MECH 423 Lab 3 Report

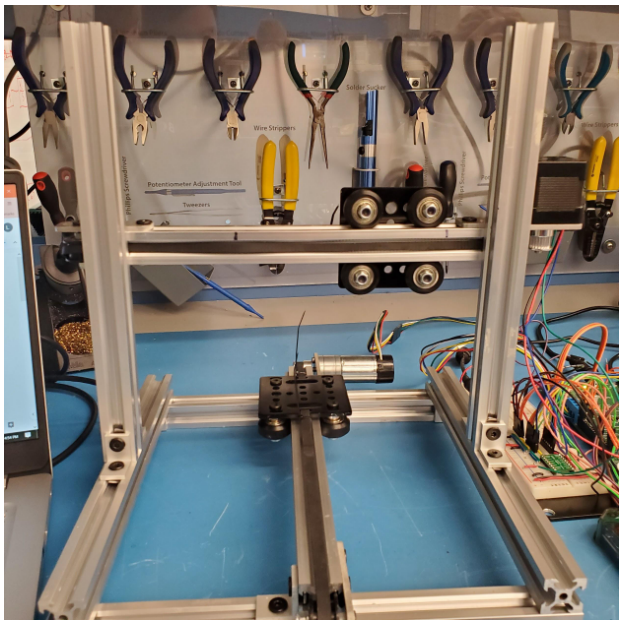
Liam Foster and Willem Van Dam

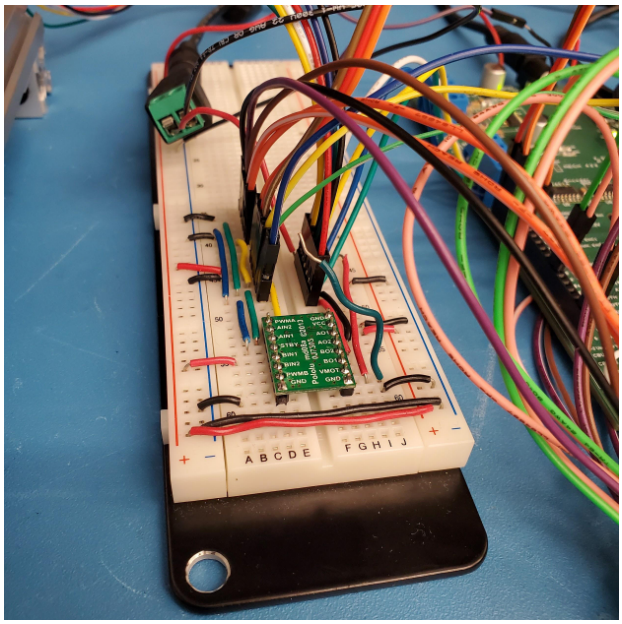
40199382 & 33500646

Apparatus and C# Program

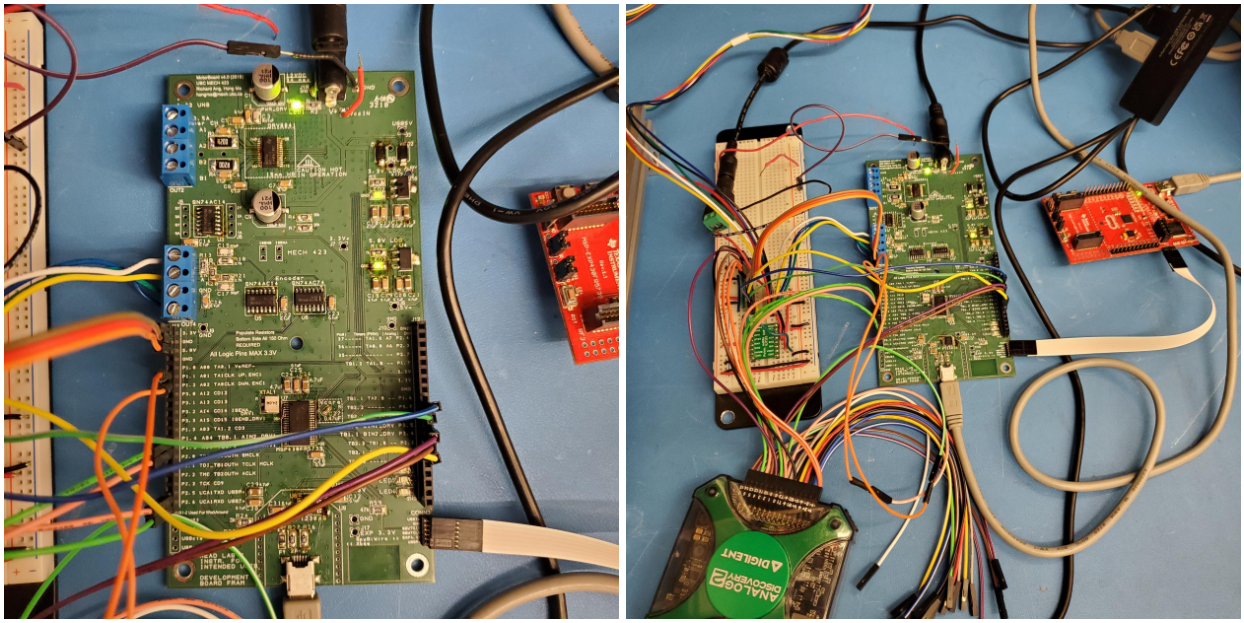
Describe your apparatus and code in your report and include a screenshot of your C# program. Attach your MCU and C# code as appendices.

