**CENG4480 Embedded System Development and Applications**

**Computer Science and Engineering Department**

**The Chinese University of Hong Kong**

**Laboratory 4: A self-balancing platform**

October, 2015

**Introduction**

In this exercise you will learn how to develop a self-balancing platform and use the accelerometer as a 2-axis tilt sensor. The theory is described in [1][2]. The procedures of the development are as follows:

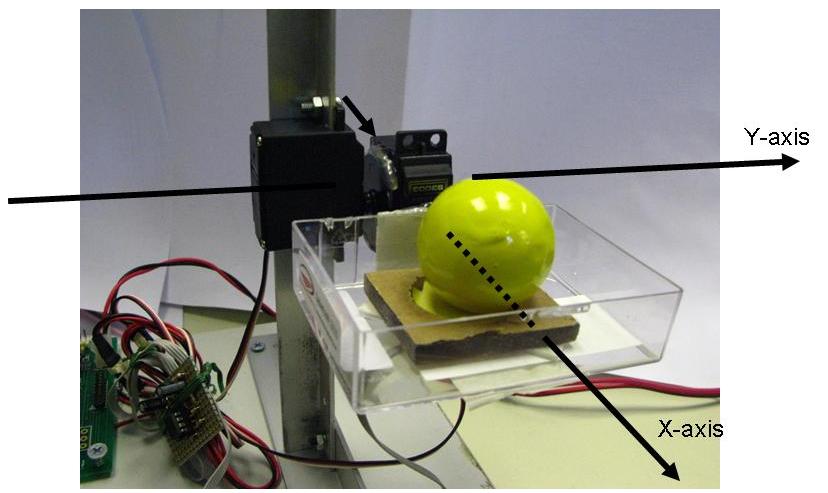
- Build a 2 channels DC amplifier for signal conditioning of the 2-axis accelerometer inputs.

- Implement a PID controller on the ARM board.

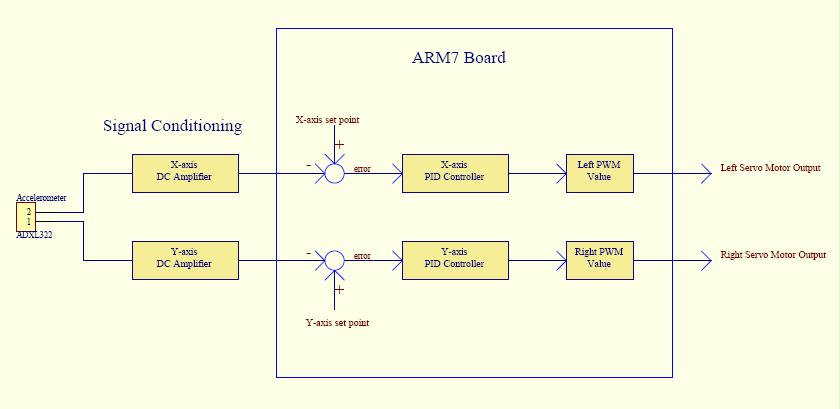
- Fine tune the PID controller constants to reach its optimal state.

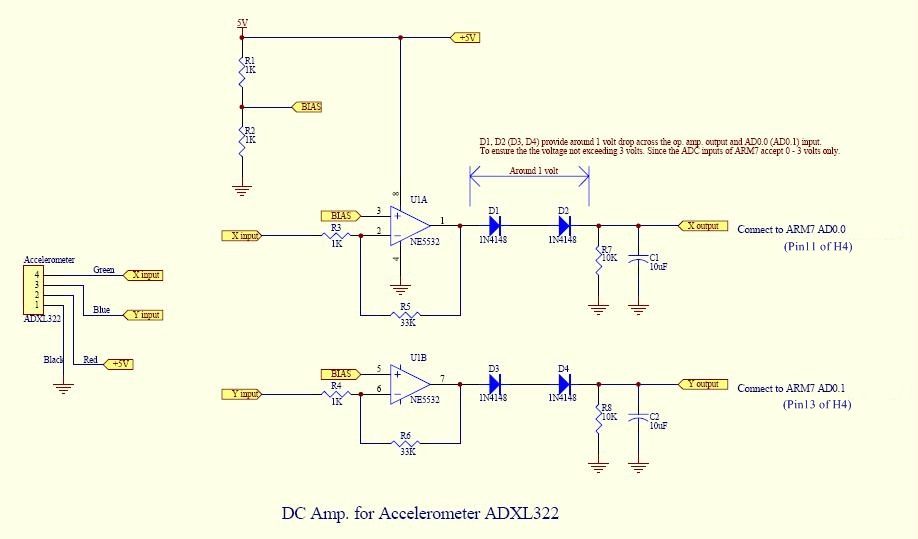
- Construct a self-balancing platform by assembling the accelerometer, ARM board and the servos mechanism together.

