```
/*
Author:  Taylor Cole Brennan
Date:   January 28, 2022
Project: Auto Brake Lights, Bike project
Class:   Junior Design 2
*/
        /* With support from:
        ===Contact & Support===
        Website: http://eeenthusiast.com/
        Youtube: https://www.youtube.com/EEEnthusiast
        Facebook: https://www.facebook.com/EEEnthusiast/
        Patreon: https://www.patreon.com/EE_Enthusiast
        Revision: 1.0 (July 13th, 2016)
        ===Hardware===
        - Arduino Uno R3
        - MPU-6050 (Available from: http://eeenthusiast.com/product/6dof-mpu-6050-accelerometer-gyroscope-temperature/)
        ===Software===
        - Latest Software: https://github.com/VRomanov89/EEEnthusiast/tree/master/MPU-6050%20Implementation/MPU6050_Implementation
        - Arduino IDE v1.6.9
        - Arduino Wire library
        ===Terms of use===
        The software is provided by EEEnthusiast without warranty of any kind. In no event shall the authors or
        copyright holders be liable for any claim, damages or other liability, whether in an action of contract,
        tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in
        the software.
```

```
       */


#include <Wire.h>
  //inputs
int left_button = 3;
int right_button = 2;
// Outputs
int left_led = 5; //digital output, indicator light
int right_led = 4; //digital output, indicator light
int left_light = 9; //digital output, tail light
int right_light = 11; //digital output, tail light


  //gyro/accel variables
float AccX, AccY, AccZ;
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ;
float roll, pitch, yaw;
float rotX, rotY, rotZ;
float y_test, p_test;
int y_count, p_count, y_target=1000, p_target=500;


  //time variables, States, and duration
unsigned long curr_millis;
float elapsedTime, currentTime, previousTime;
int left_state = HIGH, right_state = HIGH;
unsigned long butt_millis = 0, left_millis = 0, right_millis = 0;
unsigned long left_pwm_millis = 0, right_pwm_millis = 0;


// Global variables
```

```
// Thresholds and save variables
float brake_thresh = .01; //dummy value need to be calc.
float left_thresh = -4.0, left_buffer=-2.0, turn_buffer=0.7, accel_buffer=0.2;
float right_thresh = (-1)*left_thresh, right_buffer=(-1)*left_buffer;
int hold_left = 0, hold_right = 0, left_dummy, right_dummy;


  //blinker variables
int blinker_delay = 500;
int last_button = 250;
float turn, brake, turn_save, brake_save;
int pwm_tot = 10;
int pwm_on =30, pwn_high=255, pwn_low=30;


int left_pwm = LOW, right_pwm = LOW;



void setup() {
  // put your setup code here, to run once:
  pinMode(left_button, INPUT); //digital
  pinMode(right_button, INPUT); //digital
  pinMode(left_led, OUTPUT); //digital
  pinMode(right_led, OUTPUT); //digital
  pinMode(left_light, OUTPUT); //digital
  pinMode(right_light, OUTPUT); //digital
  Wire.begin();
  setupMPU();
  Serial.begin(9600);
}
```

```
void loop() {
  recordAccelRegisters();
  recordGyroRegisters();
  updateValues();
  checkInput();
  updateLights();
}

void checkInput() {
  curr_millis = millis();
  if (curr_millis - butt_millis >= last_button) {
        if (digitalRead(left_button)) {
        butt_millis = millis();
        if (hold_left == 0) {
        turn_save=pitch;
        reset_variable();
        hold_left = 1;
        digitalWrite(right_led, LOW);
        } else {
        reset_variable();
        digitalWrite(left_led, LOW);
        }
        }
        if (digitalRead(right_button)) {
        butt_millis = millis();
        if (hold_right == 0) {
        turn_save=pitch;
        reset_variable();
        hold_right = 1;
```

```
        digitalWrite(left_led, LOW);

        } else {

        reset_variable();

        digitalWrite(right_led, LOW);

        }

        }

  }

}



void updateLights() {

  curr_millis = millis();

  int pwn_value = pwm_on;

  if (y_test < yaw ) { brake_save=y_test; brake = 1;}

  if (brake && yaw<brake_save) {brake=0; brake_save=0;}

  if (brake)if(y_test-brake_save<=2) pwn_value=(pwn_high-pwn_low)*((y_test-
brake_save)*2)+pwn_low;else pwn_value=pwn_high;

  else pwn_value=pwn_low;

  if (pwn_value<pwn_low)pwn_value=pwn_low;

  else if (pwn_value>pwn_high)pwn_value=pwn_high;

  if (hold_left == 1) {

        if ((left_state == HIGH) && (curr_millis - left_millis >= blinker_delay)) {

        left_state = LOW;

        left_millis = curr_millis;

        digitalWrite(left_led, left_state);

        }

        else if ((left_state == LOW) && (curr_millis - left_millis >= blinker_delay)) {

        left_state = HIGH;

        left_millis = curr_millis;

        digitalWrite(left_led, left_state);
```

```
      }
      if (pitch < (turn_save+left_thresh)) left_dummy=1;
}
else if (hold_right == 1) {
      if ((right_state == HIGH) && (curr_millis - right_millis >= blinker_delay)) {
      right_state = LOW;
      right_millis = curr_millis;
      digitalWrite(right_led, right_state);
      }
