**Background**

The XR-3 Rhino Arm, built back in the 1980’s, was used by universities to teach basic robotics.

Someone gave the Osbourn High Robotics team the Robot Arm back around 2013. It sat untouched in our storeroom until 2023 when I decided to build a control system for using a combination of Arduinos and Raspberry Pi micro controllers.

**Goal**

The goal of this project is to provide a programming challenge for advanced robotics students.

**Description of the Robot Arm Hardware**

The robot arm has 6 joints (Waist, Shoulder, Elbow, Wrist elevation, Wrist rotation, Gripper) where each joint has a small motor, shaft encoder and a position reference point switch. Each robot arm joint can be moved to a specific position by first moving the robot to the reference point then use the shaft encoder to count the number of motor shaft rotations to reach the desired set point.

**Description of the Replacement Control System**

The design of the replacement control system uses two Arduino micro controllers to perform the real time (high speed) tasks such as monitoring the shaft encoder and reference point switch signals and control the joint motors. Each Arduino handles 3 robot joints. The Raspberry Pi provides high level commands to the Arduinos. Commands are like, “Move Joint A to the home position (reference point)” and “Move Joint A to 30 degrees (from the home position)”. The RPI communicates with the Arduino over an I2C bus (Inter-Integrated Circuit). Communication is two-way; commands are sent to the Arduino and responses are provided back to the RPI. Since the Arduino’s digital input/output ports cannot provide sufficient voltage or current to drive the joint motors, H-Bridge integrated circuits control the power to the motors. The H-Bridge allows the Arduino to control the direction and speed of the motor. The speed of the motor is controlled using pulse width modulation (PWM) fed into the enable signal of the H-Bridge.

A last-minute addition to the RPI was to add an I2C based OLED display (128 x 32-pixel display). The intent of the display is to provide an easy display of the status of operations in the RPI. The initial implementation of the Rhino Control python code does not use this feature. Test code has been written to write text to the display.

**Details of control system**

**Notes on the Robot Joint Electronics**

The Arduino inputs from the home position reference point switch are physical switches to ground. When the arm moves to the home position, the “Limit Switch” signal is connected to ground. With the Arduino input configured with the internal “pull-up resistor” enabled, the input signal will be high (5 volts) when not at the home (reference position) and 0 volts when the home position is reached.

Similarly, the joint shaft encoders signals are open-collector connections. This is the output of an LM393 comparator integrated circuit. This is like a physical switch to ground switching between 0 and 5 volts with a pullup resistor enable.

The connections to the motors are doubled up. Two ribbon cables wires to both the positive and negative sides of the motors.

**How the Arduinos are used**

One Arduino controls arm joints A, B, and C and the second controls arm joints D, E, and F. The software is designed to allow each controlled joint to have different parameters such as range of motion and speed. The joint parameters are coded into the Arduino software. To allow each Arduino to know what joints it is controlling, a single input (A3) is connected to either high or low using header pins and moveable jumpers. This input signal is also used to create the I2C address.

The I2C communications connection between the RPI and Arduino required some additional thought since RPI GPIO signals use 3.3-volt signals and the Arduino uses 5-volt signals. Research indicated when the RPI is the master and Arduino is the slave, there is no problem. The I2C bus uses open-collector connections and the communicating devices pull the data and clock lines to ground. The source voltage is provided by the master device. In this case the RPI pulls the bus up to 3.3 volts.

The Arduino Tx and Rx signals are brought out to a header. This allows the Arduino monitor to be read into the RPI. This information is not needed to control the robot arm but is used for troubleshooting. Since the RPI can only accept a single RS-232 interface, the two Arduino monitor output signals are merged with a logic gate. This does cause failed communications at times but it was easy to implement.

The H-Bridge controls the power to the motors. It provides pulse with modulated (PWM) voltage to the motors.

Since the input control signals to the H-Bridge (pins 10,15 and 2,7) will always be opposites, a digital NOT gate (7404) was used to control the second input. This saved using an Arduino output pin (which were in short supply).

The H-Bridge Enable signal (pin 9) needed a pull-down resistor to prevent uncontrolled movement. It was observed that the Arduino’s inputs/outputs go to high impedance (basically not connected) when the Arduino initializes. This brief disconnection caused the H-Bridge to be enabled momentarily.

