G-BOT-Gesture Controlled Robot

Oindrilla Ghosh Dastidar
Department of Electrical and Computer Engineering
The Ohio State University
Columbus, United States
ghoshdastidar.4@osu.edu

Parth Ketankumar Modi
Department of Electrical and Computer Engineering
The Ohio State University
Columbus, United States
modi.99@osu.edu

***Abstract* — In this project, a wheeled robot was made that can be controlled by an android device through our hand gestures, and which can stop automatically when an obstacle is on its way. Wi-Fi has been used as the communication protocol between the robot and the android device. The main aim was to make an embedded system which has all components from sensors, microcontrollers, actuators, and a communication protocol as well as uses the android device as it is currently the most utilized device and is accessible to all users.**

I. INTRODUCTION

The wheeled robot designed in this project can be controlled by users using their android devices. The sensors which are embedded within an android device, for example a gyroscope can be used to capture the gestures made by the user. This property of the android device was utilized to make this robot easily controllable by an android phone. Our bigger idea for this project is the application areas this robot can be used in. It can be used in the exploration of mines, exploration of forests, and archaeological sites where humans cannot tread. Thus, this robot would enable the users to operate a robot with their android devices and maneuver them just like a car steering.

Previous work shows that these kinds of projects have been done before in which android is used to control a wheeled robot. The novelty of this project is nevertheless the fact that the Wi-Fi communication between the android device and the robot has not been explored before. Given, the applications the robot is to be used for Wi-Fi has been chosen to give it a large range so that the user can operate it from a large distance. Thus, in this paper, the use of android sensors, Wi-Fi communication, robot motion, obstacle detection, gesture control, and graphic user interface on an android device has been brought together on the same page. The approach used to control the robot has been human gestures as well as obstacle detection. The objective of the project is that the user can maneuver the robot through gestures from very large distances even without seeing the robot and can control it through the GUI(Graphical User Interface) on the android device.

In this report, the work done has been divided into ten sections. In section I. a brief introduction is given. The system block diagram has been given in section II. In section III, the overall system architecture has been defined. The technical approach used has been explained in section IV. The details of the methodology have been described in section V. The details of the user application developed has been stated in section VI. In section VII, the experiments have been stated. In section IX the details of the bugs that can appear have been described.

II. BLOCK DIAGRAM

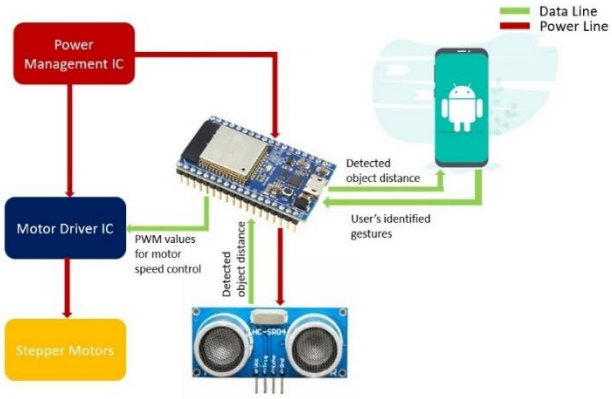

Figure 1: Block Diagram of the system

The block diagram of the system can be seen in Figure 1.

III. SYSTEM ARCHITECTURE

The different hardware components used in the system can be seen in Figure 2. The overall system consists of two main components namely the android device, and the robot. The communications between the android device and the robot take place using Wi-Fi. The robot is made up of a wheeled chassis and the power and logic circuitry which contains an