The following is the block diagram of the self-balancing platform and the 2 channels DC amplifier circuit diagram. A demo video can be found at [3].



**Fig. 1a. The experimental setup: the self-balancing platform holding a ball can be rotated in 2 axes**



**Fig. 1b.Block Diagram of the Self-balancing platform**

**Fig. 2 Two channels DC amplifier circuit**

**Objectives**

* To lean how to interface a direct current (DC) output sensor to a microcontroller
* To learn how to implement a [Proportional–Integral–Derivative PID](http://en.wikipedia.org/wiki/PID_controller) feed back control system.

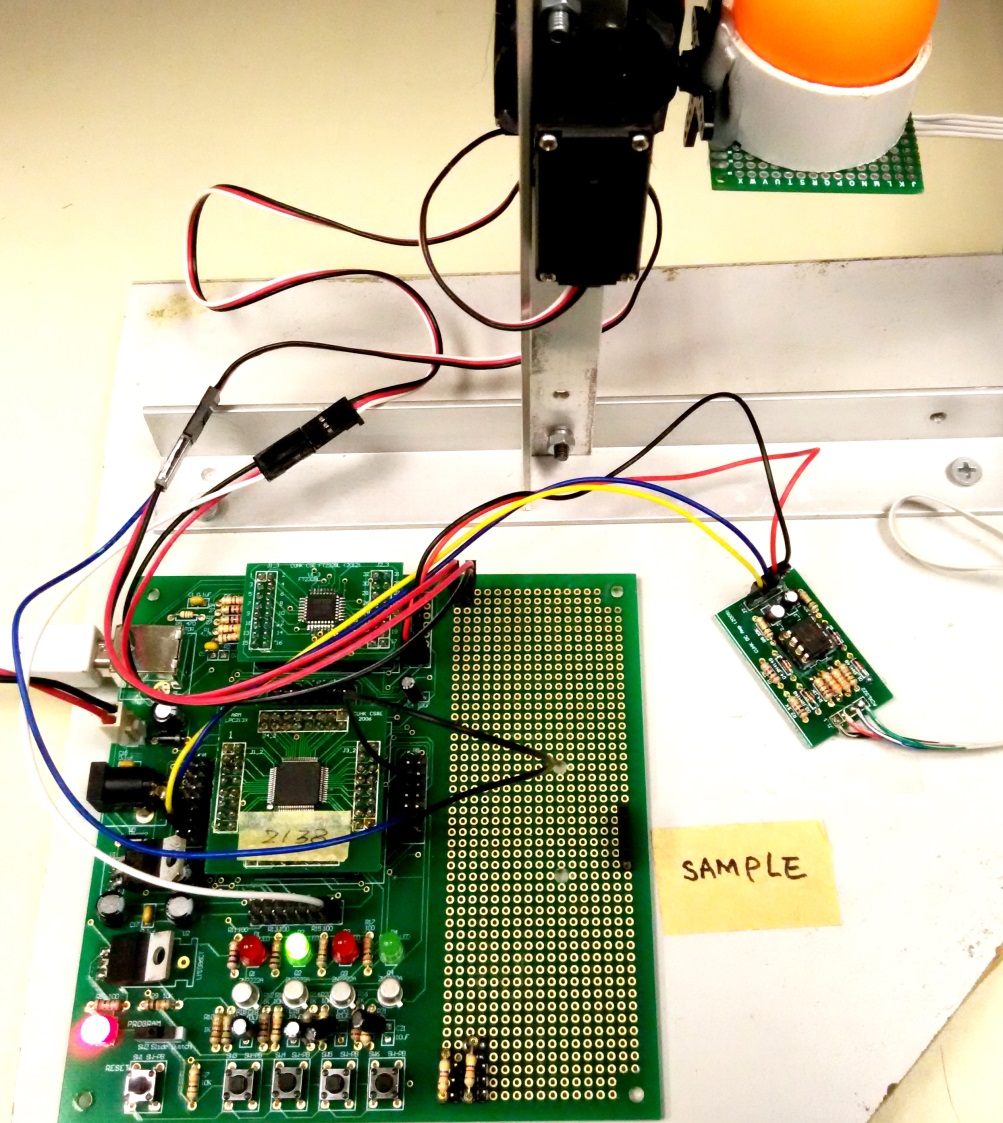
Procedures and what to submit:

