Lab 4 Notes – Group 4

June 7th (LAB 1)

ALL INVOLVED

Initial steps for this project was to get the motor running off the bench. There existed an example code the project that ran the motor. The summary of this day was spent attempting to run the motors.

- Ran into an issue where the motor example did not run the motors from the start
    - The problem was that we were calling the motor example before the *BSP_Init*, which first initializes the board. The motor example worked afterwards.
- Spent time attempting to understand the code by going over it line-by-line, reading documentation, and seeking help from the teaching team.
- It was determined that the following function runs the motor:
  StepperMotorBoardHandle->Command->Move(board, device, L6470_DIR_FWD_ID, Step)
  This only moves it by a certain step, as per the function of a stepper motor.
  Found functions such as *Run, Move, Speed, Direction, Hard/Softstop* that allowed further manipulation of the motor.
- **Adonis, Richard**: Determined how the limit switches should be wired onto the board. They were wired from the GPIO pins to the ground.
- It is important to note that the limit switches were normally closed, thus the pinout was configured such that the internal resistor in the GPIO pins were pull-up. This tells us that when the switch is not pressed, a value of zero will be sent, and when the switch is pressed, a value of 1 will be sent. Limit switches were wired to be interrupts, as discussed in the interrupts portion of the lab.
- Another file called "motordriver" was created to contain all the functions of the motor, such as stopping, running, initializing, etc, for organization.
    - There were issues attempting to call the functions in the motordriver.c file. It was difficult to navigate the KeiluVision program to create the necessary .c and .h files while also having them in the correct folder.

June 19th

- **Jonathan**: Created the test cases:
    - Objective 1 test cases – Want to interrupt to stop the motors rather than to reverse so that a false trigger cannot reverse into a hard stop:
    - Let 0 be open switch and 1 be a closed switch. Let the most significant bit be the "forward switch".
    - Expected Results:
    - 00 – The motor continuous in its current direction without stopping
    - 01 – The motor stops, will not move backward
    - 10 – The motor stops, will not move forward
    - 11 – The motor stops, will not move.

June 20th

- **Adonis:** Created the git project. This allows multiple people to access the code on different computers without having the code saved on one computer. Also we get version control in case anything happens.
    - Link to the Github page: https://github.com/adonis-mansour/MTE325-TwoAxisMachine
- **Kalen:** Made variables in C to global and declared the stop function.
- **Jonathan:** Added LSmotordriver function and defined run and stop functions.
- **Richard:** Debugging errors in the code as to why motordriver.c was not working.

June 21st (LAB 2)

ALL INVOLVED

- More debugging. It was determined that one interrupt was never declared, and we corrected how to check the status of the pin. Changed the pins to interrupt on a rising edge.
- It was also realized the GPIOB Pin 6 was a pin used by the board for resets. This was changed to Pin 8.
- The demos for the limit switches in both the axis of the system worked as expected.

July 5th (LAB 3)

ALL INVOLVED

This lab session focuses on the second demonstration. We will be characterizing the speed of the motor. Input will be a function from a user, output will be the speed of the motor in a specific direction.

Procedure of Motor Characterization:

Several attempts to measure the stepper motor RPM were made. Initially the members used a stopwatch and taped the motor shaft with a easily identifiable black electrical tape. The RPM was counted as manually. To increase efficiency, an android application was installed on one members phone, which used the phone's front IR sensor to detect the tape and thereby calculate the RPM. However, the app was ineffective when the rpm exceeded 90. As a final approach, the group used a "Beats per minute" application. Each time the motor shaft turns past a reference point marked on the motor base, a beat was tapped into the app, resembling one rotation. The final approach was proven to be the most efficient and accurate measuring up to the maximum RPMs. all results obtained through previous methods were re-obtained with the final approach.

The first test was done on a motor as a stand-alone system (not on the 2-axis machine). The following results were obtained in the order of the rows presented when the motor was moving FWD:

Input (Motor Speed)	Output (RPM)
5000	23.4
10000	47.3
15000	70.8
20000	93

25000	117
40000	125
100000	125
26000	125

It was also determined that at a speed of 0, the motor does not rotate. At a speed of 10, the rotation is just barely detectable. An input of 217 produces approx. 1 rpm.