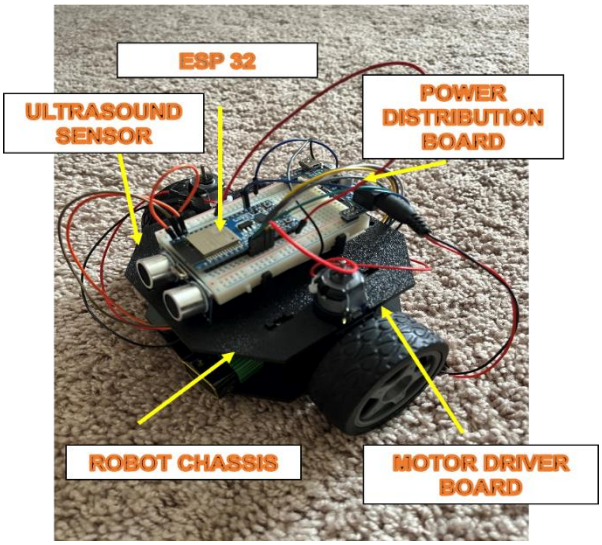

Figure 2: Hardware components on the system

ESP 32, power management IC, motor driver IC, stepper motors and an ultrasound sensor as shown in figure. ESP 32 module is used as the microcontroller for the robot and is programmed using Arduino IDE. ESP32 functions as a Wi-Fi server and creates an access point to let clients connect to it in-order to receive and send data. An android application is created using Android Studio which provides the user with an interface to control the robot. The primary function of this application is to capture and classify the users intended gesture using the gyroscope present on the device and send this classified gesture to the ESP32 using Wi-Fi. The android device running the above-mentioned application functions as a Wi-Fi client that connects to a Wi-Fi server in-order to send and receive data. On top of Wi-Fi the system incorporates the WebSockets protocol to transfer data. The OkHTTP library for android and Java applications is used in android studio to achieve data transfer using WebSockets on an android device. Whereas on the ESP32 this is achieved using the WebSocketServer library.

IV. TECHNICAL APPROACH

The android application created using android studio has four primary functions. Initially, the application requires the user calibrate and threshold gyroscope values. This enables the application to run on a wide range of android devices that contain gyroscope sensors with varying sensitivity. Secondly, the application captures different gestures depending on the change in gyroscope values due to the motion endured by the device. The application then classifies the identified gestures into decimal values indicating different types of motion that the robot can perform. Lastly, the application sends these decimal values to the ESP32 over Wi-Fi using WebSockets. The ESP32 comes with an integrated Wi-Fi module which makes the data transfer using Wi-Fi on the ESP32 relatively easy. The ESP32 is programmed using Arduino IDE and is used to control the robot motion depending on the data it receives from the android application. The robot chassis also houses an ultrasound sensor which is used to constantly monitor the area in front of the robot and detect objects that the robot can collide into. The ultrasound sensor sends the distance at which it detects the nearest object to the ESP32. This distance value is transferred back to the android device as feedback for the user to better operate the robot. As soon as the distance value received by the ESP32 from the ultrasound sensor is lower than the set threshold, the ESP32 stops the robot. At this point no forward motion of the robot is allowed as this indicates the presence of an object in front of the robot at a dangerously short distance. The robot can only move in the reverse direction until it is at a safe distance from the detected object after which normal operation is initiated again.

V. METHODOLOGY

To implement this project following hardware and software components were used:

A. Hardware Components

- ESP32 module
- AMS1117 (Power Management IC)
- L298N (Motor Driver IC)
- Stepper Motors
- HC-SR04 (Ultrasonic distance Sensor)

B. Software Components

- OkHTTP Library for Android and Java
- WebSocketServer Library for ESP8266 module
- WiFi library for ESP32 module

1. ESP32

ESP32 is a 32-bit microcontroller developed by Espressif Systems. The ESP32 houses a 2.4 GHz Wi-Fi module and a Bluetooth/Bluetooth LE module. ESP32 embeds two Xtensa 32-bit Lx6 microprocessors that have adjustable clock frequencies ranging from 80MHz to 240MHz. The ESP32 have a good physical range with a +19.5dBm output power. ESP32 supports legacy Bluetooth connections in addition to supporting Bluetooth low energy profiles including L2CAP, GAP, GATT, and SMP. ESP32 has a sleep current of 5uA, this makes it suitable for low-power applications. ESP32 also includes peripherals like Ethernet, High speed SPI, UART, I2S and I2C [1]. ESP 32 comes with two 12-bit SAR ADCs that can support 18 analog enabled pins in the ESP32 [2]. In this project an ESP32 is used as the microcontroller that is responsible to control the motion of the robot, detect objects in front of the vehicle and avoid collisions depending on the presence of objects. The resolution of the ADC is set to 10-bit for this project which provides it with 1024 divisions to control the magnitude of velocity of the robot with optimal precision. Figure 3 shows the ESP32 WROOM module.

