M.E. 530.420 Lab 9:  
Closed-Loop Digital Velocity Control  
Louis L. Whitcomb*  
Department of Mechanical Engineering  
G.W.C. Whiting School of Engineering  
The Johns Hopkins University  
This lab is due 5:30PM Tuesday November 15, 2011 at 115 Hackerman Hall  
Revision 2: November 9, 2011  
*This laboratory © Louis L. Whitcomb

Apparatus:  MaEvArM Manual  
Text Chapter 4.9  
LS7366 Quadrature decoder board  
OPA544 op-amp current-amplifier board  
Spec sheets from previous labs for LF-411, OPA544 Power OP-Amp, SG- SG-531P 5.0 mHz TTL clock, LS7366 Quadrature Decoder/Counter chip, Pololu Gear motor/Tachometer (part #1442)  
Note:  Do Pre-Lab Questions 1-6 should be done in advance of the lab if possible.

1 PRE-LAB Questions

1) Pre-Lab: Consider the following idealized motor torque model for motor torque, $t_m$, as a function of motor current $i_m$ and the motor torque constant $k_t>0$

$$\tau_m = k_t \cdot i_m$$

and the following idealized motor dynamical model for the time-derivative of shaft velocity, $\omega$, as a function of the motor-and-shaft moment of inertia, $I_m>0$, and motor torque, $t_m$.

$$I_m \dot{\omega} = \tau_m,$$  

motor dynamics model (no friction)

$$\dot{\omega} = I_m^{-1} \cdot \tau_m$$

and the derivative-feedback control law for motor current, $i_m$, as a function of actual shaft velocity, $\omega$, and the desired shaft velocity, $\omega_d$, (constant) and the derivative feedback gain, $k_P$, (a constant)

$$i_m = -k_P \cdot (\omega - \omega_d).$$

closed-loop control law

Show analytically that solutions to the closed-loop system have the property that the actual shaft velocity asymptotically approaches the desired shaft velocity, i.e.

$$\lim_{t \to \infty} (\omega - \omega_d) = 0.$$

steady-state solution – i.e. when $\dot{\omega} = 0$.

2) Pre-Lab: Consider the case where the motor has linear (dynamic) friction and the motor dynamic model is

$$I_m \dot{\omega} = \tau_m - k_D \omega,$$  

motor dynamics model (with linear friction)

where $k_D>0$ is the linear friction parameter.
In this case, if we employ the closed-loop control law given in Question 1, what is the steady state value of $\Delta \omega = (\omega - \omega_d)$?

3) **Pre-Lab:** Consider the case where the motor has coulomb (static) friction and the motor dynamic model is

$$I_m \dot{\omega} = \tau_m - k_S \text{sgn}(\omega), \quad \text{motor dynamics model (with static friction)}$$

where $k_S > 0$ is the coulomb friction parameter and

$$\text{sgn}(\omega) = \begin{cases} +1 & \text{if } \omega > 0 \\ 0 & \text{if } \omega = 0 \\ -1 & \text{if } \omega < 0 \end{cases}$$

In this case, if we employ the closed-loop control law given in Question 1, what is the steady state value of $\Delta \omega = (\omega - \omega_d)$?

4) **Pre-Lab:** From your previous DAC lab, give the relationship between MAX517 DAC command counts (0-255 decimal, $00$-$FF$ hexadecimal) and $V_{DAC}$.

5) **Pre-Lab:** Analyze the circuit in Figure 2 to derive $i_M$ as a function of $V_{CMD}$. Show your work.

6) **Pre-Lab:** Use the results of Questions 2, and 3 to derive $i_M$ as a function of DAC command counts for the above circuit.

7) **Figure 1: Block Diagram of Closed-Loop Velocity Control of the DC Motor**

2  **Laboratory**

7) Construct and test the DAC circuit shown on THE DEFT SIDE ONLLY of Figure 2 comprised of a DAC517 and a LF411. **Do not connect the output of the LF411 to the OPA544 board yet. Do not use the OPA544 board or the DC motor yet.**