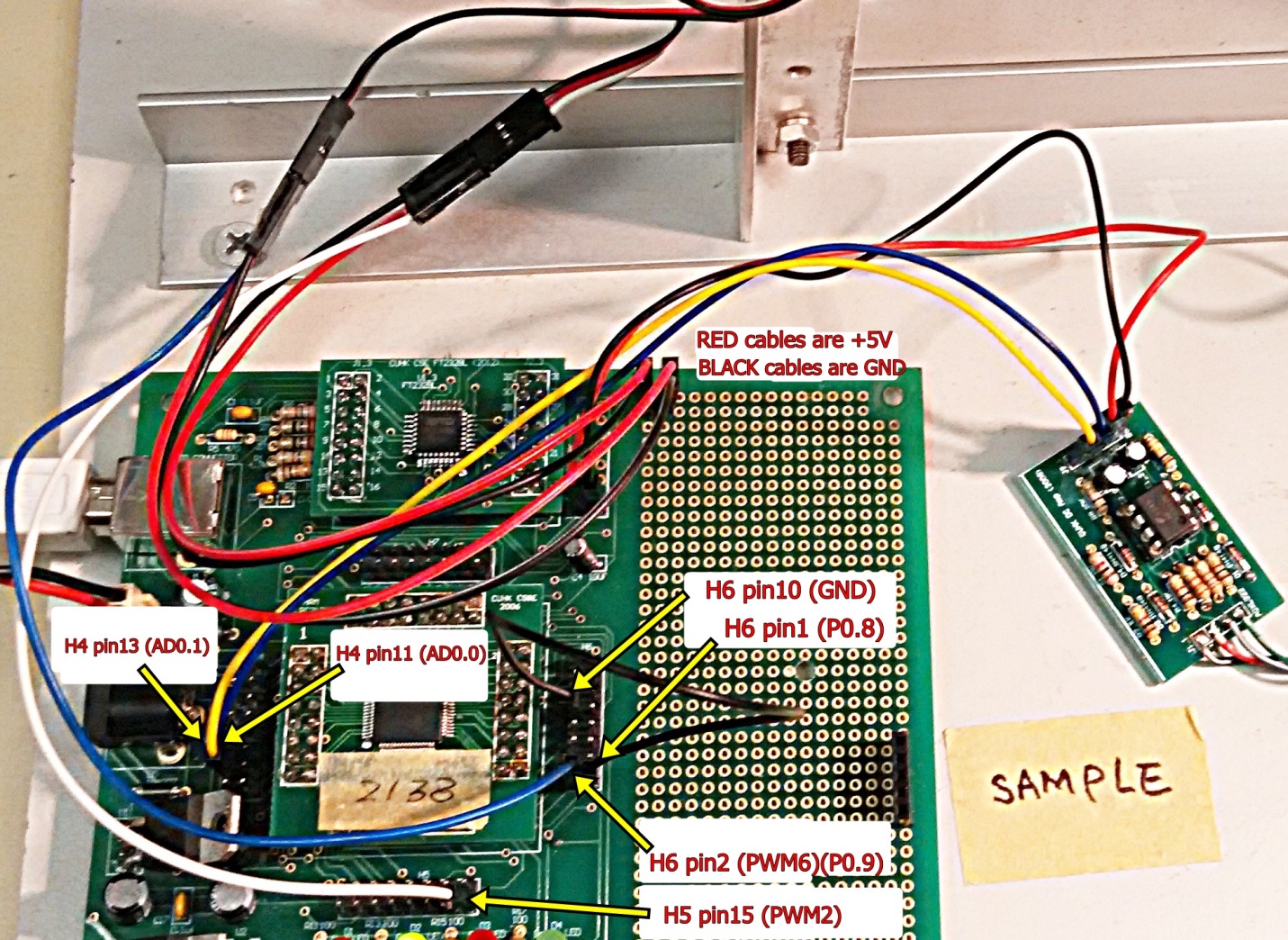
Follow the procedures of each experiment. Submit a lab report sheet with your name and student ID to the tutor after the lab. The lab report sheet should have the measurements or plots of your experiments, and answers of the questions asked in this lab manual. You may prepare the report using a computer document and use a camera to capture the waveforms and insert them in your report.