Figure 3: ESP 32 WROOM [3]

2. AMS1117 (Power Management IC)

Figure 4 shows the AMS1117 power module with 3.3V and 5V outputs. The AMS1117 is used to provide up to 1A output current and can operate down to a differential of 1V

input-to-output.AMS1117 modules are easy to use and are protected against short circuit and thermal overloads. If the temperature of the junction becomes more than 165° C, then the thermal circuitry shuts down the regulator. The AMS1117 is pin compatible with the older adjustable

Figure 4: AMS1117 Power Module

regulators [10].

3. *L298N (Motor Driver IC)*

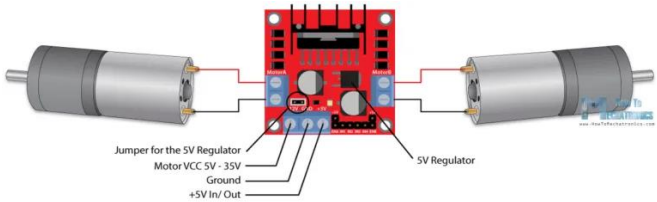

Figure 5: L298N motor driver module [4]

The drawing of an L298N can be seen in Figure 5. The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can be used to drive DC motors with voltages from 5V up to 35V, the peak current should be less than 2A [4].

4. *DC Motors*

DC motor used in our application is operated in the continuous mode. The speed of the motor can be changed by varying the voltage supplied. There are several ways to vary the voltage to the motor. The method used in our project is the PWM (Pulse Width Modulation) technique. The analog pin from the ESP 32 controller sends PWM output of 8 bit which can range from minimum value of 0 and maximum value of 255. Thus changing the output from 0 to 255 the speed of the motor can be controlled.

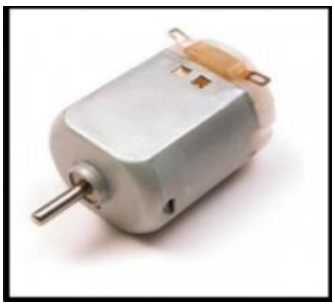

Figure 6: DC Motor [9]

These motors are rated at 3-6V DC and 200-400mA current during normal operation. But during a stall, in the case of heavy weight on the chassis or due to increased friction it can draw a stall current upto 1.5A. [9]. Figure 6 shows a common DC motor that can be used for the robot operation as in this project.

5. *HC-SR04 (Ultrasonic distance Sensor)*

Figure 7: HC-SR04 Ultrasonic Distance Sensor [5]

Figure 7 shows the image and pinout of the HC-SR04 ultrasonic distance sensor. The HC-SR04 ultrasonic distance sensor is used to provide 2cm to 400cm of non-contact measurement functionality. The ranging accuracy of this sensor can be as good as 3mm. The sensor module houses an ultrasound transmitter, receiver, and control unit. The sensor module functions by transmitting eight 40 kHz signals. Then the module waits for these signals to be received by the receiver. If the receiver receives these signals,
then the time taken for the signals to return is sent as the output to the microcontroller [5]. From this time value the distance of the object can be extracted using the following formula:

$$Distance = (Return\ time * velocity\ of\ sound)/2 \quad (1)$$

This distance value is obtained and monitored by the ESP32 and change the motion of the robot accordingly.

6. *OkHTTP Library for Android and Java*

OkHTTP is a third-party library developed by Square. It is used to send and receive HTTP-based network requests. It is built on top of the Okio library. It is also used as the underlying library for Retrofit. OkHTTP v3.5 includes support for bidirectional web sockets. To use web sockets with OkHTTP the URL should be prefixed with ws:// or wss:// [6].

7. *WebSocketServer Library for ESP8266 module*

The WebSocket library for ESP8266 is a third-party library developed by Morris Singer. The library lets the ESP8266 to become a web socket server and provides it with functions like getData() and sendData() which are used to send and receive string data. The library was developed for ESP8266 and therefore some minor changes are to be made to

this library to make it work for the ESP32. These changes are discussed later [7].

8. *WiFi library for ESP32 module*

The Wi-Fi library enables local and internet network connection. This library helps in instantiating servers, clients and send/receive UDP packets through Wi-Fi. This library also allows using the ESP32 as an access point. This allows the ESP32 to create a wireless local area network that is secure from the outside world. This connection facilitates data transfer without the need of being connected to the internet [8].