The lab apparatus and C# program interface remained the same throughout the lab. The C# program interface was added to as the lab progressed but didn't lose any of its previous functionality; it has been added to Appendix A. The MCU code was also appended throughout the lab and kept all of it's previous functionality. The final MCU code is also attached in Appendix B. The apparatus and C# UI are pictured below in Figures 1 through 5:





Figures 1 & 2 - Mechanical Gantry System and motor driver breadboard



Figures 3 & 4 - PCB connections and overall electrical circuit

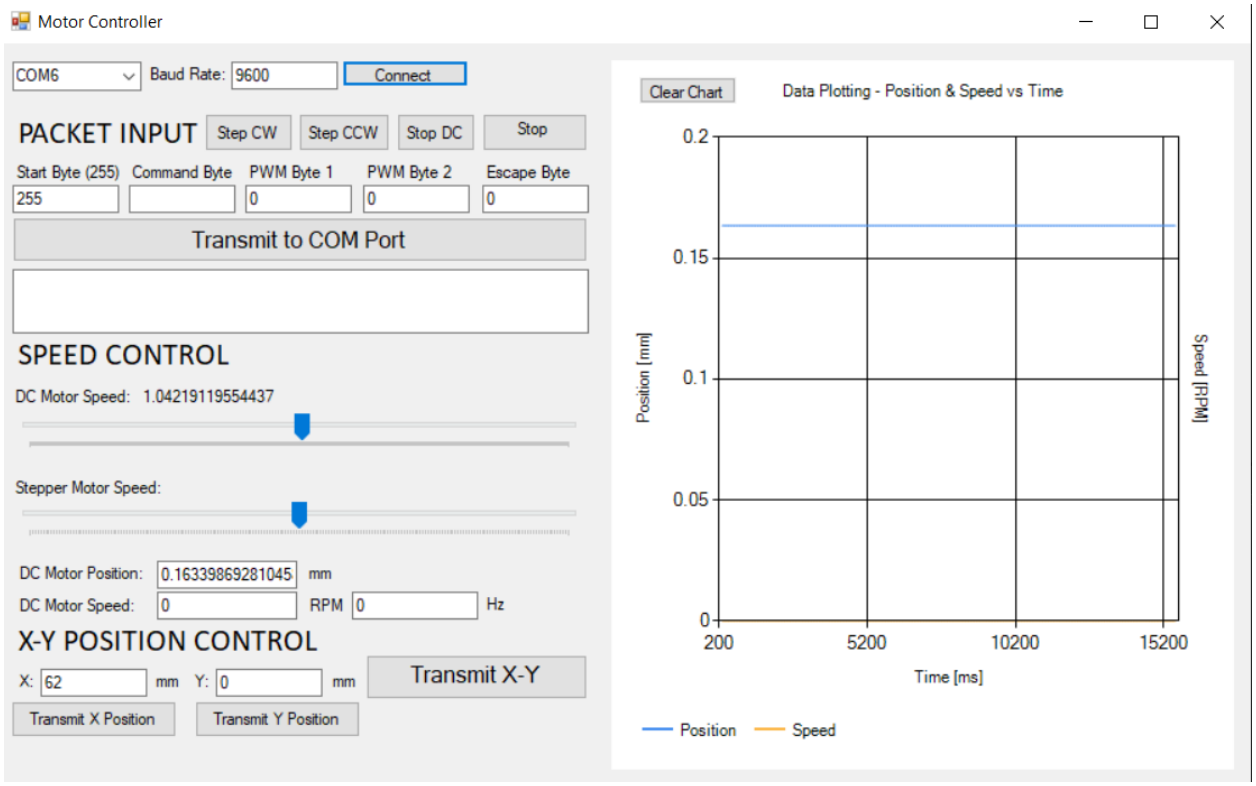


Figure 5 - User Interface Programmed with C#

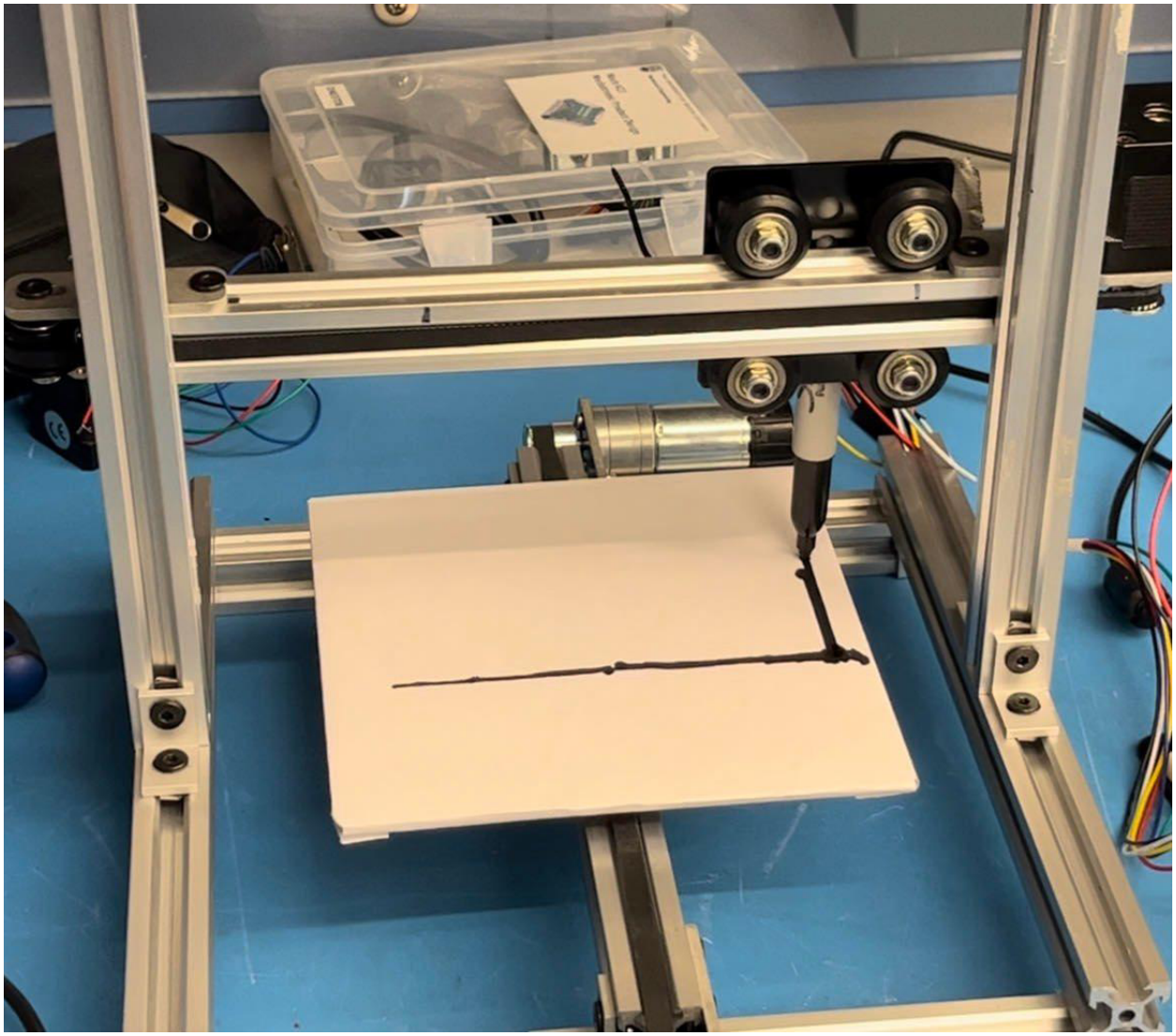


Figure 6. Mechanical Assembly with Marker and Paper

DC Motor Control

Draw an electrical schematic diagram of the minimum components necessary to drive a DC motor including the MCU, motor driver, power supply, and accessories. The drawings should be easily understandable and accurate. You should label the components, component values, as well as pin number and pin names