**EG114728 HD in AI and Robotics**

**MBS4544 Robot Sensing and Vision**

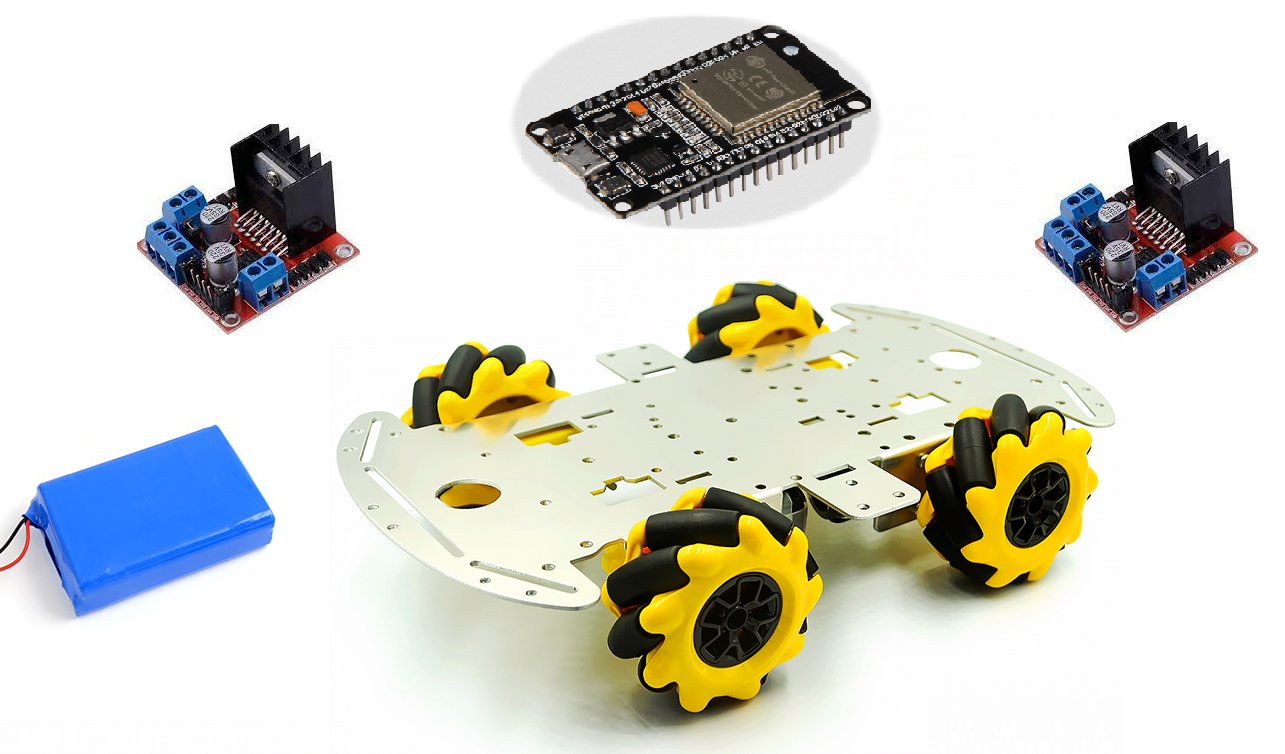
**Assignment 3**

**This assignment is submitted individually.**

**Deadline: 4 Dec 2023**

**Stage 1**

You are required to build a robot car with four mecanum wheels and develop a ESP32 code to drive the car by the PS4 controller.



A video game controller

Description automatically generated with low confidence

PS4 Controller

**Demo Video:**

[**https://vtcmca-my.sharepoint.com/:v:/g/personal/220171174\_stu\_vtc\_edu\_hk/EaBYLGn6mQFCn2PaWTtbZdsBVV-NuRBwEEURH0cs9krmjQ?e=dbbuPP**](https://vtcmca-my.sharepoint.com/:v:/g/personal/220171174_stu_vtc_edu_hk/EaBYLGn6mQFCn2PaWTtbZdsBVV-NuRBwEEURH0cs9krmjQ?e=dbbuPP)

**Code:**

#include <PS4Controller.h>

#include "esp\_bt\_main.h"

#include "esp\_bt\_device.h"

#include "esp\_gap\_bt\_api.h"

#include "esp\_err.h"

#define FORWARD 1

#define BACKWARD 2

#define LEFT 3

#define RIGHT 4

#define FORWARD\_LEFT 5

#define FORWARD\_RIGHT 6

#define BACKWARD\_LEFT 7

#define BACKWARD\_RIGHT 8

#define ROTATE\_LEFT 9

#define ROTATE\_RIGHT 10

#define STOP 0

#define BACK\_RIGHT\_MOTOR 1

#define BACK\_LEFT\_MOTOR 2

#define FRONT\_RIGHT\_MOTOR 3

#define FRONT\_LEFT\_MOTOR 4

#define MAX\_MOTOR\_SPEED 120 // about 50% duty cycle

unsigned long lastTimeStamp = 0;

//FRONT RIGHT MOTOR

int enableFrontRightMotor = 14;

int FrontRightMotorPin1 = 26;

int FrontRightMotorPin2 = 27;

//BACK RIGHT MOTOR

int enableBackRightMotor=22;

int BackRightMotorPin1=16;

int BackRightMotorPin2=17;

//FRONT LEFT MOTOR

int enableFrontLeftMotor = 32;

int FrontLeftMotorPin1 = 33;

int FrontLeftMotorPin2 = 25;

