ELEC 3907-A

Engineering Project

Winter 2023

Final Report

Gesture-controlled Alpha-bot23

Group 1C

Jenrolaoluwa Lasisi 101154388

Ansar Mukash 101165672

Enshuo(Simon) Li 101164840

Hussin Abdullah 101155276

Debjyoti Biswas (Ron) 101177900

April 12, 2023

Carleton University

Executive summary [Hussin Abdullah]

There are roughly 5.4 million people in the United States alone who are living with paralysis[1]. To these millions of people who suffer from limited motor capabilities, or are unable to utilise their legs, there must be technological advancements made that allow them to employ and operate everyday machinery and technology easily. For this reason our group has dedicated time and effort into creating a solution which allows for extra input and functionality tied into a movement that is easily done such that it allows for the broadest use-case possible. This solution can be used to empower those suffering from paralysis and limited motor function and allow them to do tasks they would otherwise be unable to; such as driving a car.

The solution we have developed is a glove which captures the tilt of the hand and uses that information to control the movement of a small radio-frequency car. Our goal by doing this is to create a system which provides added functionality to those that need it. On the glove additional functions can be added in the form of buttons which provide further functionality. The glove will communicate to the car using radio frequency waves which operate in the 2.4-2.5 GHz frequency range. It is able to send data at a rate of up to 2Mbps[2]. The data sent by these Radiofrequency modules is interpreted and stored by two arduino mini computers. The arduino nano is utilised on the glove and the arduino mega on the car. The two-system design is utilised to show the advantage and use case for the glove and the real world applications provided by the extra functionality.

The design we have come up with has many advantages that separate it from the competition. First of all is its lightweight properties. The glove features an arduino nano as its main component which weighs only 7 grams. This is incredibly important as the time the device is being used can be for hours where any unnecessary weight will cause unwanted strains. Furthermore, the design implements an environmentally ethical rechargeable battery which, coupled with low power consumption provides a solution which is environmentally conscious. Furthermore our design allows for room for growth which allows for the implementation of further functions through buttons, sensors and other modules on the glove which can provide a unique design for each individual and their specific needs.

The project will provide added functionality to those that really use it. Furthermore it shows exactly what we as engineers can do through the car and its functions. By making the car as complex as possible it shows how a complex system can be controlled by a simple input that can be utilised by anyone. The next steps will be to broaden the horizon such that the glove can be connected to more than just the car but rather to other developments that suit the user's needs. An example of this can be something small like a remote control vacuum cleaner to another control method for driving a car without the need for pedals. Overall the aim of this is to provide those who need it with extra functionality and control.

Introduction [EnshuoLi]

In this project, our team set out to create a new remote-controlled car that included an automatic braking system and a hand-controlled remote system to improve its performance and usability. Our goal was to transform our initial idea into a working prototype, showcasing our combined abilities in engineering, programming, and design. As we moved through each project stage,

we encountered various challenges and obstacles that we successfully overcame, which led to the development of a truly unique and impressive hand-controlled remote-controlled car.

This hands-on experience taught us many things about working together effectively, managing our time wisely, and finding solutions to problems that arose during the project. These important lessons will help us in any future projects we undertake.

In this detailed account of our project, We will discuss the steps we took to design and build the car, the challenges we faced, and the solutions we found. By sharing our experience, we hope to inspire others to take on similar projects and learn from the challenges and successes we encountered.

Alternative design details [Jenrolaoluwa Lasisi]

The table below highlights the part initially chosen for the project and the alternative choice. We chose our part based on the following factors: Cost, affordability, Efficiency and Carbon footprint.

Original part	**Final part**
Glove	
MPU6050	MPU6050
Arduino Nano	Arduino Nano
WIFI	NFR2401
18650 batteries	9v lithium ion batteries
Motor	
IR Distance sensor	Ultrasonic distance sensor
WIFI	NFR2401
L928N motor driver	L928N motor driver
Arduino Uno	Arduino Nano
18650 batteries	9v lithium ion batteries
5V DC motor	5V DC motor

Table 1: Original parts and final parts

Over the course of this project we iterated over the choice of alternative design. Some of this choices include:

1. **NRF2401 transceiver**: At the start of this project, we brainstormed over a range of communication systems including WIFI, bluetooth and RF. We decided to use the NRF2401 transceiver because of its ease to program, affordability and its long range.
2. **9V lithium ion rechargeable battery**: We initially decided to use the 18650 battery but because of cost, we opted for two 9V batteries. These batteries were sufficient and provided adequate power to the arduino, motor driver and motors, as well as all other subsystems on the robot.
3. **Ultrasonic distance sensor**: For obstacles detected, we initially planned to use an Infrared distance sensor. During our research, we found out that using this option will cause detection issues on transparent objects as the reflected infrared rays may not be sufficient enough to be actually sensed by the receiver. To alleviate this issue, we utilised the HC-SR04 ultrasonic distance sensor module. This new sensor uses sound for detection hence, it is not fooled by glass.
4. **Custom built chassis**: During the initial phase of the project, we decided to buy an off-the-shelve chassis. We were immediately faced with a lot of issues such as: space,strength, ease of use and Affordability. We decided to design and 3D print our chassis from scratch. Doing this allowed us to come up with a design that does not only fit our parts perfectly, but also helps us in weight savings and strength. Also since we printed it PLA, the chassis was easy to make and had a reduced carbon and pollution footprint.
5. **Arduino Mega**: We started off building the robot with an Arduino Uno but soon realised that the input and output pins on the Uno were insufficient. To combat this problem, we decided to go with the Arduino Mega which provided us with more pins. The Arduino Mega also had a flash memory of 256kB compared to the 32 kB of the Arduino uno. This extra flash space enabled our code to run more efficiently and enabled to write more code to ensure that the robot runs as optimally as possible

We believe that we have come up with the best design that is efficient, affordable, easy to manufacture and has a low carbon footprint. Other design choices listed above are great options but a combination of all the alternative parts we have used has allowed us to come up with the best version of our Alphabot23.

Technical details with sub-sections written by individuals

Glove control system design [Enshuo Li]

This part of the design of our gesture-controlled remote car system allows users to control the car using hand movements. This innovative system can assist individuals with limited mobility or those who want to use gestures for calling services. The control system is versatile and can be used for various remote devices, such as lighting systems, robotic arms, or even real cars. The system consists of a gyroscope for collecting user movement data, a transmitter to send signals to the remote car, and a glove for the user to wear. To control the remote car, we use hand rotation to indicate turning left or right, while moving the hand up and down represents forward and backward movement. We first record the movement data, which serves as a benchmark for debugging our code. We use the MPU-6050 device, a 3-axis gyroscope and accelerometer combined, to control the movement and send signals to the remote car. The device communicates with an Arduino Nano microcontroller via the I2C communication protocol. The Nano processes and analyses the data and sends messages to the transmitter, which then controls the remote car. We also use the Arduino library for MPU-6050 for coding and testing purposes.The dead zone concept is crucial in designing

the controller. It prevents the controller from being too sensitive or sluggish by filtering out minor movements. We can set a dead zone range for all axes or adjust them individually to ensure the device sends signals under reasonable situations. In conclusion, this remote control system is designed to control our remote car project and can be adapted for other purposes, such as controlling smart home devices. When coding the system, it is essential to choose an appropriate dead zone to ensure the device operates effectively and accurately. This innovative design offers a hands-free solution for controlling remote devices, providing convenience and accessibility to users.

DC motor and motor driver implementation [Ansar Mukash]

1. System description

The car's movement is achieved through four DC motors connected to an L298N motor driver. The focus of this section will be the exploration of the motors and motor driver of the remote-control car. For detailed specifications of those components can be found in section 5 of this report.

