CENG4480 Lab 2: Implementing a PID feedback control system

(CUHK CSE 2012)

1. **About this document**
   The objectives, overview and detailed content of the 2nd laboratory section of CENG4480 are listed in this document. Following this document you will be able to build a self-balancing platform with servo motors and accelerometer. Please read this document before coming to the lab.
2. **Objectives**
   a) Learn to control servo motors with MCU
   b) Learn to implement a Proportional-Integral-Derivative feedback control system
   c) Learn to implement multi-task in MCU
3. **Overview**
   a) System components
   Figure 1 is a diagram of negative feedback control loop. The self-balancing platform (illustrated in figure 2) we are going to build also contains the 3 components; they are the MCU, servo motors and accelerometer corresponding to the controller, system and sensor in figure 1.

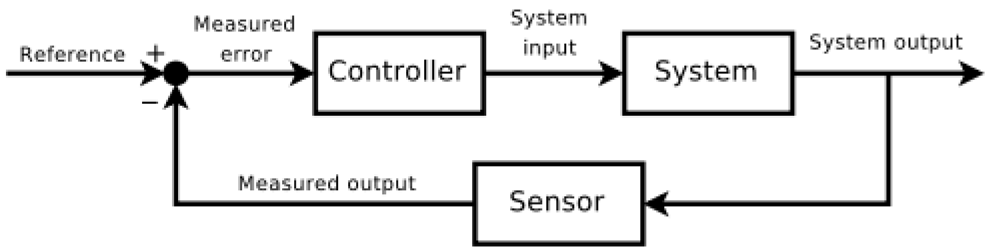

Figure 1: a diagram of negative feedback loop (from Wikipedia)

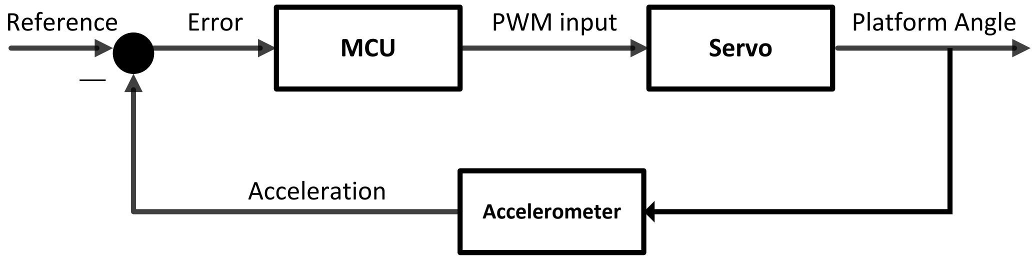

Figure 2: system components of a self-balancing platform

Figure 3 is the picture of the self-balancing system. There are two servo motors and a 2-D accelerometer ADXL322. The PID controller is implemented inside this mbed MCU.