VI. IMPLEMENTATION

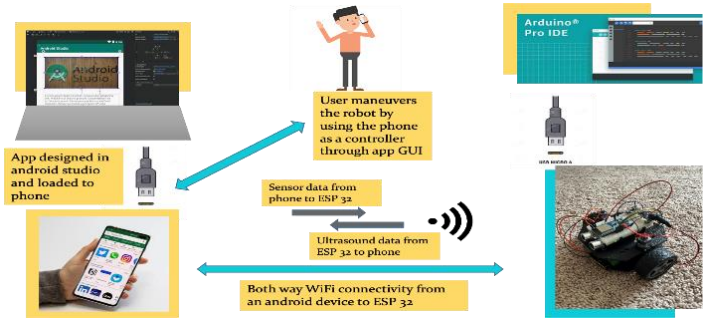

Figure 8: Depiction of the implementation of the system

A brief idea about the implementation of the system is shown in Figure 8.

A. *Implementation on the robot side*

1. *Creating an access point using ESP32*

As discussed earlier, the WiFi library for Arduino is used to create an access point and to enable a WiFi Server using the ESP32. To create an access point the WiFi.softAP() function is used in Arduino IDE. This function requires two parameters namely ssid and password, which can be randomly set by the user as per their convenience. In addition to this we require the IP address of the ESP32 which will act as a URL to which the android device will connect to. The IP address can be found by using the WiFi.softAPIP() function. Finally, the server.begin() function is used to enable the server. The logic to create an access point using ESP32 can be seen in Pseudo Code 2.

```
WiFiServer server(80)
ssid = "xxxxxx"
password = xxxxxxxx
WiFI.sofAP(ssid, password)
print(WiFi.softAPIP())
server.begin()
```

Pseudo Code 2: Code to create an access point using ESP32

2. *Creating a WebSocketServer*

The Web socket server is created using the ESP8266-WebSocket-Master library created by Morris singer. This library was developed for ESP8266 therefore to use this library for ESP32 some changes to the library are required to be made. The MD5.c file in the library source code contains three functions namely MD5Init(), MD5Update(), and MD5Final(). To use this library for ESP32 the names of these functions should be changed to MD5InitXXX(), MD5UpdateXXX(), and MD5FinalXXX() respectively. After this the library should work fine for the ESP32. As the server has been created using the WiFi library, the WebSocket library is used to connect the server with WebSocket clients. To do

```
WebSocketServer websocketserver
WifiClient client = server.available()
if client is connected and handshake is completed{
      getData()
      sendData()
}
```

Pseudo Code 3: Code to create a WebSocket Server using the WebSocket Library

this we first create an instance of the WebSocketServer. The server constantly checks if any client is available. If a client is available, then it is a client instance is created and the available client is assigned to this instance. After this the WebSocket library connects to this client and completes the handshake after which data transfer can take place using getData() and sendData(). The web sockets in this project are configured to transfer only string data therefore all the data to be transferred is first converted to string datatype and then as per requirement converted back to integer datatype. Pseudo Code 3 shows the logic to create a WebSocket server.

3. *Motor Control using data received from android device and object detected by the ultrasound sensor*

The data received from the android device using the web socket is in the form of decimal number in the range of 0 to 22

```
d = distance of object
if d is greater than 40{
      if motion = x (where 0<x<22)
            {
                  PWMDutyCycle1 = y (Depending on motion)
                  PWMDutyCycle2 = z (Depending on motion)
                  MOTOR1_TERMINAL1 = HIGH
                  MOTOR1_TERMINAL2 = LOW
                  MOTOR2_TERMINAL1 = HIGH
                  MOTOR2_TERMINAL2 = LOW
            }
      }
```

Pseudo Code 1: Code to control motor depending on the integer motion values

depending on the gesture from the user. The motion of the robot depends on the number received. The above-mentioned range of numbers is categorized to a specific motion performed by the robot.