The L298N motor driver is a popular choice for controlling DC motors. It is a high-current, dual full-bridge driver that is designed to accept standard TTL logic levels. The L298N is a dual H-bridge motor driver that can control the speed and direction of two DC motors [3]. In our case, we decided to connect four motors to one L298N. In this way, two motors on the left are controlled by pins in1 and in2, and the right two motors are controlled by pins in3 and in4.

To feed the motors, two 9V rechargeable batteries connected in series were connected to the motor driver to feed all four motors. This power proved to be enough to be able to operate all four motors. Additionally, the L298N has a 5V output pin that can be used for powering the Arduino board on the car.

Depending on the voltage polarity in the respectable pins, the motors will rotate one way or another. For example, if in1 is set to HIGH and in2 is set to LOW, the motors on the left side will rotate so that our car will start the forward movement, if we switch the polarity of the pins, the car will move backwards. If both in1 and in2 are the same, the motor will stop the rotation. By connecting the enA and enB pins, the speed of the car can now be controlled. In order to regulate the speed, PWM signals are utilised. PWM digital signals simulate an analog signal. They are used to control the speed of the motors. The PWM signal is generated by the Arduino board and sent to the motor driver. The motor driver then adjusts the voltage supplied to the motors based on the PWM signal. In our project, once the values from the accelerometer are collected, they are mapped to match the specifications of PWM of the motor driver (from 0 to 255). After that, the information of the acceleration and its direction is sent to the car and fed through the pins mentioned above.

2. Assembly process

The first step in the project was to connect the motors and motor driver to the Arduino board. The connections were made as follows:

The DC motors were connected to the L298N motor driver as per the datasheet [3].

- Connect the positive terminal of each motor to the OUT1, OUT2, OUT3, and OUT4 pins of the motor driver.
- Connect the negative terminal of each motor to the GND pin of the motor driver.
- Connect the negative terminal of each motor to the GND pin of the motor driver.

· The L298N motor driver was connected to the Arduino board.

- Connect the VCC pin of the motor driver to the 5V pin of the Arduino board.
- Connect the IN1, IN2, IN3, and IN4 pins of the motor driver to digital pins on the Arduino board.
- Connect the ENA and ENB pins of the motor driver to analog pins on the Arduino board.

The NRF24L01 RF transceiver was also connected to the Arduino board.

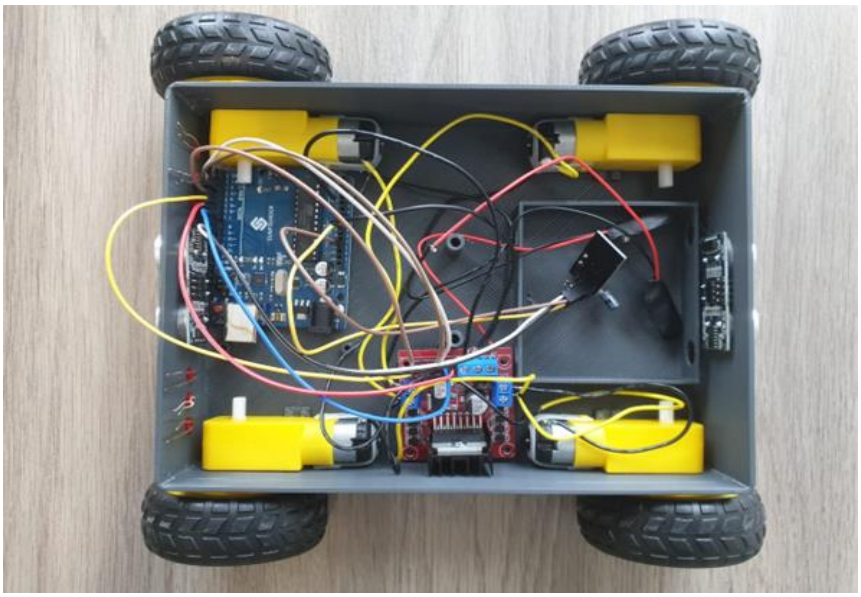


Figure 1: Car top view

3. Coding

Once the connections were made, the next step was to write the code to control the motors. The code was written in the Arduino IDE and uploaded to the Arduino board. The code involved setting the speed and direction of the motors. The speed and direction of the motors were controlled using PWM signals.

Here is the example code in Arduino IDE which is the part of the receiver code. Once the signal is received from the glove with the accelerometer values, the following code will initialise the movement of the wheels.

```
// int motor_speed = map(abs(x_tilt), 0, 100, 0, 255); // map the tilt value to motor speed
// int turn_speed = map(abs(y_tilt), 0, 100, 0, 255); // map the tilt value to turn speed

if (x_speed > 0) { // move forward
  analogWrite(enA, x_speed); // set the speed of left side wheels
  digitalWrite(in1, LOW); // set the direction of left side wheels
  digitalWrite(in2, HIGH);
  analogWrite(enB, x_speed); // set the speed of right side wheel
  digitalWrite(in3, LOW); // set the direction of right side wheel
  digitalWrite(in4, HIGH);
}
else if (x_speed < 0) { // move backward
  analogWrite(enA, x_speed); // set the speed of left side wheels
  digitalWrite(in1, HIGH); // set the direction of left side wheels
  digitalWrite(in2, LOW);
  analogWrite(enB, x_speed); // set the speed of right side wheel
  digitalWrite(in3, HIGH); // set the direction of right side wheel
  digitalWrite(in4, LOW);
}
else { // stop moving
  digitalWrite(in1, LOW); // stop left side wheels
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW); // stop right side wheel
  digitalWrite(in4, LOW);
}

if (y_speed > 0) { // turn right
  analogWrite(enA, y_speed); // set the speed of left side wheels
  digitalWrite(in1, LOW); // set the direction of left side wheels
  digitalWrite(in2, HIGH);
  analogWrite(enB, y_speed); // set the speed of right side wheel
  digitalWrite(in3, HIGH); // set the direction of right side wheel
  digitalWrite(in4, LOW);
}
else if
```

Figure 2: Arduino IDE code snippet

This code sets the speed and direction of each motor using variables (x_speed and y_speed), received from the glove and sent to the analogWrite() in a form of PWM signal. The digitalWrite() function is used to control the direction of each motor, with HIGH setting the motor to rotate forward and LOW setting the motor to rotate backward.

Another feature that is implemented into the code is the dead zone. When the accelerometer values are close to zero (no more or less than value 3), the motors are at halt, meaning in1, in2, in3, and in4 are all in digital LOW.

4. Challenges encountered

One of the challenges our team encountered while working with the DC motors was the breakability of the fused wiring. During the project, at least 2 motors had this issue, when the little metal plate that connected the outer wires to the brushes inside the motor simply broke off, and it was nearly impossible to solder. At first, we tried to disassemble the motor, and replace the brushes with the actual wires. This technique worked for a few sessions, but later failed, probably due to the unstable connection of wires inside the motor. Eventually, we replaced the motors with new ones.

All DC motors should have similar specifications, such as voltage, current, and RPM. However, in practice, the movement of the car was not perfectly straight, and it tended to slowly turn to the right as it moved forward. One of the theoretical reasons our team has concluded is that some motors performed better than the others due to different build quality and specifications. This was partially fixed by limiting the maximum speed of the car, which redistributed the power more evenly and the car was moving forward properly.

