#include <SPI.h>

#include <PID\_v1.h>

#include <nRF24L01.h>

#include <RF24.h>

#include "I2Cdev.h"

#include "LMotorController.h"

#include "MPU6050\_6Axis\_MotionApps20.h"

#if I2CDEV\_IMPLEMENTATION == I2CDEV\_ARDUINO\_WIRE

 #include "Wire.h"

#endif

#define CE\_PIN 9

#define CSN\_PIN 10

#define single\_red 30                                                 // when detect single white line, red led on

#define double\_green 31                                               // when detect double white line, green led on

//———————————————————————————//

#define MIN\_ABS\_SPEED 5 //30

#define threshold 2000

//———————————————————————————//

MPU6050 mpu;

// MPU control/status variables

bool dmpReady = false;                                                // set true if DMP init was successful

uint8\_t mpuIntStatus;                                                 // holds actual interrupt status byte from MPU

uint8\_t devStatus;                                                    // return status after each device operation (0 = success, !0 = error)

uint16\_t packetSize;                                                  // expected DMP packet size (default is 42 bytes)

uint16\_t fifoCount;                                                   // count of all bytes currently in FIFO

uint8\_t fifoBuffer[64];                                               // FIFO storage buffer

// orientation/motion variables

Quaternion q;                                                         // [w, x, y, z] quaternion container

VectorFloat gravity;                                                  // [x, y, z] gravity vector

float ypr[3];                                                         // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

//———————————————————————————//

int speedlimit = 255;

int sampletime = 5;

int moveState= 1;

int turnPWM = 10;                                                     // 0 = balance; 1 = forth; 2 = back

double originalSetpoint = 180.0;

double setpoint = originalSetpoint;

double movingAngleOffset = 0.0;

double input, output;

double Kp = 13;   //13

double Kd = 0.5; //0.5

double Ki = 100; //100

PID pid(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);

double motorSpeedFactorLeft = 0.8;

double motorSpeedFactorRight = 1;

//———————————————————————————//

// L298N

int ENA = 3;

int IN1 = 5;

int IN2 = 4;

int IN3 = 7;

int IN4 = 6;

int ENB = 8;

LMotorController motorController(ENA, IN1, IN2, ENB, IN3, IN4, motorSpeedFactorLeft, motorSpeedFactorRight);

// turn direction

int leftSpeed = 0;

int rightSpeed = 0;

int turn = 0;

// RF receiver

double data[4];

RF24 radio(CE\_PIN, CSN\_PIN);                                          // 9 and 10 are a digital pin numbers to which signals CE and CSN are connected

const byte address[6] = "00001";

// Timers

long setFeq1 = 0;

long setFeq2 = 0;

long setFeq3 = 0;

volatile bool trigger =false;

volatile bool black = false;

volatile bool white = false;

volatile bool mpuInterrupt = false;                                   // indicates whether MPU interrupt pin has gone high

void dmpDataReady()

{

  mpuInterrupt = true;

}

void setup()

{

  // join I2C bus

  #if I2CDEV\_IMPLEMENTATION == I2CDEV\_ARDUINO\_WIRE

  Wire.begin();

  TWBR = 24;                                                          // 400kHz I2C clock (200kHz if CPU is 8MHz)

  #elif I2CDEV\_IMPLEMENTATION == I2CDEV\_BUILTIN\_FASTWIRE

  Fastwire::setup(400, true);

  #endif

  // initialization

  mpu.initialize();

  devStatus = mpu.dmpInitialize();

  Serial.begin(9600);

  // gyro offsets

  mpu.setXGyroOffset(-135);

  mpu.setYGyroOffset(17);

  mpu.setZGyroOffset(-27);

  mpu.setZAccelOffset(1145);                                          // 1688 factory default for my test chip

  // set LED

  pinMode(single\_red,OUTPUT);

  pinMode(double\_green,OUTPUT);

  // set modem

  radio.begin();

  radio.openReadingPipe(0, address);

  radio.setPALevel(RF24\_PA\_MIN);

  radio.startListening();

  // make sure it worked (returns 0 if so)

  if (devStatus == 0)

  {

    // turn on the DMP, now that it's ready

    mpu.setDMPEnabled(true);

    // enable Arduino interrupt detection

    attachInterrupt(0, dmpDataReady, RISING);

    mpuIntStatus = mpu.getIntStatus();

    // set our DMP Ready flag

    dmpReady = true;

    // get expected DMP packet size for later comparison

    packetSize = mpu.dmpGetFIFOPacketSize();

    //setup PID

    pid.SetMode(AUTOMATIC);

    pid.SetSampleTime(sampletime);

    pid.SetOutputLimits(-speedlimit, speedlimit);

  }

  else

  {

    // ERROR!

