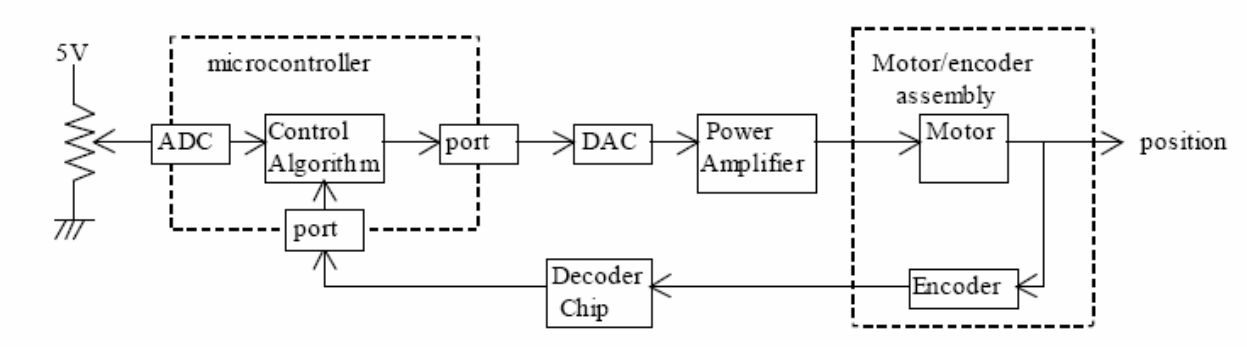
University of California, Berkeley
EE128, Fall 2005

Lab 7 – A Microcontroller Based Position/Speed Controller

Introduction

In this lab, we will develop and evaluate a microcontroller based position/speed control system. The system consists of a Atmel mircrocontroller, a Digital-to-Analog Converter, a power amplifier, a motor, an encoder, an decoder chip, and a Analog-to-Digital converter. The following block diagram shows the interconnection between these components.



Component Descriptions

As indicated in the above block diagram, an on-chip ADC channel is used to convert the analog position commend input to digital. The controller output is sent to a 12-bit DAC chip (Analog Device, AD7247A). The output of the DAC spans –5v (all zeros) to +5v (all 1's). The power amplifier is the same power amplifier used in the magnetic ball levitation experiment. The motor is a DC brush type motor and the optical encoder is by HP. The following paragraph is provided by the retailer of the motor/encoder assembly.

> Pancake Motor, Japan Servo, #DF10BE22-01, 127K9720. Permanent magnet. Reversible. Ball bearing. Continuous duty. Rated 24 VDC. No-load speed 3200 rpm @ 0.180 amp. 24 VDC. 1700 rpm 3.0 amps @ 30 oz-in load. Hewlett Packard 9100 series optical encoder on rear with 500 counts per revolution Two channel quadrature output which is TTL compatible. Encoder operates on 5 VDC @ 40 mA. max Motor dimensions: 4 3.4" max. dia. The decoder chip is HCTL-2016 by Agilent Technologies.
>
> The decoder chip is an up/down counter that counts the quadrature signals from the encoder. The output of the decoder chip is an 8-bit binary number. The counter can be reset by the microcontroller. The following figure is the schematics of the interface board.

Project Goal and Grading

The goal of the project is to close the position and speed control loop and to achieve the highest performance possible. The performance of your controller will be evaluated by the following criteria:

· Steady state error
· Frequency response
· Step transient response, i.e., overshoot, damping characteristics, settling time (both small step and
large step)
· Disturbance rejection

Your project will be graded base on the following factors:

· Controller performance (50%)
· Modeling and analysis of the system (30%)
· Clarity and completeness of the report. (20%)

The list of items you need to do is:

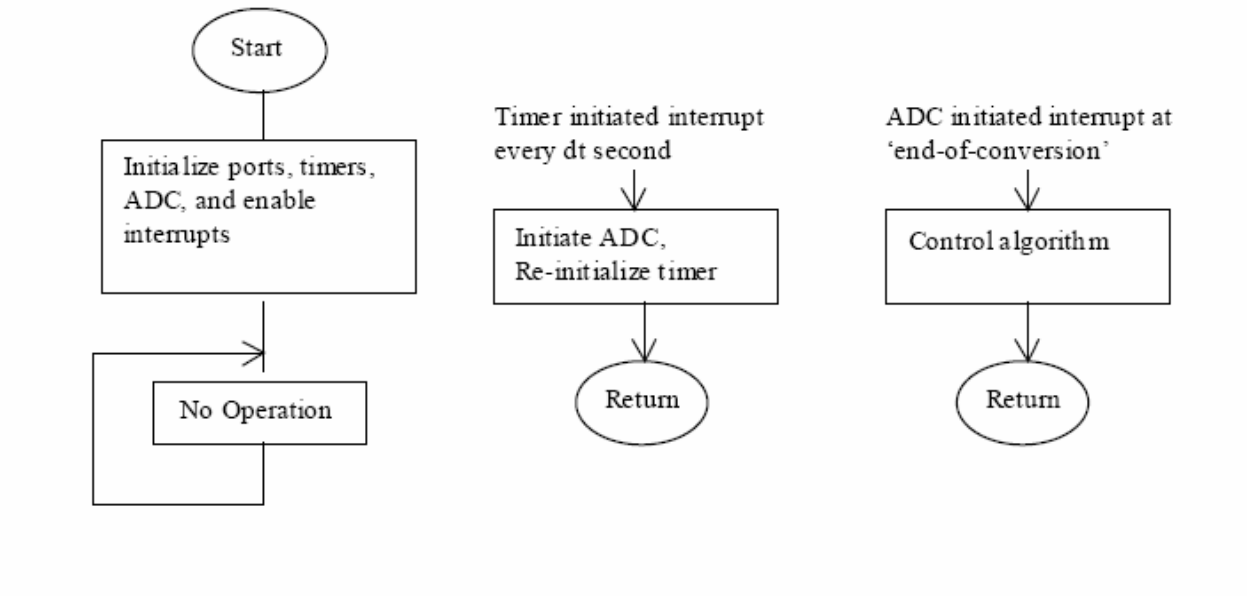
- Determine the Moment of Inertia
- Determine the Transfer function of the System + Controller
    - PID Controller (Hint: The two below are the same but with certain coefficients zero)
    - PD Controller
    - PI Controller
- Calculate coefficients for the three controllers
- Implement controllers
- Measure transient and steady-state characteristics
- Comment about what sampling frequency we should use
- Compare the experimental characteristics with the theoretical characteristics

A Sample Control Program

The following is a working sampling control program. This program closed the control loop with a simple PD controller. You can use this program as a template for your program. The logic flow of the program is shown in the flow chart below.