Programming of the transmitter side of a two-way communication module

[Hussin Abdullah]

In this project my main focus was the programming and coding of the transmitter side of a two way communication module. This communication module is needed in order to send the direction and magnitude of movement to the car from the MPU6050 accelerometer situated on the glove. Therefore, by tilting the glove about the y-axis the car should experience a forward or backwards movement. Similarly tilting the glove about the x-axis should cause the car to turn either left or right. The communication is conducted through an NRF24L01 radio frequency transmitter. This transmitter operates at 2.4Ghz. In radio communication there are two main speeds normally used. 2.4GHz, and 5GHz. In this case we decided to use a 2.4GHz frequency as although it is slower than the 5GHz frequency the main benefit of 2.4GHz is it offers superior range[4]. In this case we determined the lower speed is not a significant enough drawback whereas the extended range is a great benefit for allowing the car to travel as far as possible.

```
1  //Include Libraries
2  #include <SPI.h>
3  #include <nRF24L01.h>
4  #include <RF24.h>
5  #include <Wire.h>
6  #include <MPU6050.h>
```

Figure 3: Library declaration

The beginning of the glove is the simplest, in this section we simply declare which libraries we are using. These libraries provide ease of access as well as extra functionality when working with external hardware connected to the arduino. In this case we have both the Radio frequency module and MPU6050 6-axis gyroscope module. By including the library the arduino now understands which modules are connected and this will provide functions which we are able to call which make programming these specific modules in relation to the arduino much simpler.

```
8  //create an RF24  and MPU6050object
9  RF24 radio(9, 8);  // CE, CSN
10 MPU6050 mpu;
11
```

Figure 4: Declaring pins

The next step of the code is to declare the pins and modules. In this case the Radio frequency module simply labelled as RF24 has the CE and CSN pins connected to the arduino pins 9 and

8 respectively. This simply tells the arduino that the connections made to pins 9 and 8 are those of the CE and CSN from the Radio frequency module.

The following figure quickly shows the pins of a NRF24L01 RF module such that you get a better understanding of the connections regarding the code:

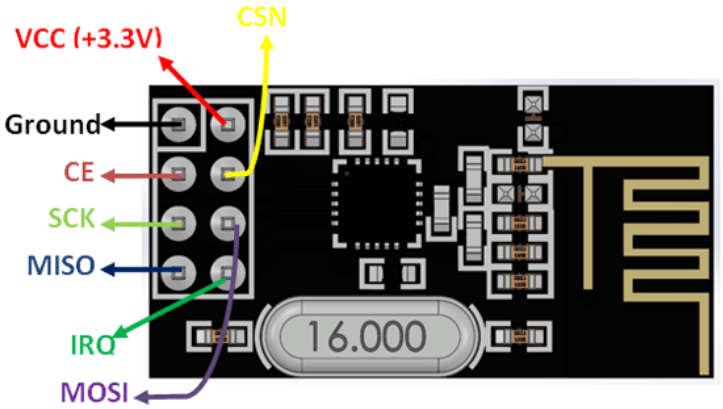


Figure 5: NRF24L01 pins

```
13  unsigned long timer = 0;
14  float timeStep = 0.01;
15
16  // Pitch, Roll and Yaw values
17  float pitch = 0;
18  float roll = 0;
19
20  //address through which two modules communicate.
21  const byte address[6] = "00001";
22
23  int text[2] = {};   //constant to store gyrodata
24
```

Figure 6: initialising values

In this part of the code I am simply initialising values that are going to be used in future portions of the code. In particular the setup and void loop. The unsigned long timer and time step creates a timer from a 32 bit unsigned long value. We use unsigned because we don't want to have negative values for time. The flow for pitch and roll are used to calculate and describe the tilt of the glove based on the tilt of the hand as demonstrated in the figure below.

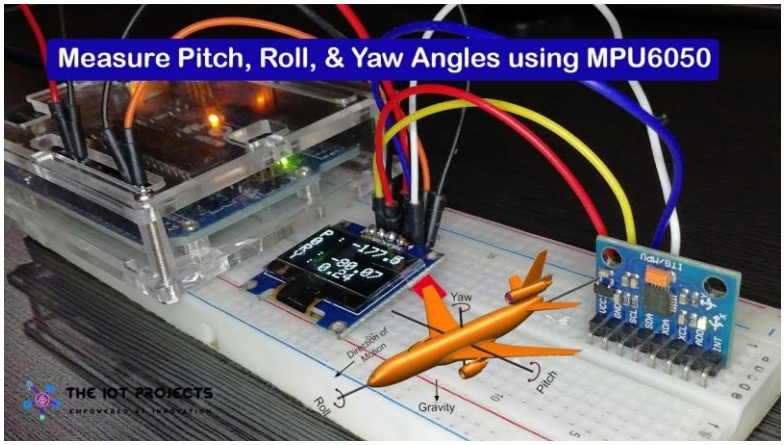


Figure 7: Demonstration of Pitch and Roll [5]

Next we must open a channel in which the two transmitters communicate to each other. In this case the channel I have decided to use is a 6 byte channel with the address value of "000001". These work essentially as IP addresses in which the receiver will only accept data from transmitters with the specified address initialised. On the receiver side the same address must be initialised for proper communication. Finally the int text[2] = {} is the constant which is used to store the data from the tilt of the glove and transmit it over to the car in order to initialise the motor driver and drive the car and set its direction;

```
25  void setup()
26  {
27
28    Serial.begin(115200);
29
30    // Initialize MPU6050
31    while(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G))
32    {
33      Serial.println("Could not find a valid MPU6050 sensor, check wiring!");
34      delay(500);
35    }
36
37    //Calibrate the gyroscope
38    mpu.calibrateGyro();
39
40    // Set threshold sensivty. Default 3.
41    mpu.setThreshold(3);
42
43    pitch = 0;
44    roll = 0;
45
46    //nrf24 setup
47    radio.begin();
48
49    //set the address
50    radio.openWritingPipe(address);
51
52    //Set module as transmitter
53    radio.stopListening();
54  }
```

Figure 8: Void Setup code transmitter

The next step is to set up the conditions for the loop. serial.begin sets the baud rate in which information is set to the receiver. In this case it is configured to send 115200 signal changes per second. This is done to accurately measure each movement by the glove and ensure there is no delay between inputs and output. By installing the proprietary library from the internet we are able to use the initialise gyroscope command which sets the gyroscope up and initialises the values to 0 based on the starting position of the hand. The next step is the threshold which is set as 3, this essentially sets a deadzone so the glove isn't too sensitive and gives unwanted inputs. Finally we set the values for pitch and roll and begin the communication to the car. Finally we have the loop which calculates the direction and magnitude of the tilt and converts

that into data that will be sent over the communication channel and into the car which will then drive the motor drivers based on the given information.

```
56  void loop()
57  {
58    timer = millis();
59
60    // Read normalized values
61    Vector norm = mpu.readNormalizeGyro();
62
63    // Calculate Pitch, Roll and Yaw
64    pitch = pitch + norm.YAxis * timeStep;
65    roll = roll + norm.XAxis * timeStep;
66
67    // Wait to full timeStep period
68    delay((timeStep*1000) - (millis() - timer));
69
70    text[0] = roll;
71    text[1] = pitch;
72
73    radio.write(&text, sizeof(text));
74
75    //Serial.println("roll");
76    Serial.println(roll);
77    //Serial.println("pitch");
78    //Serial.println(pitch);
79
80    delayMicroseconds(10);
81  }
82
```

Figure 9: Void Loop code transmitter

In this loop we constantly read the values of the pitch and roll and add a short delay such that the arduino on the car doesn't get clogged with an information overload. Next we simply set up two text packages, one which contains the data for the tilt value of the roll and another which contains the tilt value of the pitch. These are both then written into the radio and sent to the car where it will be read by the arduino mega situated on the car and transferred into data that will be used to drive the motors based on the magnitude and direction of the tilts. The aim of this code was to create the transmitter side of the communication module which sends data dependent on the tilt of the glove and uses that information to drive the motors and wheels situated on the car to get the correct turn and speed magnitude values depending on the user's input.

