Wroclaw University Of Science And Technology

BATIN TOPBAŞOĞLU 276718

CANER OLCAY 276715

PIOTR KĘDZIA 275540

ADVANCED TOPICS IN ROBOTICS

Final report

TABLE OF CONTENT

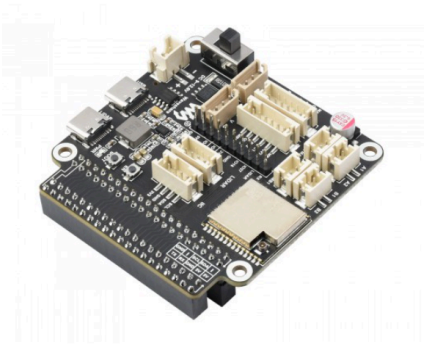
Remotely Controlled Tracked Mobile Platform - Final Report

Introduction

The Remotely Controlled Tracked Mobile Platform project aimed to develop a versatile robotic platform that could be operated remotely using an intuitive Python-based GUI. Initially, the project explored the integration of the Robot Operating System (ROS2) to manage hardware components and ensure seamless communication. However, due to time constraints and implementation challenges, the approach was revised. The final solution utilized an ESP32 microcontroller with Arduino-based programming to establish communication and integrate hardware components, including an ultrasonic sensor for object detection. This project highlights the challenges and successes of developing an accessible and functional robotic system.

ASSUMPTIONS

ESP32-WROOM-32 module



- Based on the ESP32-WROOM-32 module, supports wireless communication such as WIFI, Bluetooth and ESP-NOW
- Onboard motor control interfaces for 2x DC motor with encoder or 4x DC motor (2 groups) without encoder

- Onboard serial bus servos control interfaces for controlling up to 253 ST3215 serial bus servos and obtaining servos feedback
- Onboard 9-axis IMU to obtain attitude and heading information at any time
- Supports 7~13V power input, and can be powered directly by 2S or 3S lithium battery module
- Automatic download circuit for easy uploading programs
- Support input voltage/current monitoring
- Onboard TF card slot
- Onboard Laser Lidar interface and integrated UART to USB function
- I2C interface for connecting peripherals such as OLED, IMU, and other I2C devices
- Adapting Multi-functional extended header for additional functions, such as controlling servos or relays
- Onboard 40PIN GPIO header for connecting and powering the host computer (Raspberry Pi/Jetson Nano, etc), communicating via serial port or I2C
- Provides open-source demos and detailed tutorials for beginners, easy to get started

HC-SR04 Ultrasonic Sensor

- The HC-SR04 is an affordable and easy to use distance measuring sensor which has a range from 2cm to 400cm (about an inch to 13 feet).
- The sensor is composed of two ultrasonic transducers. One is the transmitter which outputs ultrasonic sound pulses and the other is the receiver which listens for

reflected waves. It's basically a SONAR which is used in submarines for detecting underwater objects.

The Main Specifications

Operating Voltage	5V DC
Operating Current	15mA
Operating Frequency	40KHz
Min Range	2cm / 1 inch
Max Range	400cm / 13 feet
Accuracy	3mm
Measuring Angle	<15°
Dimension	45 x 20 x 15mm

- The sensor has 4 pins. *VCC* and *GND* go to *5V* and *GND* pins on the Arduino, and the *Trig* and *Echo* go to any digital Arduino pin. Using the *Trig* pin we send the ultrasound wave from the transmitter, and with the *Echo* pin we listen for the reflected signal.

How the HC-SR04 Ultrasonic Distance Sensor Works?

- It emits an ultrasound at 40 000 Hz which travels through the air and if there is an object or obstacle on its path It will bounce back to the module. Considering the travel time and the speed of the sound you can calculate the distance.

RS775 DC Motor 12V-36V 7500-15000 RPM

- This high-performance DC motor is designed for applications requiring both speed and torque, operating within a voltage range of 12V to 36V. At 12V, it achieves a no-load speed of 7500 RPM and delivers a loaded torque of 4 kg/cm, making it ideal for robotics, RC vehicles, and power tools. With a stall torque of 16.5 kg/cm² and a maximum stall current of 25A, it offers substantial force for demanding tasks. Its compact size and lightweight design (42.3mm diameter and 340g) make it suitable for space-constrained projects, provided the motor driver supports its current and voltage requirements.