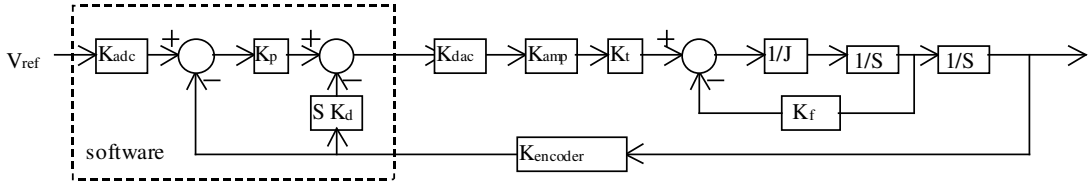
```
#include<mega16.h>

int abc=0,Cnt,LowCnt, HighCnt, adc_data, Output, speed, OldCnt;
int timer = 178;
float F_Cnt,position,F_speed,control,old_position, ref;

// This is a timer 0 initiated ISR.
interrupt [TIM0_OVF] void timer0_isr(void)
{
        ADCSRA=ADCSRA|0x40; // start ADC conversion
        TCNT0 = timer; // re-initialize counter
}

// This ISR is initiated by ADC at each end-of-conversion
interrupt [ADC_INT] void adc_isr(void)
{
        adc_data=-(ADCW<<1); //Get the Analog input
        PORTA.7=0; // Enable the output of the decoder chip & stop updating
        PORTA.6=0; //select high byte
        #asm("nop") // wait for the data
        #asm("nop") // wait for the data
        HighCnt=PIND; // Get the higher 8-bit count
        PORTA.6=1; //select low byte
        #asm("nop") // wait for the data
        #asm("nop") // wait for the data
        LowCnt=PIND; // Get the lower 8-bit count
        PORTA.7=1; // Output disable -> resume updating
        OldCnt=Cnt; // Save the old count
        Cnt= (HighCnt << 8)|LowCnt; // Combine the high and low counts
        F_Cnt = (float) Cnt;
        old_position = position;
        position=F_Cnt/318.3;
        F_speed = (position-old_position)/0.010;

        // STEP RESPONSE
        /*
        if (abc==100)
        {
                if (ref)
                {
                        ref = 0;
                }
                else
                {
                        ref = 1;
                }

                abc=0;
        }
        abc++;
        */

        // CONTROL LAW
        //-------------------------------------------------------
        control =  0.01*F_speed + 0.5*position + ref;
        //-------------------------------------------------------

        control = control/0.00244 + 2048;

        // CLIPPING/SATURATION CODE
        if (control < 0)
```

```
	{
		Output = 0;
	}
	else if (control > (2048+2047))
	{
		Output = (2048+2047);
	}
	else
	{
		Output = (int) control;
	}

	PORTB=Output; //send the low control byte to DAC
	PORTC=(Output>>8)|0xF0; //send the higher control byte
	PORTC.5=0; // strobe the data into DAC (falling edge)
	PORTC.5=1; // strobe the data into DAC (raising edge);
	//Output = (int) F_Cnt;
	Output = position/0.00244 + 2048;
	PORTB=Output; //send the low control byte to DAC
	PORTC=(Output>>8)|0xF0; //send the higher control byte
	PORTC.6=0; // strobe the data into DAC (falling edge)
	PORTC.6=1; // strobe the data into DAC (raising edge);
}

void main(void)
{
	DDRB = 0xff; // Port D all output (DAC lower 8-bit data)
	DDRC = 0xff; // Port C all output (DAC data and control)
	DDRA = 0xF0; // Port A upper nibble output (decoder control)
	DDRD = 0x0; // Port D all input (Decoder chip)
	abc=0;
	TCCR0 = 0x05; // divide by N pre-scalar (This and the next instruction
	// (the number 0x5 & 130) determine the sampling frequency).
	// you need to decide the best numbers to use.
	TCNT0=timer; // initialize timer 2
	TIMSK = 0x01; // unmask timer 2
	TIFR = 0x01; // do something to the interrupt flag reg.
	ADMUX = 0; // select analog channel 0
	ADCSRA = 0xCF; // ADC on, divide by 64, enable int. and start cover,
	#asm("sei") ; // Enable global interrupt
	while(1);
} // dummy loop
```



System Modeling

The following figure is block diagram of the system with a PD controller.



The value of the parameters are:

Kdac = 5/2048 = 0.00244 ; 2048 gives the maximum voltage output (5v).

Kamp = 1 (A/V) ;1 Amp per input voltage

Kencoer = 2000/2π=318.3 ; 2000 counts per revolution

Kadc = 1024/5=204.8 ; 5v gives 1024 (10 bit adc)

The toque constant Kt and the friction coefficient Kf can be estimated by the data (below) provided by the vender.

