ECEN301
Motor Control Assignment

Robert Reid
300385028

October 23, 2018

Abstract

The primary purpose of this assignment was to implement the knowledge and skills developed throughout this course to control the speed of a 12 volt DC motor. The main objectives for the system are to be able to control the motor speed by using pulse width modulation (PWM), returning and displaying the actual speed by calibrating it along with other acquired information, varying the speed of the motor at the user's control. Furthermore, implement and investigate the effect of using a control system to minimize error and optimize the system response. This uses the application of low-level machine coding of a microcontroller, measuring external interrupts, calibrating the interrupts to derive a speed and tuning the control system. The final result has met the majority of the specifications such as the DC motor speed being adjustable by the user, with a control system implemented, and displaying information such as the RPM.

Hardware Setup

The first part of the assignment is setting up the hardware that will drive the system once the software is implemented. The entire system consisted of five individual modules that are all to be interconnected. The microcontroller which is the heart of the system that will store and run the designed program that will control everything, powered externally from the mains and software downloaded from a PC. The DC Motor Driver, powered by an external $\pm 15V$, is used to drive the DC Motor clearly, at $\pm 12V$ but also receives a $\pm 5V$ input signal which is modulated with the power supply to vary the output. Consequently, a module being the DC motor itself which is driven from the previous, that also incorporates an encoder to produce a square-wave signal with the frequency dependant on the oscillation of the motor wheel. Although this encoder is powered independently by micro-controller at $5V$. Next module is the LCD which is a 16 character, two-line liquid crystal display, that will output text and values from the micro-controller. The last part of the system is the I/O module which outputs an 8-bit value used to vary the input speed, these bits are set with hard switches.

The first challenge of the assignment was to get the motor working with the speed dependant on varying pulse width modulation. This only includes the Microcontroller, Motor driver and Motor modules for this initial stage. The microcontroller has an internal PCA module that performs as an 8-bit pulse width modulator that is to be used as the input signal for the motor driver. To set up the PWM, firstly the PCA is turned on by setting CCON to 0x40, which is equivalent to 0100

0000 so just setting the PCA control bit HIGH and leaving the rest LOW. Next the compare 8-bit PWM is enabled using the CCAPMn register, this outputs the waveform on the CEXx pin. The PWM output can be seen below in Figure 1 which displays the variation from changing the 8-bit CCAP value which controls it. High bit values are seen to have a smaller HIGH-value response with the majority of the cycle LOW, the opposite is seen when the PWM signal is set to 0000 0001 where the low part of the cycle is minimal. This signal is modulated with the input signal of the DC motor which results in having control of the input voltage from $\frac{1}{255} * V_{supply}$ to V_{supply}.

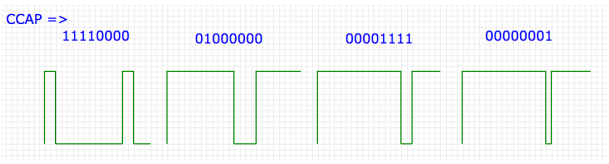

Figure 1: PWM output for varying CCAP0

With the motor working at a speed dependant on an internal register appointed in the software, the LCD display is introduced to provide an indication of the speed that is derived from the encoder's output signal. The DC motor has effectively 16 holes for each opto-switch on a gear that is at a ratio of 30:1 to the wheel which produces a square wave signal dependant on the frequency of rotations. This signal is input into the microcontroller through the CCFx bit in CCON register to generate an interrupt request. These interrupts are to be captured using the PCA module where it stores the current count of the PCA at the time of the interrupt. When there is an interrupt the system calls the handler function where the normal code execution is suspended and the interrupt service routine is run. This routine disables interrupts, checks if the interrupt is on the expected CEXx pin and if so the PCA value is read as a 16-bit register between CCAPnH and CCAPnL. This produces a current PCA value, which the previous value is subtracted from resulting in a change in the count. This change in the count is then used to calculate the RPM knowing the PCA frequency is a 12th of the micro-controllers 12Mhz. This produces the equation below:

$$RPM = \frac{F_{OSC}/12 * sec/min}{holes/sensors * Gear \quad ratio * \Delta Count} = \frac{1Mhz * 60}{16 * 30 * \Delta Count}$$

Once the $\Delta Count$ value has been stored, the overflow flag and PCA interrupts are cleared before the interrupts are all enabled again. This value is then output onto the LCD to display an RPM value in real time. This speed is verified using the tachometer and was observed to be within ± 1 of the value calculated.