The motion of the robot also depends on the object detection by the ultrasound sensor. The threshold for the object distance set by us is 40 cm. This threshold was decided after experimenting different values to know the distance required the robot to come to a complete halt. If there is an object at less than 40 cm in front of the robot, then motions involving reverse motion are only allowed. The speed of the robot is controlled by setting PWM duty cycle values of the motor driver to '300' for slow motion and '350' for fast motion.

Two PWM channels of the ESP32 are used simultaneously to independently vary the values for the two motors, this enables us to program different kinds of motions for the robot. Two IO pins are used to control each motor of the robot. Depending on the type of motion one of the IO pins is set to "HIGH" which passes the PWM duty cycle value set to one of the terminals of the motor. The other pin is set to "LOW" which passes a '0' duty cycle value essentially grounding the second terminal. This creates a voltage difference between the two terminals due to which the motor starts rotating.

To stop the rotating motor or keep it from rotating both the terminals of the motor are set to either "HIGH" or "LOW", because of this there is no voltage difference between the terminals and the motor does not rotate. The motor control logic can be understood by Pseudo Code 1.

B. Implementation on Android Side

1. Setting up the android environment

To use the Wi-Fi module of the android device internet permission must be provided. To add the internet permission to the android application the following line should be added to the android manifest file:

```
<uses-permission
android:name="android.permission.INTERNET" />
```

Next, to use the OkHTTP library in the android application the following line must be added to the dependencies section of the gradle build file of the android application:

```
implementation
'com.squareup.okhttp3:okhttp:4.9.3'
```

Finally, in-order to connect to the web socket server created by the ESP32, the android device must be connected to the access point created by the ESP32. This can be seen in Figure 9.

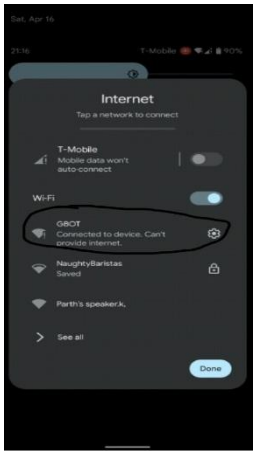

Figure 9: Connecting to G-BOT access point on android device

2. Calibrating the thresholds depending on the android device

The threshold values of the gyroscope before calibration can be seen in Figure 10. The user needs to calibrate the

Figure 10: Threshold values before calibration

gyroscope thresholds. This feature of the application is used to enable this application to be used on all the different android devices that use gyroscopes of different sensitivities. The gyroscope in an android device provides the angular velocity of the motion that it detects. This value of angular velocity has a unit of rad/s. Looking at the unit to convert the value of angular velocity to amount of rotation experienced by the device the value of angular velocity is multiplied by the amount of time taken for that angular velocity to take place.

$$Rotation\ experienced\ =\ Angular\ Velocity\ *\ time \ldots (2)$$

Moreover, the gyroscope does not provide a single value of angular velocity for the entire gesture motion, but the entire gesture motion is divided into many small motion detections. The problem caused due to this is that the rotation value provided by using this angular velocity in equation 2 provides a value of rotation for all the small motions detected by the gyroscope, this value of rotation is not what we are looking for. Therefore, to solve this problem and get the rotation value for the entire gesture motion the rotation values of all the small

```
last_update = current time

While Motion Detected{
      actual = current time
      time_elapsed = actual - last_update
      motion = gyroscope.value[1]
      rotation = motion*time_elapsed
      sum += rotation
      last_update = current time
}
```

Pseudo Code 4: Finding rotation values from angular velocity provided by the gyroscope

motions are cumulated into a sum variable. This variable provides the rotation value for the entire gesture motion. Hence, now in the sum variable we have the rotation value for the complete gesture by the user along one axis of rotation. This procedure is then repeated for the second axis of motion in consideration.

