CODE FOR CAPSTONE PROJECT

#include <Wire.h>

#include <MPU6050\_6Axis\_MotionApps20.h>

#include "CytronMotorDriver.h"

#include <BluetoothSerial.h> // Include Bluetooth library for ESP32

// MPU6050 and DMP-related declarations

MPU6050 mpu;

bool dmpReady = false;

uint8\_t mpuIntStatus;

uint8\_t devStatus;

uint16\_t packetSize;

uint16\_t fifoCount;

uint8\_t fifoBuffer[64];

// Orientation/motion vars

Quaternion q;

VectorFloat gravity;

float ypr[3];

// Motor pin definitions

const int M1A = 4; // Left motor

const int M1B = 2;

const int M2A = 19; // Right motor

const int M2B = 18;

// Create motor driver objects

CytronMD leftMotor(PWM\_PWM, M1A, M1B);

CytronMD rightMotor(PWM\_PWM, M2A, M2B);

// PID constants - may need tuning for your specific robot

float Kp = 17; //15.0 //17 //18

float Ki = 0.05; //0.03 //0.05 //0.05

float Kd = 0.13; //1.2 //1.7 //0.1

float targetAngle = 0; // Target angle for balancing

float moveTarget = 0; // Additional angle for movement

// PID variables

float lastError = 0;

float iTerm = 0;

unsigned long lastTime = 0;

// Max PWM value

#define MAX\_PWM 255 // Assuming 8-bit PWM

// Timing

unsigned long loopTime = 10; // 10ms per loop (100Hz)

unsigned long lastLoopTime = 0;

// LED blink state

bool blinkState = false;

// Bluetooth

BluetoothSerial SerialBT; // Create a BluetoothSerial object

void setup() {

Wire.begin();

Serial.begin(115200);

Serial.println(F("Initializing MPU6050..."));

mpu.initialize();

// Verify connection

Serial.println(F("Testing device connections..."));

Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

// Initialize DMP

Serial.println(F("Initializing DMP..."));

devStatus = mpu.dmpInitialize();

// Supply your own gyro offsets here, scaled for min sensitivity

mpu.setXGyroOffset(73);

mpu.setYGyroOffset(-3);

mpu.setZGyroOffset(30);

mpu.setZAccelOffset(1327);

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

if (devStatus == 0) {

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

Serial.println(F("Enabling DMP..."));

mpu.setDMPEnabled(true);

// Get expected DMP packet size for later comparison

packetSize = mpu.dmpGetFIFOPacketSize();

// Set our DMP Ready flag so the main loop() function knows it's okay to use it

Serial.println(F("DMP ready! Waiting for first interrupt..."));

dmpReady = true;

} else {

// ERROR!

// 1 = initial memory load failed

// 2 = DMP configuration updates failed

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

Serial.print(devStatus);

Serial.println(F(")"));

}

// Configure LED for output

pinMode(LED\_BUILTIN, OUTPUT);

// Initialize Bluetooth

SerialBT.begin("SelfBalancingRobot"); // Set Bluetooth device name

Serial.println("Bluetooth started, waiting for connections...");

}

void loop() {

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

if (!dmpReady) return;

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

while (fifoCount < packetSize) {

fifoCount = mpu.getFIFOCount();

// Check for timeout

if (millis() - lastLoopTime > loopTime) {

break;

}

}

// Reset interrupt flag and get INT\_STATUS byte

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!"));

} 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

fifoCount -= packetSize;

// Get Euler angles in degrees

mpu.dmpGetQuaternion(&q, fifoBuffer);

mpu.dmpGetGravity(&gravity, &q);

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

float pitch = ypr[1] \* 180/M\_PI;

// Calculate PID output

float pidOutput = computePID(pitch);

// Map PID output to motor speeds

int leftSpeed = constrain(pidOutput, -MAX\_PWM, MAX\_PWM);

int rightSpeed = constrain(pidOutput, -MAX\_PWM, MAX\_PWM);

// Set motor speeds

leftMotor.setSpeed(leftSpeed);

rightMotor.setSpeed(rightSpeed);

// Print debug info

Serial.print("Pitch: ");

Serial.print(pitch);

Serial.print(" PID Output: ");

Serial.println(pidOutput);

Serial.print("target value");

Serial.println(targetAngle);

// Blink LED to indicate activity

blinkState = !blinkState;

digitalWrite(LED\_BUILTIN, blinkState);

}

// Check for Bluetooth data

if (SerialBT.available() >= 4) { // Check if at least 4 bytes are available

uint32\_t command = 0;

for (int i = 0; i < 4; i++) {

command |= ((uint32\_t)SerialBT.read() << (i \* 8)); // Read each byte and shift it into place

}

// Print the received command

Serial.print("Received command: ");

Serial.println(command);

if (command == 201)

{ // Command to move forward

targetAngle = 1.5;

//forward

// Adjust target angle to move forward

}else if(command==202)

{

//backward

targetAngle=-1.5;

}

else if (command == 220) { // Command to stop

targetAngle = 0.0; // Reset to balance position

}

}

// Maintain loop timing

while (millis() - lastLoopTime < loopTime);

lastLoopTime = millis();

}

float computePID(float input)

{

float error = targetAngle - input;

if (abs(error) < 0.15) {

error = 0; // Don't react to very small errors

}

unsigned long now = millis();

float timeChange = (float)(now - lastTime);

float pTerm = Kp \* error;

iTerm += Ki \* error \* timeChange;

iTerm = constrain(iTerm, -MAX\_PWM, MAX\_PWM);

float dTerm = Kd \* (error - lastError) / timeChange;

lastError = error;

lastTime = now;

return pTerm + iTerm + dTerm;

}