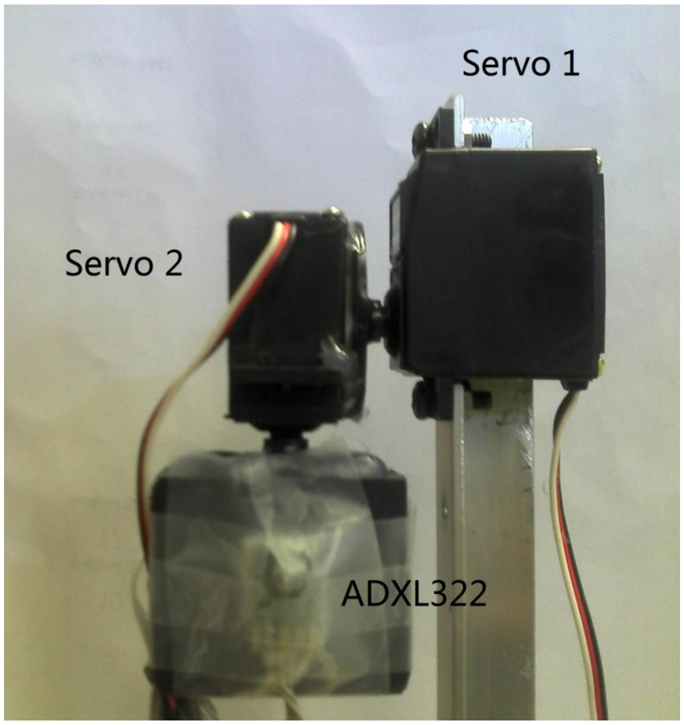

Figure 3: picture of self-balancing platform

b) PID feedback control algorithm

The PID controller calculation (algorithm) involves three separate constant parameters, and is accordingly sometimes called three-term control: the proportional, the integral and derivative values, denoted P, I, and D. Heuristically, these values can be interpreted in terms of time: P depends on the present error, I on the accumulation of past errors, and D is a prediction of future errors, based on current rate of change. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve, or the power supplied to a heating element.

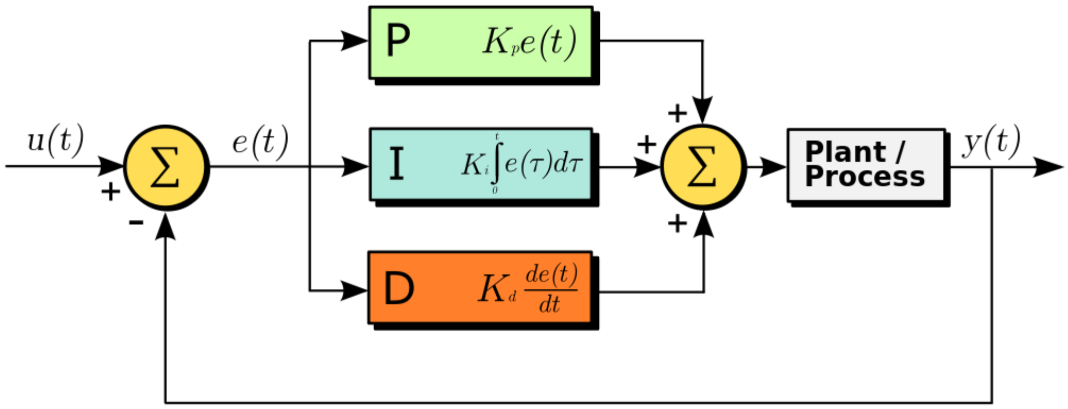

Figure 4: block diagram of PID controller

4. **Component introduction and interconnection**

   a) How to control a servo motor

   A servomotor is a rotary actuator that allows for precise control of angular position. It has 3 wires, the black and red ones should be connected to the negative and positive pole of a 5V DC power supply and the yellow line should be connected to the control signal, in our case it is a PWM output pin of mbed chip.

   To rotate the servo to a certain degree, you need a pulse width modulated (PWM) signal. A PWM signal is like the one in figure 4, the cycle suitable for this servo motor is 20ms, and the duration of positive signal in a cycle ranges from 0.5ms to 2.5ms corresponding to the 0 degree and 180 degree of the servo motor. As the signal I figure 5, the servo will rotate to 90 degree and hold there until the PWM signal changes.

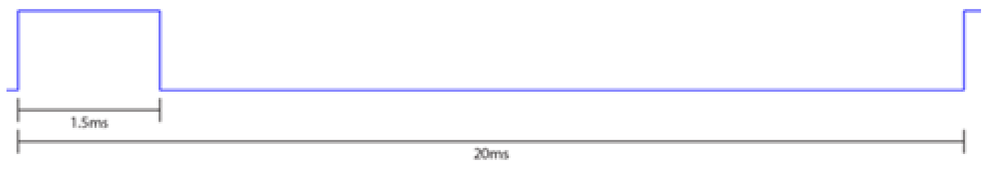

Figure 5: a 1.5ms pulse PWM signal in 20ms cycle

   b) How to generate PWM output in mbed

   Within the mbed chip, one can get PWM signal easily with very few programming work. Please refer to this for the information http://mbed.org/handbook/PwmOut . And it even provided servo control library to make the control of servo motor easier. Here's an example about how to use the library http://mbed.org/cookbook/Servo . With this servo library we can control the rotating angle of a servo motor with a floating number between 0 and 1.