Experimental procedures

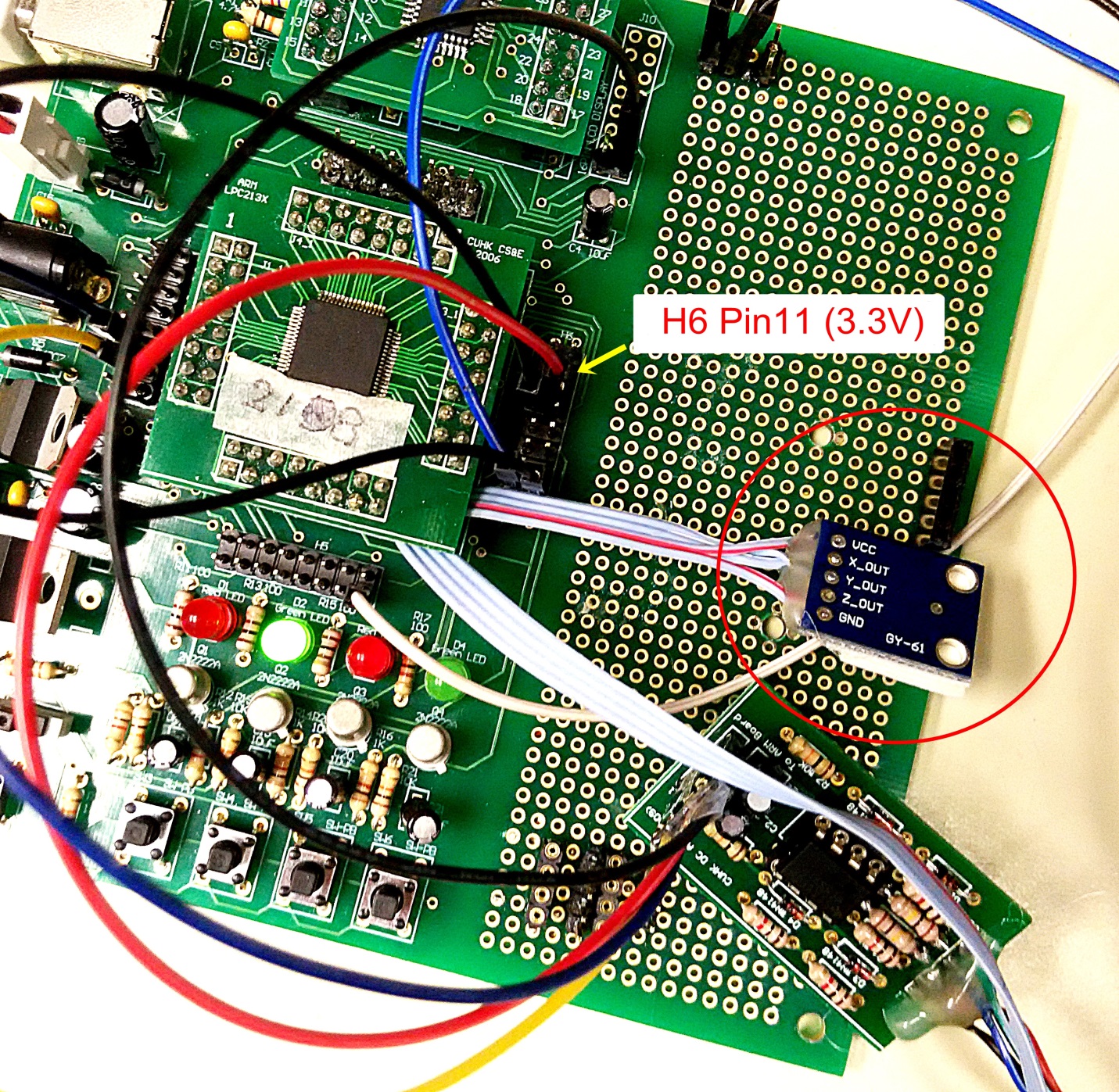
1. **Two channels DC amplifier circuit building**