Coding of the Receiver side of a two-way communication module. [RON BISWAS]

The NRF24 receiver is a wireless module that receives data packets from the transmitter through the 2.4 GHz frequency. The system consists of an RF24 receiver installed on the automobile The hand gesture control device delivers wireless signals to the RF24 receiver fitted on the automobile after detecting the hand gestures made by the user. The signals are transformed into commands for the automobile to proceed in the desired direction by the RF24 receiver. For wireless connection, the system makes use of an RF24 radio module and an Arduino microcontroller. The Arduino programming language was used to create the code for the RF24 receiver, which initialises the radio module and waits for a signal from the hand gesture control unit. When a signal is received, the code decodes it to identify the hand gesture sequence and sends commands to the motor driver module to move the automobile in that direction.

The screenshots of the code for RF24 receiver, are given below:

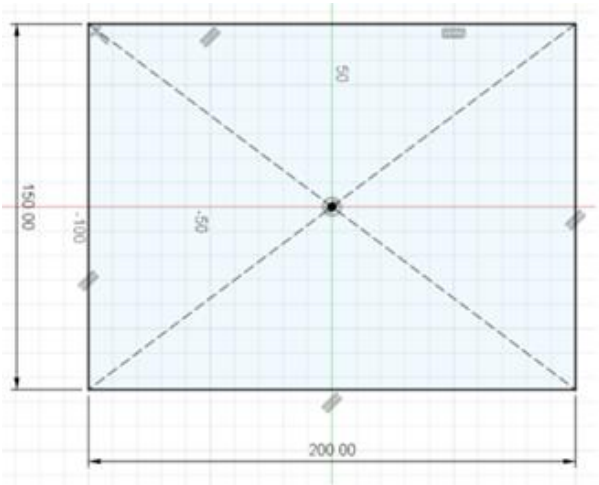
```ino
#include <Wire.h>
#include <SPI.h>
#include <RF24.h>

RF24 radio(9, 8); // CE, CSN

const uint64_t pipe = 0xE8E8F0F0E1LL; // Pipe address for communication

int enA = 6;  // PWM pin for speed control of left side wheels
int in1 = 7;  // direction control pin 1 for left side wheels
int in2 = 2;  // direction control pin 2 for left side wheels
int enB = 5; // PWM pin for speed control of right side wheel
int in3 = 3; // direction control pin 1 for right side wheel
int in4 = 4; // direction control pin 2 for right side wheel

void setup() {
  pinMode(enA, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  radio.begin(); // start the radio communication
  radio.openReadingPipe(1, pipe); // open the communication pipe
  radio.setPALevel(RF24_PA_MIN); // set the power level of the radio
  radio.startListening(); // start listening for incoming data
}

void loop() {
  if (radio.available()) { // check if there is any data available
    int x_speed, y_speed;

    radio.read(&x_speed, sizeof(x_speed)); // read the x-axis tilt value
    radio.read(&y_speed, sizeof(y_speed)); // read the y-axis tilt value

    // int motor_speed = map(abs(x_tilt), 0, 100, 0, 255); // map the tilt value to motor speed
    // int turn_speed = map(abs(y_tilt), 0, 100, 0, 255); // map the tilt value to turn speed

    if (x_speed > 0) { // move forward
      analogWrite(enA, x_speed); // set the speed of left side wheels
      digitalWrite(in1, LOW); // set the direction of left side wheels
      digitalWrite(in2, HIGH);
      analogWrite(enB, x_speed); // set the speed of right side wheel
      digitalWrite(in3, LOW); // set the direction of right side wheel
      digitalWrite(in4, HIGH);
    }
    else if (x_speed < 0) { // move backward
      analogWrite(enA, x_speed); // set the speed of left side wheels
      digitalWrite(in1, HIGH); // set the direction of left side wheels
      digitalWrite(in2, LOW);
      analogWrite(enB, x_speed); // set the speed of right side wheel
      digitalWrite(in3, HIGH); // set the direction of right side wheel
      digitalWrite(in4, LOW);
    }
    else { // stop moving
      digitalWrite(in1, LOW); // stop left side wheels
      digitalWrite(in2, LOW);
      digitalWrite(in3, LOW); // stop right side wheel
      digitalWrite(in4, LOW);
    }

    if (y_speed > 0) { // turn right
      analogWrite(enA, y_speed); // set the speed of left side wheels
      digitalWrite(in1, LOW); // set the direction of left side wheels
      digitalWrite(in2, HIGH);
      analogWrite(enB, y_speed); // set the speed of right side wheel
      digitalWrite(in3, HIGH); // set the direction of right side wheel
      digitalWrite(in4, LOW);
```

```
70        }
71        else if
72          (y_speed < 0) { // turn left
73          analogWrite(enA, y_speed); // set the speed of left side wheels
74          digitalWrite(in1, HIGH); // set the direction of left side wheels
75          digitalWrite(in2, LOW);
76          analogWrite(enB, y_speed); // set the speed of right side wheel
77          digitalWrite(in3, LOW); // set the direction of right side wheel
78          digitalWrite(in4, HIGH);
79          }
80            else { // stop moving
81          digitalWrite(in1, LOW); // stop left side wheels
82          digitalWrite(in2, LOW);
83          digitalWrite(in3, LOW); // stop right side wheel
84          digitalWrite(in4, LOW);
85            }
86      }
87    }
88
```

The way this code works is it initially configures the communication channel and data rate for the RF24 radio module. The code then waits for the hand gesture control mechanism to provide a signal. When a signal is received, the code decodes it to identify the hand gesture sequence and sends commands to the motor driver module to move the automobile in that direction.

Design and assembly of the chassis [Jenrolaoluwa Lasisi]

I designed the robot using the Autodesk Fusion 360 CAD software. I used design constraints such as size and weight to design the robot. I started sourcing the component to use in the project along with my team members. After I acquired this component, I measured each component and created a rough sketch of the robot. Some design changes were made after I showed the sketch to my team members and more iterations of the sketch was done. After creating a rough sketch of the robot, I proceeded to create the base sketch of the robot, then this sketch was extruded. Figure 2 shows the base sketch and the extruded body. The image on the left is the base sketch and the image on the right is the extruded sketch.



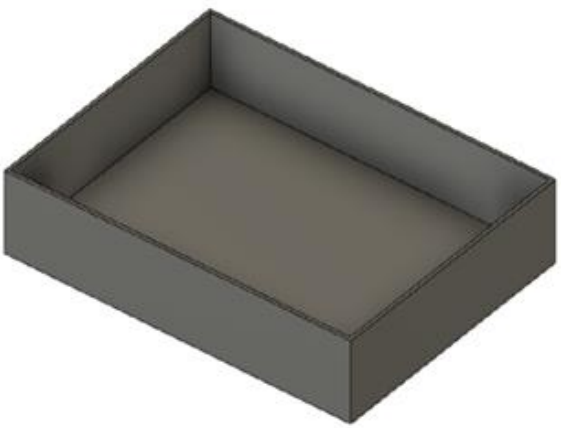


Figure 10: Base sketch and extruded body

After creating the base layer of the robot. I created a sketch to include the holes and holders for the motor. Then, I used the extrusion tools to create the holes from the sketch. Because the holes were symmetrical on both sides, I used one sketch to create all the holes. Figure 3 below shows the sketch and the extruded body. The image on the left is the base sketch and the image on the right is the extruded sketch.

