

Remote-Controlled Robotic Car

Ameen AlShaibani

University of Denver

Ritchie School Of Engineering and Computer Science

Advisors:

Dr. Wenzhong Gao

Dr. Goncalo Martins

Research Paper submitted in pursuit of Departmental Distinction in Computer Engineering

May 15, 2023

## Introduction

In recent years, the field of robotics has witnessed remarkable growth, driven by advancements in 3D printing, open-source software, and the increasing accessibility of tools for creating custom components and printed circuit boards (PCBs). As these technologies become more prevalent, researchers and enthusiasts alike are exploring innovative ways to develop user-friendly, cost-effective, and versatile robotic systems. One such application is the creation of remote-controlled robotic cars, which have the potential to revolutionize not only the field of robotics but also adjacent industries, such as automotive, logistics, and entertainment.

This research paper presents the development of a remote-controlled robotic car and the design of a user-friendly PCB motherboard using KiCad, a free Electronic Design Automation software. Our approach aims to provide a "Plug and Play" experience, enabling users to easily connect motors to the board and control their car remotely. In contrast to traditional Radio Frequency (RF) modules, our design employs Bluetooth technology, allowing users to leverage their Android smartphones as intuitive and accessible controllers.

To achieve this level of user-friendliness, we developed a custom Android application that facilitates connection and control of the car via Bluetooth. The PCB integrates a microcontroller for signal distribution and a Bluetooth module for communication with the Android app. Power distribution to all circuitry and motors is managed through a single power input, and a motor control system interprets signals from the smartphone controller. We also developed the code for the microcontroller, ensuring seamless integration of the Bluetooth module and motor control system.

This research paper provides a comprehensive account of our innovative approach to remote-controlled robotic car development. We discuss the design and implementation of the PCB, the creation of the Android application, and the integration of Bluetooth technology. We also present experimental results that demonstrate the effectiveness and versatility of our design. By pushing the boundaries of remote-controlled robotics and making custom robotic cars more accessible and convenient, our research contributes to the ongoing evolution of the robotics field.

### **Methods and Materials**

For the development of the PCB, specific components were selected based on their compatibility, efficiency, and form factor. In this section, we outline the rationale behind the chosen components, and compare them with alternative options.

#### **Microcontroller: ESP32**

The ESP32 microcontroller was selected for its integrated wireless capabilities and powerful processing features. As a dual-core, low-power microcontroller with integrated Wi-Fi and Bluetooth Low Energy (BLE) functionalities, it is well-suited for IoT applications and wireless communication. The ESP32 has 34 GPIO pins that are capable of providing digital signals as well as PWM signals which are crucial for the motor controller. The ESP32 provides ample processing power and memory for the demands of the robotic car project. The choice of the ESP32 was further motivated by its built-in BLE capabilities, which eliminated the need for a separate BLE module. This decision streamlined the design, reduced the overall footprint of the PCB, and increased system efficiency. Alternative options, such as the Arduino Nano or the Teensy, would have required an additional BLE module to be connected such as the HM-10, which would have increased complexity, size, and power consumption.

**Motor Controller: TB6612FNG**

The TB6612FNG motor driver was chosen as the motor controller due to its compact size, efficient performance, and versatility. Compared to alternatives like the L298N motor driver, the TB6612FNG does not require a large heatsink or large capacitors, which helps maintain a smaller form factor for the PCB. Additionally, the TB6612FNG offers better thermal efficiency, ensuring safe and reliable operation even under high load conditions.

The TB6612FNG is a dual-channel motor driver that can deliver up to 1.2 A per channel, with a logic voltage of 3.3V and an operating motor voltage range of 2.5-13.5 V. It features built-in thermal shutdown protection and low voltage detection, further enhancing the safety and reliability of the motor control system. The motor driver also offers a range of control options, including PWM pin to determine the speed of the motor, as well as IN1, and IN2 inputs, which provide the user options for turning the motor clockwise, counterclockwise, a short brake or a complete stop. Two TB6612FNG motor drivers are used, allowing users to drive up to 4 motors.

**Power Distribution: Buck Converters**

Efficient power distribution is crucial for the proper functioning and reliability of the PCB. To meet the voltage and current requirements of the various components, two buck converters were incorporated: the AZ1117IH-3.3TRG1 and the LM1085-5.0.

The AZ1117IH-3.3TRG1 is a low dropout linear regulator that converts input voltages up to 12V to a fixed 3.3V output, providing a maximum current of 1A. This 3.3V output powers the ESP32

microcontroller and the TB6612FNG motor driver, ensuring stable operation and mitigating the risk of component failure due to voltage fluctuations.

