**CENG4480 Embedded System Development and Applications**

**Computer Science and Engineering Department**

**The Chinese University of Hong Kong**

**Laboratory 8: Self-balancing Robot (2) (software)**

November, 2016

**Introduction**

In this lab you will complete your self-balancing robot by coding the program and tuning the parameters of the proportional-integral-derivative controller (PID) to enable your robot to perform the self-balancing act.



**Figure 1. CENG4480 self-balancing robot program flow chart**

**Objectives**

* To learn how to develop software for q computer controlled system
* To learn how to test and calibrate computer controlled electro-mechanical systems

Procedures and what to submit:

Demonstrate your robot to a tutor after your complete the experiment, no need to submit lab report.

Experimental Procedures

1. **Calculate the angle from accelerometers values**
* On the provided skeleton program Lab8.ino add the angle calculation codes as the following:

 Ayz=atan2(RwAcc[1],RwAcc[2])\*180/PI; //angle measured by accelerometer

 Ayz-=offset; //adjust to correct balance point

1. **Add the complement and Kalman filters**
* On the provided skeleton program Lab8.ino add the complement and Kalman filters codes as the following:

Angy = 0.998\*(Angy+GyroIN[0]\*interval/1000)+0.002\*Ayz; //complement filter

kang = kalmanCalculate(Angy, GyroIN[0],interval); //kalman filter

Serial.println(kang);

1. **Add the PID calculation and update the speed of motors**
* On the provided skeleton program Lab8.ino add the PID calculation and update the speed of motors code as the following:

if ((abs(kang)>=minangle)&&(abs(kang)<maxangle)){

 delta=kang;

 diff = delta - last;

 diff2 = delta - last2;

 diff = constrain(diff,-maxdiff,maxdiff);

 diff2 = constrain(diff2,-maxdiff,maxdiff);

 last2 = last;

 last = delta;

 LRspeed = P\*delta + I\*accu\*interval\*0.001 + D\*(diff\*100+diff2\*100)/interval;

 accu+=delta;

 accu = constrain(accu,-maxaccu,maxaccu);

}

else {

 LRspeed = 0;

 accu = 0;

 last=0;

 diff=0;

}

1. **Calibrate the offset**
* After adding all codes in Lab8.ino then upload it to the Arduino board
* Hold the robot vertically
* Open the COM window and find out the offset value
* Change the offset value in Lab8.ino accordingly
* Upload the Lab8.ino to the Arduino board again
1. **Tuning the PID constants**
* **For each test, change the values of the parameters P, I and D in the program and observe their effects on the performance until you find the best combinations of the parameters. You may follow the following recommendations for changing the parameters.**
* Rough tuning
	+ Step1: Increase the P value in the program in step of 50 units for each test, upload the program of each test to the Arduino and observe its performance. Stop changing P until the robot starts to oscillate (move back and forth)
	+ Step2: Increase I in step of 50 for each test so that the robot accelerates faster when off balanced
	+ Step3: Increase D in step of 10 so that the robot would move about its balanced position in a more gentle and smooth manner, and there shouldn’t be any significant overshoots
	+ Step4: If first attempt doesn’t give the satisfying results, reset PID values and start over again with different value of P
	+ Repeat the step 1,2,3,4 until you find a set of P,I,D values which give a satisfactory result
* Fine tuning: In fine tuning, PID values are restricted to neighboring values found in rough tuning described above (and smaller steps of change for fine tuning). Adjust the P,I,D values until the performance is the best.
1. **Demo to your TAs**
* The longer your robot can automatically stand up-right the better. You will get bonus marks if it can stand still for over 1 minute.