//BACK LEFT MOTOR

int enableBackLeftMotor = 23;

int BackLeftMotorPin1 = 18;

int BackLeftMotorPin2 = 19;

const int PWMFreq = 1000; /\* 1 KHz \*/

const int PWMResolution = 8; // 8-bit

const int PWMSpeedChannel\_1 = 1; // 0-15

const int PWMSpeedChannel\_2 = 2; // 0-15

const int PWMSpeedChannel\_3 = 3; // 0-15

const int PWMSpeedChannel\_4 = 4; // 0-15

void notify()

{

int LStickY = map( PS4.LStickY(), -128, 127, -254, 254);

int LStickX = map( PS4.LStickX(), -128, 127, -254, 254);

int RStickX = map( PS4.RStickX(), -128, 127, -254, 254);

if (LStickY > 30 && LStickX < -30) {moveCar(FORWARD\_LEFT);}

else if (LStickY > 30 && LStickX > 30) {moveCar(FORWARD\_RIGHT);}

else if (LStickY < -30 && LStickX < -30) {moveCar(BACKWARD\_LEFT);}

else if (LStickY < -30 && LStickX > 30) {moveCar(BACKWARD\_RIGHT);}

else if (LStickY > 90) {moveCar(FORWARD);}

else if (LStickY < -90) {moveCar(BACKWARD);}

else if (LStickX < -90) {moveCar(LEFT);}

else if (LStickX > 90) {moveCar(RIGHT);}

else if (RStickX < -50) {moveCar(ROTATE\_LEFT);}

else if (RStickX > 50) {moveCar(ROTATE\_RIGHT);}

else {moveCar(STOP);}

// Print data for debugging purpose only

if (millis() - lastTimeStamp > 100)

{

Serial.print(LStickX);

Serial.print(',');

Serial.print(LStickY);

Serial.print(',');

Serial.print(RStickX);

Serial.println();

lastTimeStamp = millis();

}

}

void onConnect()

{

Serial.println("Connected!");

}

void onDisConnect()

{

rotateMotor(BACK\_RIGHT\_MOTOR, 0);

rotateMotor(BACK\_LEFT\_MOTOR, 0);

rotateMotor(FRONT\_RIGHT\_MOTOR, 0);

rotateMotor(FRONT\_LEFT\_MOTOR, 0);

}

void moveCar(int inputValue)

{

switch(inputValue)

{

case FORWARD:

rotateMotor(FRONT\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);

rotateMotor(BACK\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);

rotateMotor(FRONT\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);

rotateMotor(BACK\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);

break;

case BACKWARD:

rotateMotor(FRONT\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);

rotateMotor(BACK\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);

rotateMotor(FRONT\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);

rotateMotor(BACK\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);

break;

case LEFT:

rotateMotor(FRONT\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);

rotateMotor(BACK\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);

rotateMotor(FRONT\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);

rotateMotor(BACK\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);

break;

case RIGHT:

rotateMotor(FRONT\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);

rotateMotor(BACK\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);

rotateMotor(FRONT\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);

rotateMotor(BACK\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);

break;

case FORWARD\_LEFT:

rotateMotor(FRONT\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);

// rotateMotor(BACK\_RIGHT\_MOTOR, STOP);

// rotateMotor(FRONT\_LEFT\_MOTOR, STOP);

rotateMotor(BACK\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);

break;

case FORWARD\_RIGHT:

// rotateMotor(FRONT\_RIGHT\_MOTOR, STOP);

rotateMotor(BACK\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);

rotateMotor(FRONT\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);

// rotateMotor(BACK\_LEFT\_MOTOR, STOP);

break;

case BACKWARD\_LEFT:

// rotateMotor(FRONT\_RIGHT\_MOTOR, STOP);

rotateMotor(BACK\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);

rotateMotor(FRONT\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);

// rotateMotor(BACK\_LEFT\_MOTOR, STOP);

break;

case BACKWARD\_RIGHT:

rotateMotor(FRONT\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);

// rotateMotor(BACK\_RIGHT\_MOTOR, STOP);

// rotateMotor(FRONT\_LEFT\_MOTOR, STOP);

rotateMotor(BACK\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);

break;

case ROTATE\_LEFT:

rotateMotor(FRONT\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);

rotateMotor(BACK\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);

rotateMotor(FRONT\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);

rotateMotor(BACK\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);

break;

case ROTATE\_RIGHT:

rotateMotor(FRONT\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);

rotateMotor(BACK\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);

rotateMotor(FRONT\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);

rotateMotor(BACK\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);

break;

case STOP:

rotateMotor(FRONT\_RIGHT\_MOTOR, STOP);

rotateMotor(BACK\_RIGHT\_MOTOR, STOP);

rotateMotor(FRONT\_LEFT\_MOTOR, STOP);

rotateMotor(BACK\_LEFT\_MOTOR, STOP);

break;

default:

rotateMotor(FRONT\_RIGHT\_MOTOR, STOP);

rotateMotor(BACK\_RIGHT\_MOTOR, STOP);

rotateMotor(FRONT\_LEFT\_MOTOR, STOP);

rotateMotor(BACK\_LEFT\_MOTOR, STOP);

break;

}

}

void rotateMotor(int motorNumber, int motorSpeed)

{

if (motorSpeed < 0 && motorNumber == 1)

{

digitalWrite(BackRightMotorPin1,LOW);

digitalWrite(BackRightMotorPin2,HIGH);

}

else if (motorSpeed < 0 && motorNumber == 2)

{

digitalWrite(BackLeftMotorPin1,LOW);

digitalWrite(BackLeftMotorPin2,HIGH);

}

else if (motorSpeed < 0 && motorNumber == 3)

{

digitalWrite(FrontRightMotorPin1,LOW);

digitalWrite(FrontRightMotorPin2,HIGH);

}

else if (motorSpeed < 0 && motorNumber == 4)

{

digitalWrite(FrontLeftMotorPin1,LOW);

digitalWrite(FrontLeftMotorPin2,HIGH);