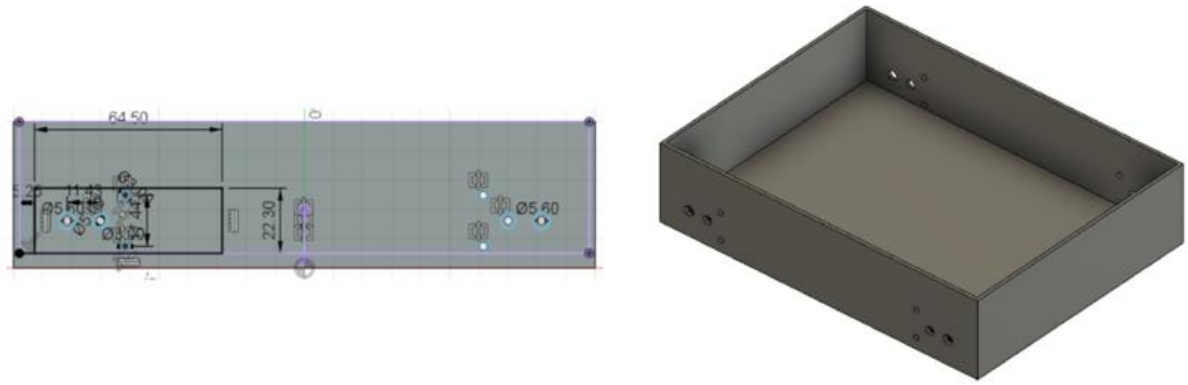


Figure 11: sketch and extruded body of the motor holes

After creating the holes on the robot, I rounded off the corners using the fillet tools. Then I created a sketch to include holes for the LEDs. Then, I used the extrusion tool to create the holes from the sketch. Figure 4 below shows the sketch and the extruded body. The image on the left is the base sketch and the image on the right is the extruded sketch.

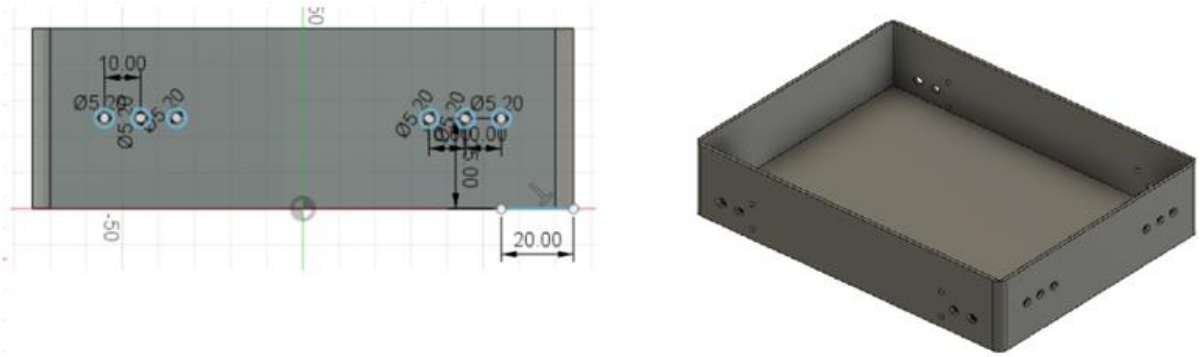


Figure 12: sketch and extruded body of the LED holes

After creating the holes for the LEDS, I created a sketch to include the holes for the distance sensors. After I created the sketch, I used the extrusion tool to create the holes for the sketch. Since the front and back distance sensor had the same Y-Z coordinates, I used only one sketch to create both holes. Figure 4 below shows the sketch and the extruded body. The image on the left is the base sketch and the image on the right is the extruded sketch.

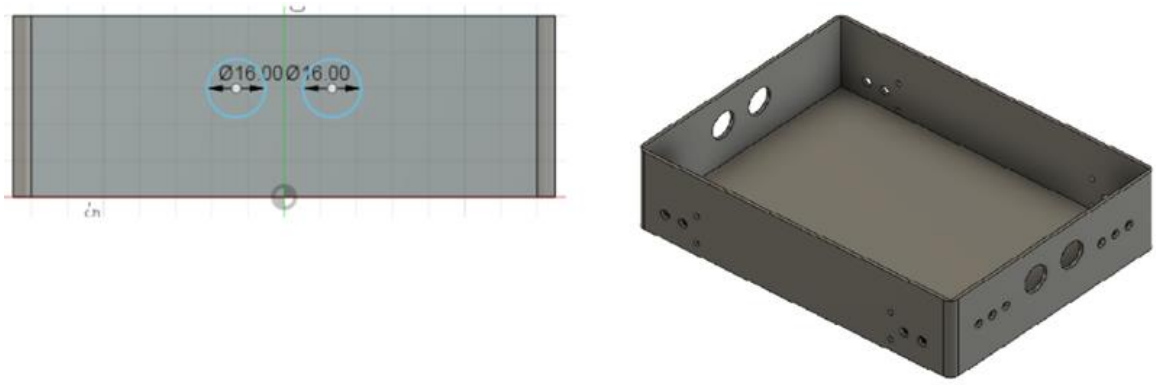


Figure 13: sketch and extruded body of the distance sensors

After creating the holes for the distance sensors, I created a sketch to include the next set of holders for the motor. After I created the sketch, I extruded the holders from the sketch. Figure 5 below shows the sketch and the extruded body. The image on the left is the base sketch and the image on the right is the extruded sketch.

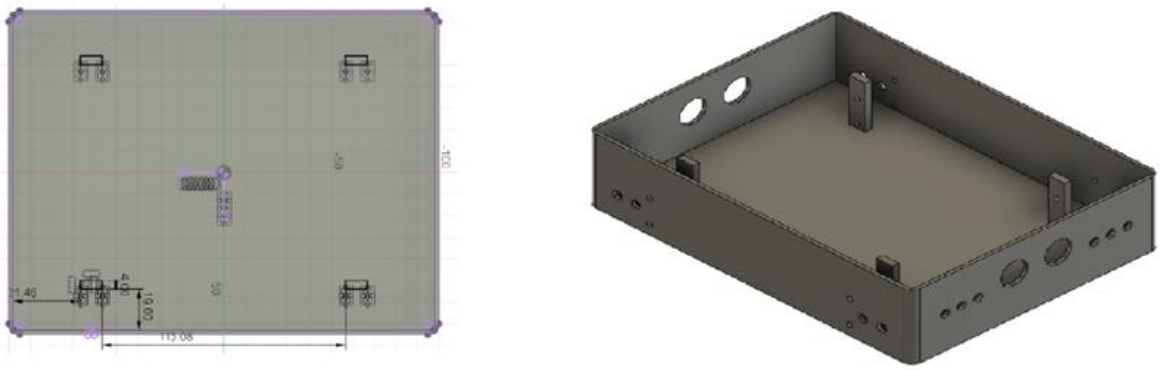


Figure 14: sketch and extruded body of the motor holders

After creating the holders for the motor, I created a sketch to include the mounting holes for the Arduino, motor driver and second layer. After I created the sketch, I extruded the mounting holes from the sketch. Figure 6 below shows the sketch and the extruded body. The image on the left is the base sketch and the image on the right is the extruded sketch.

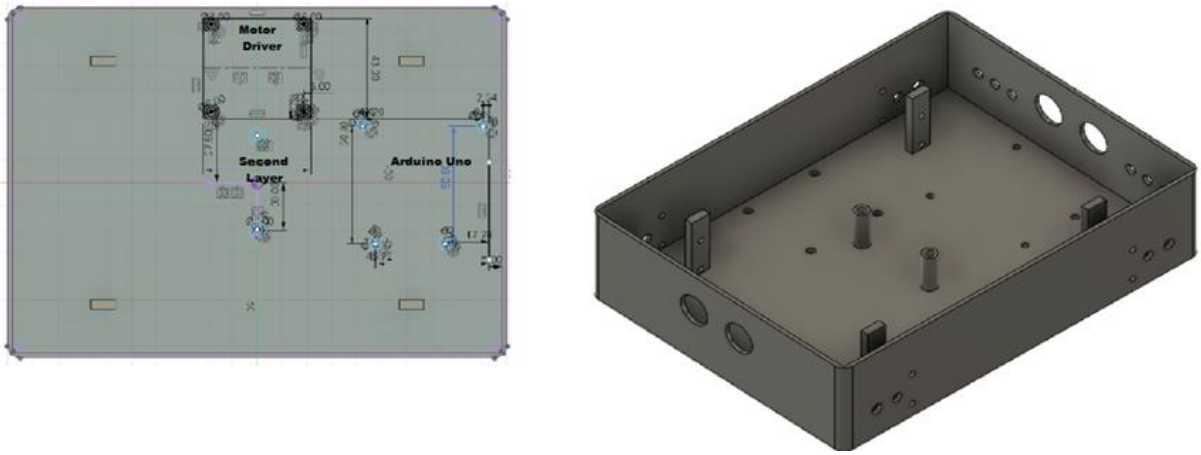


Figure 15: sketch and extruded body of the mounting holes for the Arduino, motor driver and second layer

Finally, I created a sketch to include the Battery holder. After I created the sketch, I extruded the battery holder from the sketch. Figure 7 below shows the sketch and the extruded body. The image on the left is the base sketch and the image on the right is the extruded sketch.