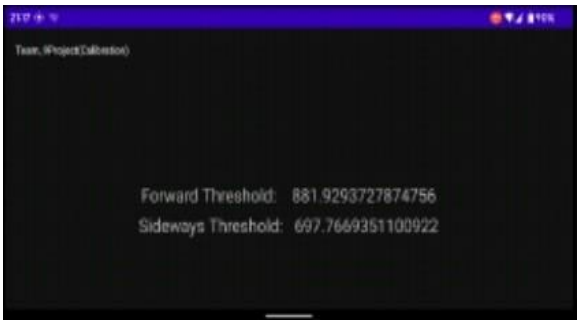

Figure 11: Threshold values after calibration

Therefore, we have two values of rotation for the user's gestures along both the axes of rotation in consideration. The user needs to perform a couple of dummy gestures to calibrate the gyroscope thresholds. These dummy gestures are a 90° forward motion and a 90° left motion. As the thresholds for the backward motion are equal in magnitude to the forward motion threshold, the backward motion threshold can be calculated by negating the forward threshold value. The relation between the left and right threshold values is same as the forward and backward motions therefore the same logic can be achieved to get the right motion threshold values from the left threshold values. Figure 11 shows the threshold values of the gyroscope after calibration is completed. Pseudo Code 4 shows the logic to get rotation values from angular velocity values acquired from the gyroscope.

3. *Identifying the gestures and displaying object distance received from ESP32*

The logic to identify the gestures in the android application is like calibrating the gyroscope thresholds,

the only difference being that this time the rotation values along both the axis is checked against the threshold set by the calibration and the gesture corresponding to that threshold value is identified as the intended gesture by the user. This project recognizes 8 different gestures depicted by different integer values. Depending on the detected gesture the corresponding integer value is assigned to that gesture

Figure 13: Screen showing object distance and motion performed by the robot

The object distance data from the ESP32 is received in the form of string. This string is then directly assigned to the textbox designated to display the distance data as textbox in android accept only string data. Figure 13 shows the screen seen by the user in the application when they are operating the robot.

4. *Categorizing the identified gesture in different motions*

With the help of the 8 recognized gestures in this project 23 different types of motions that the robot can perform using these 8 gestures are derived. These motions are assigned a pre-defined integer value in the range of 0-22. More information on all the different motions and the integer assigned to that motion is provided in Table 1.

C. *Robot electrical connections*

The electrical connections for the robot are shown in Figure 12. In the figure U105 is ESP32 module, U106 is the L298N motor drive, U101 and U102 designate the two DC motors, U107 is the AMS1117 power management module and the U106 is the HC-SR04 ultrasonic distance sensor module. The AMS1117 power management module can produce both 5V and 3.3V. The 3.3V output is used to power the ESP32 and the HC-SR04, while the 5V is used to power the L298N motor driver and the DC motors.

The echo pin on the HC-SR04 is connected to the IO22 pin of the ESP32, while its trigger pin is connected to IO23 of the ESP32.

The L298N is has 6 input pins from the ESP32. Two enable pins for each motor and 4 analog pins to provide the PWM duty cycle values to the two motors. The analog pins used for the first motor are IO27 and IO14. The enable pin for motor 1 is connected to IO25 on the ESP32. For the second motor the analog pins used on the ESP32 are IO12 and IO15.

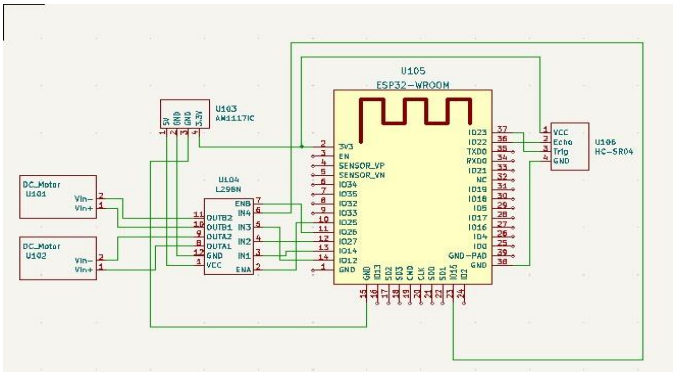

Figure 12: Schematic Diagram of the circuit

Table 1: Recognized motion for the robot and the integer number assigned to each motion