}

else if (motorSpeed > 0 && motorNumber == 1)

{

digitalWrite(BackRightMotorPin1,HIGH);

digitalWrite(BackRightMotorPin2,LOW);

}

else if (motorSpeed > 0 && motorNumber == 2)

{

digitalWrite(BackLeftMotorPin1,HIGH);

digitalWrite(BackLeftMotorPin2,LOW);

}

else if (motorSpeed > 0 && motorNumber == 3)

{

digitalWrite(FrontRightMotorPin1,HIGH);

digitalWrite(FrontRightMotorPin2,LOW);

}

else if (motorSpeed > 0 && motorNumber == 4)

{

digitalWrite(FrontLeftMotorPin1,HIGH);

digitalWrite(FrontLeftMotorPin2,LOW);

}

else if (motorSpeed == 0 && motorNumber == 1)

{

digitalWrite(FrontRightMotorPin1,LOW);

digitalWrite(FrontRightMotorPin2,LOW);

}

else if (motorSpeed == 0 && motorNumber == 2)

{

digitalWrite(FrontLeftMotorPin1,LOW);

digitalWrite(FrontLeftMotorPin2,LOW);

}

else if (motorSpeed == 0 && motorNumber == 3)

{

digitalWrite(BackRightMotorPin1,LOW);

digitalWrite(BackRightMotorPin2,LOW);

}

else if (motorSpeed == 0 && motorNumber == 4)

{

digitalWrite(BackLeftMotorPin1,LOW);

digitalWrite(BackLeftMotorPin2,LOW);

}

ledcWrite(PWMSpeedChannel\_1, abs(motorSpeed));

ledcWrite(PWMSpeedChannel\_2, abs(motorSpeed));

ledcWrite(PWMSpeedChannel\_3, abs(motorSpeed));

ledcWrite(PWMSpeedChannel\_4, abs(motorSpeed));

}

void setUpPinModes()

{

pinMode(enableFrontRightMotor,OUTPUT);

pinMode(FrontRightMotorPin1,OUTPUT);

pinMode(FrontRightMotorPin2,OUTPUT);

pinMode(enableFrontLeftMotor,OUTPUT);

pinMode(FrontLeftMotorPin1,OUTPUT);

pinMode(FrontLeftMotorPin2,OUTPUT);

pinMode(enableBackRightMotor,OUTPUT);

pinMode(BackRightMotorPin1,OUTPUT);

pinMode(BackRightMotorPin2,OUTPUT);

pinMode(enableBackLeftMotor,OUTPUT);

pinMode(BackLeftMotorPin1,OUTPUT);

pinMode(BackLeftMotorPin2,OUTPUT);

//Set up PWM for motor speed

ledcSetup(PWMSpeedChannel\_1, PWMFreq, PWMResolution);

ledcSetup(PWMSpeedChannel\_2, PWMFreq, PWMResolution);

ledcSetup(PWMSpeedChannel\_3, PWMFreq, PWMResolution);

ledcSetup(PWMSpeedChannel\_4, PWMFreq, PWMResolution);

ledcAttachPin(enableBackRightMotor, PWMSpeedChannel\_1);

ledcAttachPin(enableBackLeftMotor, PWMSpeedChannel\_2);

ledcAttachPin(enableFrontRightMotor, PWMSpeedChannel\_3);

ledcAttachPin(enableFrontLeftMotor, PWMSpeedChannel\_4);

rotateMotor(BACK\_RIGHT\_MOTOR, 0);

rotateMotor(BACK\_LEFT\_MOTOR, 0);

rotateMotor(FRONT\_RIGHT\_MOTOR, 0);

rotateMotor(FRONT\_LEFT\_MOTOR, 0);

}

void setup()

{

//

PS4.begin();

uint8\_t pairedDeviceBtAddr[20][6];

int count = esp\_bt\_gap\_get\_bond\_device\_num();

esp\_bt\_gap\_get\_bond\_device\_list(&count, pairedDeviceBtAddr);

for(int i = 0; i < count; i++)

{

esp\_bt\_gap\_remove\_bond\_device(pairedDeviceBtAddr[i]);

}

Serial.println("Previous BT device pairing cleared!!!");

delay(1000);

//

setUpPinModes();

Serial.begin(115200);

PS4.attach(notify);

PS4.attachOnConnect(onConnect);

PS4.attachOnDisconnect(onDisConnect);

PS4.begin("08:d1:f9:e9:58:b2");

Serial.println("Initialization Ready!!!");

}

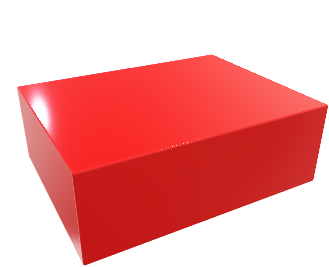
void loop()

{

}

**Stage 2**

Control the mecanum wheel robot car wirelessly by another ESP32 with an IMU connected. By tilting and rotating the IMU on your hand, the robot car should be able to move in the directions that is done in Stage 1.