5. **Implementation guide**

   a) Rotate the servo motor with keyboard input

   Follow the connection in this page http://mbed.org/cookbook/Servo . The control line of servo motor can be connected to pin 21 and the power lines be connected to the DC power supply in the lab. Remember to connect the ground pin of mbed chip and the negative pole of DC power supply, so that these two have common ground, or the system won't work. You can first test your connection with **Servo_Helloworld** program in this page. To utilize servo library you need to import it to your project first, it can be done by clicking this option

   » Import this program in the webpage.

   If your servo motor can rotate well then we can come to the second step, adjusting the PWM output with keyboard input. The 2nd example in this page http://mbed.org/handbook/SerialPC will give you some hints about how to complete this task. ***[20pts]***

b) Build the self-balancing system

i) Implementing the PID control algorithm

In this self-balancing system, there are two servo motors and a 2-D accelerometer which can measure the tilt of a platform. As in the last laboratory, we can read the voltage of accelerometer from the analog input pins. So basically when we find that certain dimension(X axis, or Y axis) is not horizontal, we can command the corresponding servo motor to rotate so that the platform can be horizontal and keep balanced.

Now let's come to the most important part of this lab work, the implementation of PID control algorithm. We know that the MCU device is a digital device, so we don't have a continuous error data but only digitalized data. So how we deal with those integration and differentiation calculation? In fact we just simply use plus and subtraction instead. To set up this system from ground, we can start testing of X axis first and if we succeed we can add another axis into the system. Below is a pseudo code of PID implementation:

```
X_error = X_axis – X_balance;  // This calculate the error of x axis

X_integration += X_error;       // This is the integration of errors, representing the past errors

X_diff = X_error – X_last_error // The differential of error

X_last_error = X_error;

PID = KP*X_error + KI*X_integration + KD*X_diff
```

After we have the PID control result, how do we combine it with the servo motor control parameter which is a value between 0 and 1? Should add this PID result to the position parameter P of servo motor or subtract it from the parameter? This depends on your own system connection, so you should observe your system for a moment and make your decision. But remember we are implementing a negative feedback control and our aim is to make each axis horizontal. Now let's come to some problems that you may meet during this lab.

Dead band: usually we don't need to set the position to an exact value, a small range of values around that ideal position are all acceptable. So when we detect that the system is in that range we can consider the system achieving the control target.

Setting limitation for the integration part: in my testing, I found that sometimes the integration part may have accumulated too much error that the other two parts cannot pull it back so that the integration part dominates the PID result. So when we are doing the integration we should set a limit for that part.

After this section you should be able to demo me a system that can achieve self-balancing at least on one axis.***[40pts]***

ii) Implement periodic multi-task within ticker

In this section we want you to use a ticker to implement multi periodic tasks. So the control of X servo and Y servo can be done in callback function of different ticker. Here's the link of how to use a tick, http://mbed.org/handbook/Ticker . In my system I set the cycle of the ticker 0.02 second. After this section you should be able to demo me a self-balancing system on both axis.***[40pts]***

6. Conclusion

After this experiment, you should have better understanding of PID control algorithm and have a better competence in programming with mbed.

7. Deadline of submission

The deadline for submission will be by 31st October 5pm. Marks will be deducted if late submission occurs. Your lab score will be deducted by 20% if the submission is one day late, deducted by 50% if two day late and if your submission is three day late, your score will be zero.