involved. I suggest using Microsoft Visio, but feel free to use any drawing programs that you are familiar with (e.g. Illustrator, Photoshop, Paint, Inkscape, Corel Draw, AutoCAD).

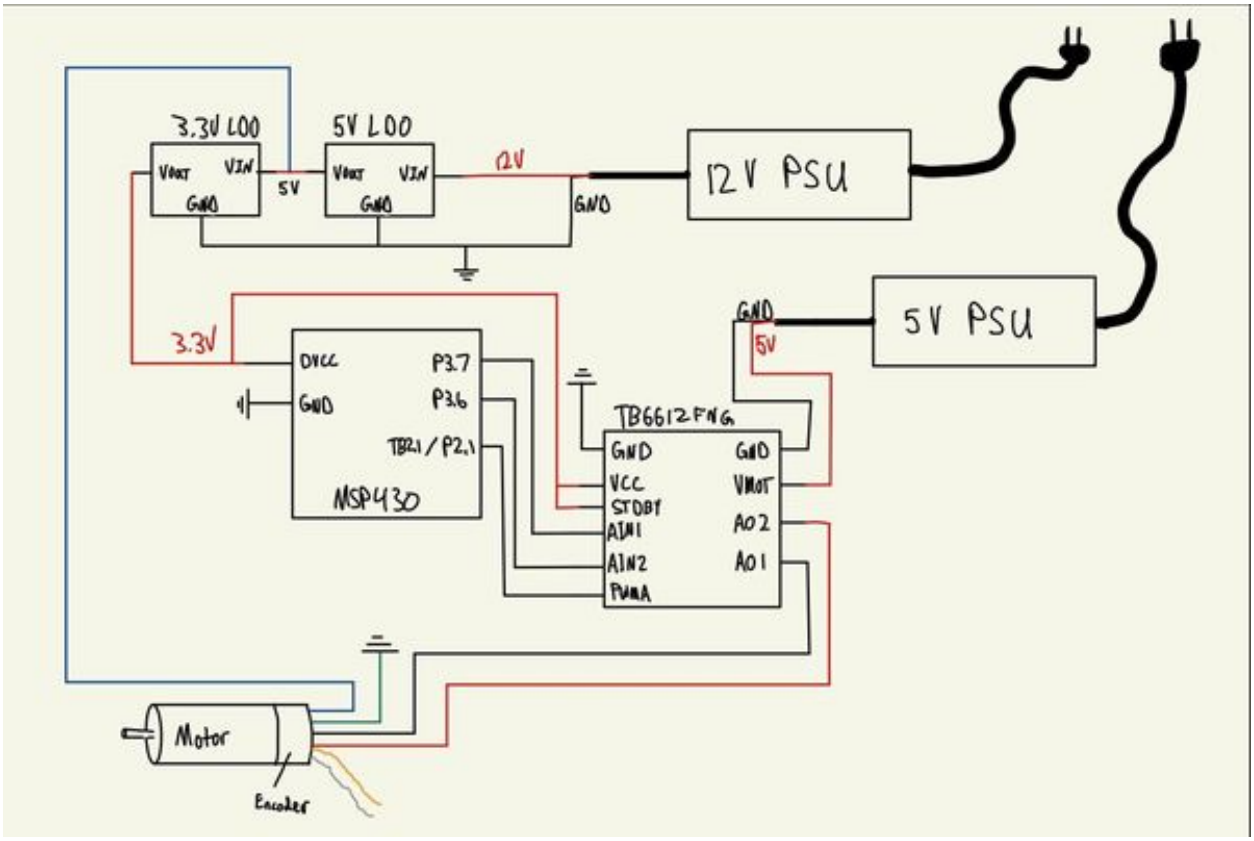


Figure 7 - DC Motor Electrical Diagram

What is the minimum PWM duty cycle for the motor driver to generate a reasonable output waveform? Check using an oscilloscope. Why does this limit exist? Discuss in your report.

The minimum PWM duty cycle that generates a ‘reasonable’ output waveform is a range between 0.15% and 0.3%. The waveform was visible but slightly unstable towards the lower end and completely stable towards the higher end. This limit exists because the transistors in the H-bridge of the motor driver have a rise time necessary when switching for the drain and source to fully open and close. At the very low PWM there is only a very tiny spike of high voltage before the next period of low voltage. The mosfets don’t have time to fully open and close in this time so there is no recognizable waveform from the output of the motor driver.