From the results, the max output is 125, capped at ~26000 input motor speed.

The reverse direction was also tested with the following results:

Input (Motor Speed)	Output (RPM)
10000	46
25000	116
35000	124
-35000	124 (negative value is ignored)

We see that the input motor speeds in the forward and reverse directions will product similar outputs, thus special accommodations need to be made.

Made a function for the chassis to move back and forth continuously. Something went wrong during the process. Ran the debugger and determined that *BSP_Init* was calling the interrupt handler. Afraid an unknown change was made to a file unintentionally. The code is running however the behavior has changed. Now, whenever the code is run, the interrupts would trigger immediately, even when disabling the interrupts for the limit switches. There is no other output to the putty except for “INTERRUPT EXECUTED” when there is usually a long display of text indicating that the board is starting up.

July 6th

ALL INVOLVED: This day focused on setting up the ADC.

- ADC setup for 12-bit resolution with single conversion mode, so data is retrieved from the ADC by polling for it repeatedly. Most of the other inputs were selected arbitrarily.
- Setup a pin for ANALOG input to read from the potentiometer.
- There was no success in getting any kind of readings from the potentiometer. A meeting with the prof. was set up to discuss possible issues.

July 9th

ALL INVOLVED: Meeting with the prof. after class to figure out ADC issues

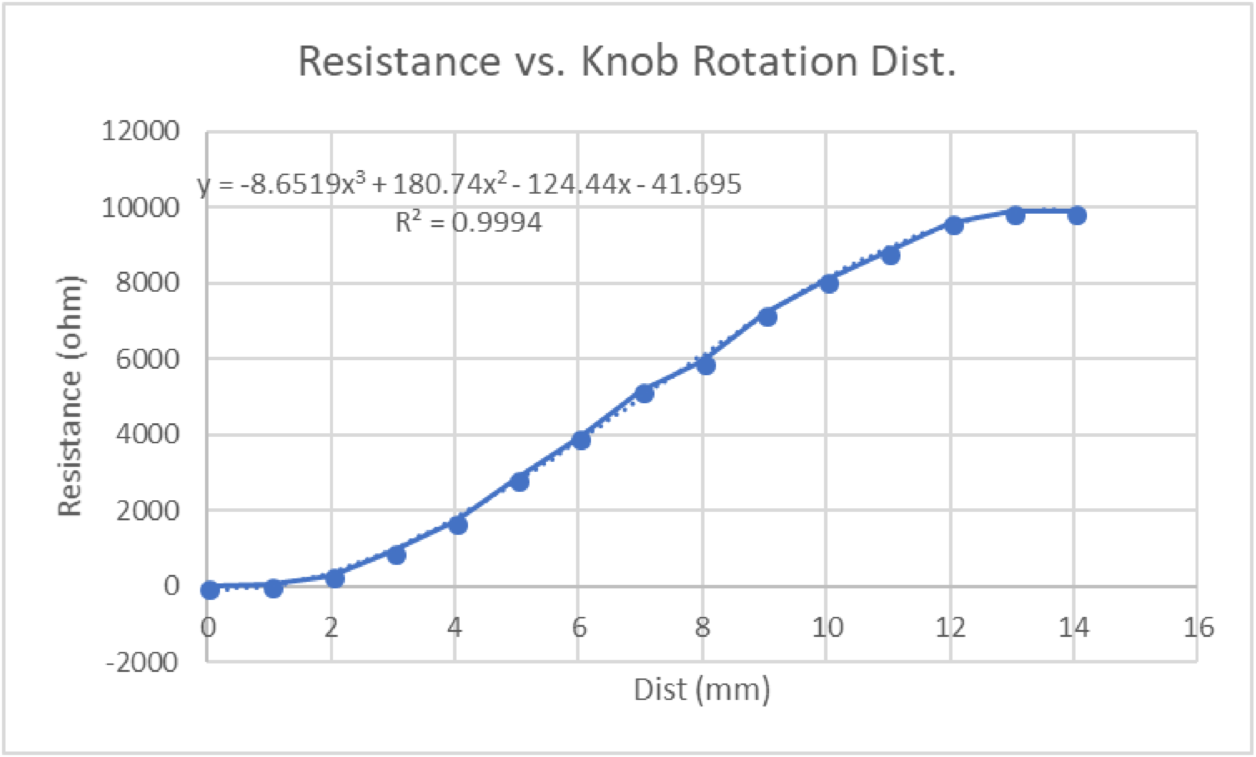
- It was found that the correct pin channel was not selected, which can be found from the documentation on the board. Also, we were not calling the Read_ADC function properly. Previously, we had the following:
  HAL_GPIO_ReadPin(PGIOA, GPIO_PIN_6); // Attempt to get a reading from the potentiometer.
  The fix was to call the function as: adcValue = Read_ADC();

