Colorado School of Mines
Department of Mechanical Engineering
MEGN 540: Mechatronics

Laboratory 3: Pulse Width Modulation and DC Motors Version 2.0

Objective: In this lab, you will familiarize yourself with the Hansen 116-1 Series DC brushed motor and using pulse width modulation to drive it.

Equipment:

- Haptic Paddle
- Atmel Studio Software
- Hansen 116-1 Series DC Motor Datasheet
- AVRISP mkII programmer
- Computer (with Windows OS)
- Matlab
- ATmega328p Datasheet

Background:

DC brushed motors are the most widely used actuators in mechatronics. A DC motor is effectively a torque transducer that converts electrical energy into mechanical energy. In electrical circuits, DC motors are often modeled as a voltage source (back-EMF), an inductor and a resistor in series. The electrical component of the motor that is modeled by this electrical circuit is known as the armature. Current that flows through the armature from the positive to negative lead will generate a torque on the motor rotor that acts in the positive direction of rotor spin, and vice versa for current that flows from the negative to positive lead. This torque is expressed as:

$$T = K_t i_{arm}$$

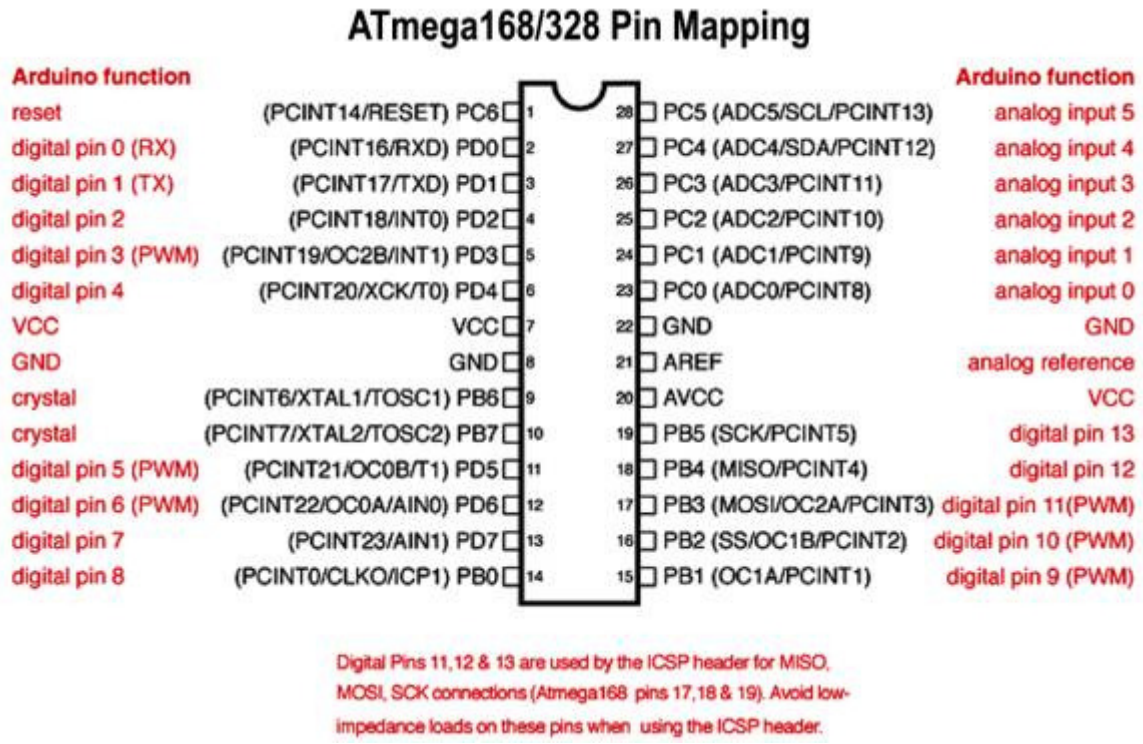
Where T is the motor torque (Nm), K_t is the torque constant of the motor (Nm/A), and i_{arm} is the armature current (A). K_t is sometimes listed as K_i in motor data sheets. The modeled voltage source is commonly called a back Electromotive Force (EMF) and will have an opposing effect to the motor power supply, and its magnitude will be proportional to the velocity of the rotor. This back EMF is expressed as:

$$V_{emf} = K_v \omega$$

Where V_{emf} denotes the back EMF (V), K_v is the speed constant or voltage constant (V*sec/rad) and ω is the rotor velocity (rad/sec) of the motor. K_v is sometimes listed as K_e in motor data sheets. Together, these two equations form the basis for basic DC motor operation.

