Advanced Surveillance an Multi-Utility Rover Project

This document outlines the modular structure for the "Advanced Surveillance and Multi-Utility Rover" project, which integrates an ESP32 for low-level control and a Raspberry Pi 4 for high-level tasks and user interface. Communication between the two main modules is facilitated by WebSockets.

1. ESP32 Firmware (Arduino)

This part of the project focuses on controlling the rover's motors and servos based on commands received from the Raspberry Pi via WebSockets.

Features:

* Motor control using an L298N motor driver.
* Servo control using a PCA9685 PWM driver.
* WebSocket client to connect to the Raspberry Pi.

Libraries Required:

* ESPAsyncWebServer (for asynchronous TCP)
* AsyncTCP
* Wire (for I2C communication with PCA9685)
* Adafruit_PWMServoDriver

Code:

```arduino
#include <WiFi.h>
#include <WebSocketsClient.h>
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

// WiFi credentials
const char* ssid = "ESP32_AP";
const char* password = "12345678";
```

```
// Raspberry Pi WebSocket server details
const char* host_ip = "192.168.4.1"; // Replace with your Raspberry Pi IP address
const int websocket_port = 81;

WebSocketsClient webSocket;
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

// Motor driver pin definitions
#define MOTOR1_IN1 16
#define MOTOR1_IN2 17
#define MOTOR2_IN3 18
#define MOTOR2_IN4 19

// Servo channel definitions (connected to PCA9685)
#define SERVO_BASE 0
#define SERVO_ARM 1

void setupMotors() {
  pinMode(MOTOR1_IN1, OUTPUT);
  pinMode(MOTOR1_IN2, OUTPUT);
  pinMode(MOTOR2_IN3, OUTPUT);
  pinMode(MOTOR2_IN4, OUTPUT);
  // Initialize motors to stop
  digitalWrite(MOTOR1_IN1, LOW);
  digitalWrite(MOTOR1_IN2, LOW);
  digitalWrite(MOTOR2_IN3, LOW);
  digitalWrite(MOTOR2_IN4, LOW);
}

void driveMotors(String direction) {
  if (direction == "forward") {
    digitalWrite(MOTOR1_IN1, HIGH); digitalWrite(MOTOR1_IN2, LOW);
    digitalWrite(MOTOR2_IN3, HIGH); digitalWrite(MOTOR2_IN4, LOW);
  } else if (direction == "backward") {
    digitalWrite(MOTOR1_IN1, LOW); digitalWrite(MOTOR1_IN2, HIGH);
    digitalWrite(MOTOR2_IN3, LOW); digitalWrite(MOTOR2_IN4, HIGH);
  } else if (direction == "left") {
    digitalWrite(MOTOR1_IN1, LOW); digitalWrite(MOTOR1_IN2, HIGH);
```

```
    digitalWrite(MOTOR2_IN3, HIGH); digitalWrite(MOTOR2_IN4, LOW);
  } else if (direction == "right") {
    digitalWrite(MOTOR1_IN1, HIGH); digitalWrite(MOTOR1_IN2, LOW);
    digitalWrite(MOTOR2_IN3, LOW); digitalWrite(MOTOR2_IN4, HIGH);
  } else { // stop
    digitalWrite(MOTOR1_IN1, LOW); digitalWrite(MOTOR1_IN2, LOW);
    digitalWrite(MOTOR2_IN3, LOW); digitalWrite(MOTOR2_IN4, LOW);
  }
}

void moveServo(uint8_t servo, int angle) {
  // Map the angle (0-180 degrees) to the PWM pulse range (typically 102-512 for
standard servos)
  int pulse = map(angle, 0, 180, 102, 512);
  pwm.setPWM(servo, 0, pulse);
}

void handleCommand(String command) {
  if (command.startsWith("move:")) {
    driveMotors(command.substring(5));
  } else if (command.startsWith("servo:")) {
    int colon = command.indexOf(":", 6);
    int servoID = command.substring(6, colon).toInt();
    int angle = command.substring(colon + 1).toInt();
    moveServo(servoID, angle);
  }
}

void webSocketEvent(WStype_t type, uint8_t * payload, size_t length) {
  switch (type) {
    case WStype_DISCONNECTED:
      Serial.println("WebSocket Disconnected!");
      break;
    case WStype_TEXT:
      Serial.printf("WebSocket received Text: %s\r\n", (char*)payload);
      String message = (char*) payload;
      handleCommand(message);
      break;
    case WStype_BIN:
      Serial.printf("WebSocket received BIN: ");
```

```
    for (int i = 0; i < length; i++) {
      Serial.printf("%02X ", payload[i]);
    }
    Serial.println();
    break;
  case WStype_PING:
    // Handle ping if needed
    break;
  case WStype_PONG:
    // Handle pong if needed
    break;
  case WStype_ERROR:
  case WStype_FRAGMENT_TEXT_START:
  case WStype_FRAGMENT_BIN_START:
  case WStype_FRAGMENT:
  case WStype_FRAGMENT_FIN:
    break;
 }
}

void setup() {
 Serial.begin(115200);
 Serial.println("ESP32 Booting...");

 WiFi.begin(ssid, password);
 Serial.print("Connecting to WiFi...");
 while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
 }
 Serial.println("\nWiFi Connected! IP address: ");
 Serial.println(WiFi.localIP());

 Wire.begin(); // Initialize I2C bus
 pwm.begin();
 pwm.setPWMFreq(50); // Standard frequency for servos

 setupMotors();
 Serial.println("Motors Initialized.");

 webSocket.begin(host_ip, websocket_port, "/");
 webSocket.onEvent(webSocketEvent);
```

```
  Serial.println("WebSocket client started.");
}

void loop() {
  webSocket.loop();
}
```