July 13th

Adonis/Jonathan: This day was focused on characterizing the potentiometer.

The potentiometer was characterized based on how the resistance changes as it is rotated. A digital multimeter was used to measure the resistance. A string was tied around the knob of the potentiometer and the distance was measured using a ruler as the knob was rotated. The results are as follows:

Dist (mm)	Resistance (Ohms)
0	3.3
1	34
2	270
3	925
4	1717
5	2850
6	3940
7	5160
8	5930
9	7200
10	8100
11	8840
12	9600
13	9900
14	9900



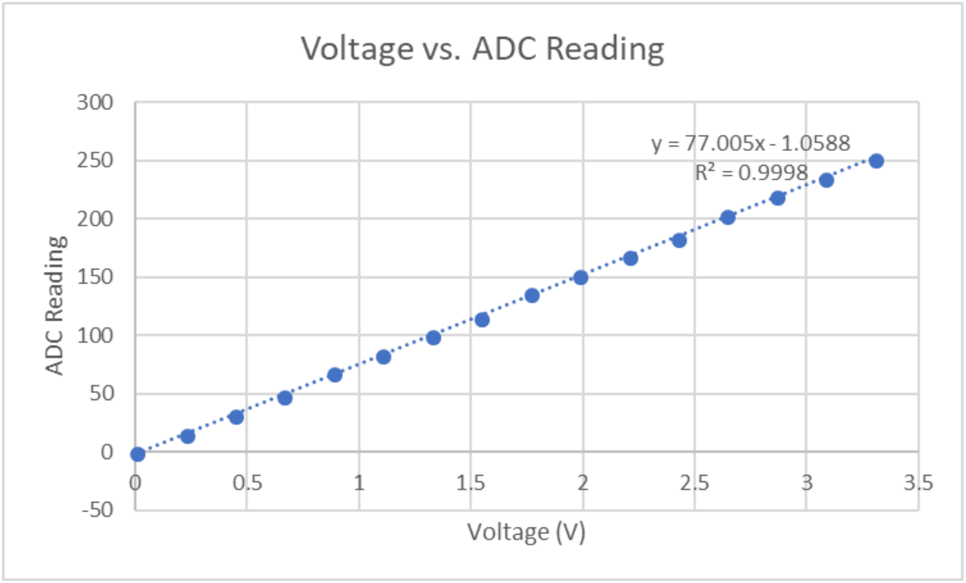
July 18th

ALL involved.

This day was focused on characterizing the ADC and creating a function to map the ADC readings from the potentiometer to a motor speed value.

The potentiometer was characterized based on how the Vout readings from the ADC vs the voltage input. The voltage in was controlled with a potentiometer and the voltage input was confirmed with a DMM. The ADC outputs values from 0 - FC in increments of 4.