Applied voltage	Current	Speed	Load torque
24	0.18A	3200rpm (335.1 rad/s)	0
24	3A	1700rpm (178 rad/s)	30 oz-in (0.212 N-m)

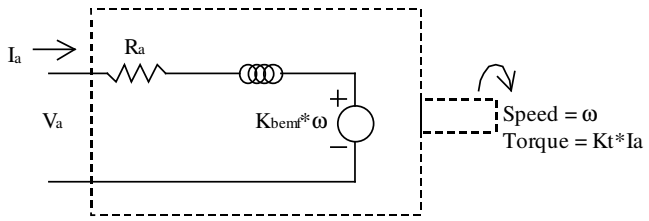
(Use unit on-line unit conversion at http://www.onlineconversion.com/torque.htm)

There are two ways to find the toque constants:

(1) Direct calculation.

Kt=0.212 Nm / 3 Amp = 0.07 Nm/Amp

(2) Use the fact that the numerical value of Kt is the same as the numerical value of the back emf constant. This is based on the conversation of power. Consider the motor equivalent circuit below:



The mechanical output power is Tourqe*speed = Kt*Ia*ω. The electrical input power is Kbemf*Ia*ω.

By equating these two terms, we get Kt*Ia*ω= Kbemf*Ia*ω, or Kt=Kbemf.

From the above table, the back emf constant (Ke) is

Kbemf = (24V-(10ohms)*(0.18A))/335.1rad/s = 0.066V/(rad/s)

where Ra=10 ohms obtained by a direct measurement of the armature winding. Base on this analysis, the toque constant is 0.066 Nm/A which is close to the 0.07 Nm/A obtained from using direct calculation. For the lab experiment, we will use the averaged value 0.068 Nm/A

The friction coefficient can now be estimated base on Kt. The torque generated by the no-load current (0.18Amp) is in fact the torque that is necessary to cancel the friction torque. If we assume the friction is linearly proportional to speed (viscous friction), the coefficient is then,

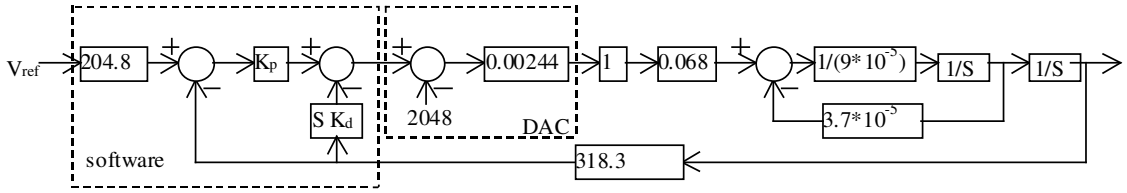
$$Kf= 0.068*0.18/335.1 = 3.7 \times 10^{-5} \text{ Nm/(rad/s)}$$

Note that the effect of the actual friction will be higher if the motor is operating at low speed and in a stop-and-go mode as in the position control mode.

The motor inertia J can be experimentally determined or by estimation. The rotor of the motor is estimated to have a weight of 0.2kg and a radius of 3cm. The inertia of a cylinder of this weight and size is

$$J=0.5*M*R^2= 0.5*0.2*0.03^2 = 9 \times 10^{-5} \text{ (kg*m}^2\text{)}$$

The following block diagram includes numerical values of all the system parameter. Note that the 2048 offset is due to the offset of the DAC (0=-5v, 2048=0v, 4096=5v).



Controller Design

Base on the above block diagram, Kp and Kd can be determined from the desired pole locations (i.e., bandwidth, damping characteristics). Note that if the closed loop poles are placed to far to the 'left', both the DAC and power amplifier may saturate. Also it is important to check the magnitude of all intermediate variable so that they stay within the range of representation of the data type (int, long, etc.)

Hint:

Avoid using floating point variable. Floating point operation slows down the program execution drastically.

Use the unused DAC channel for debugging purpose. For example, output the position, error, and/or speed to that channel so it can be observed with a scope.