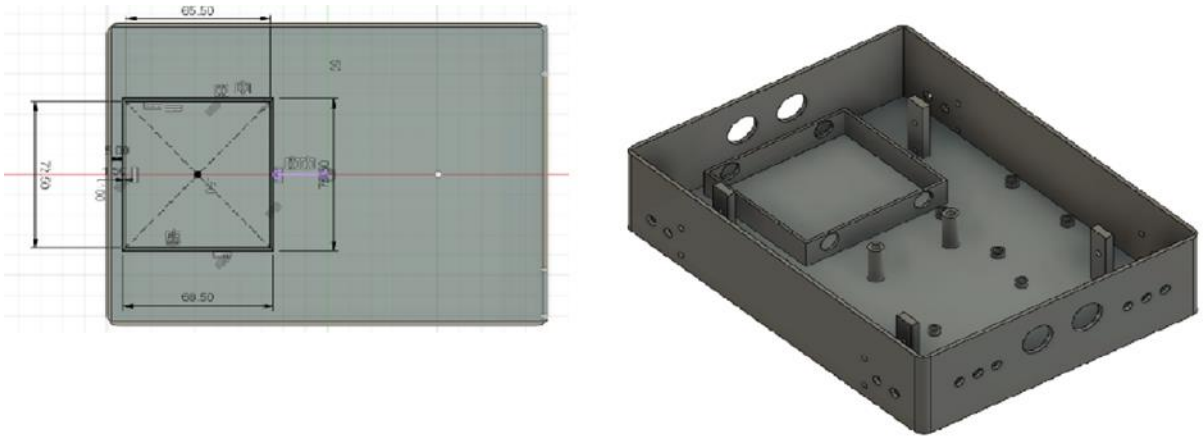


Figure 16: sketch and extruded body of the battery holder

After designing the robot, exported the design as an STL file and put in an order for it to be 3D printed. After printing, I assembled the vehicle by placing the motors, motor driver, Arduino Uno, and Batteries into the robot. Figure 7 below shows the completed and assembled design.

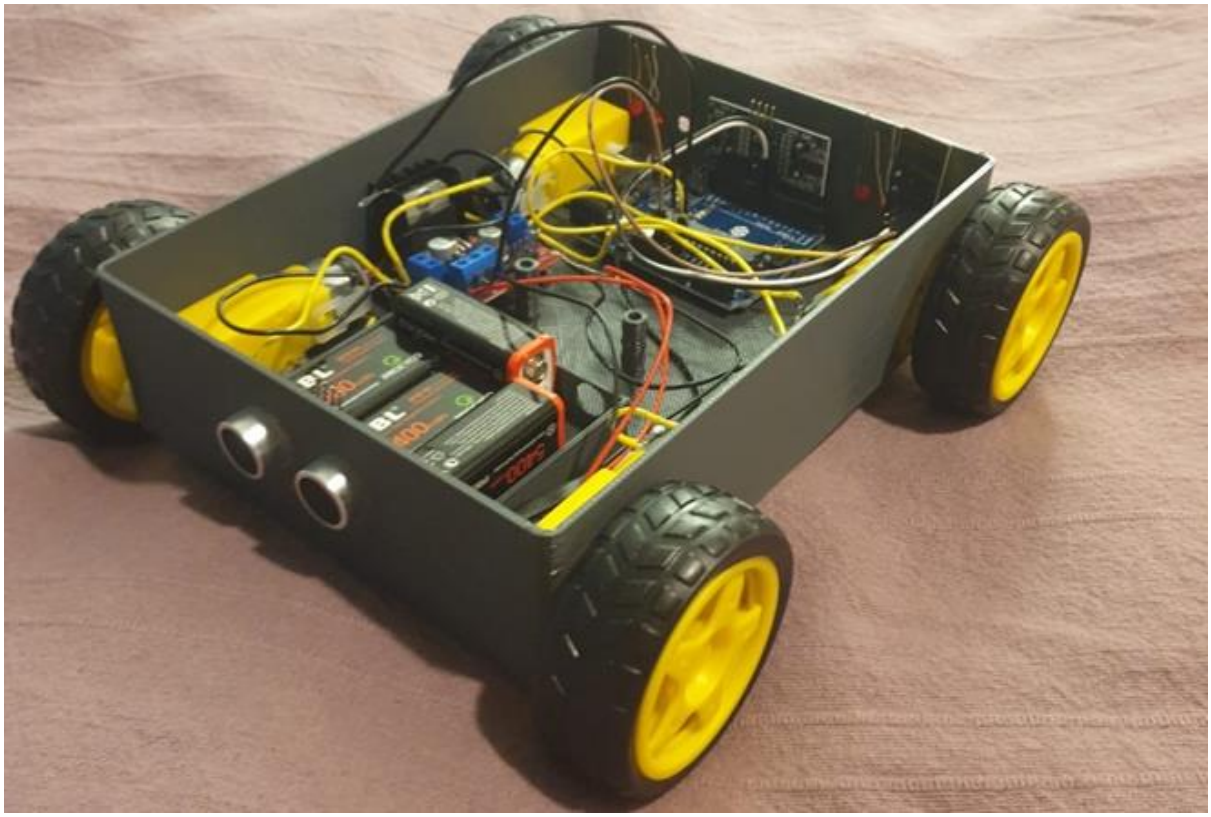


Figure 17: Completed and assembled robot

Specifications [RON BISWAS] [Hussin Abdullah]

PLA plastic (Car Chassis) - PLA is one of the easiest materials to print with because it does not require precise print settings. Temperatures are adjustable, warping is minimal, and no strong print odours are present. PLA is also well-known for producing high-quality prints. It has the best detailed reproduction of any common 3D printing filament, making it ideal for models that require a high level of aesthetics. PLA's strength is surprisingly adequate for most light prototypes and models in practical applications.[8] The statistics of this particular filament make it a great choice for our project. It has a tensile strength of about 37 Mpa which is resilient enough to endure many crashes. Furthermore, the plastic has a melting point of 173 degrees celsius, which is extremely important due to potential electrical fires or if the robot was to ever be in a situation where fire would be present. [9]

MPU6050 - The MPU6050 includes three gyroscope and accelerometer axes, for a total of six degrees of freedom, and can measure 16'384 LSB/g and 131 LSB/dps. It contains 8 pins with a 2.54 mm spacing and an I2C interface. A 16-Bit AD Converter and a Low-Dropout regulator are also included.It requires a 3.3V - 5V DC power source.[10] The MPU6050 has a minimum operating temperature of -40 degrees celsius which is extremely important as it will be situated on the outside of a glove and will need to perform in Canadian winters. Its maximum operating

temperature is 85 degrees celsius which means it can safely be used in all climates on Earth. Furthermore it has a very low standby current of 5 micro amps which means when not being utilised it has very little power draw therefore reducing the load on the batteries. [11]