* On the PCB board provided, connect the components by soldering them on the boards to build the two-channel DC amplifier circuit (as shown in Fig. 2).
* Connect the accelerometer to the amplifier inputs and connect the amplifier outputs to the ARM board as shown in the figures below. (refer to the sample)
* Attach the accelerometer to the provided platform.
* Connect two servo motors of the platform to the ARM board as shown in the figures below. (refer to the sample)

****

****

For new accelerometer GY-61, connect VCC to 3.3V (H6 Pin11) as shown on the figure below:



1. **Measure and record the amplifier output vs the tilt angle, and plot the graph**

After finishing procedure 1,

* Connect the power supply and USB cable to the ARM board.
* Download the sample program
* **Connect H6 pin1 and H6 pin10** (connect p0.8 to GND).
* Open the Hyper terminal (57600 baud rate).
* Reset the ARM board to run the sample program.
* Manually adjust the platform to the horizontal position.
* Measure the Xoutput (H4 pin13) of the amplifier, record the voltage value (Hints: use digital volt meter DVM function on the oscilloscope to measure the output voltage is more accurate and convenient)
* Press any key and record the ADC reading of Xaxis on the Hyper Terminal.
* Measure the tilt angle of the platform along the x-axis (refer to Fig.3) and record the angle. Change the tilt angle and repeat the measurement.

1. **In your lab report:**
   1. Fill in the following table and plot the graph.

|  |  |
| --- | --- |
| y-axis  x-axis | acceler_sensor_of_inverted_platforms |

**Fig. 3 (a) Top View of the Accelerometer , (b) the sensor attached to the bottom of the platform**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **X-Tilt Angle**  **(Degree)** | **-50** | **-40** | **-30** | **-20** | **-10** | **0** | **10** | **20** | **30** | **40** | **50** |
| **X-axis Output**  **(Volts)** |  |  |  |  |  |  |  |  |  |  |  |
| **X-axis ADC reading** |  |  |  |  |  |  |  |  |  |  |  |

2.5

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2.0 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| Output Voltage (Volts)  1.5 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1.0 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0.5 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0.0 |  |  |  |  |  |  |  |  |  |  |  |  |

60

50

40

30

20

10

0

-10

-20

-40

-50

-30

Tilt Angle (Degree)

1. **Question1: *Is the output voltage changes linearly with the tilt angle?***
2. **Determine and set the reference point of X-axis and Y-axis**

From the measurement results of procedure 2, obtained the Xaxis ADC reading (X0) when the tilt angle of the X-axis of the platform is 0 degree. This value (X0) is used for the X-axis set point value of the PID controller (see Fig.1 Block diagram).

**- Use X0 value to set MIDL in the sample program.**

For example, if X0 = 14700 then define the MIDL in the sample program as follow:

**#define MIDL 14700** // The set point value of accelerometer X-axis

if X0 = 15200 then define the MIDL in the sample program as follow:

**#define MIDL 15200** // The set point value of accelerometer X-axis

**- Repeat the measurement for Y0 of the Y-axis and set the MIDR in the sample program.**

1. **Compile the sample program, download and run the program**

* Disconnect the jumper between H6 pin1 and H6 pin 10.
* Run the sample program and test the performance.