Feature	Details
Operating Voltage	Minimum: 12V - Maximum: 36V
No-Load Speed at 12V	7500 RPM
Motor Power	125W
No-Load Current at 12V	1A
Loaded Current at 12V	11A
Maximum Stall Current at 12V	25A
Loaded Torque at 12V	4 kg/cm
Stall Torque at 12V	16.5 kg/cm²
Motor Weight	340 g
Motor Shaft Diameter	5 mm
Motor Shaft Length	10 mm
Motor Diameter	42.3 mm
Motor Total Length	66.3 mm (Motor part only)

Review of the Previous Work

During the initial phase of the project, we dedicated approximately one month to implementing ROS. ROS was chosen for its robust capabilities in managing hardware and its integration potential. Despite extensive efforts, several challenges arose:

- The complexity of setting up ROS for our hardware environment.
- Debugging and resolving communication issues with ROS nodes.

These challenges, combined with the approaching project deadline, led to the decision to transition from ROS to a simpler approach using Arduino for hardware communication. The revised method proved more efficient and manageable within the given timeframe.

Project Evaluation

The project achieved significant milestones through the revised implementation:

1. **ESP32 Integration**: The ESP32 microcontroller was configured using the Arduino IDE, allowing seamless communication with the hardware components. This setup enabled remote control functionality.

2. **Ultrasonic Sensor Implementation**: An ultrasonic sensor was integrated to detect obstacles and enhance the platform's navigation capabilities. The sensor was tested successfully under various conditions.

These accomplishments demonstrate the platform's ability to perform basic navigational tasks while providing a solid foundation for further enhancements.

Arduino Code(Movement and sensor definitons):

```
#include <WiFi.h>
#include <ArduinoWebsockets.h>

using namespace websockets;
```

In this part WiFi.h enables the ESP32 to connect to a WiFi network.
ArduinoWebsockets.h manages WebSocket connections for real-time communication.
using namespace websockets makes WebSocket classes accessible without prefixing them with `websockets`

```
// WiFi credentials
const char* ssid = "androidap123";
const char* password = "latwehaslo";
```

For the WiFi credentials ssid is the name of the WiFi network and password is the password for the WiFi network. It needs to be correct for communication, here it is made for phone-hotspot.

```
// WebSocket server
WebsocketsServer server;
WebsocketsClient client;  // D
```

server: Manages WebSocket connections for receiving commands.
client: Represents a connected WebSocket client.

```
// Motor control pins
const uint16_t PWMA = 25;
const uint16_t PWMB = 26;
const uint16_t AIN2 = 17;
const uint16_t AIN1 = 21;
const uint16_t BIN1 = 22;
const uint16_t BIN2 = 23;
```

The PWMA and PWMB pins control the speed of motors A and B using Pulse Width Modulation (PWM). The AIN1/AIN2 and BIN1/BIN2 pins determine the direction of the motors, allowing for forward or backward motion. Together, these pins enable precise motor control for movement and turning.

```
// Ultrasonic sensor pins
const uint8_t TRIG_PIN = 27; // TRIG pin
const uint8_t ECHO_PIN = 16; // ECHO pin
```

TRIG_PIN sends signals to the ultrasonic sensor, while ECHO_PIN receives the reflected signals from objects.

```
// Distance threshold
const uint16_t STOP_DISTANCE = 10;  // Stop if ob

// Timing variables"
unsigned long lastDistanceCheck = 0;
const unsigned long distanceInterval = 10; // Mea

// Global variable to store the latest distance
long lastDistance = 0;
```

STOP_DISTANCE defines the minimum safe distance to stop the motors. lastDistanceCheck tracks the last measurement time, while distanceInterval sets how often to measure. lastDistance stores the most recent distance value.

```
void initMotors() {
  pinMode(AIN1, OUTPUT);
  pinMode(AIN2, OUTPUT);
  pinMode(PWMA, OUTPUT);
  pinMode(BIN1, OUTPUT);
  pinMode(BIN2, OUTPUT);
  pinMode(PWMB, OUTPUT);
}
```