**Circuit Boards Details**

The were custom made by a Chinese PCB maker called EasyEDA (<https://easyeda.com/>). A Robotics team student told me how easy and inexpensive the process is. (About $25 for 5 circuit boards delivered in 10 days). The process involves drawing the circuit in an intuitive circuit CAD program then laying out the parts on the circuit board. The online tool handles the routing of the PCB lands. Unfortunately, an error was made when creating the schematic and an output signal of an IC was tied to ground. I think this was caused when I relocated a component in the schematic and the tool inadvertently connected the signal. To resolve this issue, I cut a couple of lands and added jumpers to resolve this problem. The circuit boards are the same. Header pins are used on the board to access signals.

**Power System:**

The system runs using 5 volts for the Arduino and RPI and 12 volts for the motors. Each board is designed to accept power using standard connectors. A power bus is exposed using a ribbon cable header to allow power sharing between the two circuit boards. The design original included an old-style voltage regulator (7805) to create the 5-volt system from the 12 volt supply. This was replaced with a more efficient Power Converter. The system can now be run with a single 12-volt supply for the motors and DC converted down to 5 volts for the Arduino and RPI. Testing showed the motors only draw about 0.3 amps when moving.

**Notes on the Software**

The Arduino’s code uses the C language and the RPI is using python.

The current Arduino and RPI code are very inefficient. The code has a significant amount of “near duplicate” functions which could likely be combine. The plan is to rewrite the code once the prototype code is functional. I always struggle trying to decide if I should have near duplicate functions or increase the complexity of a function so can handle more capability.

The Arduino code uses the I2C communications to trigger events. When the RPI sends a message, the “receiveEvent” routine is called and the Arduino code reads the command from the RPI and implements it

**Software Commands between RPI and Arduino**

|  |  |  |
| --- | --- | --- |
| **Command** | **Number** | **Notes** |
| Stop all | 1 |  |
| Home all | 2 |  |
| Test Home Switches | 3 |  |
| Find Home for all | 5 |  |
| Self-Test All | 6 |  |
| Home One | 10 | Moves toward home until limit switch detected |
| Set Angle | 11-xx | Not started |
| Set Count | 12-xxxx | Moves to count (Requires home prior to use) |
| << no menu >> | 13 | Move to full range position |
| << no menu >> | 14 | Home, full range, then home again |
| Find Home | 15 | Moves toward home, then away until limited then back to home |
| Self-Test | 16 | Call command 18, 19, then 17. Returns one second counter.  (Results are not displayed currently) |
| Get 1 Sec | 17 | Pulls the one second counter used with command 19 |
| Man-home | 18 | Move toward home (lower counter) for one second |
| Man-Out | 19 | Move away from home (lower counter) for one second.  (Creates one second counter in Arduino code) |
| Get Status | 90 | Read the counters from the arm |
| Get Counts | 91 | Read the status from the arm |

**Diagrams at the End of This Document**

Hand drawn schematic

CAD schematic

Board Layout

PCB error corrective actions

H-Bridge theory diagrams

Robot Joint Ribbon Connector pinout

**External References**

How to have the RPI display the Arduino Monitor output

<https://roboticsbackend.com/raspberry-pi-arduino-serial-communication/>

H-Bridge basics

<https://www.modularcircuits.com/blog/articles/h-bridge-secrets/h-bridges-the-basics/>

**Colleges which used the Rhino Arm as a project**

Portland State University

<https://web.cecs.pdx.edu/~mperkows/CLASS_479/PSUBOT2/1.%20Rhino%20Arm%20Info%202010.pdf>

<https://web.cecs.pdx.edu/~mperkows/CLASS_479/PSUBOT2/3.%20Rhino%20Arm%20Report,%20Jeff%202010.pdf>

<https://www.studypool.com/documents/8062061/link-diagrams-of-rhino-xr-3-and-scara-using-robotics-toolbox-by-peter-corke->

**Old Robots**

https://www.theoldrobots.com/rhinoarm.html

<https://www.theoldrobots.com/book80/Rhino_XR.pdf>

**Future Software Updates:**

Tolerate and report failed I2C connection

Use OLED display

Support comments in the command file

Add a pause command for command file

Revise to support multiple joints moving

**A few notes on Establishing the RPI**

Use RPI imaging tool – PRESET DATA

WIFI INFO, Status address, Root account and password, I2C bus

Enable VNC

Once started, updated Add libraries:

Set VNC password

1. Load Image
2. Configure VNC
3. Update software
4. Enable busses (GUI config tool)

Installing to use I2C. [Source: https://www.instructables.com/Raspberry-Pi-I2C-Python/]

sudo apt-get update

sudo apt-get install i2c-tools

sudo apt-get install python-smbus

sudo adduser pi i2c.

i2cdetect -y 0

>> Needed for RPI-2B

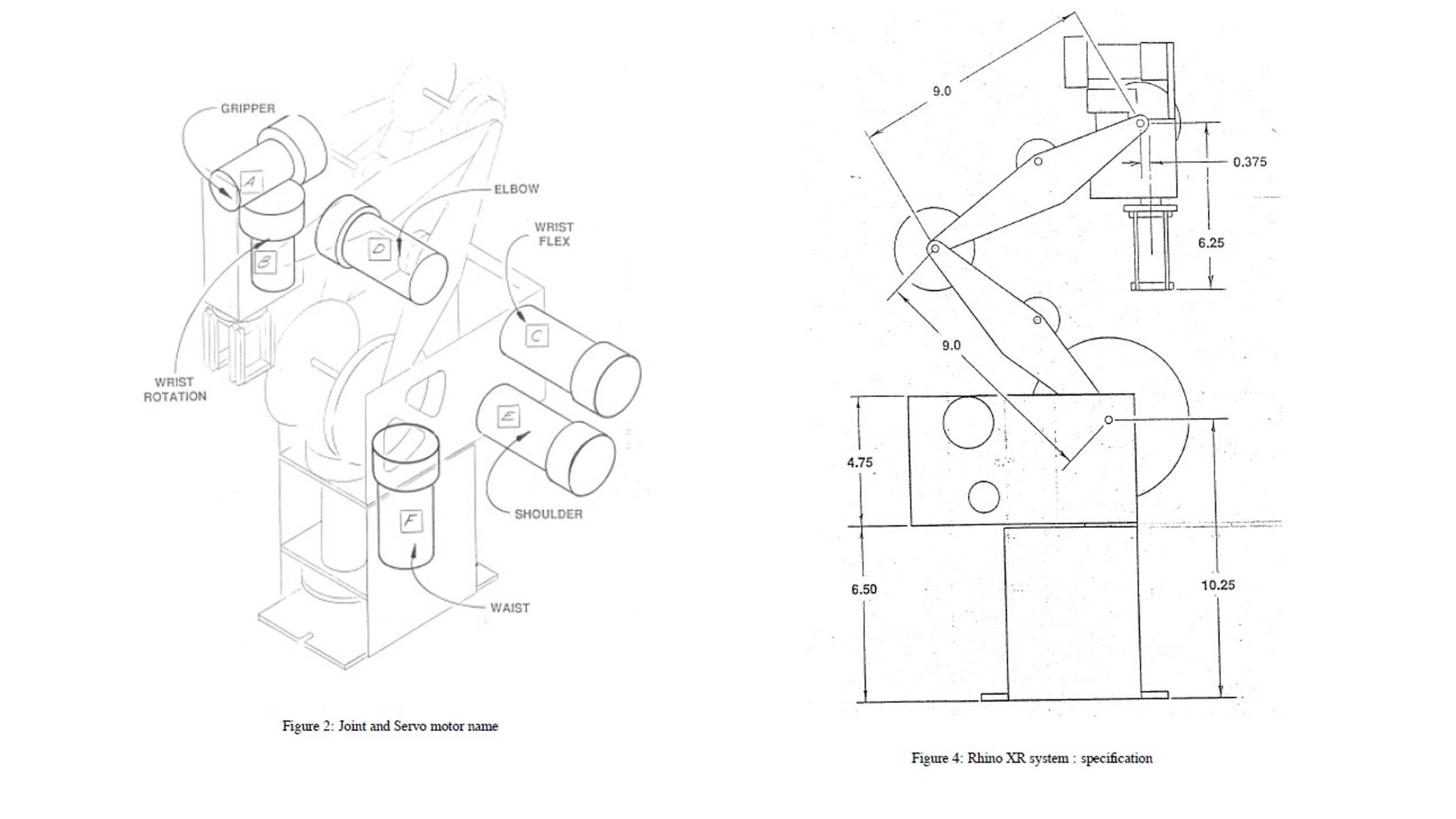
sudo vi /boot/config.txt

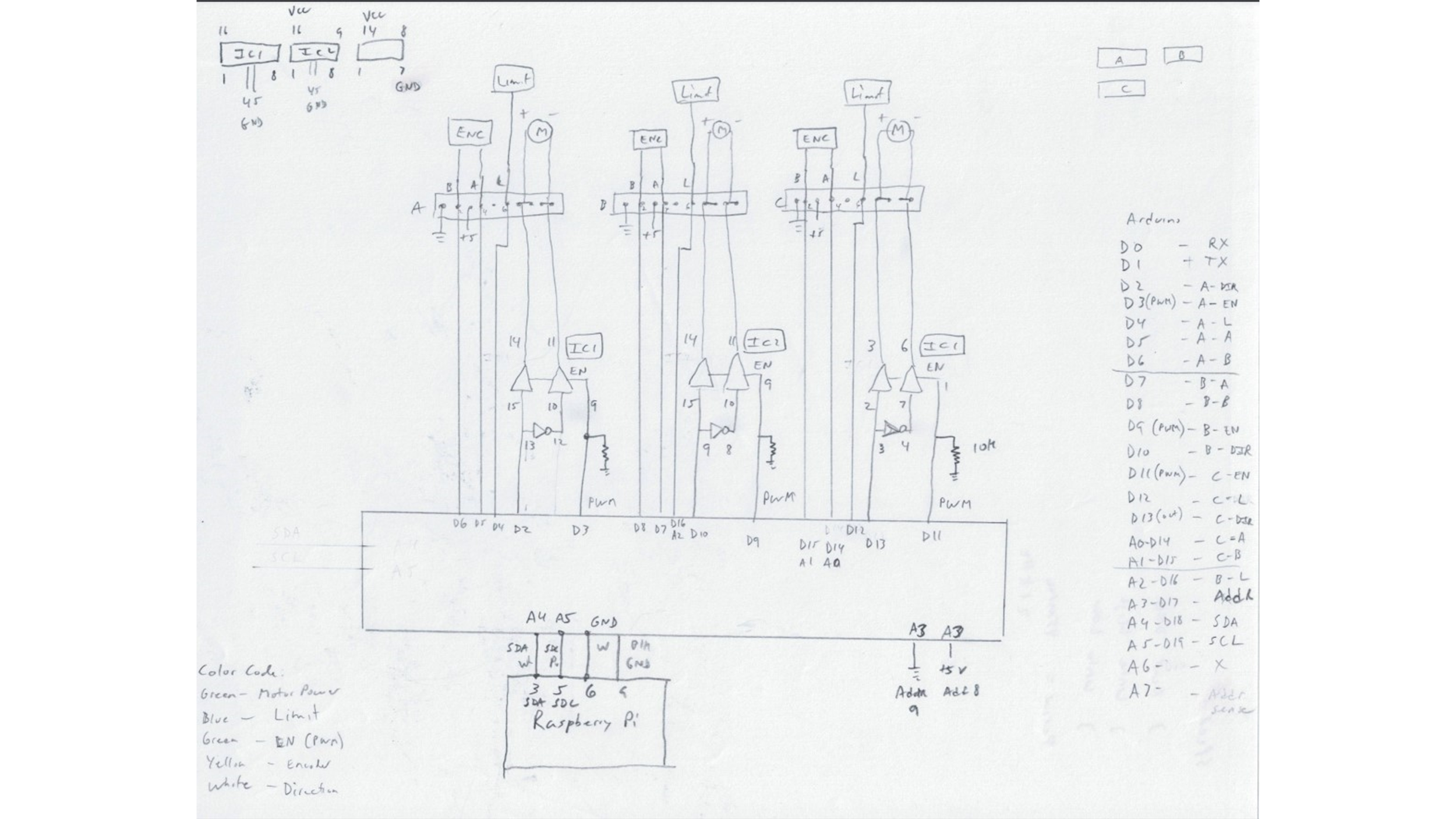
Updated line to “dtparam=i2c0=on ”

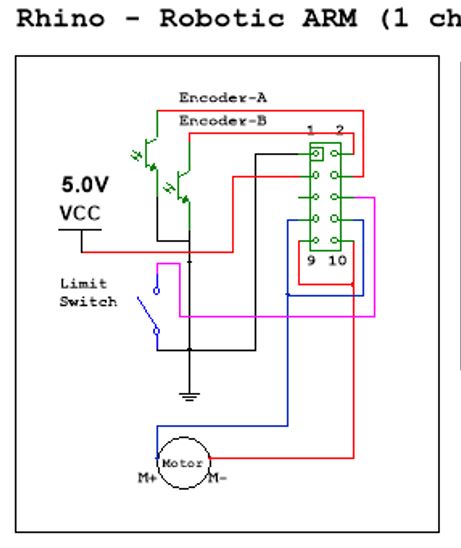
and reboot.