The last module to be implemented is the I/O module which provides the means to change the speed of the motor while running the system. This outputs 8-bits on a ribbon cable which goes directly into a port of the micro-controller. Conveniently this is the same amount of bits as the PWM register so it can be directly set, resulting in the speed being controlled by the 8 switches. The final interconnection of all these modules can be observed below in 2, where each port of the micro is labeled P1, P2 etc. The red lines represent the positive voltage lines, blue the negative, the dashed represents ground and the solid arrows represent data lines. Note that the DC motor and Encoder have separate grounds, the Encoder is grounded with the micro and the motor is grounded with the power supply and driver.

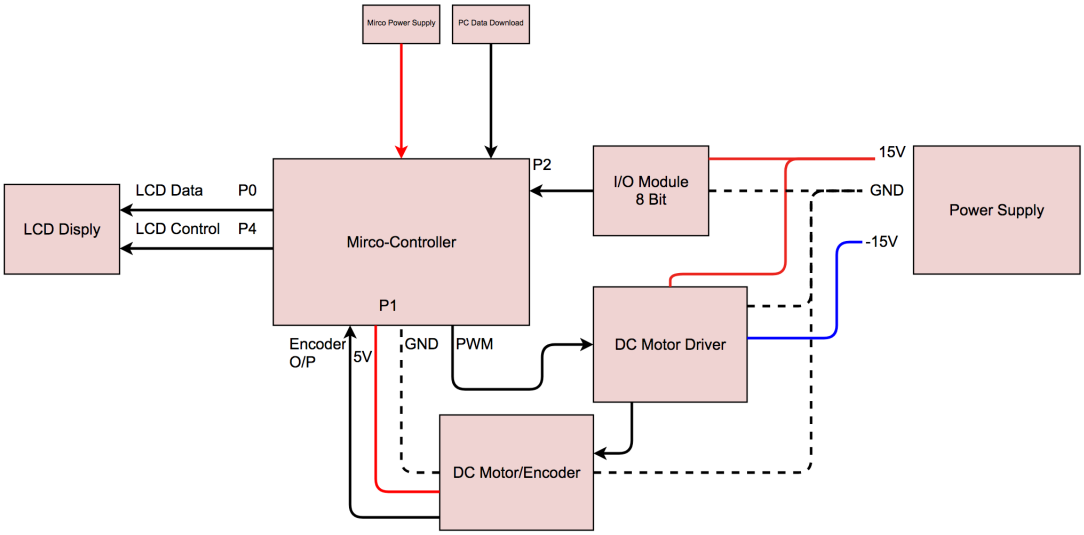

Figure 2: Block Diagram of Hardware Overview

Control System

A control system is implemented to allow the system to automatically regulate the output and set it to the desired point without external interference from the user. This is achieved by comparing the output value to the target value then adjusting accordingly, this results in an error value from the difference which is aimed to be minimum. The micro-controller is set up to output the actual RPM, the target RPM and the error value to the LCD display so the user can analyze the system. This display is shown on the right in Figure 3, after the control system has been implemented.

Figure 3: LCD Display at Full Speed, Steady State

Without any control, the relationship between the PWM and RPM can be observed. It is found that when the speed is set at the maximum (255 input to the PWM) the output RPM is 265, at the midpoint (128) the output is 135. This shows that relationship between the input and output is reasonably linear, therefore the target speed can be set directly instead of as a function. Although for inputs lower than 10 the wheel is motionless so this gives a threshold for a minimum speed. This gives a range of approximately 10 - 265 RPM, with the input for the target speed being from 0-255 directly from the 8-bit I/O module.

The control system that is implemented is a proportional-integral-derivative controller that has negative feedback in a closed loop. The proportional part of the controller is the outputs to the motor at a value which is proportional to the size of the error. With just this part of the controller, it results in an offset where the output is at a steady state below the desired RPM. This problem is overcome with the integral term which is a sum of the errors multiplied by the integral constant, Ki. This eliminates the offset error as the integral term continues to increase until the error tends to zero. The differential term provides an anticipatory action which observes how fast the error is changing

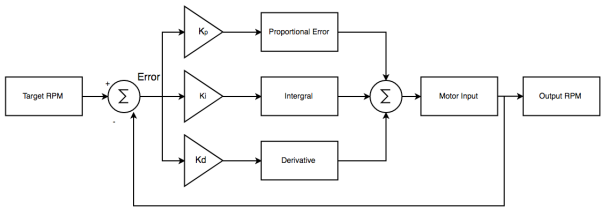

Figure 4: Closed Loop System of PID Controller

and tries to reduce the rate of change of error to zero. These three components are summed together producing the motors input value. This system is shown in the diagram on the left in Figure 4 which shows the process from the input speed to the resulting output speed, with negative feedback.

