Lab 2 — Motor Control and Fast Pulse Width Modulation

This lab is designed to introduce you to how pulse width modulation (PWM) is used to control the speed of an electric motor, how the timers of the Atmega32U4 can be configured for fast PWM mode to produce a desired PWM signal, and how to apply that to your line following algorithm.

You can find this and future lab updates at http://www.csulb.edu/~hill

Table of Contents

What is New?

New terminology and concepts are listed below in black. If you have any questions on this information, refer back to the lectures on the introduction to C++ and configuring the GPIO registers.

Data Types

```
const        // Define variable as a constant
static       // Define variable that will persist after a function return
uint8_t      // Defines the data type as an unsigned 8 bit integer (0-255)
uint16_t     // Defines the data type as an unsigned 16 bit integer (0-65,535)
```

C++ Code

```
#define Name Number // Allows the user to define Name to be equivalent to Number
#include <filename> // Used to include any built-in libraries such as avr/io.h
```

```
#include “filename” // User defined libraries or files that are in the sketch folder
```

Bit / Byte Operations

```
variable = byteValue << numberOfShifts; // Left shift byte value by the number defined
variable = _BV (bitNumber);             // Turns a bit number into a byte value
variable |= _BV(bitNumber);             // Set a specific bit in a byte value
variable &= ~_BV(bitNumber);            // Clear a specific bit in a byte value
variable = byteValue | byte Value;      //
```

Arudino Built-In Functions

Pin Configuration & Control

```
pinMode(pin_number, TYPE);         // Define a pin as an input or output
digitalRead(pin_number);           // Read a digital value for an input pin
analogRead(pin_number);            // Read an analog value from an input pin
digitalWrite(pin_number, output);// Output a digital value to an output pin
analogWrite(pin_number, PWM_Value);
```

Improving Control of the Motors

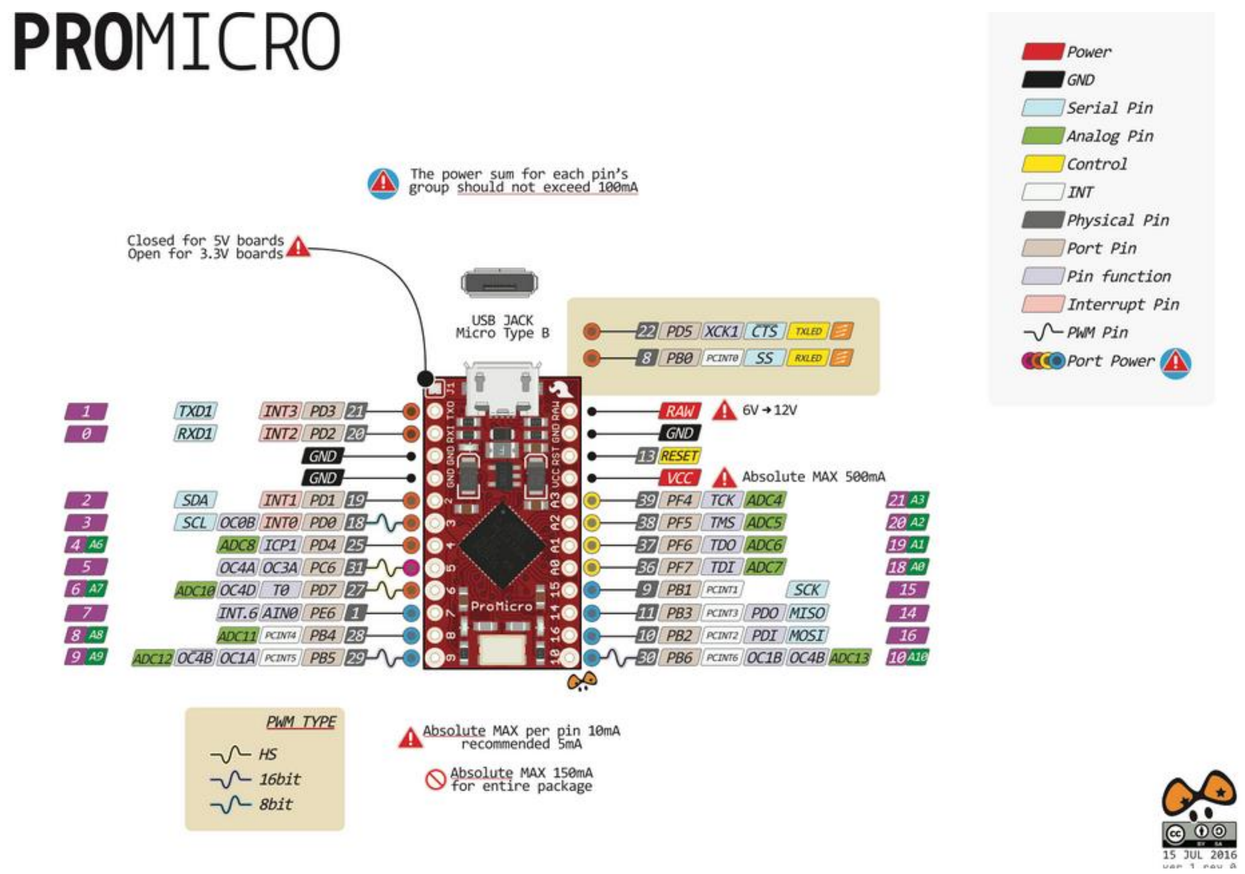
In this lab, one of the main goals is to get better control of the motors. Currently, you are operating the motors at full speed or they are completely stopped. We would like to be able to set the speed of the motors for any value in between those two extremes.