Integer Value Assigned	Rotation Axis of Gyroscope	Corresponding Robot Motion
0	Y = 0 , Z = 0	Stop
1	Y = clockwise threshold 1, Z = 0	Forward
2	Y = clockwise threshold 2, Z = 0	Forward Fast
3	Y = anti-clockwise threshold 2, Z = 0	Backward
4	Y = anti-clockwise threshold 2, Z = 0	Backward fast
5	Y = 0 , Z = anti-clockwise threshold 1	Left
6	Y = 0, Z = anti-clockwise threshold 2	Big Left
7	Y = 0 , Z = clockwise threshold 1	Right
8	Y = 0, Z = clockwise threshold 2	Big Right
9	Y = clockwise threshold 1, Z = anti-clockwise threshold 1	Forward motion while making a left turn
10	Y = clockwise threshold 2, Z = anti-clockwise threshold 2	Fast Forward motion while making a big left turn
11	Y = anti-clockwise threshold 1, Z = anti-clockwise threshold 1	Backward motion while making a left turn
12	Y = anti-clockwise threshold 2, Z = anti-clockwise threshold 2	Fast Backward motion while making a big left turn
13	Y = anti-clockwise threshold 1, Z = clockwise threshold 1	Backward motion while making a right turn
14	Y = anti-clockwise threshold 2, Z = clockwise threshold 2	Fast Backward motion while making a big right turn
15	Y = clockwise threshold 1, Z = clockwise threshold 1	Forward motion while making a right turn
16	Y = clockwise threshold 1, Z = clockwise threshold 1	Fast Forward motion while making a big right turn
17	Y = clockwise threshold 1, Z = anti-clockwise threshold 2	Forward motion while making a big left turn
18	Y = anti-clockwise threshold 1, Z = anti-clockwise threshold 2	Backward motion while making a big left turn
19	Y = anti-clockwise threshold 2, Z = anti-clockwise threshold 1	Fast Backward motion while making a left turn
20	Y = clockwise threshold 2, Z = anti-clockwise threshold 1	Fast Forward motion while making a left turn
21	Y = clockwise threshold 1, Z = clockwise threshold 2	Forward motion while making a big right turn
22	Y = anti-clockwise threshold 1, Z = clockwise threshold 2	Backward motion while making a big right turn

The enable pin for the second motor is connected to the IO26 on the ESP32.

Finally, the OUTA1 pin of the L298N should be connected to the Vin+ pin of first motor and the OUTA2 pin should be connected to the Vin- pin of the first motor. The same convention should be followed for the second motor. The first motor and the second motor can be selected by the user as per their convenience. But as it is difficult to identify OUTA1 and OUTA2 pins on the L298N, the motors should be tested after the connection is made and if the motors rotate opposite to what is expected then the connections should be interchanged.

VII. USING THE APPLICATION

A. Steps to use the Android Application

1. Install the application on an android device using the source files provided
2. Power the robot through a power source that can provide at least 1A current
3. Wait till the robot access point is available to connect on the android device
4. Connect to the robot access point (You will be required to enter the password if you are connecting for the first time. The default SSID for the robot is G-BOT and the default password is 123456789. This can be changed in the Arduino code)
5. Open the android application by the name Team_9_Project_Calibration
6. Click the calibration button on the start screen
7. Now you will see the calibration activity on the screen
8. First, rotate the phone 90° forward as if doing a forward gesture and hold for a second until a value appears on in front of the forward threshold and rotate back to the starting position
9. If step 8 was successful, then rotate the device 90° left, again hold for a second until a value appears in front of the sideways threshold
10. If step 9 is successful, then the thresholds are now calibrated, and you can go back to the start screen
11. Now, click on the start button and start operating the robot
12. Once in the start activity you can see the object distance, straight and forward
13. On rotating the device by 45° forward you can see the forward change from 0 to 1 and the robot moving forward
14. If step 13 worked, then come back to the starting position and rotate left by 45° and you will see straight turn to left and 1 appear in front of it and the robot moves left
15. You can follow the same procedure for reverse motion and right motion by rotating the device backward and right respectively
16. Once all the primary motions are performed successfully, you can refer to Table 1 for more motions that you can try out

VIII. EXPERIMENTS

Following experiment had to be implemented to find the object detection threshold:

- *Experiment 1: To find out the minimum threshold for object detection*

Setup: The assembled chassis is connected to power; the android application is installed on an android device. The android device is connected to the G-BOT access point using Wi-Fi. The G-BOT is operated in fast forward operation mode and moved towards the object setup by us for this experiment. A random threshold value is initially set. The experiment is repeated by increasing or decreasing the threshold value depending on the success or failure of the previous threshold value.