1. **The effect of sampling frequency**

The sampling frequency of the sample program is originally set to 500Hz.

**- Reduce the sampling frequency to 100Hz by changing the value of TMR0 to 138240 in sample program and recompile it. Then download and run it again.**

void init\_timer (void) {

T0PR = 0; // set prescaler to 0

**T0MR0 =138240;** // set interrupt interval to 1mS

// Pclk/500Hz = (11059200 x 5)/(4 x 100)

T0MCR = 3; // Interrupt and Reset on MR0

T0TCR = 1; // Timer0 Enable

VICVectAddr0 = (unsigned long)IRQ\_Exception; // set interrupt vector in 0

VICVectCntl0 = 0x20 | 4; // use it for Timer 0 Interrupt

VICIntEnable = 0x00000010; // Enable Timer0 Interrupt

}

***Question 2. What is the effect on the performance of the system when the sampling frequency is 100Hz?***

* **Increase the sampling frequency to 1000Hz by changing the value of TMR0 to 13824 in the sample program and recompile it. Then download and run it again.**

***Question 3. What is the effect on the performance of the system when the sampling frequency is 1000Hz? Explain your observation.***

1. **Overall tuning of the P, I D constants of the system.**

***Question 4. The deadband in sensor reading is +/-200. Estimate the deadband in degrees of our system. See [1] for the definition of deadband***

***Questions 5. You are required to move the platform without dropping the ball from -60 to 60 degrees against the X-axis as fast as you can. Tune the parameters P, I and D constants in the program so you can achieve the maximum of turns (oscillating the platform between -60 degrees to 60 degrees in the X-axis) in 30 seconds. Record the values of P,I and D used and demonstrate your results to the tutor.***

**END**

Refrences:

[1] <http://www.cse.cuhk.edu.hk/%7Ekhwong/ceg3480/lab2_self_bal_platform.ppt>

[2] <http://www.cse.cuhk.edu.hk/%7Ekhwong/ceg3480/PID_DC_motor_Control08.ppt>

[3] Youtube: A self balancing platform demo <http://www.youtube.com/watch?v=Lym2UxUh81Q>

**Appendix**

**accelerometer.c**

#include <lpc21xx.h>

extern void init\_timer(void);

#define PWM\_FREQ 276000 //set PWM frequency to 50 Hz

#define L\_DIR 0x00010000 //set p0.16 as left motor direction

#define L\_DIRinv 0x00020000 //set p0.17 as inverted left motor direction

#define R\_DIR 0x00040000 //set p0.18 as right motor direction

#define R\_DIRinv 0x00080000 //set p0.19 as inverted right motor direction

#define JUMPER 0x00000100 //set p0.8 as function selection jumper

#define NEWLINE sendchar(0x0a); sendchar(0x0d)

// Adjustable parameter /////////////////

#define INIT\_POSL 25000 // Left servo initial position

#define INIT\_POSR 18000 // Right servo initial position

#define maxaccu 200 // Maximum accumulate error

#define maxdiff 2000 // Maximum different between current and last error

#define MINOUTPUT 7000 // Minimum value of PWM output

#define MAXOUTPUT 35000 // Maximum value of PWM output

#define P 1 // Proportional constant of PID controller

#define I 1 // integral constant of PID controller

#define D 1 // Differential constant of PID controller

#define MIDL 15500 // The set point value of accelerometer X-axis

#define MIDR 15800 // The set point value of accelerometer Y-axis

// Global variables /////////////////////

long leftPWM,rightPWM; // PWM values

int tmpl,deltal,lastl,accul,diffl; // variables for left servo

int tmpr,deltar,lastr,accur,diffr; // variables for right servo

// Functions =======================================

void Init\_Serial\_A(void) {

U0LCR = 0x83; //8 bit length ,DLAB=1

U0DLL = 0x0F; //Pclk/(16\*baudrate)=(11059200 x 5)/(4 x 16 x 57600);

U0DLM = 0x00;

U0LCR = 0x03; //DLAB=0

}

char getchar(void) {

volatile char ch = '0';

while ((U0LSR&0x1)==0) // wait until receive a byte

;

ch = U0RBR; // receive character

return ch;