NRF24L01 - The NRF24L01 is a single-chip radio transceiver that operates in the 2.4 - 2.5 GHz ISM band across the world. The term transceiver refers to a module that may function as both a transmitter and a receiver. The module's power consumption is incredibly low, at 9.0mA at -6dBm output power and 12.3mA in RX mode, which is even less than an LED. When utilised in an open area with an antenna, it has a maximum range of up to 100 metres. It operates on a voltage range of 1.9V to 3.9V.[7] The transceiver can send and receive data at rates of 250kbps / 1Mbps/2Mbps, thereby having multiple potential options depending on the degree of accuracy needed. The data rate is configured through the SPI interface. Typically NRF24L01 can transmit radio frequency through 2 or 3 standard walls with a distance of 50 feet. [12]

MOTOR DRIVER- The L298N Motor Driver Module is a high-power motor driver module that can be powered by DC power source(lithium ion batteries). The L298 motor driver IC and a 78M05 5V regulator are used in this module. The L298N Module is capable of controlling up to four DC motors or two DC motors with directional and speed control. It requires a 12 V input from a DC power source.[7]

DC MOTOR- Direct current (DC) motors are electric motors that are powered by direct current (DC), such as a battery or a DC power supply. The voltage alone may be used to regulate the speed of a brushed DC motor.

ARDUINO NANO - The Arduino Nano is a compact and multifunctional microcontroller board based on the ATmega328P. The following are the Arduino Nano's important characteristics and features: ATmega328P microcontroller is used. 5V is the operating voltage. 7-12V is the suggested input voltage. 6-20V input voltage (limits), 14 digital I/O pins (of which 6 provide PWM output), 8 DC Analog Input Pins Per I/O current Pin: 20 mA, 32 KB Flash Memory (of which 2 KB used by bootloader), SRAM size: 2 KB, EEPROM size: 1 KB, Clock frequency: 16 MHz

ARDUINO MEGA- The Arduino Mega is a microcontroller board that is based on the ATmega2560. The following are the Arduino Mega's important specs and features:ATmega2560 microcontroller is used. 5V is the operating voltage. 7-12V is the suggested input voltage. 6-20V input voltage (limits) I/O digital Pins: 54 (of which 15 provide PWM output) ,16 DC Analog Input Pins Per I/O current Pin: 20 mA, 256 KB Flash Memory (of which 8 KB used by bootloader), SRAM size: 8 KB, EEPROM size: 4 KB, Clock frequency: 16 MHz.

ADDENDUM: **What would make your project 'product' obsolete? [Ansar Mukash]**

While developing a product, our team started considering its usefulness and relevance for the society in the long run. Several potential factors could render any product on the market obsolete, including advancements in technology that improve the functionality of similar products, changing customer needs that reduce the relevance or usefulness of the product, and the emergence of new competitors offering similar products or innovations, which can make one's product less competitive.

Our project may be affected by advancements in technology, which would offer more efficient realisation in the future, but besides that, our design is not going obsolete in the foreseen future. Firstly, it is innovative and very focused for a very specific market. As a result, it will have a unique selling proposition that will not have any competition in the same space. Secondly, the glove and car project can be customised to suit the specific needs and preferences of users. This makes it more appealing to a diverse range of customers, and helps to ensure that it remains relevant over time. And lastly, this project has a potential to make a significant impact on society by improving the mobility and independence of individuals with physical disabilities. This social impact will make it more meaningful and relevant, and certainly protect our product from becoming obsolete.

For these reasons, our product checks out almost all six modes of obsolescence, that are aesthetic, social, technological, economic, use, and convenience. This will push our product towards a Circular Economy contribution, minimising waste and maximising the use of resources.[13]

Conclusion and future development [Ansar Mukash]

In conclusion, our project demonstrated the capabilities of the project by implementing the movement of the car through glove manipulation, turn signals, as well as obstacle detection and avoidance. These functionalities were achieved through the rigorous research and implementation of ultrasonic sensors, RF transceiver communication module, and sequential operation of LEDs. This is not the final iteration of the design, as there are a few functions that are yet to be implemented. With further resources and time investment, the glove functionality expansion, vertical improvements to the car's chassis, as well as return home feature could be introduced.

For the glove functionality improvement, we could add an additional LED or a buzzer for the user to recognize the obstacle warning: if the ultrasonic sensors detect an object within the critical range, the LED light on the glove will start glowing, or the buzzer will start producing a sound to notify the user of danger and prevent any accidents. Another button would work as a return home function.

With the return home feature, the car could identify its current location and the dock station location, then safely find a path to that station without any human intervention. Although impressive on paper, this feature requires a lot more thought and development than our team has anticipated, therefore this feature was left out unimplemented.

And lastly, the current chassis is designed to be expanded vertically for any additional features and functionalities. This design will prevent any shifts of centre of gravity, which is crucial for the proper operation of the car. One of the modules that was planned to be installed is an accelerometer module for the return home implementation.

Nevertheless, our project was a success and has the potential to make a significant impact on the lives of those who need it.

References

[1] “Paralysis Statistics.” *Reeve Foundation*, https://www.christopherreeve.org/living-with-paralysis/stats-about-paralysis.

[2] *NRF24L01+ Datasheet - Sparkfun Electronics*. https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf.

[3] "L298N Motor Driver Datasheet," Components101, [Online]. Available: https://components101.com/sites/default/files/component_datasheet/L298N-Motor-Driver-Datasheet.pdf.

[4] *2.4 GHz vs. 5 ghz WIFI*. CenturyLink. (n.d.). Retrieved April 11, 2023, from https://www.centurylink.com/home/help/internet/wireless/which-frequency-should-you-use.html#:~:text=2.4%20GHz%20vs.%205%20GHz%3A%20Which%20frequency%20should%20you%20choose,use%20your%20WiFi%20connection%20most.

[5] YouTube. (2022, September 13). *Measure pitch roll and yaw angles using MPU6050 and Arduino*. YouTube. Retrieved April 11, 2023, from https://www.youtube.com/watch?v=ythjrfQViRQ

[6]*Joy. (n.d.). Retrieved April 12, 2023, from https://joy-it.net/en/products/SEN-MPU6050*

[7]*NRF24L01 pinout, features, Specs, working and Arduino connections*. eTechnophiles. (2022, February 15). Retrieved April 12, 2023, from https://www.etechnophiles.com/nrf24l01-pinout-features-specs-working-and-arduino-connections/

[8]*L298N motor driver module*. Components101. (n.d.). Retrieved April 12, 2023, from https://components101.com/modules/l293n-motor-driver-module

[9]*PLA 3D printing material*. All3DP. (2022, April 25). Retrieved April 12, 2023, from https://all3dp.com/2/what-is-pla-3d-printing-materials-simply-explained/

[10] Alexandra. (2022, September 13). Pla filament: The Pros and cons of this 3D printing staple material. BCN3D Technologies. Retrieved April 12, 2023, from https://www.bcn3d.com/pla-filament-stands-for-strength-temp/#:~:text=PLA's%20strength&text=Tensile%20strength%20is%2037%20Mpa,density%20is%201.3%20g%2Fcm3

[11] *MPU-6050 Invensense - Datasheet PDF, Footprint, Symbol & Technical Specs*. All About Circuits. (n.d.). Retrieved April 12, 2023, from https://www.allaboutcircuits.com/electronic-components/datasheet/MPU-6050--InvenSense/

[12] RonM9, & Instructables. (2022, May 4). *Enhanced NRF24L01 radio with a DIY dipole antenna modification.* Instructables. Retrieved April 12, 2023, from https://www.instructables.com/Enhanced-NRF24L01/

[13] B. Burns, "20 Eco Design Points," presented at Carleton University, Ottawa, ON, Canada, 2023.