What is the minimum PWM duty cycle for the motor to turn at no-load? Why does this limit exist? Discuss in your report

We found that the minimum PWM duty cycle was 2.17%. This limit exists because at lower duty cycles the mean voltage is too low to make the motor rotate. The motor also has to overcome the inertia of the motor shaft and pulley wheel as well as small amounts of friction in the motor bearings.

Stepper Motor Control

Draw an electrical schematic diagram of the minimum components necessary to drive the stepper motor including the MCU, motor driver, power supply, and accessories.

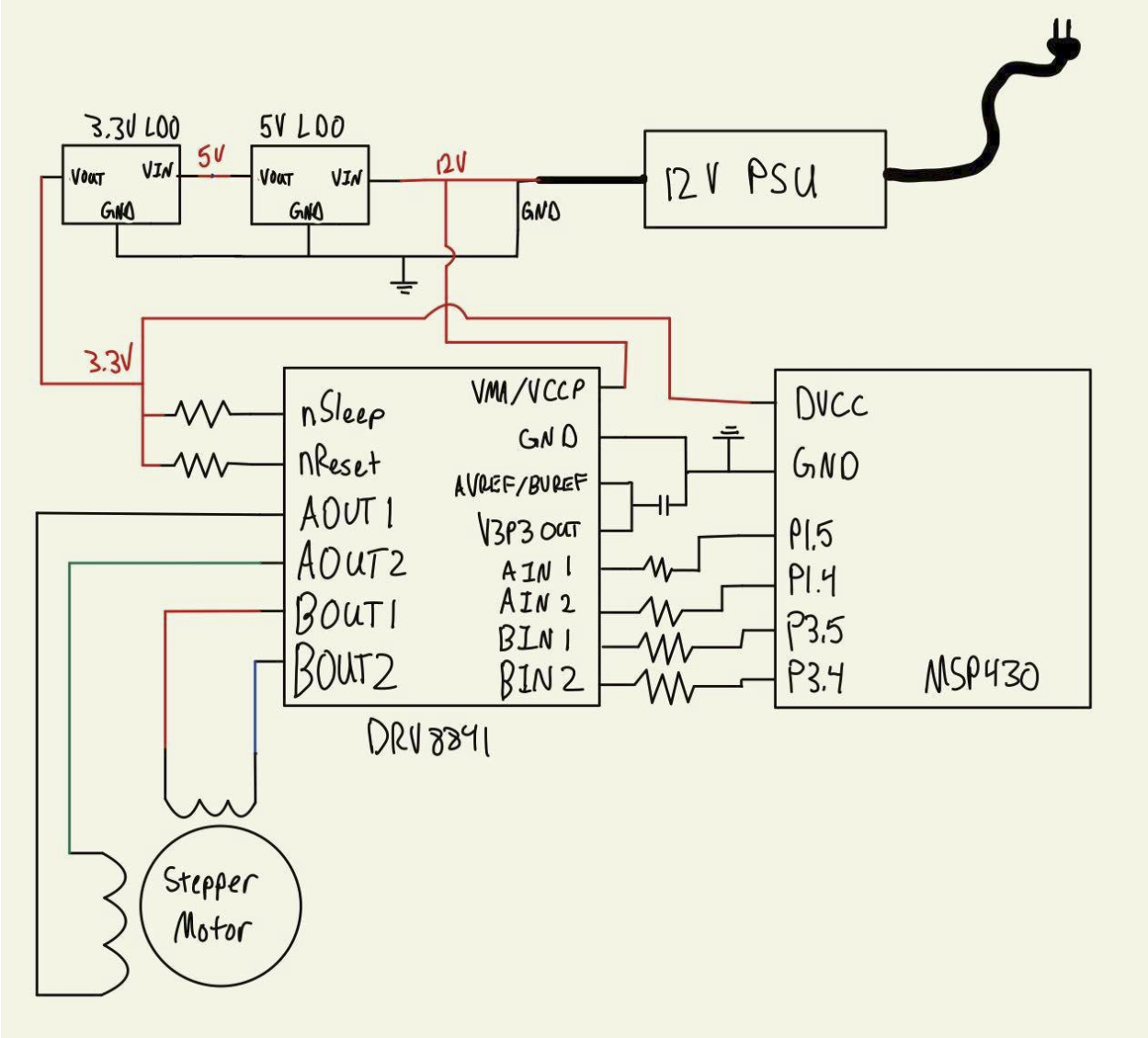


Figure 8 - Stepper Motor Electrical Diagram

What is the maximum speed of this stepper motor (in steps/s and rev/s)? Why does this limit exist? Discuss in your report.

The maximum speed of the stepper motor is 813.835 steps/s or 4.07 rev/s. This is calculated from the maximum observed speed the stepper motor could run at. This corresponded to a half-step period of 0.614ms. The speed in rev/s was then calculated using the stepper motor resolution of 200 steps/rev. This limit is due to the delay present when switching current between the stepper motor coils. Coils are unable to respond to the higher switching frequency. The full calculations are listed below:

Max speed observed at 85% of maximum possible input
Input to stepper motor switching timer period = (1 - 0.85) 65535 = 9830
With A-clock running at 16MHz:
1 step / (2*9830) ticks * 16M ticks/sec = 813.835 steps/sec
813.835 steps/s / 200 steps/rev = 4.07 rev/s

Encoder Reader

Manually turn the motor shaft and show the shaft position and rotational velocity data are correct. Save example data for your report.