With the control system implemented, the system needs to be tuned for optimal operation. This means that the motor will respond to the input with a minimal response time, minimal overshoot and minimal oscillation. These cannot all be done in practice so there is a trade-off between them. The first step in tuning the system was setting the proportional constant, so the integral and derivative were set to zero. Initially setting the constant as unity, it was observed that the motor would oscillate a lot before coming to a steady state with an offset error. This presents that the constant was much too high and needed to be decreased to increase the damping of the system. Also, it was noted that the oscillation would increase as the constant did, and once past a value 2.1, the system became unstable where the oscillation grew in amplitude. To find the optimal value the constant was decreased down from 1 to 0.3 where there was minimal oscillation and still responsive. The lower the proportional value, the less responsive the system is. The integral constant was tuned with the same approach, starting from unity and varying. At unity, it was observed that the system accelerated towards the desired speed quickly but overshot a lot. The optimal value was found to be 0.1 where there were minimal overshoot and still a reasonable response rate to the target speed. The proportional and integral values constant, the derivative term was set to unity where it caused the system to go unstable. It was found that the derivative term caused a lot more problems than it solved. Ideally it would of improved the stability and settling time of the system. It was set to 0.05 where it made only a small influence on the system resulting in the motor being driven primarily off just proportional and integral.

To keep the RPM reasonably stable so that it was less susceptible to spikes, the values were averaged using a sliding mean. This was done by storing an array of the five most recent $\Delta Count$ values then taking an average from that. This is effectively a finite impulse response filter. With the previous constraints all in place, the system responds well to all usual activity except for at low speed below 20. It was found that the motor began to oscillate around steady state, to compensate for this the speed was set directly without averaging. This fixed the problem down to about 15 RPM, where it was overcome by no longer using the PID and just setting the speed directly considering the linear relationship. A minimum speed threshold was set at 10 RPM, where the system was unresponsive therefore it was just set to zero, otherwise the integral would increase and cause spikes.

While testing the system's functionality, the wheel was physically held still while the input was set to a high value, which caused the integral to 'wind-up' as it increases continuously. This results in a large overshoot as the input is much higher than it needs to be. This was thought to be able to be fixed by capping the integral term so it could only get so high, although this meant that the term had to be reset each time the target speed was changed. This was attempted to be implemented in the lab but did not end up with good results. For example, it caused the error as shown in Figure 5, where the motor would overshoot then drop back to stop and go overshoot again. This oscillation in the behavior is very undesirable so this was not implemented. Apart from intentionally

disturbing the system, the response was very good as the wheel would reach the target speed with an error of ±1 RPM using the tachometer.

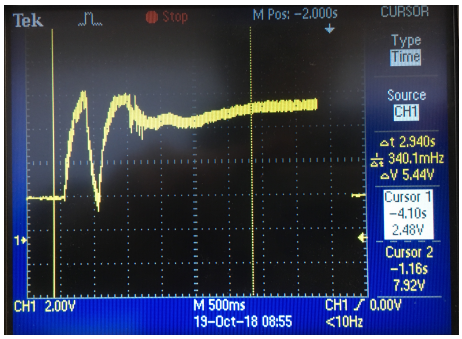

Figure 5: Oscillating Error at Step Input

To observe the response of the system on the oscilloscope the voltage across the motor input was measured, which is what is being shown in Figure 5 and 6. Figure 6 shows the system response to a step input of a target of 200 RPM. The initial climb in the voltage is due to the proportional term which causes the main increase in speed, then the integral component kicks in which minimize the offset error and goes to a steady state. Overall the control system worked well as it would meet the target speeds with a response time of 3 seconds which is measured using the oscilloscope as shown in Figure 5. While the system was already running, it responded much better to changes in target speed, with no oscillation in the steady state, and no overshoot. The less optimal part of the system was observed when responding to a step input or when disturbing the system by hand.

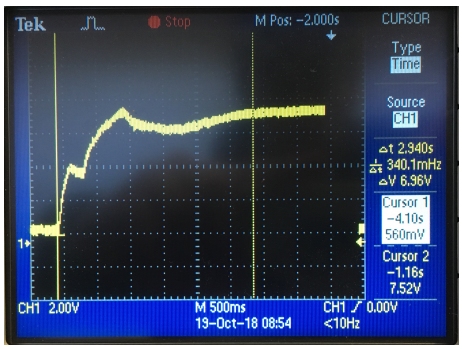

Figure 6: Step Response of System

Critically, the poorer functioning parts of the system were found to be originating from the control when facing unexpected disturbances. This resulted in overshoot and undesired oscillations. This could have been improved by implementing more restrictions on the system when in certain states, such as implementing an integral cap. Another sector in the system that could be improved is the range of speed. Instead of having the target speed congruent with the 8-bit input, the max input should be equal to the maximum possible output, so in this case 265. This would require a non-linear function between the input value and target speed. In addition, the step response of the system could be cleaned up to have a more direct and smooth increase to the target speed. This would be done by finer tuning of the PID controller.