ESP32+IMU+battery

**Sender Code:**

**#include <esp\_now.h>**

**#include <WiFi.h>**

**#include <Wire.h>**

**const int IMU\_ADDR = 0x68; // I2C address for IMU. ADO=5V -> 0x69**

**int16\_t accel\_X, accel\_Y, accel\_Z, temp\_raw, gyro\_X, gyro\_Y, gyro\_Z; // variables for raw sensor data**

**double aX, aY, aZ, theta\_acc, phi\_acc; //accel values and theta and phi calculated from accelerometer**

**double theta\_acc\_previous = 0, theta\_acc\_LP, phi\_acc\_previous = 0, phi\_acc\_LP;**

**double gX, gY, gZ, theta\_gyro = 0, phi\_gyro = 0, psi\_gyro;**

**double theta\_compli, phi\_compli; //theta and phi calculated from accelerometer and gyroscope**

**unsigned long previousTime, elapseTime;**

**double dt;**

**int n\_forward, n\_backward, n\_left, n\_right;**

**//insert mac address of receiver**

**//uint8\_t broadcastAddress[] = {0xE8, 0x68, 0xE7, 0x22, 0xC3, 0xF0};**

**uint8\_t broadcastAddress[] = { 0x3C, 0x61, 0x05, 0x03, 0x95, 0x58 };**

**// 3C:61:05:03:95:58**

**//structure of the message to send**

**//must be matched to the receiver structure**

**typedef struct struct\_message {**

**int forward;**

**int backward;**

**int left;**

**int right;**

**} struct\_message;**

**//create a new variable to store the value of "struct\_message"**

**struct\_message myData;**

**//create a new variable to store information about the peer**

**esp\_now\_peer\_info\_t peerInfo;**

**long serial\_print\_pmillis = 0;**

**bool can\_print = 0;**

**//define a callback function that will be executed when a message is sent**

**//simpfly print if the message was successfully delivered or not**

**void OnDataSent(const uint8\_t \*mac\_addr, esp\_now\_send\_status\_t status) {**

**//if (can\_print) {**

**Serial.println(WiFi.macAddress());**

**Serial.print("\r\nLast Packet Send Status:\t");**

**Serial.println(status == ESP\_NOW\_SEND\_SUCCESS ? "Delivery Success" : "Delivery Failed");**

**//}**

**}**

**void setup() {**

**// put your setup code here, to run once:**

**Serial.begin(115200);**

**//set the device as a WiFi station**

**WiFi.mode(WIFI\_STA);**

**//initialize ESP-NOW**

**if (esp\_now\_init() != ESP\_OK) {**

**Serial.println("Error initializing ESP-NOW");**

**return;**

**}**

**//once ESP-NOW is successfully init**

**//register a callback function that will be called when a message is sent**

**esp\_now\_register\_send\_cb(OnDataSent);**

**//pair with anoother ESP-NOW device to send data**

**//register peer**

**memcpy(peerInfo.peer\_addr, broadcastAddress, 6);**

**peerInfo.channel = 0;**

**peerInfo.encrypt = false;**

**//add peer**

**if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK) {**

**Serial.println("Failed to add peer");**

**return;**

**}**

**//setup for I2C communicate of mpu9250**

**Wire.begin();**

**Wire.beginTransmission(IMU\_ADDR); //communicate with IMU**

**Wire.write(0x6B); //PWR\_MGMT\_1 register**

**Wire.write(0x0);**

**Wire.endTransmission(true);**

**Wire.beginTransmission(IMU\_ADDR); //communicate with IMU**

**Wire.write(0x1C); //ACCEL\_CONFIG register**

**Wire.write(0b00000000); //Bit3 & Bit 4 -> 0,0 -> +/-2g**

**Wire.endTransmission(true);**

**Wire.beginTransmission(IMU\_ADDR); //communicate with IMU**

**Wire.write(0x1B); //GYRO\_CONFIG register**

**Wire.write(0b00000000); //Bit3 & Bit 4 -> 0,0 -> +/-250deg/s**

**Wire.endTransmission(true);**

**Wire.beginTransmission(IMU\_ADDR); //communicate with IMU**

**Wire.write(0x37); // IMU INT PIN CONFIG**

**Wire.write(0x02); // 0x02 activate bypass in order to communicate with magnetometer**

**Wire.endTransmission(true);**

**delay(100);**

**previousTime = millis();**

**}**

**void loop() {**

**can\_print = 0;//millis() - serial\_print\_pmillis > 500;**

**if (millis() - serial\_print\_pmillis > 500) serial\_print\_pmillis = millis();**

**// put your main code here, to run repeatedly:**

**Wire.beginTransmission(IMU\_ADDR);**

**Wire.write(0x3B); // start from ACCEL\_XOUT\_H**

**Wire.endTransmission(false);**

**Wire.requestFrom(IMU\_ADDR, 14, true); //request a total of 14 bytes**

**// read registers from IMU**

**accel\_X = Wire.read() << 8 | Wire.read(); //read registers 0x3B (ACCEL\_XOUT\_H) & 0x3C (ACCEL\_XOUT\_L)**

**accel\_Y = Wire.read() << 8 | Wire.read(); //read registers 0x3D & 0x3E**

**accel\_Z = Wire.read() << 8 | Wire.read(); //read registers 0x3F & 0x40**

**temp\_raw = Wire.read() << 8 | Wire.read(); //read registers 0x41 & 0x42**

**gyro\_X = Wire.read() << 8 | Wire.read(); //read registers 0x43 (GYRO\_XOUT\_H) & 0x44 (GYRO\_XOUT\_L)**

**gyro\_Y = Wire.read() << 8 | Wire.read(); //read registers 0x45 & 0x46**

**gyro\_Z = Wire.read() << 8 | Wire.read(); //read registers 0x47 & 0x48**

**Wire.endTransmission(true);**

**aX = accel\_X / 16384.0; // convert the raw data to g value. 1g = 9.8 m/s2**

**aY = accel\_Y / 16384.0;**

**aZ = accel\_Z / 16384.0;**

**if (can\_print) {**

**Serial.print("ax:\t");**

**Serial.print(aX);**

**Serial.print(",ay:\t");**

**Serial.print(aY);**

**Serial.print(",az:\t");**

**Serial.println(aZ);**

**}**

**//--------------Getting Roll(phi) and Pitch(theta) from accelerometer readings only-------------------//**

**theta\_acc = atan2(aX, aZ) \* (180 / PI) - 44; // approximation of theta (pitch). use either this or the eauation right below.**