The LM1085-5.0 is a 5V low dropout linear regulator capable of delivering up to 3A of output current. This buck converter was selected to supply power to the motors, providing them with a stable and reliable source of energy. Its ability to handle high current loads ensures consistent motor performance, even in demanding conditions.

### **Chassis and Motors**

For testing and validation of the PCB, an off-the-shelf acrylic chassis with four TT DC gearbox motors was utilized. These motors are rated for operation within a voltage range of 3 to 6V. At 5V, the motors exhibit a no-load current draw of approximately 155mA, achieving 185 RPM. In a stalled state, the current draw increases to 1.2A per motor.

In summary, the ESP32 microcontroller and the TB6612FNG motor driver were selected based on their integrated capabilities, compact form factor, and efficient performance. By incorporating these components into the PCB design, the system achieves a high level of functionality and versatility, while ensuring reliable and precise control of the remote-controlled robotic car. The combination of the AZ1117IH-3.3TRG1 and LM1085-5.0 buck converters facilitate efficient power distribution for the ESP32 microcontroller, TB6612FNG motor driver, and TT DC gearbox motors, as they were chosen to meet the power consumption demands of the components of the micro-controller, motor controller and motors. The power calculation table provides a detailed breakdown of the power requirements for each component under different

load conditions, further demonstrating the suitability of the chosen buck converters for this application.

*Table 1: Power calculations for components that operate on 3.3V*

COMPONENT	VOLTAGE (V)	CURRENT (mA)	POWER (W)
<b>ESP32</b>	3.3	600	1.98
<b>2 X TB6612FNG</b>	3.3	3	0.00495
<b>TOTAL</b>		603	1.98495
<b>AZ1117IH-3.3TRG1</b>	3.3V	1000	3.3W

*Table 2: Power Calculation for components that operate on 5V.*

COMPONENTS	VOLTAGE (V)	CURRENT (A)	POWER (W)
<b>4 X TT MOTORS (NO LOAD)</b>	5	0.62	3.1
<b>4 X TT MOTORS (STALLED)</b>	5	4.8	24
<b>LM1085-5.0</b>	5	3	15

A notable limitation of the design is that the current pull of all four motors when stalled is 4.8A which is greater than the amount of current that the LM1085-5.0 can provide. This means that when all motors stall, the buck converter will operate at its limits and will overheat, which may cause the IC to fail. Thus, it is advised not to stall all the motors simultaneously for long periods of time.

## Schematic Design

To design the schematics incorporating the ESP32 along with the TB6612FNG driver, I referenced the TB6612FNG datasheet which includes a typical application circuit. The circuit

includes two capacitors of 0.1uF and 10uF placed in parallel for noise absorption on both the power pins VCC and VM. Similarly, the buck converters also used decoupling capacitors on the input and output pins. A 10uF capacitor is used on the input and a 100uF capacitor is used on the output pins for both buck converters. Finally, screw terminals are used as connectors for the board to ensure ease of connection for all types of motors. Additionally, an M3 mounting hole is used to secure the board if needed. The final schematic design is shown in Figure 1.