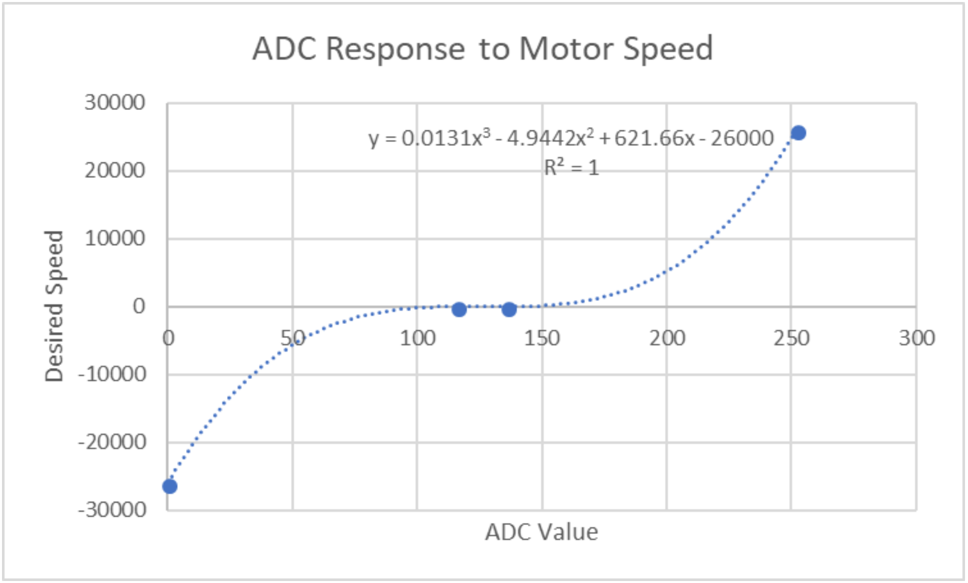
Voltage	ADC Reading	ADC Hex
0	0	0
0.22	16	0010
0.44	32	0020
0.66	48	0030
0.88	68	0044
1.1	84	0054
1.32	100	0064
1.54	116	0074
1.76	136	0088
1.98	152	0098
2.2	168	00a8
2.42	184	00b8
2.64	204	00cc
2.86	220	00dc
3.08	236	00ec
3.3	252	00fc



The next step was to finally relate the potentiometer input to a speed value. We saw that the ADC was linear, thus no accommodations were necessary to account for it. On the other hand, the potentiometer had a slight plateau to it. We had the option of accounting for the potentiometer plateau, but we just assumed it to be linear to make calculations simpler. Therefore, it was concluded that no necessary accommodations were needed for the ADC and the potentiometer since they were considered linear.

We also wanted a deadspot in the potentiometer where it would give a value of 0 for the motor speed and then slowly ramp up to the max motor value once the potentiometer reaches its extremes. We characterized a function based on the following:

ADC	Speed
0	-26000
16	
32	
48	
68	
84	
100	
116	0
136	0
152	
168	
184	
204	
220	
236	
252	26000



The deadspot can be seen around the 116 and 136 marks. The reason for this is to provide an appropriate range on the potentiometer knob where the motor will not be running.

Finally, the functions to convert from the ADC to the motor speed were made in code.

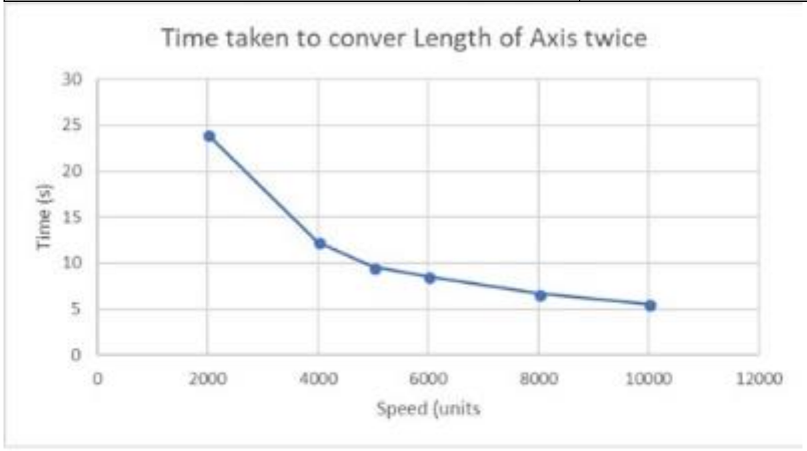
July 19th (LAB 4)