Basically configures all motor-related pins as output pins for controlling motor behavior.

```
void forwardA(uint16_t pwm) {
  digitalWrite(AIN1, LOW);
  digitalWrite(AIN2, HIGH);
  analogWrite(PWMA, pwm);
}

void forwardB(uint16_t pwm) {
  digitalWrite(BIN1, HIGH); // Adjusted for correct forward logic
  digitalWrite(BIN2, LOW);
  analogWrite(PWMB, pwm);
}

// Backward movement for Motor A and B
void backwardA(uint16_t pwm) {
  digitalWrite(AIN1, HIGH);
  digitalWrite(AIN2, LOW);
  analogWrite(PWMA, pwm);
}
```

forwardA/B moves motors A and B forward at a specified speed, while backwardA/B moves them backward. turnLeft and turnRight handle turning by controlling motor directions, and stopMotors halts both motors by setting PWM signals to 0.

```
// Measure distance using ultrasonic sensor
long measureDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH);
  long distance = duration / 58; // Convert duration to
  return distance;
}
```

Sends a short pulse (10 µs) to the ultrasonic sensor via `TRIG_PIN`.
Measures the time taken for the reflected pulse to reach `ECHO_PIN` using `pulseIn`.
Converts the measured time into distance in centimeters (`duration / 58`).

```
// Handle incoming WebSocket commands
void handleCommand(String command) {
  if (command == "forward") {
    forwardA(50);
    forwardB(50);
  } else if (command == "backward") {
    backwardA(50);
    backwardB(50);
  } else if (command == "left") {
    turnLeft(50);
  } else if (command == "right") {
    turnRight(50);
  } else if (command == "stop") {
    stopMotors();
  } else {
    Serial.println("Unknown command: " + command);
  }
}

void setup() {
```

Executes motor actions (forward, backward, left, right, stop) via WebSocket commands and logs unknown commands for debugging.

```
void setup() {
  // Initialize Serial for debugging
  Serial.begin(115200);

  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("WiFi connected!");
  Serial.print("ESP32 IP Address: ");
  Serial.println(WiFi.localIP());

  // Start WebSocket server
  server.listen(8080);
  Serial.println("WebSocket Server started!");

  // Initialize motors
  initMotors();
```

Serial.begin(115200) initializes serial communication for debugging. The code connects to WiFi using the provided SSID and password, starts a WebSocket server on port 8080, and initializes the motors and ultrasonic sensor pins.

```
void loop() {
  // Accept a new WebSocket client connection if there's none yet
  if (!client.available()) {
    client = server.accept(); // Accept new client connection
    Serial.println("WebSocket client connected!");
  }

  // Measure distance at regular intervals
  unsigned long currentMillis = millis();
  if (currentMillis - lastDistanceCheck >= distanceInterval) {
    lastDistanceCheck = currentMillis;
    lastDistance = measureDistance();
    Serial.print("Distance: ");
    Serial.print(lastDistance);
    Serial.println(" cm");

    // If an object is too close, stop motors
    if (lastDistance <= STOP_DISTANCE) {
      Serial.println("Object detected too close! Stopping motors.");
      stopMotors();
    }
  }
```

The code handles WebSocket clients by accepting new connections and processing motor control commands. It continuously measures distance every 10 milliseconds, logs it to the serial monitor, and stops the motors if an obstacle is detected closer than STOP_DISTANCE. Additionally, it reads and executes WebSocket commands such as forward, backward, left, right, and stop.

Summary

This code integrates:

- **WiFi Communication**: Connects to a network and provides WebSocket functionality.
- **Motor Control**: Provides basic movement (forward, backward, turning) and stopping.
- **Ultrasonic Sensor**: Continuously measures distance and avoids obstacles.
- **WebSocket Commands**: Allows remote control of the robot's motors via WebSocket.

Python(Controller Part):

```python
import asyncio
import websockets
from pynput import keyboard
```

All imported libraries:

1. asyncio - used for asynchronous programming, allowing the program to perform non-blocking operations.
2. websocket - for creating WebSocket clients and servers. Here, it's used to connect to and communicate with the ESP32 robot over WebSocket.
3. pynput.keyboard - A module to listen to and control keyboard inputs. It captures user commands for controlling the robot.

```python
ESP32_WS_URI = "ws://192.168.16.204:8080"  # Replace with your ESP32 IP and port
COMMANDS = {'w': "forward", 's': "backward", 'a': "left", 'd': "right"}
STOP_COMMAND = "stop"
QUIT_COMMAND = "quit"
RECONNECT_DELAY = 2  # Delay between reconnection attempts
```