**// theta\_acc = atan2(aX, sqrt(pow(aY,2)+pow(aZ,2)))\*(180/PI); // another approximation on theta (pitch). Either use this or the equation right above.**

**phi\_acc = atan2(aY, aZ) \* (180 / PI) + 44; // approximation of phi (roll). use either this or the eauation right below.**

**// phi\_acc = atan2(aY, sqrt(pow(aX,2)+pow(aZ,2)))\*(180/PI);// another approximation on phi (roll). Either use this or the equation right above.**

**// simple Low Pass Filter (LPF) added for theta(pitch) and phi(roll)**

**theta\_acc\_LP = 0.7 \* theta\_acc\_previous + 0.3 \* theta\_acc; // adjust the ratio of preivous theta value and the newly measured value**

**phi\_acc\_LP = 0.7 \* phi\_acc\_previous + 0.3 \* phi\_acc; //gs only-----**

**theta\_acc\_previous = theta\_acc\_LP;**

**phi\_acc\_previous = phi\_acc\_LP;**

**//------------end of Roll(phi) and Pitch(theta) from accelerometer readings---------//**

**//--------------Getting Roll(phi) and Pitch(theta) from gyroscope readin--------------//**

**gX = gyro\_X / 131.0 - 5.8; // convert the raw data to rad/s. "- 5.8" is the gyro error**

**gY = gyro\_Y / 131.0 - 6;**

**gZ = gyro\_Z / 131.0;**

**elapseTime = millis() - previousTime;**

**dt = elapseTime / 1000.0;**

**previousTime = millis();**

**theta\_gyro = theta\_gyro - gY \* dt; // pitch angle from gyro**

**phi\_gyro = phi\_gyro + gX \* dt; // roll angle from gyro**

**psi\_gyro = psi\_gyro + gZ \* dt; // yaw angle from gyro (not quite useful, only incremental value)**

**//-----------end of Roll(phi) and Pitch(theta) from gyroscope readings --------------------//**

**// ------- Complimentary filter from accelerometer and gyroscope data ---------//**

**// Complimentary filter uses a mix of two sensor data with**

**//a specific ratio which depend on which one you trust better**

**theta\_compli = 0.95 \* (theta\_compli - gY \* dt) + 0.05 \* (theta\_acc);**

**phi\_compli = 0.95 \* (phi\_compli + gX \* dt) + 0.05 \* (phi\_acc);**

**// ------- end of Complimentary filter from accelerometer and gyroscope data -----//**

**//-------conditional limition of reading value (lowerLimit - larger than && upperLimit - smaller than)-------//**

**if (theta\_acc < -8 && theta\_acc > -50 && phi\_acc < -2 && phi\_acc > -40) { n\_forward = 1; } //statement of forward**

**else {**

**n\_forward = 0;**

**}**

**if (theta\_acc > 10 && theta\_acc < 50 && phi\_acc < -2 && phi\_acc > -40) { n\_backward = 1; } //statement of backward**

