

Smart Motor Syringe System

Connor Wilson, *Member, IEEE,*
EE Dept. Senior Project, *California Polytechnic San Luis Obispo*, Winter 2022
Advisor: Dr. Benjamin G. Hawkins

Abstract—In this document a proof-of-concept design is developed and implementation to showcase the latest development in an ongoing effort to produce a series of automated syringe pumps.

I. INTRODUCTION

HERE is a desire to control flow rates in syringes in a steady and automated manner in the medical field; whether that be IV fluids or micro-fluids. According to PR NewsWire, the automated syringe pump market is currently valued at \$4 billion USD and is set to double its growth by 2030 [1]. These numbers suggest administering fluids in a controlled manner is a profitable and valuable piece of technology.

However, this task is difficult to do manually, requiring precise digital control to achieve success. Because this device is related to medical and bio-engineering research, greater emphasis is placed on maintaining consistent flow rates and precision control. To achieve this key feature, typical syringe pumps utilize the precision of micro-controllers, large stepper motors, and reduction gear ratios in the drive train.

With the introduction of precision control, keeping this project low cost is a secondary goal to make medical grade technology readily available. While existing products in the market are numerous, they are traditionally priced above \$1000 USD and are not easily programmable nor user friendly. Using readily available, non-proprietary hardware and software, I aim to provide reliable automated syringe systems for a fraction of the cost. This system of syringe pumps will also be controlled from a single controller, allowing the user to operate without manually addressing individual devices. This hub controller will add hierarchy that provides a more native and intuitive interface for the user, saving time and other resources used during setup.

It should be noted that my contribution to this project does not constitute a finished product but rather an ongoing pursuit for the minimum viable product as defined by Doctor Ben Hawkins.

II. BACKGROUND

A. Purpose of a Syringe Pump

A syringe pump (or driver) is defined as a “small infusion pump, used to gradually administer small amounts of fluid (with or without medication) to a patient or for use in chemical and biomedical research.” [2] The need for slow incorporation of fluids is used in many applications. Intravenous (IV) therapies, for example, require syringe

pumps in order to deliver medication over the course of several minutes or hours. Automating these doses removes most human error in high-risk procedures while saving the medical staff man-hours.



Fig. 1: Example of a Syringe Pump's IV Application [3]

Additional uses for syringe pumps lie in micro-fluid research applications such as: enzyme kinetics, chemical synthesis, and potentially dangerous chemical reactions. For this project, the immediate use of this product will be for Dr. Ben Hawkins's research in enzyme solution gradients.

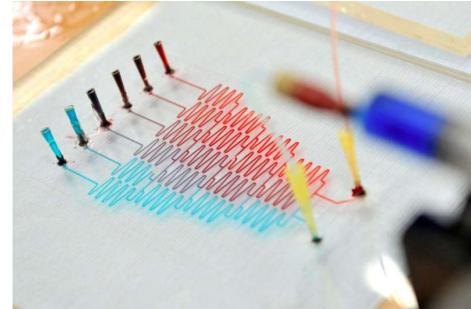


Fig. 2: Example of Microfluid Application [4]

B. Description of the Smart Motor Syringe Pump

The SMSS (Smart Motor Syringe Pump) performs as the defined syringe pump, including but not limited to: output accurate/precise flow rates, display input/output hardware parameters, and control several syringe pumps, all while remaining under \$500 USD.

The SMSS also aims to solve two problems, the first being inconsistent flow rates. Looking at existing technology, most low end pumps work on the basis of summing impulses over time such as the NE-300 in Figure 3. An ideal pump would have smooth flow and quick transient responses. However, the main concern for the SMSS in this stage is to maintain a smooth flow rate and address sluggish response times in another prototype. Quintessentially, the SMSS is

built to output flow rates that are precise and accurate. An understanding of the system's plant and controller interface is required to operate within the desired tolerances and well worth the time investment. This will be discussed later in the paper.