Conclusion

Overall, the motor control system met the majority of the specifications; varying the motor speed with pulse width modulation, displaying the output of the motor on a LCD display, calculating the actual speed, changing the motor speed externally and implementing a control system to minimize the error factor. The most successful parts of the system were in the accuracy of the steady-state target speed, the core parts of the motor control and the implementation of the embedded system. This all resulted in a functioning motor that could vary between a minimum of 10 revolutions per minute to a maximum of 255, motionless for anything below the minimum. Throughout this assignment, the concepts of low-level programming embedded systems, digital-to-analog conversion, and control systems have been implemented collectively to control the speed of a DC motor.

Appendix

PIN	Analog Input Channel	USED	Function
P1.0	Ext CLK Input for T/C2		
P1.1	Trigger Input for T/C2		
P1.2	PCA Ext CLK input		
P1.3	PCA module 0 input/PWM output	✓	PWM Output to DC Motor
P1.4	PCA module 1 input/PWM output		
P1.5	PCA module 2 input/PWM output	✓	Encoder Input from DC Motor
P1.6	PCA module 3 input/PWM output		
P1.7	PCA module 4 input/PWM output		
P1.8	GND	✓	GND for Encoder
P1.9	+5V	✓	Supply for Encoder

Table 1: Port 1 I/O Description

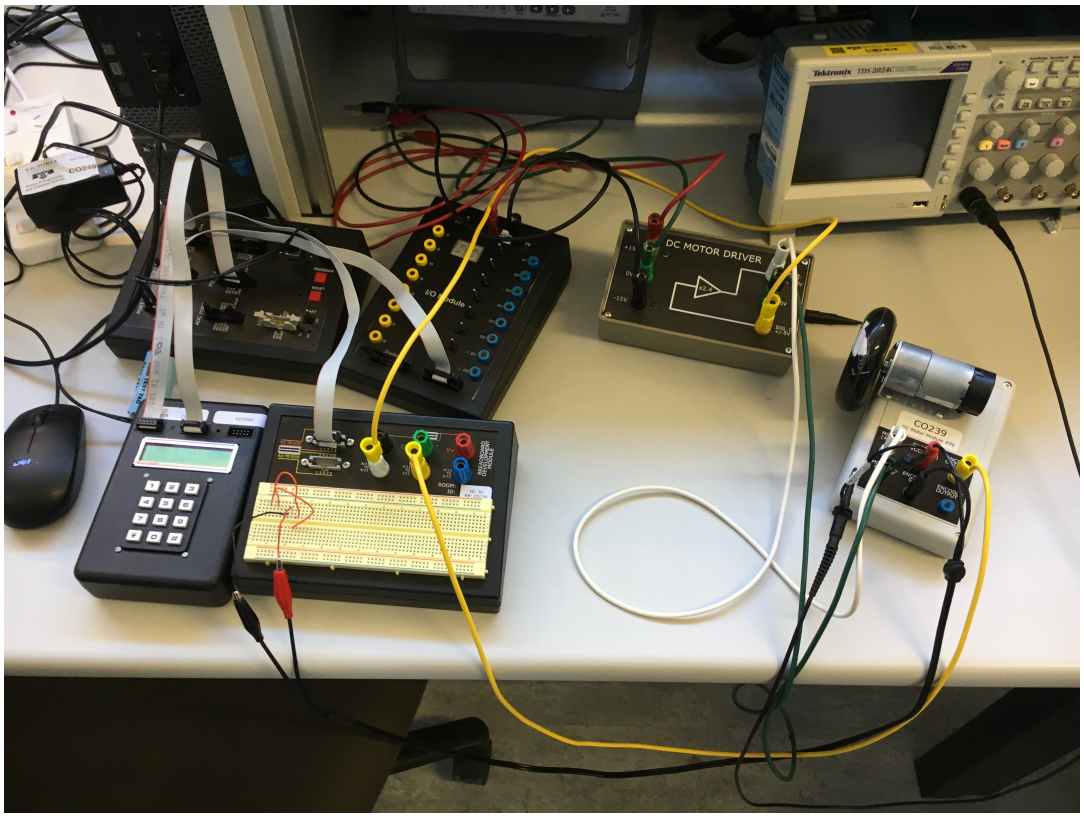

Figure 7: Set up of System in Lab

```c
/***  Include Files  ******************************************************/

#include <t89c51ac3.h>
#include <string.h>
#include "phys340libkeil.h"
#include <stdio.h>

/***  Declare Vaiables  ***************************************************/
char RPMread [33];
char Error [33];
char input [33];
int previous =0;
int current = 0;
int time = 0 ;
int speed = 0;
int in;
int speeds[5];
int err;
int intergral;
int derivative;
int RPM;
int prev_error = 0;
int prevT;

void MyIntHandler(void) interrupt 6{ //external interrupt address set

    EA=0; //disable interupts
    if(CCF0 == 1){//only if specific interrupt occurs

    current = (CCAP0H << 8)| CCAP0L; //16bit value for time

    if(current > previous){
    time = current - previous;//calculate time between interrupts
    previous = current; //update value
    }

    else{
    time = 0xFF00 - previous + current;//time between interrupts for past overflow
    previous = current; //update value
    }
    }
    CF = 0; //clears overflow flag
    CCF0 = 0; //clear PCA capture interupts
    EA = 1; //Enable all interupts


}

void MotorIn(int in){
  CCON = 0x40;//Turn on PCA
  CCAPM2 = 0x42;//Enable compare module and PWM
  CCAP2H = 255 - in;//set PWM from 0x00 - 0xFF4
  //255 - speed as stores down from 255, stores from right to left in the address
}

void captInter(){
  EA = 1; //enable interupts
  EC = 1; //enable PCA interupt