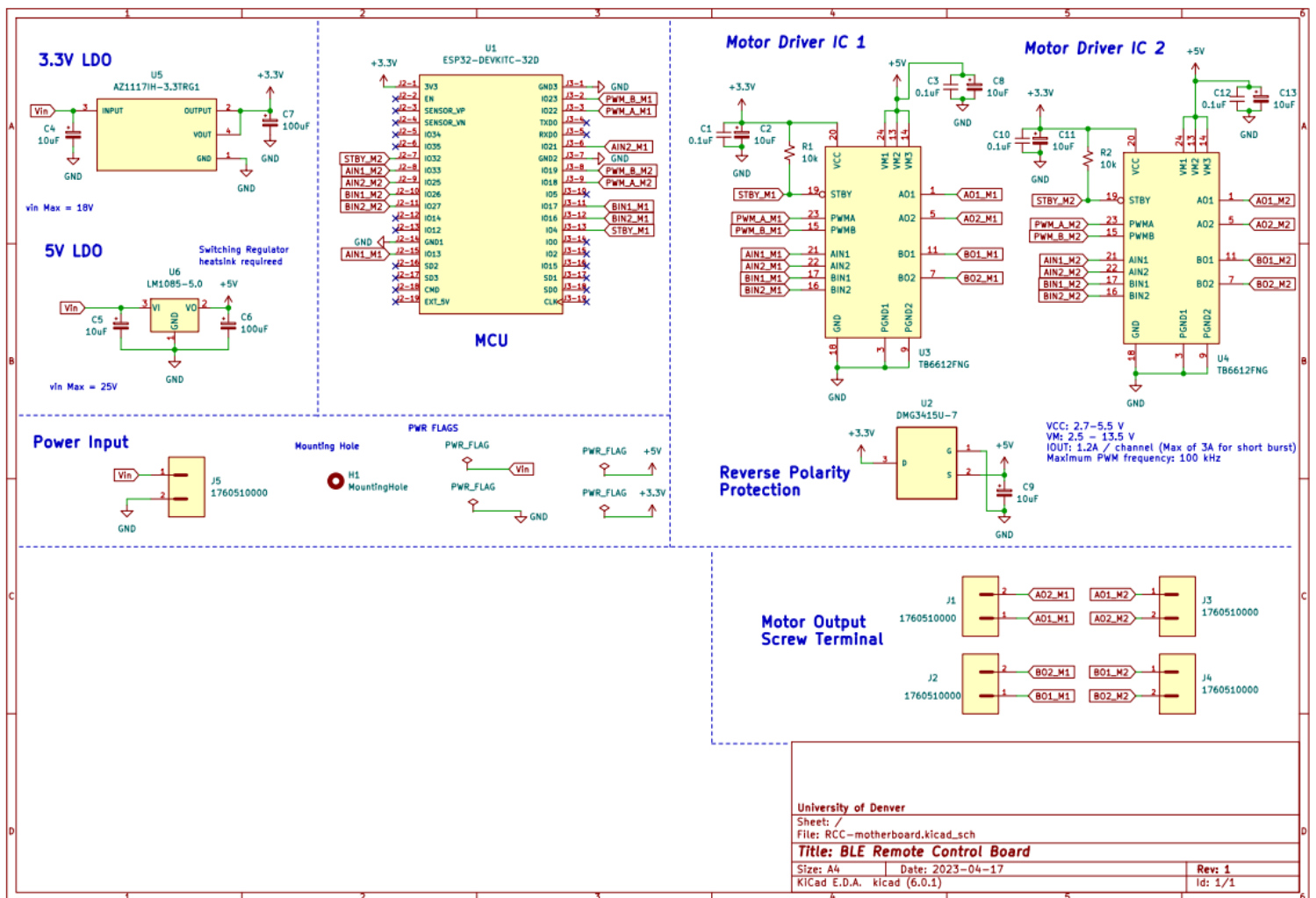


Figure 1: Schematic of Remote-Control Board

## Layout Design

Some of the main board constraints when designing a PCB are the minimum clearances between traces, the minimum trace width, minimum via size, and minimum through hole size. These constraints vary based on the manufacturer, and thus I used the specification provided by JLCPCB which are one of the largest PCB manufacturers in the world. Figure 2 shows all the constraints of the board.









Copper			
	Minimum clearance:	<input type="text" value="0.254"/>	mm
	Minimum track width:	<input type="text" value="0.254"/>	mm
	Minimum annular width:	<input type="text" value="0.2"/>	mm
	Minimum via diameter:	<input type="text" value="0.5"/>	mm
	Copper to hole clearance:	<input type="text" value="0.3"/>	mm
	Copper to edge clearance:	<input type="text" value="0.4"/>	mm
Holes			
	Minimum through hole:	<input type="text" value="0.3"/>	mm
	Hole to hole clearance:	<input type="text" value="0.5"/>	mm

Figure 2: PCB board constraints

Another important consideration for the board is deciding on the appropriate trace widths that can accommodate the current running through them. As constrained by the buck converters, the 3.3V buck converter can provide up to 1 A, while the 5V buck can provide up to 3A. To calculate the required trace widths an equation is provided in IPC-2221, which is a standard for designing PCB boards.

$$I = K * \Delta T^B * (W * H)^C$$

where:



$I$ : is the maximum current that the trace can handle

$\Delta T$ : The temperature rise above ambient in degrees Celsius

$W$ : The width of the trace in mils

$H$ : The thickness of the trace in mils

$K$ ,  $b$  and  $c$  are constants that are determined by whether the trace is external or internal. The board is designed as a 2 layer board, thus all the traces on the board are external. For external traces, the values of the constants are  $K = 0.048$ ,  $b = 0.44$ , and  $c = 0.725$ . Typical values for temperature rise is 10 C, and the thickness of the board is 1 oz/ft<sup>2</sup>.