**else {**

**n\_backward = 0;**

**}**

**if (theta\_acc > 2 && theta\_acc < 60 && phi\_acc < -12 && phi\_acc > -50) { n\_left = 1; } //statement of left**

**else {**

**n\_left = 0;**

**}**

**if (theta\_acc > 3 && theta\_acc < 50 && phi\_acc > 10 && phi\_acc < 120) { n\_right = 1; } //statement of right**

**else {**

**n\_right = 0;**

**}**

**//------------------------------End of limition-------------------------------//**

**//set the variables values to send**

**if (can\_print) Serial.print("255\*aY:\t");**

**if (can\_print) Serial.println(255 \* aY);**

**myData.forward = abs(255 \* aY \* n\_forward);**

**myData.backward = 255 \* aY \* !(n\_forward || n\_left || n\_right);**

**myData.left = constrain(map(aX, 1, -1, 255, -255),-255,255);**

**myData.right = constrain(map(aX, 1, -1, -255, 255),-255,255);**

**/\***

**//set the variables values to send**

**myData.forward = n\_forward;**

**myData.backward = n\_backward;**

**myData.left = n\_left;**

**myData.right = n\_right;**

**\*/**

**if (can\_print) Serial.print("theta\_acc :");**

**if (can\_print) Serial.println(theta\_acc);**

**if (can\_print) Serial.print("phi\_acc :");**

**if (can\_print) Serial.println(phi\_acc);**

**// Serial.print("theta\_acc\_LP :"); Serial.println(theta\_acc\_LP);**

**// Serial.print("phi\_acc\_LP :"); Serial.println(phi\_acc\_LP);**

**// Serial.print("theta\_gyro :"); Serial.println(theta\_gyro);**

**// Serial.print("phi\_gyro :"); Serial.println(phi\_gyro);**

**// Serial.print("psi\_gyro :"); Serial.println(psi\_gyro);**

**// Serial.print("theta\_compli :"); Serial.println(theta\_compli);**

**// Serial.print("phi\_compli :"); Serial.println(phi\_compli);**

**if (can\_print) {**

**Serial.print("forward :");**

**Serial.println(n\_forward);**

**Serial.print("backward :");**

**Serial.println(n\_backward);**

**Serial.print("left :");**

**Serial.println(n\_left);**

**Serial.print("right :");**

**Serial.println(n\_right);**

**}**

**//send the message via ESP-NOW**

**esp\_err\_t result = esp\_now\_send(broadcastAddress, (uint8\_t \*)&myData, sizeof(myData));**

**if (result == ESP\_OK) {**

**/\*if (can\_print)\*/ Serial.println("Sent with success");**

**} else {**

**/\*if (can\_print)\*/ Serial.println("Error sending the data");**

**}**

**//delay(50);**

**}**

**Code Receiver:**

**#include <esp\_now.h>**

**#include <WiFi.h>**

**#define FORWARD 1**

**#define BACKWARD 2**

**#define LEFT 3**

**#define RIGHT 4**

**#define FORWARD\_LEFT 5**

**#define FORWARD\_RIGHT 6**

**#define BACKWARD\_LEFT 7**

**#define BACKWARD\_RIGHT 8**

**#define ROTATE\_LEFT 9**

**#define ROTATE\_RIGHT 10**

**#define STOP 0**

**#define BACK\_RIGHT\_MOTOR 1**

**#define BACK\_LEFT\_MOTOR 2**

**#define FRONT\_RIGHT\_MOTOR 3**

**#define FRONT\_LEFT\_MOTOR 4**

**#define speed 130 // about 50% duty cycle**

**#define MAX\_MOTOR\_SPEED 255**

**unsigned long lastTimeStamp = 0;**

**//FRONT RIGHT MOTOR**

**int enableFrontRightMotor = 14;**

**int FrontRightMotorPin1 = 26;**

**int FrontRightMotorPin2 = 27;**

**//BACK RIGHT MOTOR**

**int enableBackRightMotor=22;**

**int BackRightMotorPin1=16;**

**int BackRightMotorPin2=17;**

**//FRONT LEFT MOTOR**

**int enableFrontLeftMotor = 32;**

**int FrontLeftMotorPin1 = 33;**

**int FrontLeftMotorPin2 = 25;**

**//BACK LEFT MOTOR**

**int enableBackLeftMotor = 23;**

**int BackLeftMotorPin1 = 18;**

**int BackLeftMotorPin2 = 19;**

**const int PWMFreq = 1000; /\* 1 KHz \*/**

**const int PWMResolution = 8; // 8-bit**

**const int PWMSpeedChannel\_1 = 1; // 0-15**

**const int PWMSpeedChannel\_2 = 2; // 0-15**

**const int PWMSpeedChannel\_3 = 3; // 0-15**

**const int PWMSpeedChannel\_4 = 4; // 0-15**

**void moveCar(int inputValue)**

**{**

**switch(inputValue)**

**{**

**case FORWARD:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);**

**rotateMotor(FRONT\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);**

**break;**

**case BACKWARD:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**rotateMotor(FRONT\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**break;**

**case LEFT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**rotateMotor(FRONT\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);**

**break;**

**case RIGHT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);**

**rotateMotor(FRONT\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**break;**

**case FORWARD\_LEFT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_RIGHT\_MOTOR, STOP);**

**rotateMotor(FRONT\_LEFT\_MOTOR, STOP);**

**rotateMotor(BACK\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);**

**break;**

**case FORWARD\_RIGHT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, STOP);**

**rotateMotor(BACK\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);**

**rotateMotor(FRONT\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_LEFT\_MOTOR, STOP);**

**break;**

**case BACKWARD\_LEFT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, STOP);**

**rotateMotor(BACK\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**rotateMotor(FRONT\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_LEFT\_MOTOR, STOP);**