      else if ((right_state == LOW) && (curr_millis - right_millis >= blinker_delay)) {
      right_state = HIGH;
      right_millis = curr_millis;
      digitalWrite(right_led, right_state);

      }
      if (pitch > (turn_save+right_thresh)) right_dummy=1;

} else {
      left_state = HIGH;
      right_state = HIGH;
}

pwmLights(pwn_value);

if (pitch > (turn_save+left_buffer) && left_dummy){
      hold_left =0;
      left_dummy=0;
      digitalWrite(left_led, LOW);
      turn_save=0;
```

```
  }
  else if (pitch < (turn_save+right_buffer) && right_dummy){
        hold_right = 0;
        right_dummy=0;
        digitalWrite(right_led, LOW);
        turn_save=0;
  }
  //turn off lights with turn value
}

void pwmLights(int pwn_value) {
  curr_millis = millis();
  if (left_state == HIGH) {
        analogWrite(left_light, pwn_value);
  } else {
        analogWrite(left_light, LOW);
  }
  if (right_state == HIGH) {
        analogWrite(right_light, pwn_value);
  } else {
        analogWrite(right_light, LOW);
  }
}

void setupMPU(){
  Wire.beginTransmission(0b1101000); //This is the I2C address of the MPU (b1101000/b1101001 for
AC0 low/high datasheet sec. 9.2)
  Wire.write(0x6B); //Accessing the register 6B - Power Management (Sec. 4.28)
  Wire.write(0b00000000); //Setting SLEEP register to 0. (Required; see Note on p. 9)
```

```
  Wire.endTransmission();

  Wire.beginTransmission(0b1101000); //I2C address of the MPU

  Wire.write(0x1B); //Accessing the register 1B - Gyroscope Configuration (Sec. 4.4)

  Wire.write(0x00000000); //Setting the gyro to full scale +/- 250deg./s

  Wire.endTransmission();

  Wire.beginTransmission(0b1101000); //I2C address of the MPU

  Wire.write(0x1C); //Accessing the register 1C - Acccelerometer Configuration (Sec. 4.5)

  Wire.write(0b00000000); //Setting the accel to +/- 2g

  Wire.endTransmission();

}



void recordAccelRegisters() {

  Wire.beginTransmission(0b1101000); //I2C address of the MPU

  Wire.write(0x3B); //Starting register for Accel Readings

  Wire.endTransmission();

  Wire.requestFrom(0b1101000,6); //Request Accel Registers (3B - 40)

  AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // X-axis value

  AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // Y-axis value

  AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // Z-axis value //Store last two bytes into
accelZ

  processAccelData();

}



void processAccelData(){

  accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) - 0.58; // AccErrorX
~(0.58) See the calculate_IMU_error()custom function for more details

  accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI) + 1.58; //
AccErrorY ~(-1.58)

  }
```

```
void recordGyroRegisters() {

  previousTime = currentTime;        // Previous time is stored before the actual time read

  currentTime = millis();            // Current time actual time read

  elapsedTime = (currentTime - previousTime) / 1000;// Divide by 1000 to get seconds

  Wire.beginTransmission(0b1101000); //I2C address of the MPU

  Wire.write(0x43); //Starting register for Gyro Readings

  Wire.endTransmission();

  Wire.requestFrom(0b1101000,6); //Request Gyro Registers (43 - 48)

  GyroX = (Wire.read() << 8 | Wire.read()) / 131.0; // For a 250deg/s range we have to divide
first the raw value by 131.0, according to the datasheet

  GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;

  GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;

  processGyroData();

}



void processGyroData() {



  GyroX = GyroX + 2.5; // GyroErrorX ~(-0.56)

  GyroY = GyroY - 1.80; // GyroErrorY ~(2)

  GyroZ = GyroZ + 0.79; // GyroErrorZ ~ (-0.8)

        if (( GyroX * elapsedTime) < -.05 || ( GyroX* elapsedTime) > .05)

  gyroAngleX = gyroAngleX + GyroX * elapsedTime; // deg/s * s = deg

        if (( GyroY * elapsedTime) < -.05 || ( GyroY* elapsedTime) > .05)

  gyroAngleY = gyroAngleY + GyroY * elapsedTime;



}

void   updateValues(){

  float new_pitch= ((0.96 * gyroAngleY + 0.04 * accAngleY)+2.51);

  y_test = yaw;
```

```
  p_test = pitch;


        //check if we are getting good data, or if it's random
  if (( GyroZ * elapsedTime)< (-1)*accel_buffer || ( GyroZ * elapsedTime) > accel_buffer)
        {yaw =  yaw + GyroZ * elapsedTime; y_count=0;}
  else y_count+=1;
  //reset axis and varibles if reached count
  if (y_count == y_target)
        {yaw =0; y_count=0; brake=0;}


        //check if we are getting good data, or if it's random
  if (( p_test - new_pitch)  < (-1)*turn_buffer || ( p_test - new_pitch) > turn_buffer)
        {pitch = new_pitch;  p_count=0;}
  else p_count+=1;
        //reset axiz and variable if reach count
  if (p_count == p_target) {p_count=0; gyroAngleY=0; reset_variable();}
  // roll = 0.96 * gyroAngleX + 0.04 * accAngleX;


  Serial.print(roll);
  Serial.print("/");
  Serial.print(pitch);
  Serial.print("/");
  Serial.println(yaw);


  }


void reset_variable()
{
        left_dummy=0; right_dummy=0; hold_left=0;hold_right=0;
```

```
}
```