With these values, the minimum trace width of the board, which is 0.254 mm can handle around 870mA, while a 1.4mm trace width is required to handle up to 3A. However, it is not desirable to have very large trace widths since it would complicate the wiring on the PCB and may lead to requiring the board size to increase. To mitigate this issue, a 5V power plane is placed on the front copper layer. This means that all 5V components are connected through a large copper fill on the top layer of the board, allowing large amounts of current to flow through with no heating issues. On the other hand, the small 0.254 mm trace width can be used to connect the 3.3V components as well as signal traces. The bottom layer of the board is a ground plane, which allows components to be easily connected to ground with the help of vias.

The final size of the board is 66x54 mm, mainly comprised of an ESP 32 micro-controller, along with 5 screw terminals, 1 for power and 4 screw terminals to accommodate 4 motors. There is a mounting hole placed in the middle of the board, below the ESP32 to ensure that the board can

be secured and stabilized if needed. The layout of the board is shown in Figure 3, while a Top and Bottom 3D view of the board is shown in Figures 4 and 5.

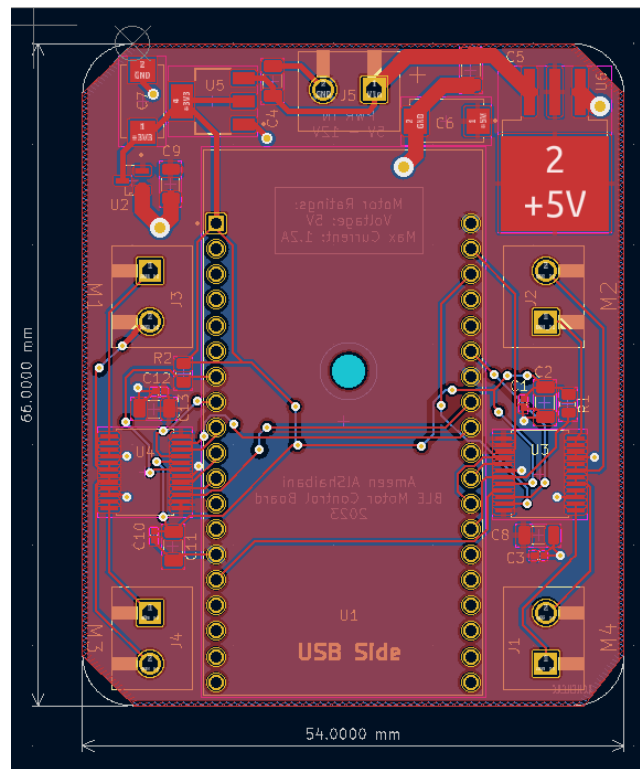


Figure 3: Layout of Remote-Control Board

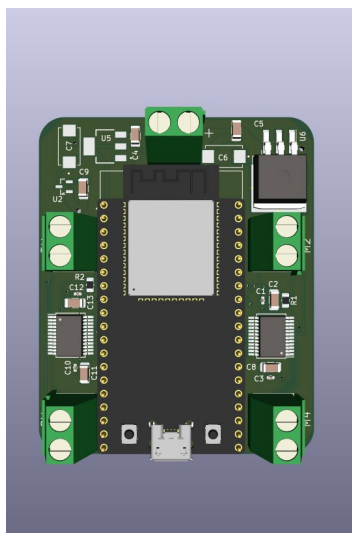


Figure 4: Top View of Remote-Control Board

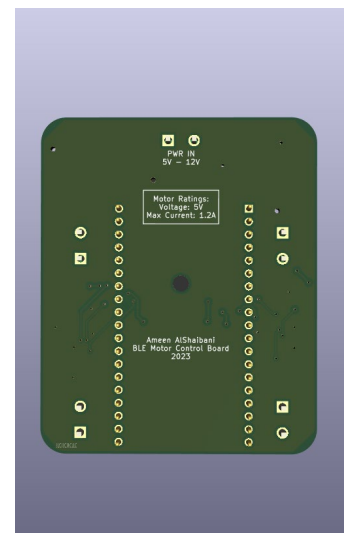


Figure 5: Bottom View of Remote-Control Board

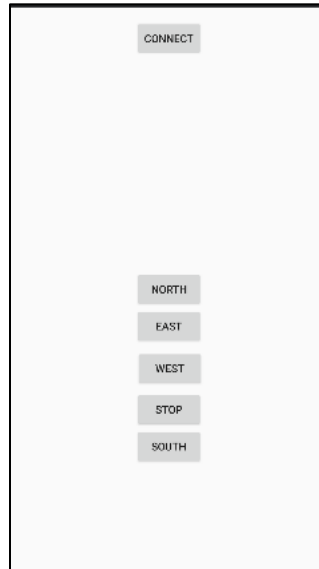
## App Development

To test that the PCB works, a simple firmware was created for the ESP32, and a very simple test application was made. The firmware used the BluetoothSerial library to use classic Bluetooth to connect to the application and receive characters from it. Based on the received characters, the controller would spin the motors at a constant PWM in the specified direction. The code for the ESP32 Firmware is provided in Appendix A. Table 1 shows all the functions that can be used:

*Table 3: Functions to control the car*

Function	Description
North()	Makes the car go forward.
South()	Makes the car go backward.
East()	Makes the car turn towards the east/clockwise.
West()	Makes the car turn towards the west/counterclockwise.
Stop()	Makes the car stop.

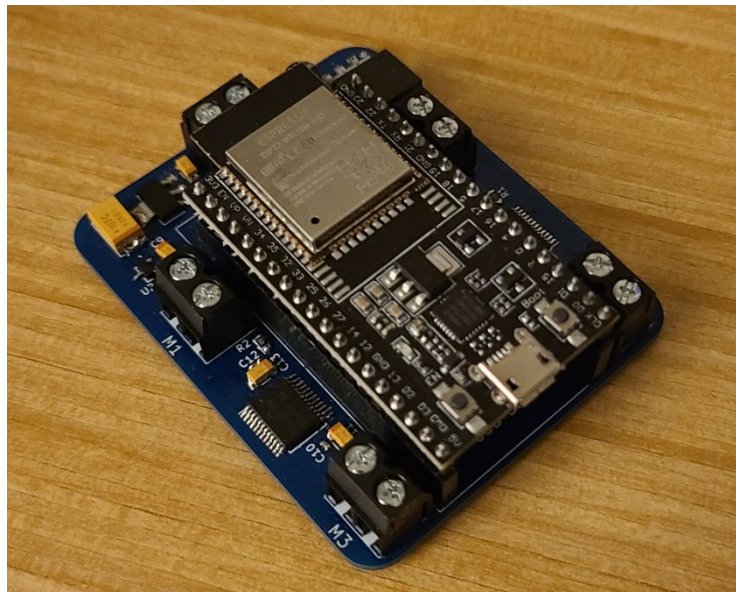
For the phone application, it was developed in Java using the Android Studio development environment. A very simple app was created that was able to connect to the ESP32 micro-controller, and several buttons were added to indicate which direction the car should move in such as North, South, East, West, and Stop. A picture of the layout of the application is provided in Figure 4.



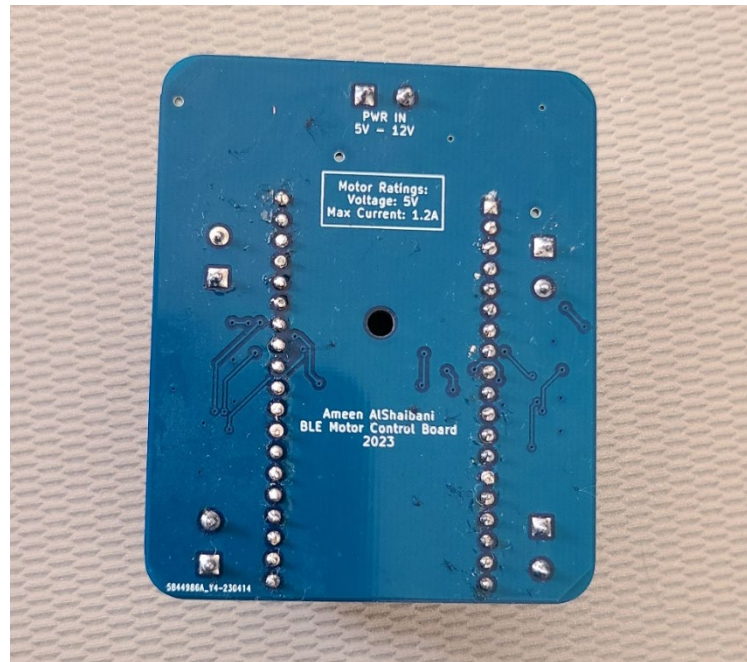
*Figure 4: Layout of Android Application*

## Results

The components for the board were ordered from Digikey and the PCB board was ordered from JLCPCB. The components were all hand soldered onto the board. The result is shown Figures 4 and 5:

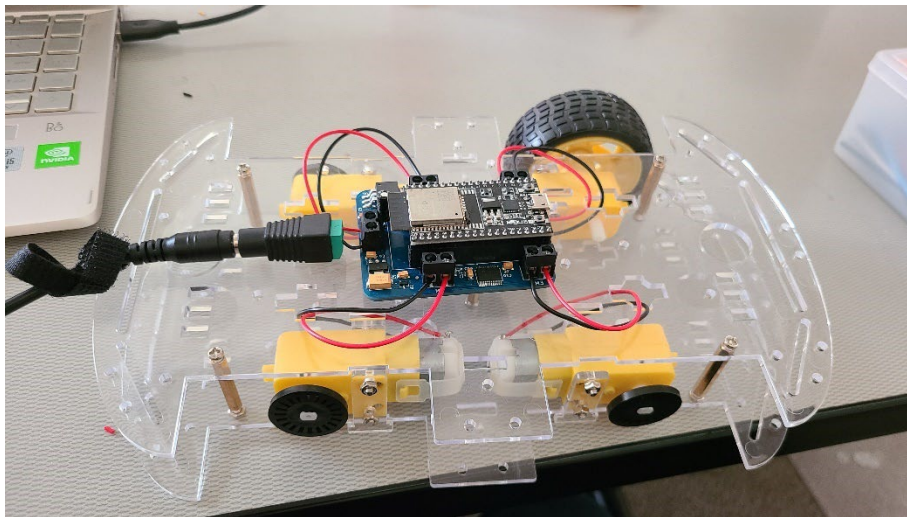


*Figure 5: Front View of Soldered Remote-Control Board*



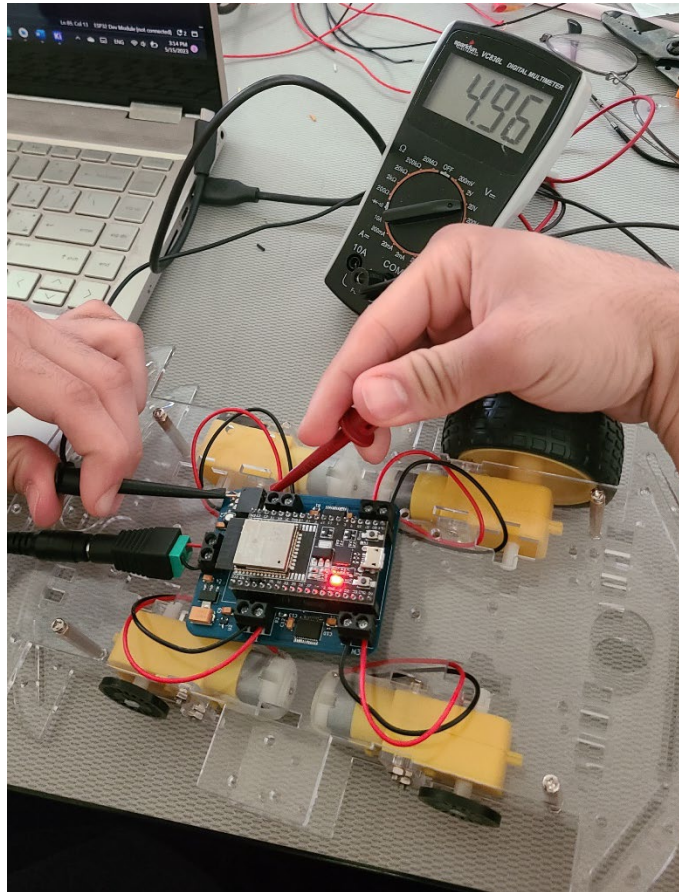
*Figure 6: Back View of Soldered Remote-Control Board*

To test the device, all the motors were connected to the appropriate terminals, and 12V was supplied from an AC-to-DC adapter to the power input pins. The hookup of the system is shown in Figure 7:



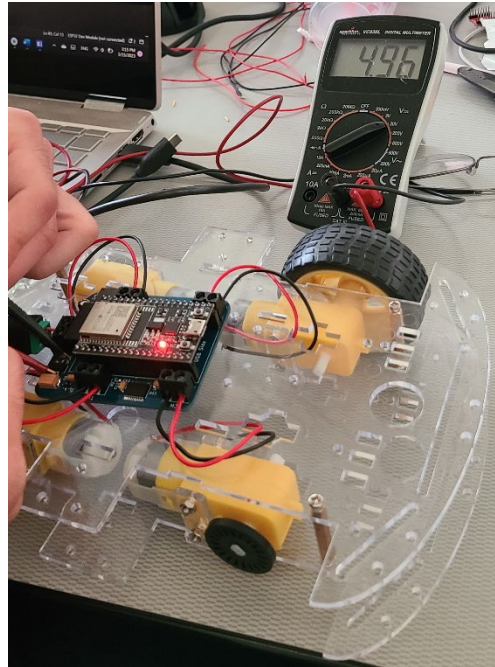
*Figure 7: Hookup of Remote-Control Board with Chassis*