**break;**

**case BACKWARD\_RIGHT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_RIGHT\_MOTOR, STOP);**

**rotateMotor(FRONT\_LEFT\_MOTOR, STOP);**

**rotateMotor(BACK\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**break;**

**case ROTATE\_LEFT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);**

**rotateMotor(FRONT\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**break;**

**case ROTATE\_RIGHT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);**

**rotateMotor(FRONT\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);**

**rotateMotor(BACK\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);**

**break;**

**case STOP:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, STOP);**

**rotateMotor(BACK\_RIGHT\_MOTOR, STOP);**

**rotateMotor(FRONT\_LEFT\_MOTOR, STOP);**

**rotateMotor(BACK\_LEFT\_MOTOR, STOP);**

**break;**

**default:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, STOP);**

**rotateMotor(BACK\_RIGHT\_MOTOR, STOP);**

**rotateMotor(FRONT\_LEFT\_MOTOR, STOP);**

**rotateMotor(BACK\_LEFT\_MOTOR, STOP);**

**break;**

**}**

**}**

**void moveCar(int inputValue,byte Speed)**

**{**

**switch(inputValue)**

**{**

**case FORWARD:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, Speed);**

**rotateMotor(BACK\_RIGHT\_MOTOR, Speed);**

**rotateMotor(FRONT\_LEFT\_MOTOR, Speed);**

**rotateMotor(BACK\_LEFT\_MOTOR, Speed);**

**break;**

**case BACKWARD:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, -Speed);**

**rotateMotor(BACK\_RIGHT\_MOTOR, -Speed);**

**rotateMotor(FRONT\_LEFT\_MOTOR, -Speed);**

**rotateMotor(BACK\_LEFT\_MOTOR, -Speed);**

**break;**

**case LEFT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, Speed);**

**rotateMotor(BACK\_RIGHT\_MOTOR, -Speed);**

**rotateMotor(FRONT\_LEFT\_MOTOR, -Speed);**

**rotateMotor(BACK\_LEFT\_MOTOR, Speed);**

**break;**

**case RIGHT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, -Speed);**

**rotateMotor(BACK\_RIGHT\_MOTOR, Speed);**

**rotateMotor(FRONT\_LEFT\_MOTOR, Speed);**

**rotateMotor(BACK\_LEFT\_MOTOR, -Speed);**

**break;**

**case FORWARD\_LEFT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, Speed);**

**rotateMotor(BACK\_RIGHT\_MOTOR, STOP);**

**rotateMotor(FRONT\_LEFT\_MOTOR, STOP);**

**rotateMotor(BACK\_LEFT\_MOTOR, Speed);**

**break;**

**case FORWARD\_RIGHT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, STOP);**

**rotateMotor(BACK\_RIGHT\_MOTOR, Speed);**

**rotateMotor(FRONT\_LEFT\_MOTOR, Speed);**

**rotateMotor(BACK\_LEFT\_MOTOR, STOP);**

**break;**

**case BACKWARD\_LEFT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, STOP);**

**rotateMotor(BACK\_RIGHT\_MOTOR, -Speed);**

**rotateMotor(FRONT\_LEFT\_MOTOR, -Speed);**

**rotateMotor(BACK\_LEFT\_MOTOR, STOP);**

**break;**

**case BACKWARD\_RIGHT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, -Speed);**

**rotateMotor(BACK\_RIGHT\_MOTOR, STOP);**

**rotateMotor(FRONT\_LEFT\_MOTOR, STOP);**

**rotateMotor(BACK\_LEFT\_MOTOR, -Speed);**

**break;**

**case ROTATE\_LEFT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, Speed);**

**rotateMotor(BACK\_RIGHT\_MOTOR, Speed);**

**rotateMotor(FRONT\_LEFT\_MOTOR, -Speed);**

**rotateMotor(BACK\_LEFT\_MOTOR, -Speed);**

**break;**

**case ROTATE\_RIGHT:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, -Speed);**

**rotateMotor(BACK\_RIGHT\_MOTOR, -Speed);**

**rotateMotor(FRONT\_LEFT\_MOTOR, Speed);**

**rotateMotor(BACK\_LEFT\_MOTOR, Speed);**

**break;**

**case STOP:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, STOP);**

**rotateMotor(BACK\_RIGHT\_MOTOR, STOP);**

**rotateMotor(FRONT\_LEFT\_MOTOR, STOP);**

**rotateMotor(BACK\_LEFT\_MOTOR, STOP);**

**break;**

**default:**

**rotateMotor(FRONT\_RIGHT\_MOTOR, STOP);**

**rotateMotor(BACK\_RIGHT\_MOTOR, STOP);**

**rotateMotor(FRONT\_LEFT\_MOTOR, STOP);**

**rotateMotor(BACK\_LEFT\_MOTOR, STOP);**

**break;**

**}**

**}**

**void rotateMotor(int motorNumber, int motorSpeed)**

**{**

**if (motorSpeed < 0 && motorNumber == 1)**

**{**

**digitalWrite(BackRightMotorPin1,LOW);**

**digitalWrite(BackRightMotorPin2,HIGH);**

**}**

**else if (motorSpeed < 0 && motorNumber == 2)**

**{**

**digitalWrite(BackLeftMotorPin1,LOW);**

**digitalWrite(BackLeftMotorPin2,HIGH);**

**}**

**else if (motorSpeed < 0 && motorNumber == 3)**

**{**

**digitalWrite(FrontRightMotorPin1,LOW);**

**digitalWrite(FrontRightMotorPin2,HIGH);**

**}**

**else if (motorSpeed < 0 && motorNumber == 4)**

**{**

**digitalWrite(FrontLeftMotorPin1,LOW);**

**digitalWrite(FrontLeftMotorPin2,HIGH);**

**}**

**else if (motorSpeed > 0 && motorNumber == 1)**

**{**

**digitalWrite(BackRightMotorPin1,HIGH);**

**digitalWrite(BackRightMotorPin2,LOW);**

**}**

**else if (motorSpeed > 0 && motorNumber == 2)**

**{**

**digitalWrite(BackLeftMotorPin1,HIGH);**

**digitalWrite(BackLeftMotorPin2,LOW);**

**}**

**else if (motorSpeed > 0 && motorNumber == 3)**

**{**

**digitalWrite(FrontRightMotorPin1,HIGH);**

**digitalWrite(FrontRightMotorPin2,LOW);**

**}**

**else if (motorSpeed > 0 && motorNumber == 4)**

**{**

**digitalWrite(FrontLeftMotorPin1,HIGH);**

**digitalWrite(FrontLeftMotorPin2,LOW);**

**}**

**else if (motorSpeed == 0 && motorNumber == 1)**

**{**

**digitalWrite(FrontRightMotorPin1,LOW);**

**digitalWrite(FrontRightMotorPin2,LOW);**

**}**

**else if (motorSpeed == 0 && motorNumber == 2)**

**{**

**digitalWrite(FrontLeftMotorPin1,LOW);**

**digitalWrite(FrontLeftMotorPin2,LOW);**

**}**

**else if (motorSpeed == 0 && motorNumber == 3)**

**{**

