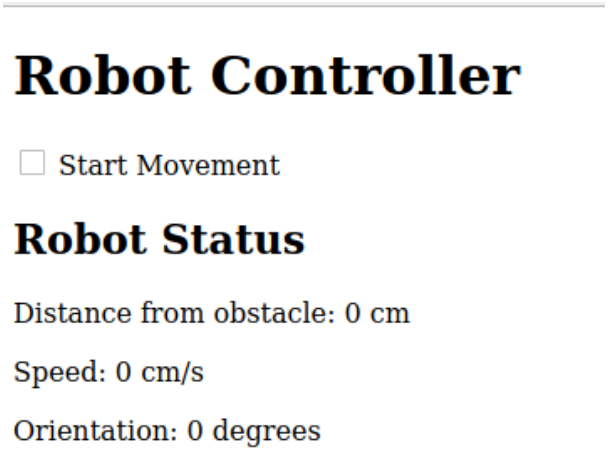

Figure 6: Output

j) Modify the webserver.js file to change the behaviour of the checkboxes to establish a localhost socket connection to the python script and to send the corresponding data items that will trigger the movement

```
var http = require('http').createServer(handler);
var fs = require('fs');
var io = require('socket.io')(http);
var { exec } = require('child_process');

// Start the HTTP server on port 8080
http.listen(8080);

// Serve the HTML file on client request
function handler(req, res) {
  fs.readFile(__dirname + '/public/index.html', function(err, data) {
    if (err) {
      res.writeHead(404, { 'Content-Type': 'text/html' });
      return res.end("404 Not Found");
    }
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write(data);
    return res.end();
  });
}

// Handle incoming socket connections
io.sockets.on('connection', function(socket) {

  // Listen for 'startMovement' event from client
  socket.on('startMovement', function(start) {
    if (start) {
      // Execute the Python script to start robot movement
      exec('python3 ' + __dirname + '/Python_4_j.py', (error, stdout,
          stderr) => {
        if (error) {
```

```
          console.error(`exec error: ${error}`);
          return;
        }

        var data;
        try {
          // Try to parse the Python script output
          data = JSON.parse(stdout);
        } catch (e) {
          console.error(`JSON parse error: ${e}`);
          data = { message: "Error parsing robot data" };
        }

        // Add movement status message and send to client
        data.message = "Movement completed: " + data.message;
        socket.emit('robotData', data);
      });
    }
  });
});
```

Listing 7: Java Example