When all the devices are powered, the micro-controller did not function properly. The micro-controller would not output a voltage on its GPIO pins. After testing with a multimeter, it was discovered that the buck converters on the board were shorted, since both the 3.3V and the 5V buck converters had an output of 5V. Pictures of the multimeter for both the 3.3V and 5V buck converters are shown in Figure 8 and Figure 9:



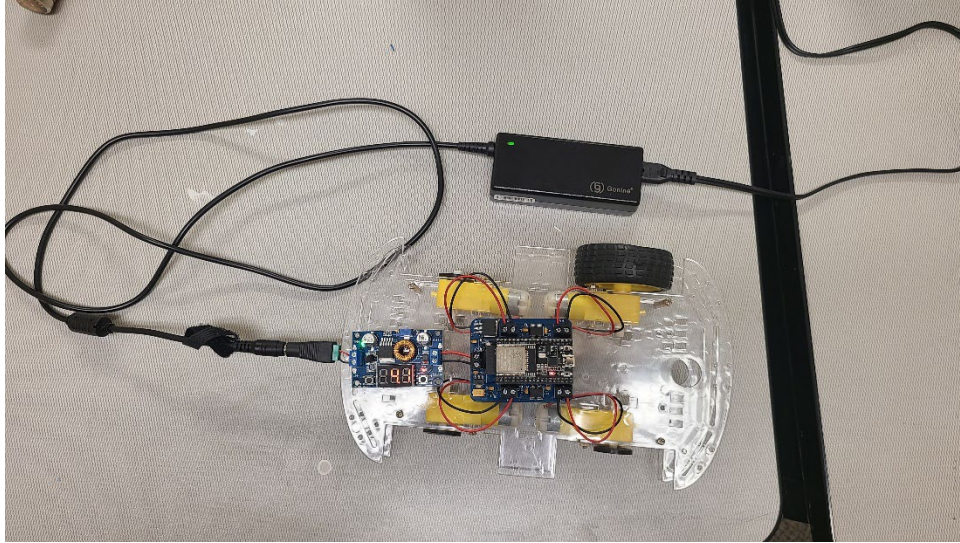
*Figure 8: Voltmeter placed across the ground and output pins of 5V buck converter.*





*Figure 9: Voltmeter placed across the ground and output pins of 3.3V buck converter.*

Thus, the 3.3V pin on the ESP32 micro-controller was not being properly powered. Luckily, the motors used in the chassis are capable of operating between the range of 3V to 6V. Thus, instead of supplying 12 volts, a 4.3V volt power input was used with the help of a small buck converter. When the input to the board is around 4.3V with a tolerance of 0.2V, the buck converters step down the voltage to around 3.3V which allows both the micro-controller and the motors to function. The final hookup of the system is shown in Figure 10:



*Figure 10: Hookup of Remote-Control Board with Chassis and External Buck Converter*

After properly powering the micro-controller, all the motors were successfully run. First, the power is connected to the Remote control driver. Afterwards, the android application is launched, the connect button is pressed to connect the android application to the ESP32 device using Bluetooth. After the successful connection, all the buttons which correspond to the functions outlined in Table 3 were tested, and they were all successful.

### **Conclusion**

Overall, the Remote-Control board was successful in being able to connect to an application using Bluetooth, and controlling motors based on instructions received from the Android application. The on-board buck converters unfortunately did not function as intended, and a second iteration of the board may be necessary to fix the problem. However, all the signal distribution on the board works as intended, and the TB612FNG motor driver successfully drove all the motors.



Several improvements can be made to the project. First, both the firmware and the android application can be revamped to allow the user to control both direction and the speed of the car directly from the application. The Android application can be worked on to improve the user interface, and even create a joystick like button in the app, which would allow the user to control both the direction and the speed of the motor directly from the application. Similarly, the firmware for the ESP32 can be recreated to receive PWM level signals directly from the application, and more sophisticated motor control function can be created to allow direct control of the direction of the car. Finally, to reduce power consumption, the firmware and the bluetooth application can be remade to use Bluetooth Low Energy (BLE) technology instead of Bluetooth Classic, since BLE is a more sophisticated protocol that allows the ESP32 bluetooth module to turn off when not being used, instead of Bluetooth Classic which uses consumers power at all times even when it is not being used. This would greatly extend the battery life of the remote-controlled car