**END**

**Answer:**

**#include <Wire.h>**

**#include <math.h>**

**#include <L3G4200D.h>**

**#include <TimerOne.h>**

**L3G4200D gyro;**

**#define Register\_ID 0**

**#define Register\_2D 0x2D**

**#define Register\_X0 0x32**

**#define Register\_X1 0x33**

**#define Register\_Y0 0x34**

**#define Register\_Y1 0x35**

**#define Register\_Z0 0x36**

**#define Register\_Z1 0x37**

**#define PI 3.14159265358979f**

**#define P 720 // Proportional constant of PID controller**

**#define I 190 // integral constant of PID controller**

**#define D 37 // Differential constant of PID controller**

**#define maxdiff 1000**

**#define maxaccu 1000**

**#define offset -9.5 //more +ve, more directed to Arduino side**

**#define resolution 1000**

**#define minangle 0.2**

**#define maxangle 15**

**const int DIRL1 = 4;**

**const int CLKL = 5;**

**const int DIRR1 = 11;**

**const int CLKR = 7;**

**const int DIRL2 = 10;**

**const int DIRR2 = 6;**

**char firstSample; //marks first sample**

**int ADXAddress = 0xA7 >> 1; // the default 7-bit slave address**

**int reading = 0;**

**int val=0;**

**int X0,X1,X\_out;**

**int Y0,Y1,Y\_out;**

**int Z1,Z0,Z\_out;**

**double Axz,Ayz;**

**double Angx,Angy;**

**double delta,diff,last,diff2,last2,accu;**

**double kang;**

**unsigned long lastMilli;**

**unsigned long interval; //interval since previous analog samples**

**float RwAcc[3]; //projection of normalized gravitation force vector on x/y/z axis, as measured by accelerometer**

**float GyroIN[3]; //gyro value from sensor**

**unsigned int counter\_l,counter\_r;**

**unsigned int speed\_l,speed\_r;**

**unsigned char dir\_l,dir\_r;**

**void setup()**

**{**

 **Wire.begin();**

 **Serial.begin(9600);**

 **pinMode(CLKL, OUTPUT);**

 **pinMode(DIRL1, OUTPUT);**

 **pinMode(DIRL2, OUTPUT);**

 **pinMode(CLKR, OUTPUT);**

 **pinMode(DIRR1, OUTPUT);**

 **pinMode(DIRR2, OUTPUT);**

 **//pinMode(13, OUTPUT);**

 **Timer1.initialize(5000); // initialize timer1, and set 0.5 milisecond period**

 **Timer1.attachInterrupt( timerIsr ); // attach the service routine here**

 **delay(100);**

 **// enable to measure g data**

 **Wire.beginTransmission(ADXAddress);**

 **Wire.write(Register\_2D);**

 **Wire.write(8); //measuring enable**

 **Wire.endTransmission(); // stop transmitting**

 **// enable gyro**

 **Wire.begin();**

 **gyro.enableDefault();**

 **accu = 0;**

 **last = 0;**

 **last2 = 0;**

 **firstSample = 1;**

**}**

**void loop()**

**{**

 **float LRspeed;**

 **static unsigned long newMilli; //new timestamp**

 **newMilli = millis(); //save the time when sample is taken**

 **Read\_acc();**

 **Read\_gyro();**

 **//compute interval since last sampling time in millisecond**

 **interval = newMilli - lastMilli;**

 **lastMilli = newMilli; //save for next loop, please note interval will be invalid in first sample but we don't use it**

 **Ayz=atan2(RwAcc[1],RwAcc[2])\*180/PI; //angle measured by accelerometer**

 **Ayz-=offset; //adjust to correct balance point**

 **Angy = 0.998\*(Angy+GyroIN[0]\*interval/1000)+0.002\*Ayz; //complement filter**

 **kang = kalmanCalculate(Angy, GyroIN[0],interval); //kalman filter**

 **Serial.print(kang);**

 **LRspeed = 0;**

 **////////////////////////////**

 **if ((abs(kang)>=minangle)&&(abs(kang)<maxangle)){**

 **delta=kang;**

 **diff = delta - last;**

 **diff2 = delta - last2;**

 **diff = constrain(diff,-maxdiff,maxdiff);**

 **diff2 = constrain(diff2,-maxdiff,maxdiff);**

 **last2 = last;**

 **last = delta;**

 **LRspeed = P\*delta + I\*accu\*interval\*0.001 + D\*(diff\*100+diff2\*100)/interval;**

 **accu+=delta;**

 **accu = constrain(accu,-maxaccu,maxaccu);**

 **}**

 **else {**

 **LRspeed = 0;**

 **accu = 0;**

 **last=0;**

 **diff=0;**

 **}**

 **Serial.print(", ");**

 **Serial.print(LRspeed/1000);**

 **Serial.println(" ");**

 **if(LRspeed<=0) {**

 **dir\_l = 0;**

 **dir\_r = 0;**

 **}**

 **else {**

 **dir\_l = 1;**

 **dir\_r = 1;**

 **}**

 **LRspeed = abs(LRspeed);**

 **LRspeed = constrain(LRspeed,0,resolution);**

 **speed\_l=LRspeed;**

 **speed\_r=LRspeed;**

**}**

**/// --------------------------**

**/// ISR Timer Routine**

**/// --------------------------**

**void timerIsr() {**

 **// Toggle LED**

 **//digitalWrite( 13, digitalRead( 13 ) ^ 1 );**

 **counter\_l++;**

 **counter\_r++;**

 **if (speed\_l!=0) {**

 **if (counter\_l >= (resolution/speed\_l)){**

 **counter\_l = 0;**

 **if (dir\_l) {**

 **digitalWrite(DIRL1,LOW); // DIR Motor 1**

 **digitalWrite(DIRL2,HIGH); // DIR Motor 1**

 **}**

 **else {**

 **digitalWrite(DIRL1,HIGH);**

 **digitalWrite(DIRL2,LOW);**

 **}**

 **// The width of LOW depends on speed\_l**

 **digitalWrite(CLKL,HIGH);**

 **}**

 **else**

 **digitalWrite(CLKL,LOW);**

 **}**

 **if (speed\_r!=0) {**

 **if (counter\_r >= (resolution/speed\_r)){**

 **counter\_r = 0;**

 **if (dir\_r) {**

 **digitalWrite(DIRR1,LOW); // DIR Motor r**

 **digitalWrite(DIRR2,HIGH); // DIR Motor r**

 **}**

 **else {**

 **digitalWrite(DIRR1,HIGH);**

 **digitalWrite(DIRR2,LOW);**

 **}**

 **// The width of LOW depends on speed\_r**

 **digitalWrite(CLKR,HIGH);**

 **}**

 **else**

 **digitalWrite(CLKR,LOW);**

 **}**

**}**

**void Read\_acc() {**

 **//--------------X**

 **int i;**

 **X\_out=0;**

 **for(i=0;i<10;i++) {**

 **Wire.beginTransmission(ADXAddress); // transmit to device**

 **Wire.write(Register\_X0);**

 **Wire.write(Register\_X1);**

 **Wire.endTransmission();**

 **Wire.requestFrom(ADXAddress,2);**

 **if(Wire.available()<=2)**

 **{**

 **X0 = Wire.read();**

 **X1 = Wire.read();**

 **X1=X1<<8;**

 **X\_out+=(X0+X1);**

 **}**

 **}**

 **X\_out/=10;**

 **//------------------Y**

 **Wire.beginTransmission(ADXAddress); // transmit to device**

 **Wire.write(Register\_Y0);**

 **Wire.write(Register\_Y1);**

 **Wire.endTransmission();**

 **Wire.requestFrom(ADXAddress,2);**

 **if(Wire.available()<=2)**

 **{**

 **Y0 = Wire.read();**

 **Y1 = Wire.read();**

 **Y1=Y1<<8;**

 **Y\_out=Y0+Y1;**

 **}**

 **//------------------Z**

 **Wire.beginTransmission(ADXAddress); // transmit to device**

 **Wire.write(Register\_Z0);**

 **Wire.write(Register\_Z1);**

 **Wire.endTransmission();**

 **Wire.requestFrom(ADXAddress,2);**

 **if(Wire.available()<=2)**

 **{**

 **Z0 = Wire.read();**

 **Z1 = Wire.read();**

 **Z1=Z1<<8;**

 **Z\_out=Z0+Z1;**

 **}**

 **RwAcc[0]=X\_out/256.0;**

 **RwAcc[1]=Y\_out/256.0;**

 **RwAcc[2]=Z\_out/256.0;**

**}**

**void Read\_gyro(){**

 **gyro.read();**

 **GyroIN[0]=gyro.g.x/256.0;**

 **GyroIN[1]=gyro.g.y/256.0;**

 **GyroIN[2]=gyro.g.z/256.0;**

**}**

**// Kalman filter module**

 **float Q\_angle = 0.001;**

 **float Q\_gyro = 0.003;**

 **float R\_angle = 0.03;**

 **float x\_angle = 0;**

 **float x\_bias = 0;**

 **float P\_00 = 0, P\_01 = 0, P\_10 = 0, P\_11 = 0;**

 **float dt, y, S;**

 **float K\_0, K\_1;**

 **float kalmanCalculate(float newAngle, float newRate,int looptime) {**

 **dt = float(looptime)/1000;**

 **x\_angle += dt \* (newRate - x\_bias);**

 **P\_00 += - dt \* (P\_10 + P\_01) + Q\_angle \* dt;**

 **P\_01 += - dt \* P\_11;**

 **P\_10 += - dt \* P\_11;**

 **P\_11 += + Q\_gyro \* dt;**

 **y = newAngle - x\_angle;**

 **S = P\_00 + R\_angle;**

 **K\_0 = P\_00 / S;**

 **K\_1 = P\_10 / S;**

 **x\_angle += K\_0 \* y;**

 **x\_bias += K\_1 \* y;**

 **P\_00 -= K\_0 \* P\_00;**

 **P\_01 -= K\_0 \* P\_01;**

 **P\_10 -= K\_1 \* P\_00;**

 **P\_11 -= K\_1 \* P\_01;**

 **return x\_angle;**

 **}**