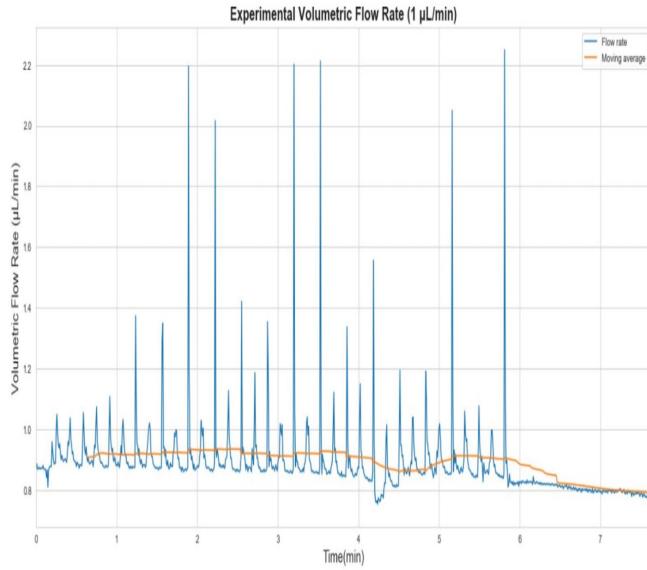


Fig. 3: Output Flow from a NE-300 Syringe Pump [5]

The second problem arises with the addition of other syringe pumps: multiple pumps will often require a synchronized start/stop time from which to collect data from. The main/sub hierarchy implemented aims to synchronize the start/stop of several units and eliminate the errors presented when triggering each pump manually.

Looking broadly at this project, the SMSS aligns with a research market: laboratory applications that deliver small amounts of fluid while collecting data on the system. It should be noted that while syringe pumps can differ in applications, their internal structure is the same: they all consist of a stepper motor, pusher block, and a syringe holder (Figure 4).

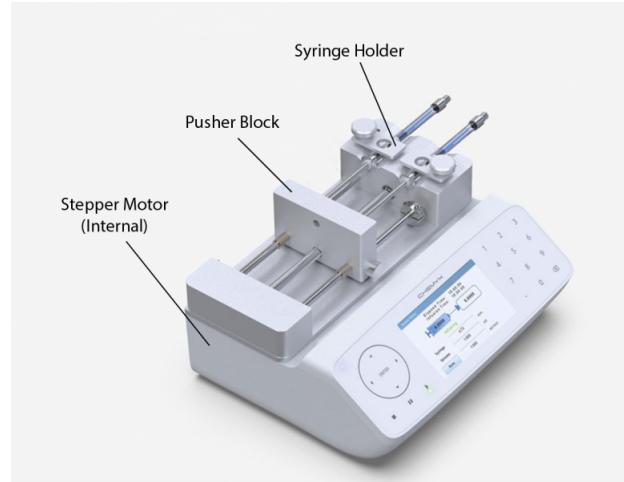


Fig. 4: Components of a Syringe Pump [6]

The SMSS makes use of the Raspberry Pi 3's computing power by implementing a display for the user; a feature forgotten on some devices on the market. The user interface prompts for user requests but also informs the user of resulting error codes, run time, and hardware issues with their requests.

III. PRODUCT PLANNING AND RESEARCH

Evaluation of the SMSS and its viability as a product requires an analysis of existing technology on the market. Only when I looked at the existing products could I determine the engineering requirements and priority of each.

A. Marketing

This subsection is focused on the marketing aspect of a syringe pump in an attempt to translate customer needs into a viable and cost effective product.

1) Marketing Research: Using Google's tool for search trends, there was a low number of searches in the US and internationally for syringe pumps. This signals that the amount of new customers looking to purchase an SMSS is small, or that existing customers in the market do not use Google to search for new products. These customers might look to a medical supplier or academic group when looking for a new syringe pump. The Dublin PRNewswire [1] has stated that the IV pump market is valued at \$4 billion USD (in 2020) and is expected to double by 2030. This speculation is pre-COVID but it does hold some merit. Looking at Figure 5, there appears to be a larger customer base outside the US; range for Figure 5 is $\in (12-211)$ hits in January 2021.



Fig. 5: Global Google Search of "Syringe Pump" 2021 [7]

Further information was gathered in Figure 6 when searching for syringe pumps: there are many applications and characteristics for existing products on the market. My sole customer's [Dr. Hawkins] request for a research-oriented device represents smaller portion when compared to infusions, but it should be noted that the SMSS can achieve the minimum viable product for multiple applications. Qualifying for various FDA medical device classes will be left up to the next iteration of the SMSS.