  CR = 1;//turn PCA timer on
```

```
60   CMOD = 0x01; //enable CF bit in CCOn register to generate interrupt
61   CCAPM0 = 0x21; //positive edge triggered, ECCF enabled
62 }
63
64 int slid_avg(int speeds[5]){
65   int k;
66   int sum = 0; //sum of all values
67   int average = 0; //declare average value to be returned
68   for(k = 0; k < 5; k++){ //iterate through array of values and increment sum
69      sum = sum + speeds[k];
70     }
71   average = sum/5; //calculate average
72   return average; //return average
73   }
74
75 /***  Main Function  *****************************************************/
76 void main(){
77     int i;
78
79     //declare the float variables for the controller, floats as < 1
80     float Kp=0.3; //Proportional Variable
81     float Ki=0.1; //Intergral Variable
82     float Kd=0.05;  //Derivative Variable
83     initLCD(); //initialise LCD display
84     captInter(); //.initialise interupts
85     in = P2; //set the input value from the P2 Port
86
87   while(1){
88     int Target =P2;//set target to input from I/O module
89
90     //set the the intergral back to zero every time the input speed changes
91     if(Target != prevT){
92     intergral = 0; //set to zero
93     prevT = Target; //store previous speed
94     }
95
96     for(i =0; i<=4;i++){
97      speeds[i]=speeds[i+1];  //increment array of speeds
98     }
99     speeds[4] = time; //add new value
100
101    RPM = 125000/slid_avg(speeds);//take average and calculate RPM
102    if(RPM <= 20){RPM=125000/time;}//dont use average below 20 as it causes errors
103
104    err = Target - RPM; //calculate error value
105    intergral = intergral + err; //calculate intergral, value adds on from previous
106    derivative = err - prev_error; //calclate derivative term between errors
107
108    if(derivative < 0){derivative =0;} //limit der term to zero so cant go ngtve
109    //if(intergral > 3000){intergral =3000;} //limit intgl term to stop int windup
110
111
112    // calculate the control value
113    in = (Kp * err) + ( Ki * intergral)+(Kd*derivative);
114
115    if(Target <=15){in=Target;RPM=Target;} //limit PID to 15rpm as below causes
       errors
116    if(Target <=10){in=0;RPM=0;}  //limit speed to 10 RPM as doesnt go below this.
117
```

```
118     MotorIn(in);  //set speed with PID control
119
120     prev_error = err; //set previous error for next loop
121
122     sprintf(RPMread,"RPM: %d ", RPM); //Covert RPM to int
123     sprintf(input,"Tgt: %d ", Target); //Convert Target to int
124     sprintf(Error,"Err: %d", Target-RPM); //Calcualte Error and display as int
125
126     // Write Display, delay, then clear
127     writeLineLCD(RPMread);
128     writeLineLCD(input);
129     writeLineLCD(Error);
130     delaya(10000);
131     clearLCD();
132     }
133   }
```