## Appendix A: ESP32 Firmware:

```
#include <Arduino.h>
#include "BluetoothSerial.h"

const int M1_1 = 33;
const int M1_2 = 25;
const int M1_PWM = 18;

const int M2_1 = 17;
const int M2_2 = 16;
const int M2_PWM = 23;

const int M3_1 = 26;
const int M3_2 = 27;
const int M3_PWM = 19;

const int M4_1 = 21;
const int M4_2 = 13;
const int M4_PWM = 22;

const int STBY_M1M3 = 32;
const int STBY_M2M4 = 4;

BluetoothSerial SerialBT;

String device_name = "ESP32-BT-Slave";

void setup()
{
  Serial.begin(115200);

  pinMode(M1_1,OUTPUT);
  pinMode(M1_2,OUTPUT);
  pinMode(M1_PWM,OUTPUT);

  pinMode(M2_1,OUTPUT);
  pinMode(M2_2,OUTPUT);
  pinMode(M2_PWM,OUTPUT);

  pinMode(M3_1,OUTPUT);
  pinMode(M3_2,OUTPUT);
  pinMode(M3_PWM,OUTPUT);

  pinMode(M4_1,OUTPUT);
  pinMode(M4_2,OUTPUT);
  pinMode(M4_PWM,OUTPUT);

  pinMode(STBY_M1M3,OUTPUT);
  pinMode(STBY_M2M4,OUTPUT);

  digitalWrite(STBY_M1M3,HIGH);
  digitalWrite(STBY_M2M4,HIGH);
```

```

    SerialBT.begin(device_name); //Bluetooth device name
    Serial.printf("The device with name \"%s\" is started.\nNow you can pair it with Bluetooth!\n",
device_name.c_str());

    delay(1000);
}

void loop() {
    if (Serial.available()) {
        SerialBT.write(Serial.read());
    }
    if (SerialBT.available()) {
        char data = SerialBT.read();
        Serial.write(data);
        if(data == 'N') {
            north();
        } else if(data == 'E'){
            east();
        } else if(data == 'W'){
            west();
        } else if(data == 'S'){
            south();
        } else if(data == 'O'){
            stop();
        }
    }
    delay(20);
}

void north() {
    runMotor1(1);
    runMotor2(1);
    runMotor3(1);
    runMotor4(1);
}

void south() {
    runMotor1(0);
    runMotor2(0);
    runMotor3(0);
    runMotor4(0);
}

void east() {
    runMotor1(0);
    runMotor2(1);
    runMotor3(0);
    runMotor4(1);
}

void west() {
    runMotor1(1);
    runMotor2(0);
    runMotor3(1);
    runMotor4(0);
}

```

```

}

void stop(){
  analogWrite(M1_PWM,0); //Speed control of Motor A
  analogWrite(M2_PWM,0); //Speed control of Motor A
  analogWrite(M3_PWM,0); //Speed control of Motor A
  analogWrite(M4_PWM,0); //Speed control of Motor A
}

void runMotor1(int dir){
  if (dir == 0) {
    digitalWrite(M1_1,HIGH); //Motor A Rotate Clockwise
    digitalWrite(M1_2,LOW);
  } else {
    digitalWrite(M1_1,LOW); //Motor A Rotate Clockwise
    digitalWrite(M1_2,HIGH);
  }
  analogWrite(M1_PWM,124); //Speed control of Motor A
}

void runMotor2(int dir){
  if (dir == 0) {
    digitalWrite(M2_1,HIGH); //Motor A Rotate Clockwise
    digitalWrite(M2_2,LOW);
  } else {
    digitalWrite(M2_1,LOW); //Motor A Rotate Clockwise
    digitalWrite(M2_2,HIGH);
  }
  analogWrite(M2_PWM,124); //Speed control of Motor A
}

void runMotor3(int dir){
  if (dir == 0) {
    digitalWrite(M3_1,LOW); //Motor A Rotate Clockwise
    digitalWrite(M3_2,HIGH);
  } else {
    digitalWrite(M3_1,HIGH); //Motor A Rotate Clockwise
    digitalWrite(M3_2,LOW);
  }
  analogWrite(M3_PWM,124); //Speed control of Motor A
}

void runMotor4(int dir){
  if (dir == 0) {
    digitalWrite(M4_1,HIGH); //Motor A Rotate Clockwise
    digitalWrite(M4_2,LOW);
  } else {
    digitalWrite(M4_1,LOW); //Motor A Rotate Clockwise
    digitalWrite(M4_2,HIGH);
  }

  analogWrite(M4_PWM,124); //Speed control of Motor A
}

```