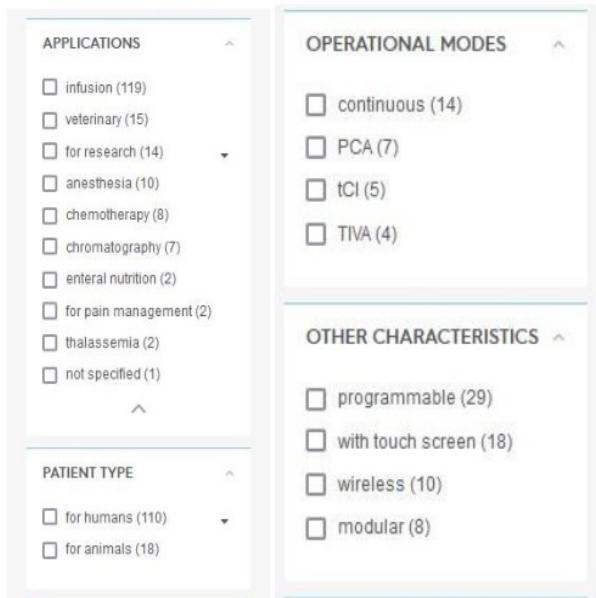


Fig. 6: Options for Pumps in MedicalExpo.com [8]

2) Customer Archetype: The Smart Motor Syringe System has many established competitors in the market but none have a portion over 40%. All of these competitors have costly devices nearing \$5000 USD but come with features such as failure alarms, custom fluid capacity syringes, unique flow patterns, and battery backup power. However, after viewing a handful of device data sheets on MedicalExpo.com [8], very few competitors provide testing data or flow rate tolerances. With our main customer being research oriented, the sensitivity of the pump's output is important. I believe bolstering and advertising the engineering requirements on top of the SMSS's steady output as the minimum

viable product will carve out a spot in the syringe pump market.

The table below provides three examples of the customer base.

TABLE I: Customer Archetype

Customer	Description	Reason	Product Use
Research Student	This group is likely to adopt new technologies with an emphasis on easy-use DIY and low cost.	This group aims to utilize the minimum viable product aspect while remaining open to learning experiences around DIY and emerging tech. The ability to buy multiple units and sync them is important for testing data.	This group will utilize this product frequently but not in a commercial setting. I can expect his product to remain relevant for a long period until replacement over the course of a decade.
Remote Medical Worker	These customers are looking for a robust solution that can be torn open and repaired. Cost is likely to be a factor.	Practicing medicine in remote locations means that supplies are limited. The need for medical attention will still require medical equipment.	This group will likely run the equipment into the ground and replace parts themselves. Cost is a strong factor as well as robustness and open-source likeness
Third-World Medical Center	These customers care most about the cost of the product as well as the functionality. MVP and functionality are critical	Currency may not be equal to USD and require a large portion of the budget to buy basic medical equipment.	[Same as above]

The following table outlines competitors in the market. It should be noted that most companies are overseas and have a large pool of products related to the medical field. This list is not shy of catheters, IV fluids, syringes, thermal components, 3-D imaging machines, and other augmented devices. One example is Norgren, a large Switzerland company. They have an international presence here in the US that shouldn't be excluded when researching domestic markets in the US. I should mention that it is not easy to track companies as a whole entity, but rather its branches and partners. Wellead is an example of a company here in the US that is difficult to contact or buy from locally. I believe a portion of these companies have third parties to sell and establish their branding within the medical market.

The table below provides four examples of competing companies.

TABLE II: Market Competitors

Market Leader	Description
COMEN [8]	Specializes in veterinary medical devices (Located in China)
wellead [8]	Specialized in a wide variety of syringe pumps and medical equipment. One of the top 10 medical companies in this field. (Located in USA)
MedRena [8]	A small company with a good product line. (Located in China)
NORGREN [8]	A pipetting company that makes channel pumps and syringe pumps. (Located in Switzerland)

3) *Marketing Requirements:* The SMSS's marketing requirements are derived from the business model and marketing data model in Appendix D Figure 47/48 with Professor Ben Hawkins. In depth, Dr. Hawkins's technological needs are a large influence on the project. That being said, the SMSS stands to be a marketable product in the medical sector.

TABLE III: Marketing Requirements

#	Marketing Requirement	Justification
1	High precision/accuracy flow rates	Constitutes the use of automation, to avoid human inconsistencies and save labor.
2	Programmable flow rates	Customers need various flow rates for different applications.
3	Modular	Customer's require multiple pumps to be controlled in a single project while being able to change setup and hierarchy.
4	Affordable cost (Priced \$500)	This cheaper price point appeals to budget-concerned customers
5	Easy User Interface	Users should enjoy using the device.
6	Reliable	Critical work depends on this device

The highest priority of the project is to produce high precision/accuracy flow rates from a standard syringe. The customer, Dr. Hawkins, requested that the pump perform to the same tolerance as existing products while avoiding impulse problems. This requirement does inadvertently imply the use of water in the syringe pumps: the viscosity of the fluid will change the flow rate and the accuracy of the device.