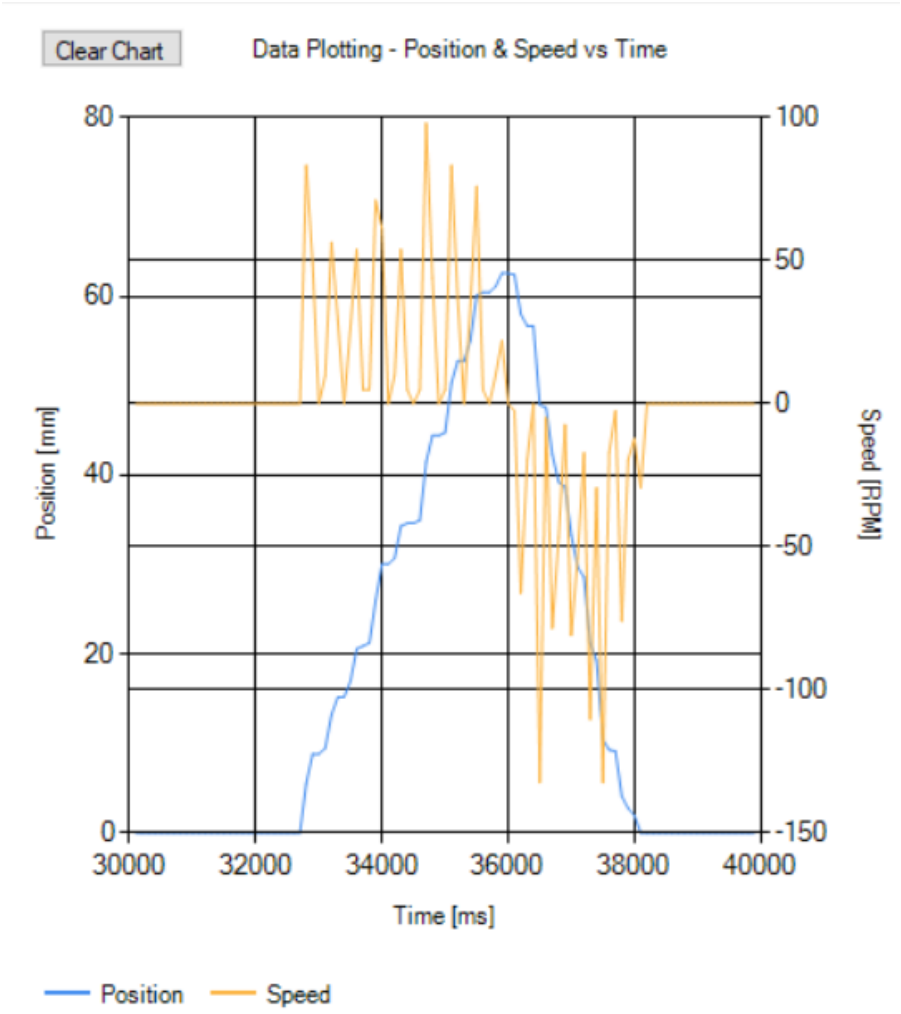


Figure 9 - DC motor position and speed vs time under manual rotation

The position and speed of the DC motor under manual rotation are plotted above. The position of the carriage was measured and found to match the encoder position plotted here. The speed also seems reasonable for manual rotation. The spikes in speed correspond to the readjustment of the fingers around the motor

Incorporate elements from Exercise 1 to generate a PWM signal to turn the motor. Measure the rotate rate versus PWM duty cycle. Describe the experiment and results in your report.

The rotate rate of the DC motor was calculated for 10 PWM duty cycles from 10% to 100%. Therefore 10 series of data were recorded. The encoder counts were used to determine the position and the instantaneous velocity was calculated from the positions for each data point. The velocities were then averaged over the course of the applied duty cycle to determine the rotate rate in RPM. The resulting relationship between DC motor rotate rate and PWM duty cycle is plotted below:

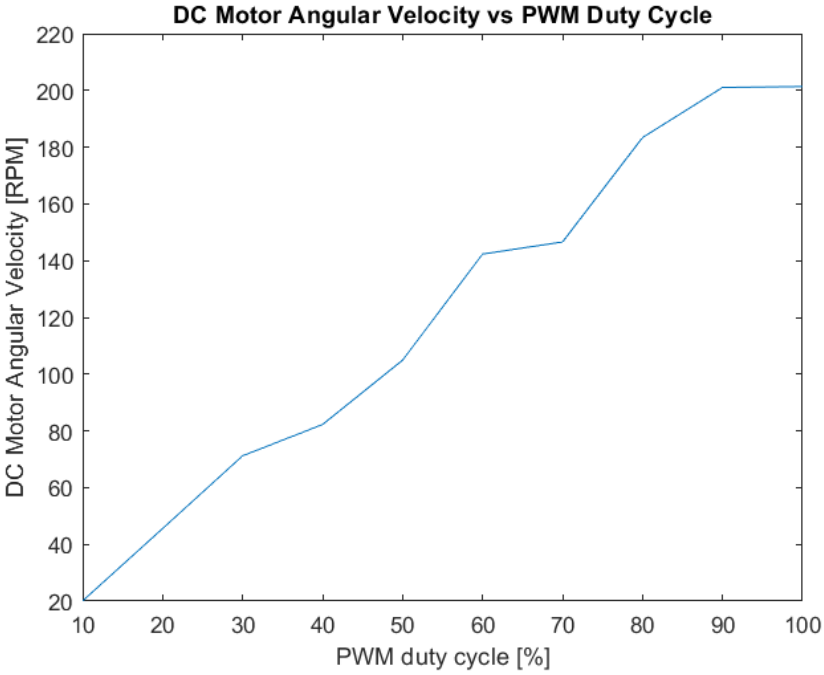


Figure 10 - DC Motor Angular Velocity vs PWM Duty Cycle

As can be seen in the plot above, the relationship is approximately linear for lower duty cycles and then the slope slowly levels off for higher duty cycles as it approaches 100%. This is because the MOSFETs in the H-bridge motor driver are unable to switch on or off fast enough at the low and high PWM duty cycles. This leads to a smaller difference in output velocity at higher PWM duty cycles.