}

void sendchar(char ch) {

while( (U0LSR&0x40)==0 );

U0THR = ch; // Transmit next character

}

int print(char \*p) {

while(\*p!='\0') {

sendchar(\*p++);

}

return(0);

}

void putint(int count) {

sendchar('0' + count/10000);

sendchar('0' + (count/1000) % 10);

sendchar('0' + (count/100) % 10);

sendchar('0' + (count/10) % 10);

sendchar('0' + count % 10);

}

int read\_sensor(int channel)

{

int temp;

ADCR=0x1<<channel; //Set channel SEL field

ADCR|=0x1200200; //Start conversion now; ADC is operational;

//11 clocks/ 10 bits; Software conversion mode;

//CLKDIV = 2; ADclk = Pclk/(CLKDIV+1)

//ADclk = (11059200 x 5)/(4 x 3) = 4.608MHz

while(((temp=ADDR)&0x80000000)==0); //check DONE flag

temp>>=6; //ADDR p6-p15 = RESULT

temp&=0x3ff;//10 bits result

return (temp\*33); //return 0-3V value precision is 1024

}

//================================================

int main(void)

{

long tmpjp;

PINSEL0 = 0x00000005; // set p0.0 to TXD0, p0.1 to RXD0 and the rest to GPIO

PINSEL1 = 0x00400000; // set p0.27 to ad0.0 and the rest to GPIO

PINSEL1 |= 0x01000000; // set p0.28 to ad0.1

PINSEL0 |= 0x00008000; // set p0.7 to PWM2

PINSEL0 |= 0x00080000; // set p0.9 to PWM6

Init\_Serial\_A(); // Init COM port

NEWLINE;

print("================================================================"); NEWLINE;

print("\* \*"); NEWLINE;

print("\* CUHK Computer Science and Engineering Department \*"); NEWLINE;

print("\* LPC2138 ARM Board \*"); NEWLINE;

print("\* \*"); NEWLINE;

print("\* Auto balancing platform by using 2-axis accelerometer \*"); NEWLINE;

print("\* (8/2015) \*"); NEWLINE;

print("\* \*"); NEWLINE;

print("================================================================"); NEWLINE;

NEWLINE;

tmpjp = IO0PIN & JUMPER; // check function selection jumper

if(tmpjp==0) { // if jumper is set then print X, Y value

while(1) {

print("Xaxis = ");

putint(read\_sensor(0));

print(" ; Yaxis = ");

putint(read\_sensor(1));

NEWLINE;

NEWLINE;

print("Press any key to show X and Y value.");

NEWLINE;

NEWLINE;

getchar();

}

}

else { // else run self balancing demo

init\_timer(); // Init TIMER 0

PWMPCR=0x4000; // enable pwm6

PWMPCR|=0x0400; // enable pwm2

PWMMCR=0x0002;

PWMMR0 = PWM\_FREQ; //set PWM frequency to 50 Hz

//set robot to full speed

leftPWM=INIT\_POSL;

rightPWM=INIT\_POSR;

PWMMR2 = leftPWM; //set left motor PWM width to full speed

PWMMR6 = rightPWM; //set right motor PWM width to full speed

PWMLER = 0x45; //enable match 0,2,5 latch to effective

PWMTCR=0x09;

//set p0.16-p0.19 as output

IO0DIR|=L\_DIR;

IO0DIR|=L\_DIRinv;

IO0DIR|=R\_DIR;

IO0DIR|=R\_DIRinv;

// enable PWM outputs

IO0SET|=L\_DIR;

IO0CLR|=L\_DIRinv;

IO0SET|=R\_DIRinv;

IO0CLR|=R\_DIR;

// initialize variables

lastl=0;

accul=0;

diffl=0;

lastr=0;

accur=0;

diffr=0;

while(1) {

}

}

}

void \_\_irq IRQ\_Exception()