**digitalWrite(BackRightMotorPin1,LOW);**

**digitalWrite(BackRightMotorPin2,LOW);**

**}**

**else if (motorSpeed == 0 && motorNumber == 4)**

**{**

**digitalWrite(BackLeftMotorPin1,LOW);**

**digitalWrite(BackLeftMotorPin2,LOW);**

**}**

**ledcWrite(PWMSpeedChannel\_1, abs(motorSpeed));**

**ledcWrite(PWMSpeedChannel\_2, abs(motorSpeed));**

**ledcWrite(PWMSpeedChannel\_3, abs(motorSpeed));**

**ledcWrite(PWMSpeedChannel\_4, abs(motorSpeed));**

**}**

**void setUpPinModes()**

**{**

**pinMode(enableFrontRightMotor,OUTPUT);**

**pinMode(FrontRightMotorPin1,OUTPUT);**

**pinMode(FrontRightMotorPin2,OUTPUT);**

**pinMode(enableFrontLeftMotor,OUTPUT);**

**pinMode(FrontLeftMotorPin1,OUTPUT);**

**pinMode(FrontLeftMotorPin2,OUTPUT);**

**pinMode(enableBackRightMotor,OUTPUT);**

**pinMode(BackRightMotorPin1,OUTPUT);**

**pinMode(BackRightMotorPin2,OUTPUT);**

**pinMode(enableBackLeftMotor,OUTPUT);**

**pinMode(BackLeftMotorPin1,OUTPUT);**

**pinMode(BackLeftMotorPin2,OUTPUT);**

**//Set up PWM for motor speed**

**ledcSetup(PWMSpeedChannel\_1, PWMFreq, PWMResolution);**

**ledcSetup(PWMSpeedChannel\_2, PWMFreq, PWMResolution);**

**ledcSetup(PWMSpeedChannel\_3, PWMFreq, PWMResolution);**

**ledcSetup(PWMSpeedChannel\_4, PWMFreq, PWMResolution);**

**ledcAttachPin(enableBackRightMotor, PWMSpeedChannel\_1);**

**ledcAttachPin(enableBackLeftMotor, PWMSpeedChannel\_2);**

**ledcAttachPin(enableFrontRightMotor, PWMSpeedChannel\_3);**

**ledcAttachPin(enableFrontLeftMotor, PWMSpeedChannel\_4);**

**rotateMotor(BACK\_RIGHT\_MOTOR, 0);**

**rotateMotor(BACK\_LEFT\_MOTOR, 0);**

**rotateMotor(FRONT\_RIGHT\_MOTOR, 0);**

**rotateMotor(FRONT\_LEFT\_MOTOR, 0);**

**}**

**// structure example to receive data**

**// must be matched to the sender structure**

**typedef struct struct\_message {**

**int forward;**

**int backward;**

**int left;**

**int right;**

**} struct\_message;**

**// Create a struct\_message called myData**

**struct\_message myData;**

**// callback function that will be executed when data is received**

**void OnDataRecv(const uint8\_t \* mac, const uint8\_t \*incomingData, int len) {**

**memcpy(&myData, incomingData, sizeof(myData));**

**Serial.print("Bytes received: ");**

**Serial.println(len);**

**Serial.print("forward: ");**

**Serial.println(myData.forward);**

**Serial.print("backward: ");**

**Serial.println(myData.backward);**

**Serial.print("left: ");**

**Serial.println(myData.left);**

**Serial.print("right: ");**

**Serial.println(myData.right);**

**Serial.println();**

**}**

**void setup() {**

**// put your setup code here, to run once:**

**// Initialize Serial Monitor**

**Serial.begin(115200);**

**// Set device as a Wi-Fi Station**

**WiFi.mode(WIFI\_MODE\_STA);**

**// Init ESP-NOW**

**if (esp\_now\_init() != ESP\_OK) {**

**Serial.println("Error initializing ESP-NOW");**

**return;**

**}**

**// Once ESPNow is successfully Init, we will register for recv CB to**

**// get recv packer info**

**esp\_now\_register\_recv\_cb(OnDataRecv);**

**setUpPinModes();**

**}**

**void loop() {**

**// put your main code here, to run repeatedly:**

**if (myData.forward >= 10) {moveCar(FORWARD ,myData.forward );}**

**else if (myData.backward >= 10) {moveCar(BACKWARD,myData.backward);}**

**else if (myData.left >= 10) {moveCar(LEFT ,myData.left );}**

**else if (myData.right >= 10) {moveCar(RIGHT ,myData.right );}**

**/\***

**if (myData.forward == 1) {moveCar(FORWARD);}**

**else if (myData.backward == 1) {moveCar(BACKWARD);}**

**else if (myData.left == 1) {moveCar(LEFT);}**

**else if (myData.right == 1) {moveCar(RIGHT);}**

**\*/**

**// else if (RStickX < -50) {moveCar(ROTATE\_LEFT);}**

**// else if (RStickX > 50) {moveCar(ROTATE\_RIGHT);}**

**// else if (LStickY > 50 && LStickX < -50) {moveCar(FORWARD\_LEFT);}**

**// else if (LStickY > 50 && LStickX > 50) {moveCar(FORWARD\_RIGHT);}**

**// else if (LStickY < -50 && LStickX < -50) {moveCar(BACKWARD\_LEFT);}**

**// else if (LStickY < -50 && LStickX > 50) {moveCar(BACKWARD\_RIGHT);}**

**else{moveCar(STOP);}**

**}**

**Example code**

case FORWARD:

      rotateMotor(FRONT\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);

      rotateMotor(BACK\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);

      rotateMotor(FRONT\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);

      rotateMotor(BACK\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);

      break;

    case BACKWARD:

      rotateMotor(FRONT\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);

      rotateMotor(BACK\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);

      rotateMotor(FRONT\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);

      rotateMotor(BACK\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);

      break;

    case LEFT:

      rotateMotor(FRONT\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);

      rotateMotor(BACK\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);

      rotateMotor(FRONT\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);

      rotateMotor(BACK\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);

      break;

    case RIGHT:

      rotateMotor(FRONT\_RIGHT\_MOTOR, -MAX\_MOTOR\_SPEED);

      rotateMotor(BACK\_RIGHT\_MOTOR, MAX\_MOTOR\_SPEED);

      rotateMotor(FRONT\_LEFT\_MOTOR, MAX\_MOTOR\_SPEED);

      rotateMotor(BACK\_LEFT\_MOTOR, -MAX\_MOTOR\_SPEED);

      break;

Reference:

<https://github.com/wysh2503/MBS4544_Robot_Sensing_and_Vision/blob/main/PS4-ESP32-MecanumWheel/PS4-ESP32-MecanumWheel.ino>