// add turning code here- make rover turn if distance is less than, say 6 inches again

// make turning stop when yaw ~ 87 degrees

// \*\*stop for a second

// read distance

// if distance > 6 inches, move forward 12 inches (obst is 8 inches long)

// if dist < 6 inches, turn 180 degrees (from the right if we turned right by 90 degrees, left if left)

// turn back by 90 degrees (if right, left. vice versa)

// turn right by 90 degrees

// repeat from \*\*

// MOSI = 51, MISO = 50, SCK = scl = 52, CS = = 53

// sda = data // scl goes to scl,

// Get Gyroscope up and running

#include <Wire.h>

const int MPU = 0x68; // MPU6050 I2C address

float yaw = 0;

float GyroZ, gyroAngleZ, GyroErrorZ;

float elapsedTime, currentTime, previousTime;

int c = 0;

char getstr;

int leftSpeed = 255;

int rightSpeed = 150;

//goToRightWall() goToLeftWall() goToMiddle() goToBottomLeftCorner() -- will orient us in a 0,0 axis

//readDistance()

/\* NECESSARY VARIABLES:

float distance

bool wallOnLeft

bool wallOnRight

bool wallInFront

\*/

#define leftIn1 26 // bottom left wheel

#define leftIn2 28 //

#define leftIn3 24

#define leftIn4 22

#define leftENA A0

#define leftENB A1

#define rightIn1 52

#define rightIn2 50

#define rightIn3 46 // bottom right wheel

#define rightIn4 48 //

#define rightENA A15

#define rightENB A14

#define leftTrig A2

#define leftEcho A3

#define rightTrig A12

#define rightEcho A13

#define frontLeftTrig A4

#define frontLeftEcho A5

#define frontRightTrig A10

#define frontRightEcho A11

#define backTrig A6

#define backEcho A7

void setup()

{

Serial.begin(9600);

pinMode(leftIn1, OUTPUT); pinMode(leftIn2, OUTPUT);

pinMode(leftIn3, OUTPUT); pinMode(leftIn4, OUTPUT);

pinMode(leftENA, OUTPUT); pinMode(leftENB, OUTPUT);

pinMode(rightIn1, OUTPUT); pinMode(rightIn2, OUTPUT);

pinMode(rightIn3, OUTPUT); pinMode(rightIn4, OUTPUT);

pinMode(rightENA, OUTPUT); pinMode(rightENB, OUTPUT);

/\* Wire.begin(); // Initialize comunication

Wire.beginTransmission(MPU); // Start communication

Wire.write(0x6B); // Communicate with register 6B

Wire.write(0x00); // Reset by placing a 0 into the 6B register

Wire.endTransmission(true);

calculate\_IMU\_error(); \*/

}

void loop()

{

forward();

delay(3000);

backward();

delay(3000);

}

void forward()

{

Serial.println("Moving Forward");

analogWrite(leftENA, leftSpeed);

digitalWrite(leftIn1, LOW);

digitalWrite(leftIn2, HIGH);

analogWrite(leftENB, leftSpeed);

digitalWrite(leftIn3, LOW);

digitalWrite(leftIn4, HIGH);

analogWrite(rightENA, rightSpeed);

digitalWrite(rightIn1, LOW);

digitalWrite(rightIn2, HIGH);

analogWrite(rightENB, rightSpeed);

digitalWrite(rightIn3, LOW);

digitalWrite(rightIn4, HIGH);

}

void backward()

{

Serial.println("Moving Backward");

analogWrite(leftENA, leftSpeed);

digitalWrite(leftIn1, HIGH);

digitalWrite(leftIn2, LOW);

analogWrite(leftENB, leftSpeed);

digitalWrite(leftIn3, HIGH);

digitalWrite(leftIn4, LOW);

analogWrite(rightENA, rightSpeed);

digitalWrite(rightIn1, HIGH);

digitalWrite(rightIn2, LOW);

analogWrite(rightENB, rightSpeed);

digitalWrite(rightIn3, HIGH);

digitalWrite(rightIn4, LOW);

}

void right() // You will need to fill in this function

{

an alogWrite(leftENA, leftSpeed); //going forwards

digitalWrite(leftIn1, LOW);

digitalWrite(leftIn2, HIGH);

analogWrite(leftENB, leftSpeed);

digitalWrite(leftIn3, LOW);

digitalWrite(leftIn4, HIGH);

analogWrite(rightENA, rightSpeed); //going backwards

digitalWrite(rightIn1, HIGH);

digitalWrite(rightIn2, LOW);

analogWrite(rightENB, rightSpeed);

digitalWrite(rightIn3, HIGH);

digitalWrite(rightIn4, LOW);

}

void left() // You will need to fill in this function

{

analogWrite(leftENA, leftSpeed);

digitalWrite(leftIn1, HIGH);

digitalWrite(leftIn2, LOW);

analogWrite(leftENB, leftSpeed);

digitalWrite(leftIn3, HIGH);

digitalWrite(leftIn4, LOW);

analogWrite(rightENA, rightSpeed);

digitalWrite(rightIn1, LOW);

digitalWrite(rightIn2, HIGH);

analogWrite(rightENB, rightSpeed);

digitalWrite(rightIn3, LOW);

digitalWrite(rightIn4, HIGH);

}

void stop()

{

// Print a message to the serial monitor saying that the rover has stopped

Serial.println("Stopped");

// Switch leftENA and rightENB to low so that all motors stop turning

analogWrite(leftENA, LOW);

analogWrite(leftENB, LOW);

analogWrite(rightENA, LOW);

analogWrite(rightENB, LOW);

}

float read\_yaw() {

previousTime = currentTime; // Use the time when read\_yaw was last called as the previous time

currentTime = millis(); // Update the value of the current time

elapsedTime = (currentTime - previousTime) / 1000; // Find the elapsed time and divide by 1000 to get seconds

Wire.beginTransmission(MPU); // Start Communication

Wire.write(0x47); // Start with the first of the two registers that hold the GYRO\_ZOUT data (There are GYRO\_ZOUT\_H and GYRO\_ZOUT\_L)

Wire.endTransmission(false); // End the communication

Wire.requestFrom(MPU, 2, true); // Read the values of the two bytes that contain the data we need

GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0; // Combine the two bytes by left shifting the first 8 bits and adding on the second set of 8 bits

// Correct the outputs with the calculated error values

GyroZ = GyroZ - GyroErrorZ; // GyroErrorZ ~ (-0.8)

// Find the angle in degrees by multiplying the angular velocity, which is in deg/s, by time to get the change in angle and then add that to the initial angular position

yaw = yaw + GyroZ \* elapsedTime;

// Print the values on the serial monitor

return yaw;

}

float calculate\_IMU\_error() {

// Read gyro values 200 times

while (c < 200) {

Wire.beginTransmission(MPU); // Start communication

Wire.write(0x47); // Start with the first of the two registers that hold the GYRO\_ZOUT data (There are GYRO\_ZOUT\_H and GYRO\_ZOUT\_L)

Wire.endTransmission(false); // End the communication

Wire.requestFrom(MPU, 2, true); // Read the values of the two bytes that contain the data we need

GyroZ = Wire.read() << 8 | Wire.read(); // Combine the two bytes by left shifting the first 8 bits and adding on the second set of 8 bits

GyroErrorZ = GyroErrorZ + (GyroZ / 131.0); // Sum the average of the readings so that they can be averaged

c++;

}

GyroErrorZ = GyroErrorZ / 200; // Average the gyro readings so that we know what the 0 point is

return GyroErrorZ;

}