ATmega328p Registers

What follows is a discussion of the pin layout and registers of our ATmega328p microprocessor. Through the previous labs, you have already become familiar with the registers of Atmega328p that are used for serial communication and analog-digital conversion (ADC). In this lab, the focus will be on the registers concerned with Pulse Width Modulation (PWM).



Timers for PWM

In order to generate a PWM signal to control the motors, first you will need to familiarize yourself with the relevant timers on the MCU. Although, a delay function such as the _delay_ms() command can be used for handling some timing tasks, this usually makes the control loop unacceptably slow in performing any other tasks requiring fast response. For such scenarios, timers are especially useful, as they run asynchronous to the main processor, and allow for multiple tasks to occur without pausing the entire program (e.g. running motors while taking sensor readings). The Atmega328p chip on the Uno has two 8-bit timers and one 16-bit timer. For this lab, you will be using Timer 2, which has an 8-bit resolution. More information on controlling Timer 2 can be found in Section 18 of the ATmega328p datasheet.

The Timer/Counter Control Registers (TCCR2A and TCCR2B) have bits that determine the counting sequence of the timer. For example, if you would like the timer to reset whenever it gets to a target value, you would use “Clear Timer on Compare Match” (CTC) mode. It is also possible to select different types of PWM using these two registers. Refer to Table 18-8 in the ATmega328p datasheet (pg. 155) to determine which bits in the Timer/Counter Control Registers (TCCR2A and TCCR2B) are used to set different counting modes. The code will look like:

```
TCCR2A = (X << WGM21) | (X << WGM20);
TCCR2B = (X << WGM22);
```

The value **X** is either 0 or 1 depending on the timer mode you want to use. Remember that the default setting for all bits is 0, so unless you are changing a bit from 1 to 0, you can delete that term from the code above and just set the bit(s) you want to set to 1.

Note that in Table 18-8, the timer counts up to a value defined for OCRA. Because you are using Timer 2, you will be using OCR2A. The timer will count up to your defined value for OCR2A and then reset, determining the loop rate. OCR2A can be any integer from 0 to 255, as it is an 8-bit timer. In PWM operation, this sets the duty cycle, which is explained later.

You can also use the timer in interrupt mode. Interrupts are events that require instant attention of the microcontroller. This makes it especially useful for applications involving reading sensors or applying outputs (actuation). The general layout for a program with interrupts is as follows:

```
1.  ISR(TIMER2_COMPA_vect) {
2.          //Perform action as OCR2A resets to 0
3.  }
4.
5.  int main() {
6.          //Set up timer counting sequence (PWM, CTC, etc)
7.          //Enable interrupt mode on timer
8.          //Define OCR2A
9.  }
```

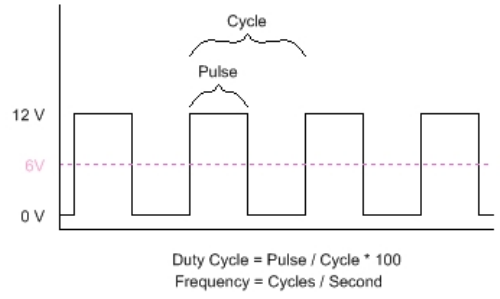
The Interrupt Service Routine at line 1 introduces the interrupt actions into the program. The function parameter indicates the type of interrupt: for a Timer/Counter Match, TIMERx_COMPA_vect is used. For a Timer Overflow, TIMERx_OVF_vect is used. One thing to note is that Timers can be used without interrupts (as you have probably used in the previous lab), and that interrupts provide a different option/way to get the behavior you want from the MCU. You can pursue either method to complete this lab.