Result: When the value of threshold is set to 40 cm. The robot comes to a halt with still a safe distance away from the object even when operated in Fast Forward motion.

Analysis: By repeating the above experiment for a few times, it was noticed from the threshold value and the distance remaining after the vehicle completely stopped that while being operated in the Fast Forward operation mode the robot took around 20 cm distance to come to a complete stop. This shows that the stopping distance of the robot is 20 cm. Therefore the threshold was set to 40 cm which is double the stopping distance which makes it safe to operate as well as the threshold is not very large to affect the performance of the robot.

IX. POTENTIAL BUGS FOUND

While operating the robot a couple of bugs were found:

1. Inaccurate object distance values: While operating the robot it was found that the object distance provided by the HC-SR04 varied very rapidly and by a huge margin. This sometimes happened if the object was very close to robot. This bug can be solved by replacing the ultrasound sensor by a more accurate sensor like an ultraviolet distance sensor or a LIDAR.
2. Very fast gesture by the user: If the user made a very fast gesture the application sometimes loses its calibration as it is not able to register the fast motion. The reason of this is that the fast motion takes place in a very small period of time which results in accurate value of rotation degree. To solve this bug the logic for gesture recognition must be tweaked so that it can exclude the fast gesture motions.

X. CONCLUSIONS

It was greatly intriguing and exciting to take up this project and solve the challenges that came in the way. The applications of this robot can be in varied places like in mines for exploration, in forests as well as in-arena gaming applications. Thus, further improvements can be made to the design to make it more robust as well as user friendly.

In the future, multiple ultrasound sensors can be used to give a better understanding of the environment. A single ultrasound sensor has a very limited range. Thus, multiple

ultrasound sensors can be used to bring into consideration the full range of the local environment of the robot. In the future, a camera can be used to used instead of ultrasound sensors. Ultrasound sensors are not always reliable and can give varying readings based on the position of the obstacle. Thus, a camera image could be sent to the user and the user can decide what action to take by looking at the GUI of the android device. As mentioned, it can be used to explore areas that cannot be tread by humans therefore in such areas even voice control can be integrated if the user is unable to use gesture control for maneuvering the robot. Also, in our demonstration, we had used an adaptor as the batteries were not able to provide enough ampere hours to drive the motors during stall conditions, thus, in the future LiPo batteries can be used to drive this robot as well as make it portable.

XI. REFERENCES

[1] "Analog to Digital Converter - ESP32 - — ESP-IDF Programming Guide v4.2 documentation," docs.espressif.com. https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/api-reference/peripherals/adc.html#:~:text=The%20ESP32%20integrates%20two%2012 (accessed May 03, 2022).

[2] "Modules | Espressif Systems," www.espressif.com. https://www.espressif.com/en/products/modules

[3] "ESP32 IoT WiFi BLE Module with Integrated USB," store.ncd.io. https://store.ncd.io/product/esp32-iot-wifi-ble-module-with-integrated-usb/ (accessed May 03, 2022)

[4] "Arduino DC Motor Control Tutorial - L298N | PWM | H-Bridge - HowToMechatronics," HowToMechatronics, Feb. 08, 2019. https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/

[5] L. Reese, "The working principle, applications and limitations of ultrasonic sensors," Microcontrollertips.com, 2019. https://www.microcontrollertips.com/principle-applications-limitations-ultrasonic-sensors-faq/

[6] "Using OkHttp | CodePath Android Cliffnotes," guides.codepath.com. https://guides.codepath.com/android/Using-OkHttp (accessed May 03, 2022)

[7] morrissinger, "morrissinger/ESP8266-Websocket," GitHub, Apr. 27, 2016. https://github.com/morrissinger/ESP8266-Websocket

[8] "WiFi - Arduino Reference," www.arduino.cc. https://www.arduino.cc/reference/en/libraries/wifi/ (accessed May 03, 2022)

[9] "ESP32 with DC Motor - Control Speed and Direction | Random Nerd Tutorials," May 17, 2018. https://randomnerdtutorials.com/esp32-dc-motor-l298n-motor-driver-control-speed-direction/

[10] "datasheet AMS1117," www.digchip.com. https://www.digchip.com/datasheets/parts/datasheet/015/AMS1117-pdf.php (accessed May 03, 2022).