2. Raspberry Pi Web Server (Python Flask + WebSocket)

This part runs on the Raspberry Pi and provides the web interface for controlling the rover and viewing the camera stream. It uses Flask for the web server and Flask-SocketIO for WebSocket communication with the ESP32.

Features:

- Hosts a web UI for rover control.
- Streams video from a connected camera.
- Establishes a WebSocket connection with the ESP32 to send control commands.

Libraries Required:

Bash

```bash
pip install flask flask-socketio eventlet opencv-python
```

Code (app.py):

Python

```python
from flask import Flask, render_template, Response
from flask_socketio import SocketIO, emit
import cv2
import socket
import time

app = Flask(__name__)
```

```python
socketio = SocketIO(app, cors_allowed_origins="*")
ESP32_IP = "192.168.4.2"  # Replace with your ESP32 IP address if different
ESP32_PORT = 81

esp_socket = None

def connect_esp32():
    global esp_socket
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((ESP32_IP, ESP32_PORT))
        print(f"Connected to ESP32 at {ESP32_IP}:{ESP32_PORT}")
        esp_socket = s
        return True
    except ConnectionRefusedError:
        print(f"Connection to ESP32 at {ESP32_IP}:{ESP32_PORT} refused.")
        return False
    except Exception as e:
        print(f"Error connecting to ESP32: {e}")
        return False

@socketio.on('connect')
def handle_connect():
    print("Client connected")
    if esp_socket is None:
        connect_esp32()

@socketio.on('disconnect')
def handle_disconnect():
    print("Client disconnected")
    global esp_socket
    if esp_socket:
        esp_socket.close()
        esp_socket = None

@socketio.on('control')
def handle_control(data):
    print(f"Sending to ESP32: {data}")
    if esp_socket:
        try:
```

```
            esp_socket.send((data + "\n").encode())
        except BrokenPipeError:
            print("Connection to ESP32 lost. Attempting to reconnect...")
            global esp_socket
            esp_socket.close()
            esp_socket = None
            connect_esp32()
            if esp_socket:
                esp_socket.send((data + "\n").encode())
            else:
                print("Failed to reconnect to ESP32.")
        except Exception as e:
            print(f"Error sending data to ESP32: {e}")
    else:
        print("Not connected to ESP32. Attempting to reconnect...")
        connect_esp32()
        if esp_socket:
            esp_socket.send((data + "\n").encode())
        else:
            print("Failed to reconnect to ESP32.")

def gen_frames():
    cap = cv2.VideoCapture(0) # Use 0 for default camera, or specify the device index
    if not cap.isOpened():
        print("Cannot open camera")
        return
    while True:
        success, frame = cap.read()
        if not success:
            print("Can't receive frame (stream end?). Exiting ...")
            break
        try:
            _, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
        except Exception as e:
            print(f"Error encoding frame: {e}")
            break
```

```
    cap.release()

@app.route('/video')
def video():
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/')
def index():
    return render_template('index.html')

if __name__ == '__main__':
    connect_esp32() # Initial connection attempt
    socketio.run(app, host='0.0.0.0', port=5000, debug=True)
```