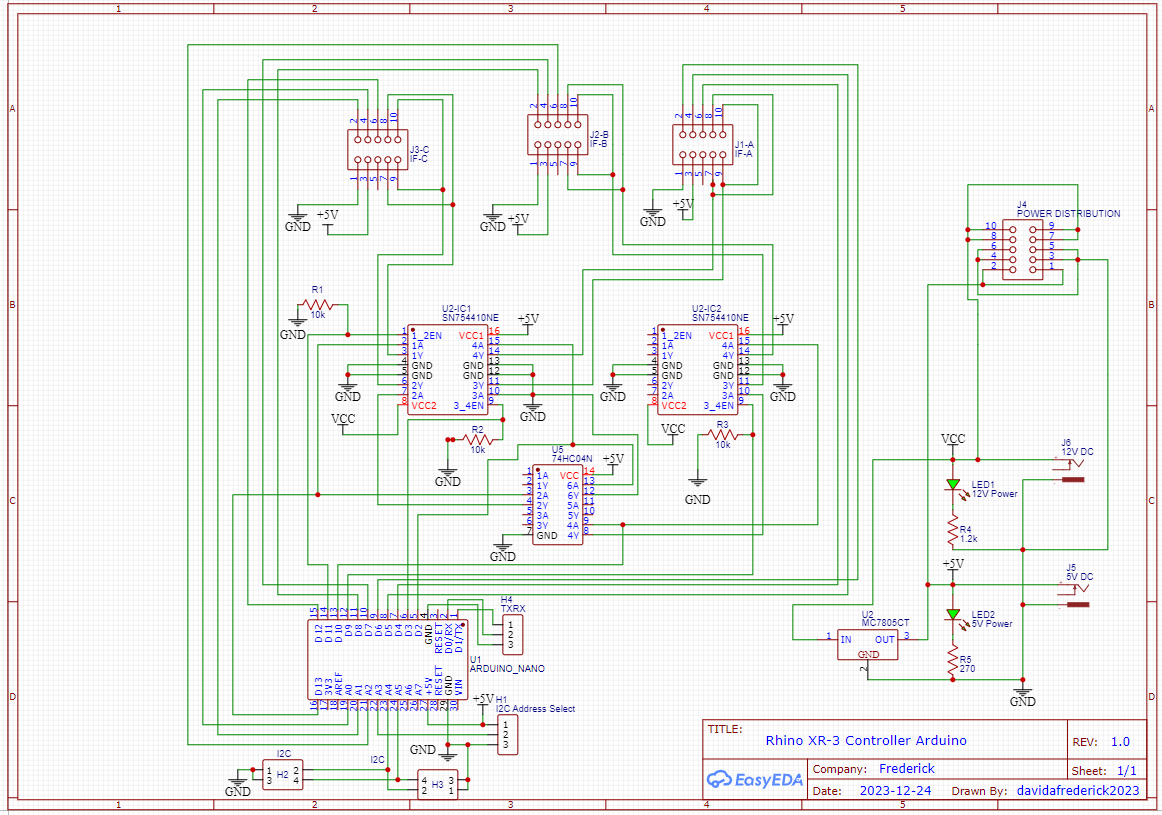
I2C

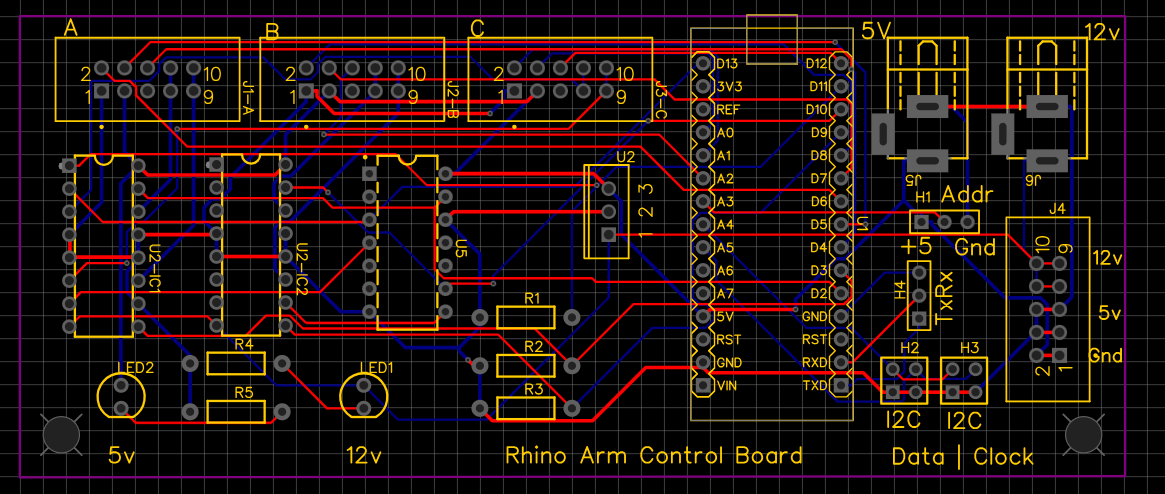
Add dtparam=i2c1=on (or dtparam=i2c0=on on old models)