Draw an electrical schematic diagram of the minimum components necessary to obtain and process signals from the encoder, including the MCU, latches, power supply, and accessories.

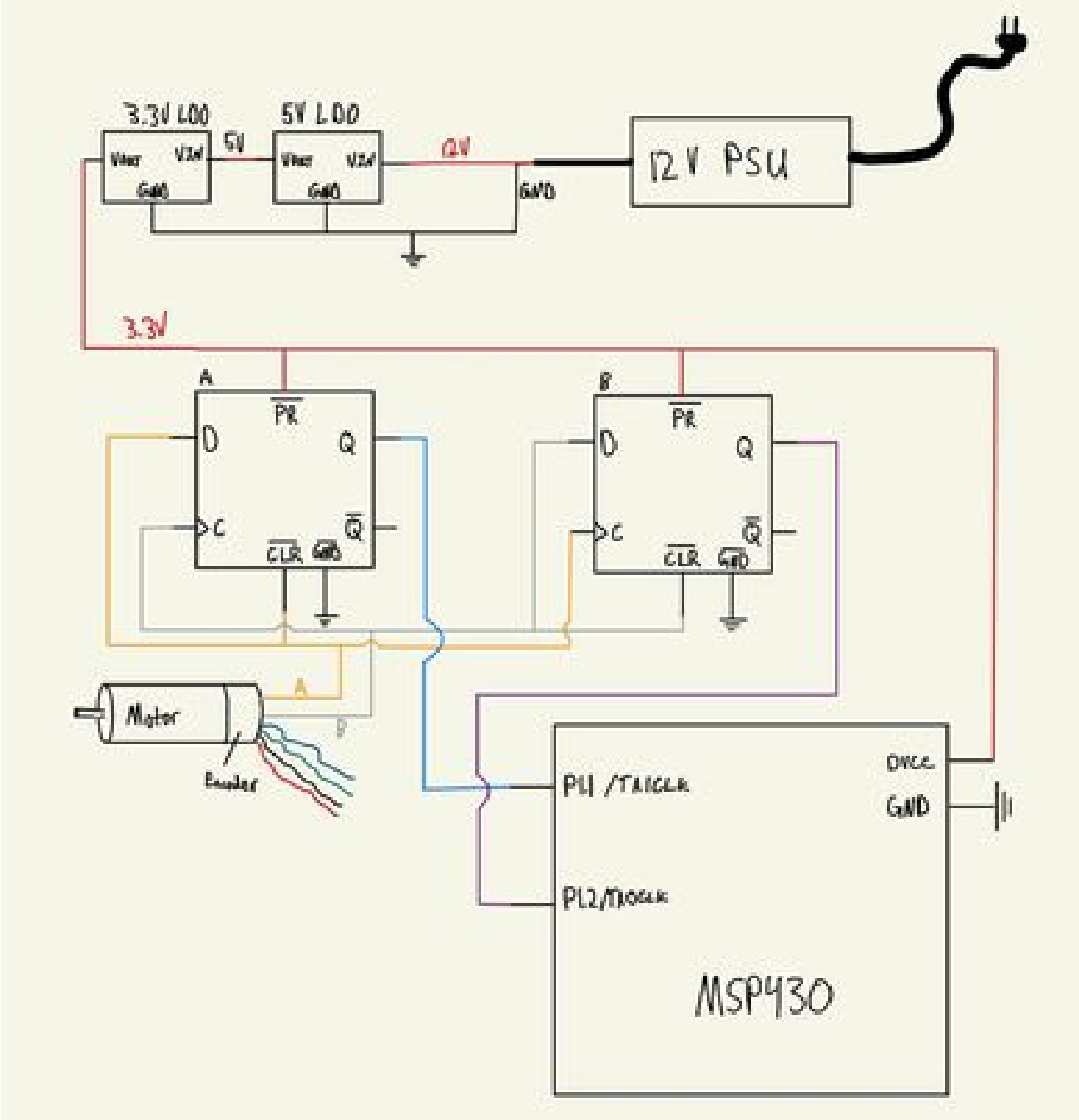


Figure 11 - Encoder Reader Electrical Diagram

Closed Loop Control

Describe your work in your report. Include block diagrams and equations of your feedback control system. Also, include a copy of your MCU code as an appendix.

To determine the closed loop control of the DC motor system, a block diagram was sketched to determine the order of the system and the transfer function from the input position to the actual position read by the encoder. The block diagram is depicted below in Figure 12.

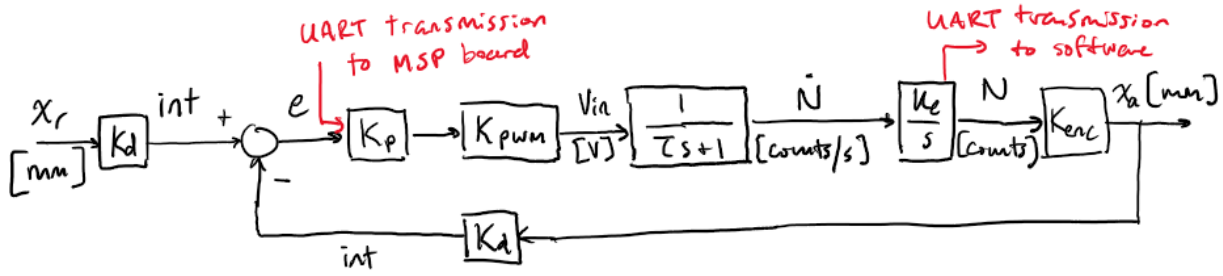


Figure 12 - Closed Loop Control Block Diagram

The transfer function was then determined resulting in the following equation:

$$G_{cl}(s) = \frac{x_a}{x_r} = \frac{K_d(\tau s + 1)s}{\tau s^2 + s + K_T}, \; K_T = K_d K_p K_{PWM} K_v K_e K_{enc}$$

The parameters were determined as follows:

K_d scales the two byte integer to the length of the axes, measured at 123mm.
K_d = 65,535/123mm = 532.8 /mm

K_{PWM} scales the error to a mean voltage input to the motor. The max voltage to the motor is 5V
$K_{PWM} = V_s$ / 65,535 = 5V / 65535 = 7.63 x 10^{-5} V

K_e is 4 to convert the quadrature signal into a number of encoder counts.