- The DAC power (+5V) and ground (0V) should be sourced directly from the MaEvArM.
- **Remember to use a 1K Ohm resistor when connecting SDA to D1.**
- Remember to include the pull-up resistors on SDA and SCK.
  i) **HI:** Write a small program with loop which sends DAC commands 0x00 through 0xFF in sequence. You may have such a program from a previous lab. Hand in a copy of your program
  ii) **HI:** Connect CH1 of your scope to Pin 1 (OUT0) of the MAX517 DAC.
  iii) **HI:** Connect CH2 of your scope to Pin 2 (OUTPUT) of the LF411.
  iv) **HI:** Use your program to verify that:
      - Your DAC produces (at terminal $V_{DAC}$) voltage levels from 0.0V rising to +5.0V.
      - The output of on Pin 6 of the LF411 should vary from +2.5V falling to -2.5V

ii) **HI:** Hand in an annotated scope plot.
Figure 2: A Circuit for Analog Current Control of a DC Motor from the Basic Stamp
8) Now construct and test the entire circuit shown in Figure 2. Use a current amplifier board with a OPA544 power op-amp.

- Configure your power supplies for +/- 18V SERIES operation, and connect the 0V terminal to earth ground. Ask your TA if you are unclear about this point.
- Modify your DAC test program to command a triangle (sawtooth) waveform with voltage range of +2.0V to +3.0V.
- Verify that your DAC produces (at terminal $V_{\text{CMD}}$) a voltage levels from -0.5V to +0.5V (i.e. ½ Volt). Print a scope plot.
- Connect Oscilloscope probes in the following manner:
  i) Connect CH 1 of your scope to $V_{\text{CMD}}$, $V_{\text{IN}}$ (i.e. the non-inverting input of the OPA544 op-am).
  ii) Connect CH 2 of your scope to the inverting input to the OPA544 op-amp.
  iii) Connect CH 3 of your scope to the OUT+ terminal of the op-amp board.
  iv) Connect CH 4 of your scope to the OUT- terminal of the op-amp board.
  v) **HI:** Print and annotate a scope plot

b) **HI:** Now immobilize the motor output shaft motor shaft (i.e. the shaft connected to the encoder wheel) to observe the effect of back-emf on the system behavior. Print and annotate another plot. What is different between these two plots? Explain what is going on here.

9) Set up and connect the quadrature decoder board to the MaEvArM. See Figure 3 for a circuit diagram layout. This is the same circuit that you constructed in a previous lab. Remember to power your encoder board directly from the MaEvArM board’s +5V and GND.
Figure 3: Quadrature Decoder Interface Circuit Diagram Sketch
10) Implement a software program on your MaEvArM to (approximately) implement the derivative-feedback proportional velocity control law

\[ i_m = -k_p \cdot (\omega - \omega_d) \]

Does this controller behave as you expected? Why or why not?

**HI:** Hand in your program and a sample terminal screen of the actual program.

- Experiment with different values of the desired (constant) shaft velocity, \( \omega_d \). In particular, try \( \omega_d = 100, 200, 300 \) radians/second counts/cycle. Note and comment on the degree to which your controller achieves the desired shaft velocity.
- Experiment with different values of the derivative feedback gain, \( k_p \), (a constant). What happens if \( k_p \) is very small? What happens if \( k_p \) is very large?

My sample program to implement this velocity feedback controller is the following:

```c
// Simple Velocity Control Program
// 2011-10-30 LLW & YK
#include "maevarmUSB.h"
#include "maevarmGEN.h"
#include "maevarmTWI.h"
#include "maevarmSPI.h"

// declare constant PI
#define PI 3.1415926535897932384626433832795

// ----------------------------------------------------
// function to initialize the quadrature decoder
// ----------------------------------------------------
void initialize_quad()
{
    //Initialize SPI as master
    //Leave SPI_CLK to 1/4 of CPU clock
    //Leave data order at MSB first
    SPI_MasterInit();

    // Write to MRO
    SS_low(); //bring SS line low to start
    SPI_send(0x88); //0x88 (10001000) = write to Mode Register 0
    SPI_send(0xC3); //0xC3 (11000011) = 4xQuad, freerun,
    //disable index, async index, fclk/2
    SS_high(); //pull SS line high to end

    // Write to MRI
    SS_low(); //bring SS line low to start
    SPI_send(0x90); //0x90 (10010000) = write to Mode Register 1
    SPI_send(0x02); //0x02 (00000010) = two-byte count,

    //enable, no flags
    SS_high(); //bring SS line high to end

    return;
}
```
// function to read from quadrature decoder
unsigned int read_quad()
{
    // declare working variables
    unsigned char MSB;
    unsigned char LSB;
    unsigned int counter_value;