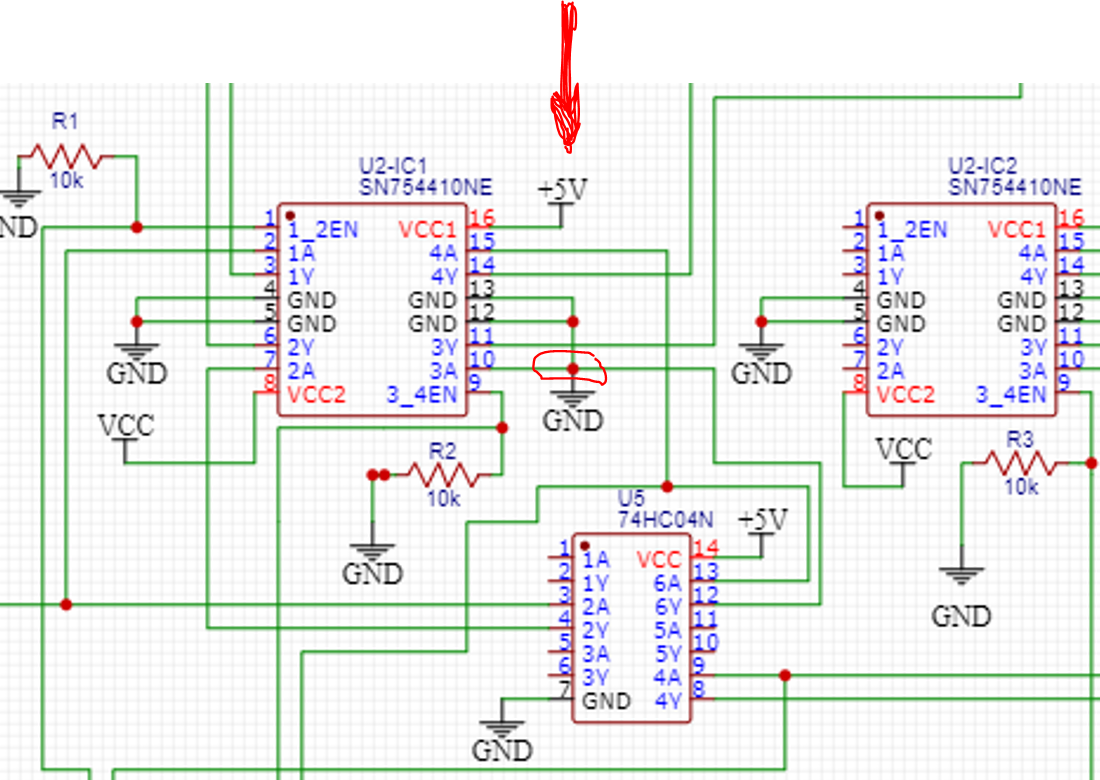












*Error in the Board Layout process. For some reason, Pin 10 of IC-1 was grounded during the schematic creation*