- The motors were still not running ever since the end of LAB 3 (July 5th). To recap, the issue is that we cannot get any part of our code working without first triggering an interrupt and the program locking itself in the interrupt handler without escaping.
- We asked a TA for assistance, and after some time, it was finally determined that there are other processes that also use the interrupt handler. In our interrupt handler, we just wrote the code to stop the motor, but after changing it to the following:

  ```
  if (GPIO_Pin == GPIO_PIN_8 || GPIO_Pin == GPIO_PIN_6)
  {
          stop_motor();
  }
  ```

  Where we add an if statement to check for the triggered GPIO pins, the code runs without any problems.
- We also tested the potentiometer and ADC and it works with the motors as well.
- We finally resumed the characterization of the motor on the 2-axis machine, a continuation from July 5th (LAB 3) where we only tested the motor as a standalone system. The difference is rather than counting the rpm, we timed the platform to complete one cycle. The following are the test results:

INPUT (Motor Speed)	OUTPUT (Time, sec)
2000	24
4000	12.3
5000	9.6
6000	8.6
8000	6.7
10000	5.6
15000	Too fast, limit switch was not able to react in time

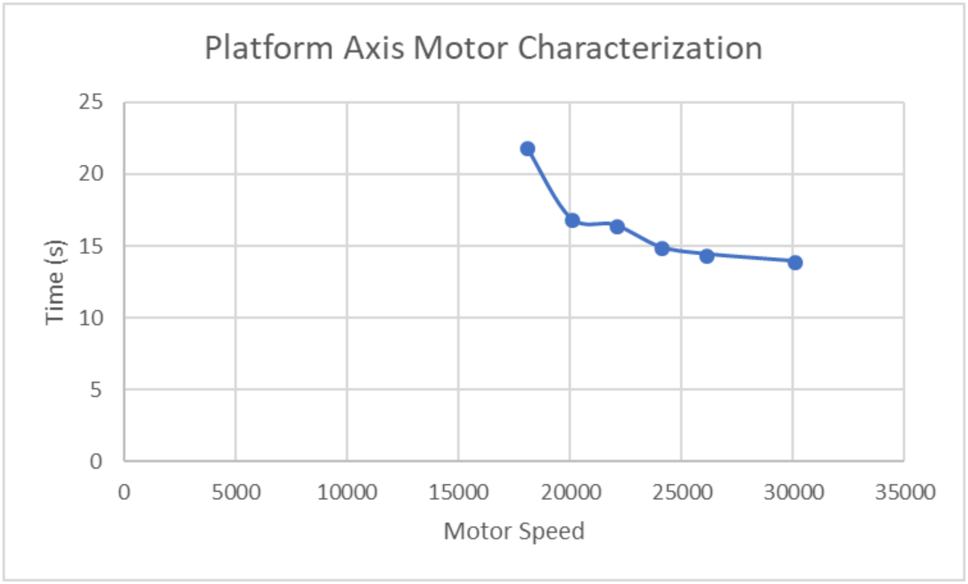


From the graph above, the motor speed reaches a plateau after an input of 10,000. The motor takes exponentially longer to complete a cycle as the speed is decreased.

Furthermore, the test was changed to go from speed of 2000 to 10000 at 2000 intervals rather than up to 26000. We set the speed of the motor to 15000, but it was hitting the limit switch too hard from our observations. We concluded that a limit of 10000 is appropriate. This changes our potentiometer to motor speed value since we will have to modify our code to not go over 10000.

We also performed the motor characterizations on the 2nd axis of the 2-axis machine to try and put our hand into the bonus. The difference is that we only counted the time for a completion of half the cycle. We also used high motor speed inputs (> 18000) since any lower values will take some time to complete. We ended up with the following results:

INPUT (Motor Speed)	OUTPUT (Time, sec)
18000	22
20000	17
22000	16.5
24000	15
26000	14.5
30000	14



There is a little more error in the data compared to the 1st axis, but the general trend is still seen. There is an approximately exponentially decaying relationship to the curve, and the plateau is still visible after the 30000 motor speed mark.

Finally, we probed the USB serial bus to the submission portion of the lab.