All defined constants:

1. ESP32_WS_UR - The WebSocket URL of the ESP32. IT NEEDS TO BE REPLACED WITH THE ACTUAL IP ADDRESS AND PORT OF ESP32!.
2. All commands used for controlling the robot
3. Reconnection delay - time (in s) to wait before attempting to reconnect if theWebSocket connection fails.

```python
async def send_command(websocket, command):
    """Send a command to the WebSocket server."""
    try:
        await websocket.send(command)
        print(f"Command sent: {command}")
    except websockets.ConnectionClosed:
        print("Connection closed, unable to send command.")
        raise  # Allow the exception to propagate for reconnect handling
```

Defining function responsible for sending commands (like “forward etc) to the ESP32 via the WebSocket.

```python
async def keyboard_control(websocket, exit_event):
    """Listen to keyboard input and send commands to the robot."""
    print("Use W, A, S, D keys to control the robot. Press Q to quit.")
    current_command = None
    active_key = None

    async def send_loop():
        """Send commands in a continuous loop."""
        nonlocal current_command
        while not exit_event.is_set():  # Exit loop when event is set
            if current_command:
                try:
                    await send_command(websocket, current_command)
                except websockets.ConnectionClosed:
                    break  # Exit loop on disconnection
            await asyncio.sleep(0.1)  # Send commands every 100ms

    def on_press(key):
        """Handle key press events."""
        nonlocal current_command, active_key
        if hasattr(key, 'char') and key.char in COMMANDS:
            if key.char != active_key:  # Prevent duplicate commands
                active_key = key.char
                current_command = COMMANDS[active_key]
        elif hasattr(key, 'char') and key.char == 'q':
            print("Exiting control mode...")
            current_command = QUIT_COMMAND
            exit_event.set()  # Set the exit event to stop the asyncio loop
```

```python
    def on_release(key):
        """Handle key release events."""
        nonlocal current_command, active_key
        if hasattr(key, 'char') and key.char == active_key:
            active_key = None
            current_command = STOP_COMMAND

    listener = keyboard.Listener(on_press=on_press, on_release=on_release)
    listener.start()

    await send_loop()
```

Keyboard control is continuously monitoring keyboard input and sends appropriate commands to the ESP32. It periodically sends the current command every 10ms(if any). The most important in this part is on_press function that maps key presses (w,a,s,d) to a robot commands and sets them as current_command, pressing q stops the whole process. Here we are also using “hasattr” which is a built-in python function that checks if an object has a specific attribute. It is not needed specifically right now but it adds possibilities to make a more complicated “key” (like ctrl/ shift + other char) to the control system. After releasing any key, the stop_command is being sent.

```python
async def connect_to_esp32():
    """Establish connection to ESP32 with retry logic."""
    while True:
        try:
            print("Connecting to ESP32...")
            websocket = await websockets.connect(ESP32_WS_URI)
            print("Connected to ESP32.")
            return websocket
        except Exception as e:
            print(f"Connection failed: {e}. Retrying in {RECONNECT_DELAY} seconds...")
            await asyncio.sleep(RECONNECT_DELAY)


async def stop_robot(websocket):
    """Ensure the robot stops when quitting."""
    try:
        print("Stopping the robot...")
        await send_command(websocket, STOP_COMMAND)
    except Exception as e:
        print(f"Error while stopping the robot: {e}")

```

This function is responsible for connection to the ESP32 WebSocket server with retry logic. Here is where the reconnection delay comes to life. Here is also defined stop.

```python
async def main():
    """Main program loop."""
    websocket = None
    exit_event = asyncio.Event()  # Event to signal the stop of the loop
    try:
        while True:
            websocket = await connect_to_esp32()
            await keyboard_control(websocket, exit_event)
            if exit_event.is_set():
                break
    except websockets.ConnectionClosed:
        print("WebSocket connection lost. Reconnecting...")
    finally:
        if websocket:
            await stop_robot(websocket)  # Ensure the robot stops before quitting
            try:
                await websocket.close()
                print("WebSocket connection closed.")
            except Exception as e:
                print(f"Error closing WebSocket: {e}")


# Run the main event loop
asyncio.run(main())
```