The following web sites provide detailed materials and example codes regarding AVR programming with timers and PWM.
http://maxembedded.com/2011/07/avr-timers-ctc-mode/
http://maxembedded.com/2011/08/avr-timers-pwm-mode-part-i/
http://maxembedded.com/2012/01/avr-timers-pwm-mode-part-ii/
http://www.avrfreaks.net/forum/tut-c-newbies-guide-avr-timers?page=all

---

Procedure:

Pulse Width Modulation
With the basics of timers covered, you can now apply PWM on the Haptic Paddle motor. Pulse Width Modulation is a method that allows one to apply a



particular voltage on the motor in the form of a square wave. The amount of voltage applied on the motor is related to the Duty Cycle, which is the percentage of time the square wave spends in its "On" state.

1. Use Table 18.8 (pg. 155) to determine the bit settings for the first "Fast PWM" mode of operation. Also, you want the Compare Output register to clear when it reaches its maximum value. Table 18.3 (pg. 153) indicates the bit values you need to set in the Timer/Counter Control Register to clear OC2A upon a compare match.
2. Write a program that sends half voltage (6V) from your power supply (12V) to the motor. Refer to the modified Ardumoto schematic (which was provided in HW2) and remember that the motor is connected to Ardumoto Output B in order to determine the appropriate DDRx and PORTx values for the direction and PWM pins. Also, you will need to set Arduino Pin 8 (PORTx0) to 1. This pin acts as an enable, which will prevent the motor from receiving voltage unless it is activated.
3. In order to change the direction of the motor, you will need to invert the PWM signal. Table 18.3 indicates which bits of TCCR2B to change to invert the signal. Also, you will need to set the motor direction pin (PORTx5) to 1. Write a program that runs the motor clockwise at 6V voltage for 1 sec, stops it for 1 sec, runs it counterclockwise at the same voltage for 1 sec, stops for 1 sec, and repeats. Hint: The Bitwise Operation for clearing a bit is a &= ~(1 << b)

Voltage vs. Velocity

Important: In this lab, you will need to connect the provided 12V DC power supply to the Uno, which will power the motor. Without a power supply, the motor would be actuated from the power it will draw from the USB port of your computer (through Uno), but it can demand more current than is available at the USB port and temporarily or permanently damage the port.

1. Try commanding different voltages to the motor and observe the steady state angular velocity of the handle. Notice that as you decrease voltage, there is a point at which the motor no longer moves. The range of voltages that are too low to move the motor are called the "deadband". Determine the deadband of your system (in Volts, for both movement directions).
2. Modify your program to stream filtered velocity values in real-time via serial port and plot it using Matlab (RealTimePlot.m). You can bring in code or programs you used in the previous lab for this purpose.
3. Choose at least 8 different voltage levels outside your deadband (for both directions) and record steady state velocity value vs. each voltage level. Calculate the velocity as a function of voltage. Make sure that your velocity readings make sense (do a visual check for a speed you can observe, for example 4 rpm = 24 deg/sec).
4. Run your motor at its nominal voltage of 12V and determine its no load speed.

Deliverables:

1. Demo: Determine the voltage level that gives you ~24 deg/sec at the handle/knob. Have your haptic paddle motor rotate at this voltage clockwise for 1 sec, stop for 1 sec, rotate counterclockwise at the same voltage for 1 sec, stop for 1 sec, and repeat. Show the velocity

measurements in real-time in Matlab during this operation (using RealTimePlot.m). Make sure that your measured values are sensible. If not, debug your code until you get correct readings. Show and explain your C code.
2. Report: Provide a plot of angular velocity vs. voltage for your system. Indicate the deadband. Explain the reasons for the deadband.
3. Report: On the Hansen 116-1 motor datasheet, the corresponding motor is the one listed on top (116-1121627). Compare your recorded no load speed to the value reported in the datasheet. Discuss the reasons for any difference between the two values. Noise cannot be a reason (the components of the velocity measurement system you developed have sufficient precision that the noise should be negligible compared to the differences in values mentioned). **Note** that the speed of the motor shaft is different than the speed of the knob, due to the pulley ratio. This ratio is **80:22**.
4. Report: Most research grade haptic systems are built with current control instead of voltage control. Based on your knowledge (or further research) about DC motors, propose an explanation for why that would be the case.