    // 1 = initial memory load failed

    // 2 = DMP configuration updates failed

    // (if it's going to break, usually the code will be 1)

    Serial.print(F("DMP Initialization failed (code "));

    Serial.print(devStatus);

    Serial.println(F(")"));

  }

}

void loop()

{

  unsigned long currentMillis = millis();

  if ((currentMillis - setFeq2) >= 20){

    detectLines();

    setFeq2 = currentMillis;

  }

  // if programming failed, don't try to do anything

  if (!dmpReady) return;

  // wait for MPU interrupt or extra packet(s) available

  while (!mpuInterrupt && fifoCount < packetSize){

    //no mpu data - performing PID calculations and output to motors

    pid.Compute();

    //Serial.println("test");

    leftSpeed = output + turn;

    rightSpeed = output - turn;

    motorController.move(leftSpeed, rightSpeed, MIN\_ABS\_SPEED);

    if (radio.available()) {

      radio.read(&data, sizeof(data));

      //Serial.println(data[2]);

      turn = map(data[1], 0, 1023, -turnPWM, turnPWM);

      if (data[2] != 512){

        moveBackForth();

      }

    }

  }

  // reset interrupt flag and get INT\_STATUS byte

  mpuInterrupt = false;

  mpuIntStatus = mpu.getIntStatus();

  // get current FIFO count

  fifoCount = mpu.getFIFOCount();

  // check for overflow (this should never happen unless our code is too inefficient)

  if ((mpuIntStatus & 0x10) || fifoCount == 1024)

  {

    // reset so we can continue cleanly

    mpu.resetFIFO();

    Serial.println(F("FIFO overflow!"));

    // otherwise, check for DMP data ready interrupt (this should happen frequently)

  }

  else if (mpuIntStatus & 0x02)

  {

    // wait for correct available data length, should be a VERY short wait

    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO

    mpu.getFIFOBytes(fifoBuffer, packetSize);

    // track FIFO count here in case there is > 1 packet available

    // (this lets us immediately read more without waiting for an interrupt)

    fifoCount -= packetSize;

    Serial.println(fifoCount);

    mpu.dmpGetQuaternion(&q, fifoBuffer);

    mpu.dmpGetGravity(&gravity, &q);

    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

    input = ypr[1] \* 180/M\_PI + 180;

  }

}

void detectLines(){

  int read\_dd = RCTime(25);

  //Serial.println(read\_dd);

  unsigned long currentMillis = millis();

  digitalWrite(single\_red, LOW);

  digitalWrite(double\_green, LOW);

  if(read\_dd < threshold && trigger == false){

    trigger = true;

    setFeq3 = currentMillis;

  }

  if(read\_dd < threshold && trigger == true && black == false && white == false){

    if ((currentMillis - setFeq3) >= 1500){

      digitalWrite(single\_red, HIGH);

      digitalWrite(double\_green, HIGH);

      trigger = false;

    }

  }

  if(read\_dd > threshold && trigger == true && black == false){

    setFeq1 = currentMillis;

    black = true;

  }

  if(read\_dd < threshold && black == true && white ==false){

    white=true;

  }

  if(black == true){

    if ((currentMillis - setFeq1) >= 600){

      if(white == true){

        digitalWrite(double\_green, HIGH);

        trigger = false;

        black = false;

        white = false;

      }else{

        digitalWrite(single\_red, HIGH);

        trigger = false;

        black = false;

        white = false;}

    }

  }

}

void moveBackForth(){

  if (data[2] > 612){

    movingAngleOffset = (data[2] - 612)/200;

    setpoint = originalSetpoint - movingAngleOffset;

    //Serial.println(setpoint);

  }

  else if (data[2] < 412){

    movingAngleOffset = (412 - data[2])/75;

    setpoint = originalSetpoint + movingAngleOffset;

    //Serial.println(setpoint);

  }

  else{

    setpoint = originalSetpoint;

  }

}

long RCTime(int sensorIn){

   long duration = 0;

   unsigned long startTime = millis();

   pinMode(sensorIn, OUTPUT);     // Make pin OUTPUT

   digitalWrite(sensorIn, HIGH);  // Pin HIGH (discharge capacitor)

   delay(1);                      // Wait 1ms

   pinMode(sensorIn, INPUT);      // Make pin INPUT

   digitalWrite(sensorIn, LOW);   // Turn off internal pullups

   while(digitalRead(sensorIn) && (millis() - startTime < 30)){  // Wait for pin to go LOW

      duration++;

  }

  return duration;

}