Appendices

Appendix A: Overall team contribution:

GROUP MEMBER	CONTRIBUTION
Hussin Abdullah	20%
Ansar Mukash	20%
Jenrolaoluwa Lasisi	20%
Enshuo Li	20%
Debjyoti Biswas (Ron)	20%

Table 2: Team contribution

Appendix B: Robot and Glove code:

Robot code:

```
//Include Libraries

#include <SPI.h>

//#include <nRF24L01.h>

#include <RF24.h>
```

```
//Global definition

//LEFT MOTORS

#define in1 7

#define in2 2

#define enA 6


//RIGHT MOTORS

#define in3 3

#define in4 4

#define enB 5


#define dead_zone_acc 2

#define dead_zone_tilt 3

#define threshold 0

#define tilt_threshold 70



//TRANSCIEVER ADDRESS

const byte address[6] = "00001";


//Acceleration and tilt data from the transmitter

float acc;

float tilt;


//Acceleration and tilt data output to the motor from the transmitter

int motor_acc;

int motor_tilt;
```

```
int max_power = 80;


//HC-SR04 distance pin
  int trig_1 = 11;   //trig of first distance sensors
  int echo_1 = 10;   //echo of first distance sensors
  long duration_1;
  int distance_1;

int const trig_2 = 12;   //trig of second distance sensors
int const echo_2 = 13;   //echo of second distance sensors
long duration_2;
int distance_2;

//create an RF24 object
RF24 radio(9, 8);  // CE, CSN

void setup() {
  // put your setup code here, to run once:

  Serial.begin(9600);

  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);

  digitalWrite(in1, LOW);
```

```
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
  digitalWrite(enA, LOW);
  digitalWrite(enB, LOW);

  radio.begin();

  //set the address
  radio.openReadingPipe(0, address);

  //Set module as receiver
  radio.startListening();

  motor_acc = 0;
  motor_tilt = 0;

  //set the distance sensor pin
  pinMode(trig_1, OUTPUT);
  pinMode(echo_1, INPUT);
    pinMode(trig_2, OUTPUT);
  pinMode(echo_2, INPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  int text[2] = {};
```

```
 if (radio.available())
 {
  //Recieves the data from the transciever
  radio.read(&text, sizeof(text));
  acc = text[0];
  tilt = text[1];
 }


 //Acceleration and deceleration
 if (acc > dead_zone_acc) forward(acc);
 else if (acc < -dead_zone_acc) backward(acc);
 //else halt();

 else if(tilt > dead_zone_tilt) left(tilt);
 else if(tilt < -dead_zone_tilt) right(tilt);
 else halt();

 //Serial.println("motor_acc");
 //Serial.println(distance_1);
 //Serial.println("tilt");
 //Serial.println(tilt);
}

//Implements forward motion with built in turning
void forward(int acc)
{
 //set direction
```

```
digitalWrite(in1, HIGH);
digitalWrite(in2, LOW);
digitalWrite(in3, HIGH);
digitalWrite(in4, LOW);

//gives analog input
motor_acc = map(acc, 0, acc, threshold, max_power);

 //read forward distance sensor
  digitalWrite(trig_1, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trig_1, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig_1, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration_1 = pulseIn(echo_1, HIGH);
  // Calculating the distance
  distance_1 = duration_1 * 0.034 / 2;
  Serial.println(distance_1);

  //else if(distance_1 > 25)distance_1 = 26;

////Checks if the user is turnning left or right
if(distance_1 >= 25)
{
  if (tilt < dead_zone_tilt)
  {
    motor_tilt = map(tilt, 0, tilt, threshold, tilt_threshold);
```

```
      analogWrite(enB, motor_acc);
      analogWrite(enA, motor_acc - motor_tilt);
    }
    else if (tilt > dead_zone_tilt)
    {
      motor_tilt = map(tilt, 0, tilt, threshold, tilt_threshold);
      analogWrite(enB, motor_acc - motor_tilt);
      analogWrite(enA, motor_acc);
    }
    else
    {
      analogWrite(enA, motor_acc);
      analogWrite(enB, motor_acc);
    }
  }

  else
    {
      //halt();
      digitalWrite(in1, LOW);
      digitalWrite(in2, LOW);
      digitalWrite(in3, LOW);
      digitalWrite(in4, LOW);
      motor_acc = 0;
      delay(1500);
    }
}
```

```
//Implements bawards motion with built in turning
void backward(int acc)
{
  //set direction
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);

  //gives analog input
  motor_acc = map(acc, 0, acc, threshold, max_power);

  //read forward distance sensor
    digitalWrite(trig_2, LOW);
    delayMicroseconds(2);
    // Sets the trigPin on HIGH state for 10 micro seconds
    digitalWrite(trig_2, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig_2, LOW);
    // Reads the echoPin, returns the sound wave travel time in microseconds
    duration_2 = pulseIn(echo_2, HIGH);
    // Calculating the distance
    distance_2 = duration_2 * 0.034 / 2;


    if(distance_2 >= 25)
  {
    //Checks if the user is turnning left or right
    if (tilt < dead_zone_tilt)
```

```
  {
    motor_tilt = map(tilt, 0, tilt, threshold, tilt_threshold);
    analogWrite(enB, motor_acc);
    analogWrite(enA, motor_acc - motor_tilt);
  }
  else if (tilt > dead_zone_tilt)
  {
    motor_tilt = map(tilt, 0, tilt, threshold, tilt_threshold);
    analogWrite(enB, motor_acc - motor_tilt);
    analogWrite(enA, motor_acc);
  }
  else
  {
    analogWrite(enA, motor_acc);
    analogWrite(enB, motor_acc);
  }
 }


 else
  {
    //halt();
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);
    motor_acc = 0;
    delay(1500);
  }
}
```

```
void left(int tilt)
{
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);

  motor_tilt = map(tilt, 0, tilt, threshold, tilt_threshold);
  analogWrite(enA, motor_tilt);
  analogWrite(enB, 0);
}

void right(int tilt)
{
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);

  motor_tilt = map(tilt, 0, tilt, threshold, tilt_threshold);
  analogWrite(enB, motor_tilt);
  analogWrite(enA, 0);
}

void halt()
{
```

```
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
  motor_acc = 0;
}
```

Glove code:

```
 //Include Libraries
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Wire.h>
#include <MPU6050.h>

//create an RF24  and MPU6050object
RF24 radio(9, 8);  // CE, CSN
MPU6050 mpu;

// Timers
unsigned long timer = 0;
float timeStep = 0.01;

// Pitch, Roll and Yaw values
float pitch = 0;
float roll = 0;
```

```
//address through which two modules communicate.
const byte address[6] = "00001";

int text[2] = {};   //constant to store gyrodata

void setup()
{

  Serial.begin(9600);

  // Initialize MPU6050
  while(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G))
  {
    Serial.println("Could not find a valid MPU6050 sensor, check wiring!");
    delay(500);
  }

  //Calibrate the gyroscope
  mpu.calibrateGyro();

  // Set threshold sensivty. Default 3.
  mpu.setThreshold(3);

  pitch = 0;
  roll = 0;

  //nrf24 setup
```

```
  radio.begin();

  //set the address
  radio.openWritingPipe(address);

  //Set module as transmitter
  radio.stopListening();
}

void loop()
{
  timer = millis();

  // Read normalized values
  Vector norm = mpu.readNormalizeGyro();

  // Calculate Pitch, Roll and Yaw
  pitch = pitch + norm.YAxis * timeStep;
  roll = roll + norm.XAxis * timeStep;

  // Wait to full timeStep period
  delay((timeStep*1000) - (millis() - timer));

  text[0] = roll;
  text[1] = pitch;

  radio.write(&text, sizeof(text));
```

```
  //Serial.println("roll");
  Serial.println(pitch);
  //Serial.println("pitch");
  //Serial.println(pitch);

  delayMicroseconds(1);
}
```