Here is the main control loop for the program.
Flow:

1. Connects the the ESP32
2. Starts the keyboard control loop to the listen for user input
3. Handles reconnection logic if the WebSocket connection is lost
4. Ensures the robot is stopped and the WebSocket is closed before exiting

```
Connecting to ESP32...
Connected to ESP32.
Use W, A, S, D keys to control the robot. Press Q to quit.
wCommand sent: forward
Command sent: forward
Command sent: forward
Command sent: forward
Command sent: stop
Command sent: stop
Command sent: stop
Command sent: stop
aCommand sent: left
Command sent: left
Command sent: left
Command sent: left
Command sent: left
aaaaCommand sent: left
aaaCommand sent: left
aaCommand sent: stop
Command sent: stop
dCommand sent: right
Command sent: right
Command sent: right
Command sent: right
```

```
19:26:18.256 -> Object detected too close! Stopping motors.
19:26:18.389 -> Received command: forward
19:26:18.389 -> Distance: 6 cm
19:26:18.389 -> Object detected too close! Stopping motors.
19:26:18.551 -> Received command: forward
19:26:18.551 -> Distance: 7 cm
19:26:18.551 -> Object detected too close! Stopping motors.
19:26:18.597 -> Received command: forward
19:26:18.597 -> Distance: 8 cm
19:26:18.597 -> Object detected too close! Stopping motors.
19:26:18.808 -> Received command: forward
19:26:18.808 -> Distance: 6 cm
19:26:18.808 -> Object detected too close! Stopping motors.
19:26:18.808 -> Received command: forward
19:26:19.036 -> Received command: forward
19:26:19.036 -> Distance: 7 cm
19:26:19.036 -> Object detected too close! Stopping motors.
19:26:19.036 -> Received command: forward
19:26:19.212 -> Received command: forward
19:26:19.212 -> Distance: 36 cm
19:26:19.255 -> Received command: forward
19:26:19.436 -> Received command: forward
19:26:19.436 -> Distance: 33 cm
19:26:19.436 -> Received command: forward
```

Main output, the first image is from python, the second image is from arduino ide

How it is controlled:

The robot is controlled through WebSocket communication between the Python script and the ESP32 module. Users can press `W`, `A`, `S`, and `D` keys on the keyboard to send directional commands ("forward," "backward," "left," "right"). When a key is pressed, the Python script sends the corresponding command to the ESP32, which controls the motors via the Arduino code. The robot stops when the key is released.

Experiments

Several tests were conducted to validate the platform's functionality:

1. **Communication Testing**: The connection between the ESP32 and the Python GUI was tested to ensure reliability and responsiveness.
2. **Ultrasonic Sensor Testing**: The sensor's ability to detect obstacles at various distances was evaluated, confirming its effectiveness in real-world scenarios.
3. **System Performance**: The entire system was tested in different environments to assess its stability and responsiveness under various conditions.

The results showed that the platform performs reliably within its design parameters, although improvements can be made to optimize performance.

Task Distribution Among Team Members

The project was completed with the collaboration of all team members, with each member contributing to specific parts of the project as follows:

1. **Piotr Kędzia** (275540):
    - Developed the Python-based GUI for robot control.
    - Implemented WebSocket communication between the Python script and the ESP32 module.
    - Conducted debugging and ensured stable communication for user inputs.

2. **Caner Olcay** (276715):
    - Worked on Arduino-based programming for the ESP32 module.
    - Configured motor controls and integrated the ultrasonic sensor for obstacle detection.
    - Tested and validated the hardware setup.

3. **Batın Topbaşoğlu** (276718):
    - Compiled the final project report.
    - Assisted with troubleshooting during the hardware implementation phase..
    - Conducted research on the ESP32 microcontroller and provided insights for its integration.

Conclusions

Despite initial challenges with ROS integration, the project successfully met its primary objectives:

- A remotely controlled robotic platform was developed with ESP32, Arduino, and Python ControllerI.
- Ultrasonic sensors were integrated, providing basic navigation and obstacle detection capabilities.
- The system demonstrated flexibility and scalability, making it a valuable platform for future projects.

This project underscores the importance of adaptability in project development and highlights the potential for continued development. Future iterations could reintroduce ROS or similar frameworks to expand functionality and enhance system capabilities.