What is Pulse Width Modulation?

Additional information can be found in the Timer with PWM lecture on the class website.

Using the Timers in Fast PWM Mode

As you learned in lecture, the timers can be configured to generate the desired PWM values and output them on specific pins. Those pins are indicated in the figure below. If you plan to use this feature for the timers, you must use one of the five available PWM pins. You will notice that we are already using PD7 and PB5 for our PWM control signals for our motor driver.



The Timer with PWM lecture goes into detail about how this entire system works and you should understand that before we move forward. For the actual implementation, you can use the built-in function of analogWrite(). All of the configuration that was described in the lecture is handled for you in the analogWrite() and the specific limitations are described here.

Finding the Speed to Match Both Motors

As stated before in Lab 1, there is a high probability that your two motors are not equal. There could be a physical difference in the gearing of the motors or their quality, which translates to a difference in maximum speeds. It could cause the robot to veer towards the right or left when you have both motors running at the maximum speed. In order to address this issue, we will be manually finding the ideal speed for each individual motor that has them moving in sync.

This portion of the lab involves a lot of trial and error, so it would be ideal to create a small setup to verify the success. You should have a straight line that is about two feet long and boundaries on either side of the line to clearly indicate when the robot veers off course. You will be manually controlling the speed of each motor using the analogWrite() function. Create a new project and copy over all of the code from Lab 1 except for the line following algorithm. In the main loop, you will only have the code to control the speed of each motor.

Begin with both motors running at full speed (PWM value of 255) and note if the robot deviates from the line. You should be able to determine which motor is causing the problem and you can then lower the speed of the faster motor. Repeat this process until the motors are in sync. Write this information as a comment in your code.

Stall Speed and the Minimum Speed

With the ability to accurately control the motors, there are some cases that you will want to avoid such as stalling the motors. This occurs when the speed of the motor is not enough to overcome the internal friction of the gears and nothing is moving. The motor is on at this point and it is consuming power but there is no physical motion. This can damage the motor and should be avoided at all costs. In order to find this value, there is a slight alteration that needs to be made to the circuit. In order to easily vary the speed instead of using the software to do it, we can utilize a potentiometer in a voltage divider circuit to provide the PWM value to be tested.

Start by connecting a 1 KΩ resistor to VCC (3.3V) and a 10 KΩ potentiometer. The potentiometer should then be connected to ground. Use a jumper wire to connect one of your available analog pins on the Pro Micro to the junction between the 1 KΩ resistor and the potentiometer. Use the analogRead() function to obtain the PWM value that will be used for the test. You will need to multiply that value by $\frac{255}{1023}$ in order to convert it to a valid PWM value for the analogWrite() function. Modify the code that controls one of your motors so that it now uses the value from the voltage divider as the PWM value as the speed. Vary the resistance for the potentiometer until you find the stall speed. Obtain that value by using the Serial.print() function to display the current PWM value.

After identifying the stall speed, we would like to find the minimum speed that will have the motor moving. It would be best to avoid turning the motor on and off because of how jerky that will make the movements of the robot. By obtaining the minimum speed to get the motor moving, we can keep the motor on and barely moving while it adjusts to get back on the line. Using the same setup for finding the stall speed, determine the **minimum speed** for each motor and note that in your comments.

Updating Lab 1

Now that you have identified all of the values that are needed to properly control the speed of the motors, we can update the line following algorithm from Lab 1 to rely less on making adjustments. Ideally, the robot should always move forward and never veer off a straight line with this change.

You will need to replace the `digitalWrite(PWMA, HIGH);` and `digitalWrite(PWMB, HIGH);` with the corresponding `analogWrite()` calls with the proper maximum speeds.

IMPORTANT: Do the same for the code that turns off the motors to adjust the robot. It should now make the motor run at a speed **not less than the minimum speed** instead of turning it off. Start with a speed halfway between the minimum speed and full speed. If the robot does not react quickly enough then gradually decrease the speed of the motor. Again to not go below the minimum speed of the motors.

There was not much that changed from Lab 1 but your line following algorithm should now be more efficient than before.

Lab 2 Deliverable(s)

All labs should represent your own work - DO NOT COPY.

Submit all of the files in your sketch folder. Make sure that the code compiles without any errors.

Checklist

Remember before you turn in your lab...

1. You have identified and indicated what the PWM value should be for each of the motors to run at full speed. For example, the left motor could have a value of 255 while the right motor has a value of 233.
2. You have identified and indicated what the stall speed is. This is when the motor does not turn on its own and the motor is not off.
3. You have identified and indicated what is the minimum speed where the motor will slowly turn.
4. The speed you used to update your line following algorithm. Your line following algorithm uses the analogWrite() function to control the speed of the motors. It should use this speed for adjusting when the robot veers off the line.