0.1. PCB

- *LED Demux:* Our circuit board's primary purpose is to enable us to indicate our robot's status (Error, Ready, Linear Closed-loop, PWM, Find Targets Velocity, Angular Closed-loop, and Waiting) visually, while occupying only 4 output pins on our microcontroller. Our PCB incorporated a 3 to 8 demuxer, a resistor, and ports for 8 LEDs to achieve this. The final PCB directed 4 inputs from the arduino to A0, A1, A3, and OE0 on the demux. LE and OE1 were also required to be grounded to logical low for the demux to work correctly.

'HC237, 'HCT237 TRUTH TABLE

INPUTS						OUTPUTS							
LE	OE_0	$\overline{OE}_1$	A_2	A_1	A_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
X	X	H	X	X	X	L	L	L	L	L	L	L	L
X	L	X	X	X	X	L	L	L	L	L	L	L	L
L	H	L	L	L	L	H	L	L	L	L	L	L	L
L	H	L	L	L	H	L	H	L	L	L	L	L	L
L	H	L	L	H	L	L	L	H	L	L	L	L	L
L	H	L	L	H	H	L	L	L	H	L	L	L	L
L	H	L	H	L	L	L	L	L	L	H	L	L	L
L	H	L	H	L	H	L	L	L	L	L	H	L	L
L	H	L	H	H	L	L	L	L	L	L	L	H	L
L	H	L	H	H	H	L	L	L	L	L	L	L	H
H	H	L	X	X	X	Depends upon the address previously applied while LE was at a logic low.							

H = High Voltage Level, L = Low Voltage Level, X = Don't Care

Figure 1: Truth table for the demux on our PCB. Inputs from A0-A2 set which output Y0-Y7 to set.

- *VCC and GND Rails:* The PCB also cleaned up our robot's wiring by supplying 5V power rails for our arduino and motor encoders. These power rails were 8 pin headers connected together by 17 mil traces. Our trace width of 17 mil we determined using an online calculator. We set the width so that a 1.4 mil thick copper trace would have a 10 degrees farenheit increase in temperature when conducting 1 Amp at 70 degrees farenheit. 1 Amp is sufficient to handle the Arduino's max current of 1 Amp, in addition to the comparitavly small current requirements of the demux and encoders.

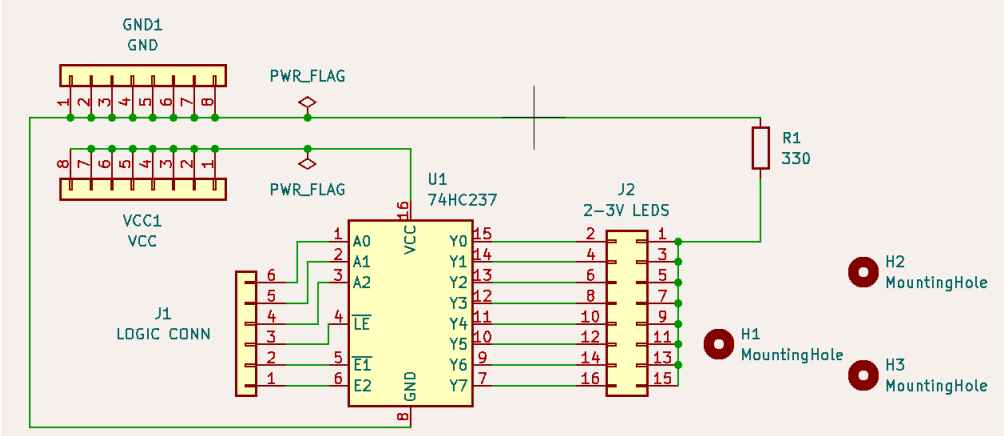

Figure 2: PCB schematic from KiCad

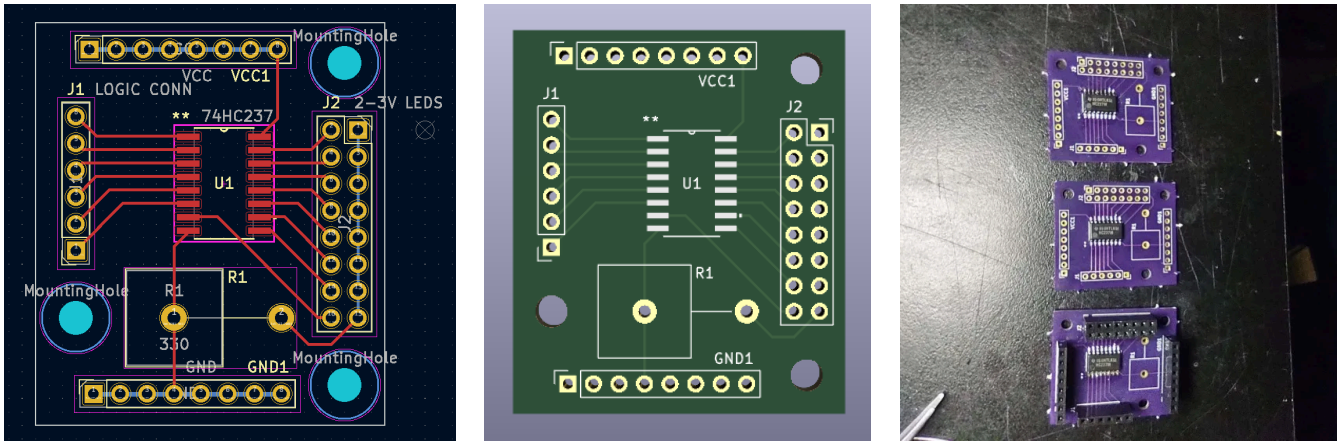

Figure 3: PCB design, render, and final result

0.2. Serial Communication

We designed and implemented our own two way serial communication protocal, using Python's struct library as a basis. With struct, we defined 11 message types where each message would begin with a byte that specified which message type was being sent, and end with the n character. Python and Arduino's serial libraries already handled buffering for us, so for both we would read a line of bytes from serial, and use Python's struct and C++'s packed structs to decode the sequence of bytes into bytes, ints, and floats. Then we can define specific behavior for each message on a switch statement for the first byte in each message.