{

tmpl = read\_sensor(0); // read X-axis value

if (tmpl>=(MIDL+200)) { // if X-axis value >= setpoint plus 200

deltal = (tmpl - (MIDL+200))/200; // calculate the error and normalize it

diffl = deltal-lastl; // caculate the different between current and last error

if(diffl<maxdiff) { // ignore if the error different > max. difference

// this prevent the noise due to undesired movement of accelerometer

lastl = deltal; // save error as the last error

leftPWM = leftPWM - (P\*deltal - I\*accul + D\*diffl);// update the left PWM value by PID

if (leftPWM<MINOUTPUT) leftPWM = MINOUTPUT;// limit the PWM value to its minimum

if(accul<maxaccu) accul += deltal/200; // ensure the integral not exceed the maximum

PWMMR2=leftPWM; // set the left PWM output

PWMLER = 0x44; //enable match 2,6 latch to effective

}

}

else if (tmpl<=(MIDL-200)) { // if X-axis value <= setpoint plus 200

deltal = ((MIDL-200) - tmpl)/200; // calculate the error and normalize it

diffl = deltal- lastl; // caculate the different between current and last error

if(diffl<maxdiff) { // ignore if the error different > max. difference

// this prevent the noise due to undesired movement of accelerometer

lastl = deltal; // save error to the last error

leftPWM = leftPWM + P\*deltal + I\*accul + D\*diffl; // update the left PWM value by PID

if (leftPWM>MAXOUTPUT) leftPWM = MAXOUTPUT; // limit the PWM value to its maximum

if(accul>0) accul -= deltal/200; // ensure the integral not less than zero

PWMMR2=leftPWM; // set the left PWM output

PWMLER = 0x44; //enable match 2,6 latch to effective

}

}

////////////////////////////////////////////////////

tmpr = read\_sensor(1); // read Y-axis value

if (tmpr>=(MIDR+200)) { // if Y-axis value >= setpoint plus 200

deltar = (tmpr - (MIDR+200))/200; // calculate the error and normalize it

diffr = deltar-lastr; // caculate the different between current and last error

if(diffr<maxdiff) { // ignore if the error different > max. difference

// this prevent the noise due to undesired movement of accelerometer

lastr = deltar; // save error as the last error

rightPWM = rightPWM - (P\*deltar - I\*accur + D\*diffr); // update the right PWM value by PID

if (rightPWM<MINOUTPUT) rightPWM = MINOUTPUT; // limit the PWM value to its minimum

if(accur<maxaccu) accur += deltar/200; // ensure the integral not exceed the maximum

PWMMR6=rightPWM; // set the right PWM output

PWMLER = 0x44; //enable match 2,6 latch to effective

}

}

else if (tmpr<=(MIDR-200)) { // if Y-axis value <= setpoint plus 200

deltar = ((MIDR-200) - tmpr)/200; // calculate the error and normalize it

diffr = deltar- lastr; // caculate the different between current and last error

if(diffr<maxdiff) { // ignore if the error different > max. difference

// this prevent the noise due to undesired movement of accelerometer

lastr = deltar; // save error as the last error

rightPWM = rightPWM + P\*deltar + I\*accur + D\*diffr; // update the right PWM value by PID

if (rightPWM>MAXOUTPUT) rightPWM = MAXOUTPUT; // limit the PWM value to its maximum

if(accur>0) accur -= deltar/200; // ensure the integral not exceed the maximum

PWMMR6=rightPWM; // set the right PWM output

PWMLER = 0x44; //enable match 2,6 latch to effective

}

}

//////////////////////////////////////////////////////////

T0IR = 1; // Clear interrupt flag

VICVectAddr = 0; // Acknowledge Interrupt

}

/\* Setup the Timer Counter 0 Interrupt \*/

void init\_timer (void) {

T0PR = 0; // set prescaler to 0

T0MR0 = 27648; // set interrupt interval to 1mS

// Pclk/500Hz = (11059200 x 5)/(4 x 1000)

T0MCR = 3; // Interrupt and Reset on MR0

T0TCR = 1; // Timer0 Enable

VICVectAddr0 = (unsigned long)IRQ\_Exception; // set interrupt vector in 0

VICVectCntl0 = 0x20 | 4; // use it for Timer 0 Interrupt

VICIntEnable = 0x00000010; // Enable Timer0 Interrupt

}