K_{enc} converts the number of counts to a linear position on the gantry axis. This is calculated from the timing belt tooth pitch, the number of teeth on the DC motor pulley, the DC motor gear ratio, and the encoder count resolution.
K_{enc} = pitch*teeth/(gear ratio * encoder count resolution)
= 2mm/tooth * 20teeth/rev / (20.4 * 48 counts/rev) = 0.041mm/counts

K_p is the proportional constant determined through tuning. We started with 1/τ and tuned from there until we reached 9.68.

Finally the time constant τ and K_v were determined by measuring the rise time of the DC motor. Starting from rest, when the DC motor's velocity is ~63% from the steady-state velocity, one time constant has passed. Using this relationship we measured the time constant for 25%, 50%, 75%, and 100% duty cycles and averaged them to determine τ. To determine K_v, the steady-state velocity was divided by the mean voltage corresponding to the input PWM duty cycle. This was then averaged to determine the scaling factor between the input voltage V_{in} and the output encoder speed.

τ = 0.02375s, K_v = 0.0003135 counts/Vs

The steady-state velocities vs the PWM duty cycles are plotted below:

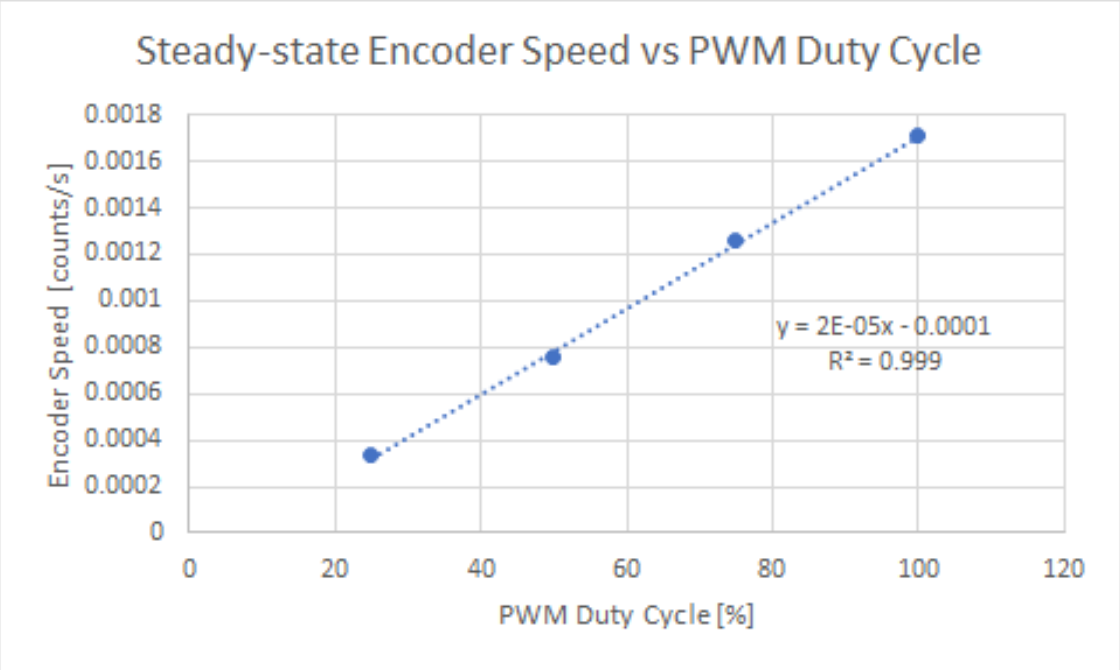


Figure 13 - Steady-state encoder speed vs PWM duty cycle

2-Axis Control

Describe your approach, apparatus and code in your report and include a screenshot of your C# program and a picture of your piece of paper. Attach your MCU and C# code as appendices.

Approach

The approach to implement the 2 axis control was to treat the DC motor as if it was a stepper motor. All of the commands given to the MSP430 were in absolute X and Y locations after being zeroed. This was done by storing the current encoder value and the stepper position value between commands. When given the position in X and Y coordinates the number of half steps needed for the stepper to reach that point were calculated and set as the Y position goal. Then the distance the X axis needed to travel from where it was currently (determined using the encoder value) was divided by the number of Y steps to determine how long an X “step” will need to be. The Y axis and X axis control loops were then started using two different interrupt timers. Every time the Y control loop executed, the reference point for the X control loop was incremented by the value of xStep. This would then trigger the X control loop (running faster than the Y control loop) to use the closed loop control and make it to the next X reference point. The Y control loop also incremented the Y location as usual which caused the stepper PWM timer interrupts to update to the new stepper phase. This then repeats until the Y location has reached the Y position goal. At this point if X reference is not equal to the final X location goal it sets the final X reference to this goal. This allows the DC motor to still reach the accuracy it had before, after any potential rounding errors in calculating the X step size.

In cases where only the Y value changed the X step would be equal to zero and the X control flag (triggering the closed loop controller) would be turned off immediately. In cases where only the X value changed the Y location would equal the goal and the final X reference would be set to the goal. Ideally this would be implemented where if the X value only changed it would still split the X travel up into ~600 steps and loop through in the same way as explained above. This would allow the movements that involve only X to be at the users speed input.

The speed for the 2 axis control loop was controlled by changing the time delay before the Y control loop executes as this is where the y location and x location increment.