3. Frontend UI (HTML + JavaScript)

This HTML file provides the user interface elements for controlling the rover's movement and servos, and for displaying the video stream. It uses JavaScript and Socket.IO client library to communicate with the Raspberry Pi's web server.

Code (templates/index.html):

HTML

```html
<!DOCTYPE html>
<html>
<head>
  <title>Rover Control</title>
  <script src="[https://cdn.socket.io/4.7.4/socket.io.min.js](https://cdn.socket.io/4.7.4/socket.io.min.js)"></script>
  <style>
    body { font-family: sans-serif; display: flex; flex-direction: column; align-items: center; }
    h1 { margin-bottom: 20px; }
    img { margin-bottom: 20px; border: 1px solid #ccc; }
    .controls { margin-bottom: 20px; }
    button { margin: 5px; padding: 10px 20px; font-size: 16px; cursor: pointer; }
    .servo-control { margin-bottom: 15px; display: flex; flex-direction: column; align-items: center; }
    input[type="range"] { width: 300px; margin-top: 5px; }
    label { margin-top: 5px; font-size: 14px; color: #555; }
  </style>
```

```html
</head>
<body>
  <h1>Advanced Surveillance Rover</h1>
  <img src="/video" width="640" height="480" alt="Video Stream"/>
  <div class="controls">
    <button onclick="send('move:forward')">Forward</button><br>
    <button onclick="send('move:left')">Left</button>
    <button onclick="send('move:stop')">Stop</button>
    <button onclick="send('move:right')">Right</button><br>
    <button onclick="send('move:backward')">Backward</button>
  </div>

  <div class="servo-control">
    <label for="baseServo">Base Servo (0-180°)</label>
    <input type="range" id="baseServo" min="0" max="180" value="90" oninput="send('servo:0:' +
this.value)">
    <span id="baseServoValue">90</span>
  </div>

  <div class="servo-control">
    <label for="armServo">Arm Servo (0-180°)</label>
    <input type="range" id="armServo" min="0" max="180" value="90" oninput="send('servo:1:' +
this.value)">
    <span id="armServoValue">90</span>
  </div>

  <script>
   const socket = io();

   const baseServoSlider = document.getElementById('baseServo');
   const baseServoValueSpan = document.getElementById('baseServoValue');
   const armServoSlider = document.getElementById('armServo');
   const armServoValueSpan = document.getElementById('armServoValue');

   function send(cmd) {
    socket.emit('control', cmd);
   }

   baseServoSlider.oninput = function() {
    send('servo:0:' + this.value);
    baseServoValueSpan.textContent = this.value;
   }
```

```
  armServoSlider.oninput = function() {
    send('servo:1:' + this.value);
    armServoValueSpan.textContent = this.value;
  }
 </script>
</body>
</html>
```

Summary:

- The ESP32 firmware handles the low-level control of motors and servos based on simple text commands received via WebSocket.
- The Raspberry Pi hosts a Flask web server that serves the user interface and streams video. It uses Flask-SocketIO to establish a WebSocket connection with the ESP32 and forward control commands.
- The frontend UI (HTML and JavaScript) provides buttons for movement control and range sliders for servo control. It uses the Socket.IO client library to send commands to the Raspberry Pi.