    SS_low(); // bring SS line low to start

    SPI_send(0x60); // 0x60 (011000000) = read from counter

    MSB = SPI_read(); // read first byte
    LSB = SPI_read(); // read second byte

    SS_high(); // pull SS line high to end

    // assemble a 16 bit unsigned int from the two bytes
    counter_value = (MSB << 8) + LSB;

    return counter_value;
}

// function to write to DAC
void write_DAC(char value)
{
    TWI_start(); // generate start condition
    TWI_tx_address(0x58); // send address byte
    TWI_tx_data(0x00); // send command byte
    TWI_tx_data(value); // send DAC value
    TWI_stop(); // generate stop condition
    return;
}
int main()
{
    // declare and initialize local variables.
    // quad_delta should be signed
    unsigned int quad_count = 0;
    unsigned int quad_last = 0;
    int quad_delta = 0;

    long int pos_count = 0;
    double pos_radians = 0.0;
    double pos_radians_last;
    double vel_rps;
    double vel_rps_command;
    double vel_rps_error;

    int kount = 0;

    // set command velocity in radians/sec
    vel_rps_command = +200.0;

    int DAC_value_unclipped;
    unsigned char DAC_value; //output value to DAC

    //initialize usb
    usb_initialize();

    // initialize the SPI interface and the quadrature decoder chip
    initialize_quad();
    // Initialize TWI/I2C for communication with DAC
    TWI_enable(); //enable TWI module
    TWI_bitrate(10); //set the value of bitrate register to 10
// main loop ----------------------------------------
while(1)
{
   //update quad_last
   quad_last = quad_count;
   //read and store value into quad_count
   quad_count = read_quad();
   //calculate quad_delta
   quad_delta = quad_count - quad_last;
   // keep 32 bit absolute position
   pos_count = pos_count + quad_delta;
   // convert counts to position in radians
   pos_radians_last = pos_radians;
   pos_radians = (pos_count / 48.0) * 2.0 * PI;
   // compute angular velocity
   vel_rps = (pos_radians - pos_radians_last) / 0.01;
   vel_rps_error = vel_rps - vel_rps_command;
   // compute the dac command
   // define Kp in units of (DAC counts)/(rad/sec)
   #define Kp 0.25
   DAC_value_unclipped = 127 + (vel_rps_error * Kp);
   // do not command more 127 +/- 100 DAC counts
   #define MAX_DAC (127+100)
   #define MIN_DAC (127-100)
   if(DAC_value_unclipped > MAX_DAC)
      DAC_value = MAX_DAC;
   else if(DAC_value_unclipped < MIN_DAC)
      DAC_value = MIN_DAC;
   else
      DAC_value = DAC_value_unclipped;
   write_DAC(DAC_value);
   //print quad_count (unsigned) and quad_delta (signed) values
   usb_printf("count=%04X delta=%+6d vel=%+10.3f (r/s) vel_ref=%+10.3f (r/s) vel_err=%+10.3f (r/s) DAC=%5d
",
         quad_count, quad_delta, vel_rps, vel_rps_command, vel_rps_error, DAC_value);
   //delay 10 ms
   delay_ms(10);
   // uncomment the following section of code to change the velocity command direction every 2 seconds
   kount = kount +1;
   //if(kount > 200)
   // {
   //  kount = 0;
   //  vel_rps_command = - vel_rps_command;
   // }
}
}
11) Examine the SS signal with your scope to determine the period and frequency of your control program.
12) Experiment with changing the SIGN of the velocity command by uncommenting the last few lines of the program. **HI:** What do you observe?

- Be inquisitive.
- Ask questions.
- Be creative.
- Discover.
- Remember to show your work.
- Typed or hand-written lab reports are OK. Messy or ambiguous lab reports will be rejected.
Cover Sheet for 530.420 Lab #9
Closed-Loop Digital Velocity Control

Use this cover sheet for your lab writeup.

My Secret Code: _______
Fill in the secret code which was provided to you on your graded lab#1.

My Lab Station: _______

My Partner’s Secret Code: _______

Lab Station Clean!
TA’s Signature & Date:

Your TA will sign here after you have finished your lab, cleaned up your lab station to perfection, and shown your lab station to your TA.