Extra tuning of the Kp value and the tolerance for the X location was required as the extra instructions involved with controlling and updating the Y axis, and the extra friction of the sharpie on the paper resulted in either overshoot, or not enough acceleration from the X axis to match the Y axis speed.

Results

The final assessment of the gantry was to trace the following coordinates on a piece of paper.

Table 1. Final Assessment Locations

Location	Absolute X (mm)	Absolute Y (mm)	Velocity (%)
1	50	0	100
2	50	50	50
3	0	0	20
4	70	20	60
5	0	50	80
6	0	0	10

The results from the final assessment of the gantry are shown in the figure below. The lines highlighted in yellow were traveled during the final tuned run. The other lines were previous untuned attempts.

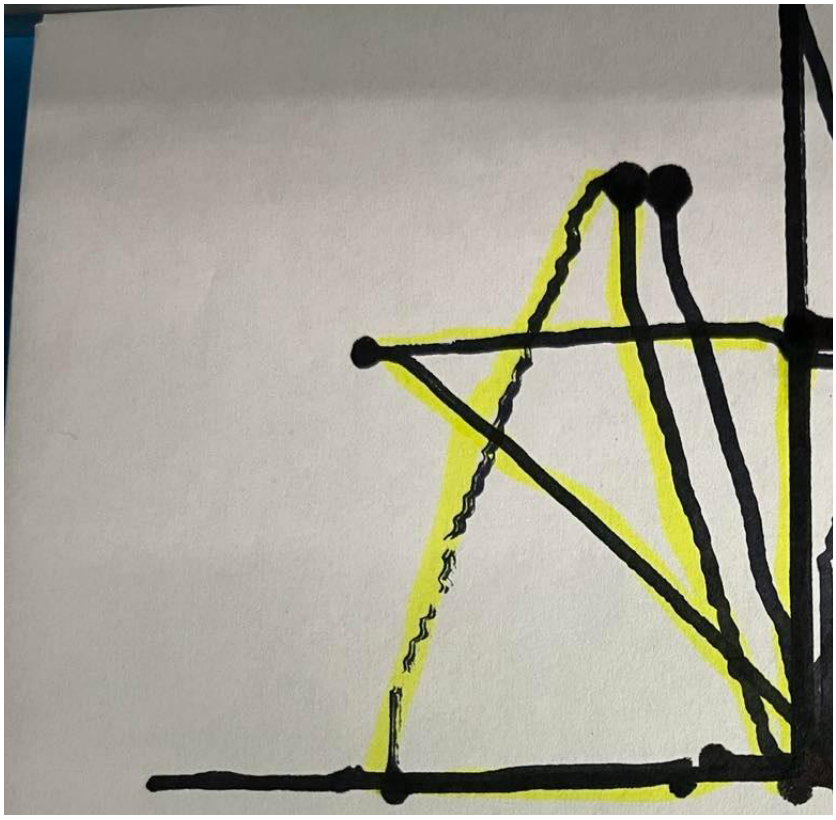


Figure 14. Final Assessment Results

As seen from the results above the performance on straight lines was almost perfect. There was a slight dip in the top right 90deg corner but that was due to the sharpie and carriage's lack of rigidity when in the -X direction. This was also seen in the angled line going from top left to bottom right, and the angled line going from top center to bottom right. There was a short period of lag between when the X axis started moving and the marker started moving along the paper. Those curved effects at the beginning and end of the line were also due to the DC motor not having enough acceleration at the beginning and end to catch up with the stepper speed. This is partially due to extra static friction at the beginning but also due to the lower Kp value we chose to eliminate oscillations.

Discussion

To improve the results a higher Kp value would be used to allow the DC motor to accelerate faster, but an integrator would need to be used to reduce the steady state error and a differentiator would be used to reduce the oscillations and overshoot from the higher Kp and Ki values. The challenge with implementing this would be lack of calculation cycles during each X control loop. This would require a faster clock speed (the MSP430 can run faster than the 16MHz we were using, but would require more tuning of timer interrupt periods to implement) or to optimize the interrupts and run most

of the controls in the main loop of the program, and just switch flags and simple outputs in the actual interrupts.

What are the advantages/disadvantages of using a stepper motor? What are the advantages/disadvantages of using a DC motor? If you were designing a similar machine would you use stepper or DC motors and why? How would you implement a CW or CCW curve? What about a non-symmetrical curve? Discuss in your report.

The advantages of using a stepper motor are high positional accuracy and torque at low speeds. Controlling the position of a stepper motor is much easier than a DC motor although it requires a micro-controller. Stepper motors are unable to run at higher speeds though and are less efficient.

DC motors work well at high speeds and have better efficiency. With a microcontroller and lots of spare time, the DC motor's positional accuracy can be improved using closed loop controls.

If we were to design a similar machine we would use DC motors with encoders. If a stepper motor were to skip a step due to an interrupt getting delayed, or a dip in power supply level, or one of the other many problems we ran into, the position will be off by that amount for the rest of the test. These incremental errors will increase and eventually become significant enough to result in large positional errors. In the case of a DC motor and encoder the control loop can account for errors in delays or overshoot and adjust accordingly. The DC motor also runs accurately at low or high speeds The challenges with the DC motor would be complexity of implementation in hardware and firmware, but the benefits of the control loop outweigh the downsides.

For a circular curve in the CW or CCW direction, the radius would need to be defined as an input. The motor position would be calculated according to the circle equation, $x^2+y^2=R^2$. If using DC motors, a minimum radius would exist where the DC motor is unable to respond to a small enough input. If stepper motors were used with micro-stepping, there would be no minimum input radius as the step size can be as small as desired. The feedback with DC motors would still be more advantageous for higher accuracy.

For non-symmetrical curves, the coordinates of the curve would be split into linear segments and the motors would follow these linear slopes. The size of the segments will be determined by the minimum slope possible with the DC motor. This is because at a low enough input, the DC motor is unable to respond.