The next highest priority is programmable flow rates: allowing the user to achieve a desired flow rate from

the available hardware in a jog or timed manner. This requirement does also inadvertently imply that the only mechanical signal acting on the syringes are unit steps that only produce constant flow. Additional mechanical signals such as a ramp, triangle, or exponential function are not within the scope of this version of the SMSS.

A modular marketing requirement states the user has ability to use a single pump or scale the project up to three pumps. In another frame, the SMSS function without a complete set of devices.

An affordable price point is market disrupting while outperforming the competition with other higher priority qualities. An affordable cost also dictates a lower cost per iteration of the project; this student project will continue to build in complexity without inflating in cost.

An easy interface is a key point as the user shouldn't be required to code or write scripts to use the device. A low learning curve for the device keeps the SMSS competitive and relevant with current Cal Poly research projects outside the electrical engineering department.

Reliability is the last marketing requirement and it is last because the SMSS is still in a prototype state. Reliability for the SMSS is hardware robustness and resistance to various types of noise in the field. This could be interpreted in many ways: study housing, noise rejection, operating in partial failure modes, etc.

B. Engineering Requirements

The engineering requirements seen in the table below are derived from the marketing requirements in Appendix D Table III. In order to meet the market requirements of the customer, these engineering requirements were tested and validated before release. This list is not all encompassing nor in order of priority, but does outline the minimum viable product to remain competitive in the market. It should also be noted that the requirements derived here are set for this specific version of the SMSS and not guaranteed to hold between generations.

Verification of these requirements covers a wide array of testing setups and is discussed in the Testing section of the paper.

TABLE IV: Engineering Requirements

Market Req.	Engineering Req.	Justification
4	Total production shouldn't exceed \$500	This covers cost (with net profit) while remaining competitive in the low-end market
1,6	The pump shall hold a minimum flow rate of $0.1 \frac{mL}{hr}$ within $\pm 0.01 \frac{mL}{sec}$ of target rate.	Similar products have this tolerance and the application of the pump needs to remain relevant for medical purposes
1,6	The pump shall hold a maximum flow rate of $100 \frac{mL}{hr}$ within $\pm 0.01 \frac{mL}{sec}$ of target rate.	Similar products have this tolerance and the application of the pump needs to remain relevant for medical purposes
4	The pump shall not contain proprietary software or hardware.	I want to avoid costly parts that could be difficult to alter.
3	The pump must be able to house syringes of size 2mL to 20mL	Dynamic range of syringes is necessary for compatibility of products in the field.
2,3	The hub controller must be able to handle 3 pumps with same flow rates.	Scaling to 3 pumps is defined by the customer
5	The controller shall have a GUI.	The product needs to be useful for a wide audience
3,4	The drive train and housing of the pump can be 3D printed.	The product should remain available to a wide range of personnel and institutions.
1,2,5	The controller must be able to output digital log information on output flow rates: 255 Samples $\in (1\text{min}, 30\text{min}, 1\text{hr}, 6\text{hr}, 24\text{hrs}, 2\text{wk})$	The product needs to be able to provide researchers with data they can analyze.

C. Scheduling

The current GANTT chart on the SMSS is seen in Appendix D Figure 49. This GANTT was utilized to the best of its predictions but was not a living document for the duration of the quarter.

D. Bill of Materials

The SMSS project ideally requires funding for parts and labor. The chart in Appendix D Figure 50 outlines the current status of the market pricing as of March 2022 and the stockroom of this continuous project. Furthermore, my contribution to this project is highlighted in orange and added to the overall cost of the project. The total funding required for the system is below \$500 USD with some items missing documentation from previous work.

Additional man hours have been added for adequate time to debug and perform software validations. It was increased from 20 hours a week to 45 given the condensed time frame and large portions of software.

IV. SYSTEM DESIGN

The Smart Motor Syringe System is a mixture of both hardware and software. The following section discusses both aspects of the device in detail.

A. System Hierarchy

The SMSS has many layers to its complexity. For the sake of clarity, a functional decomposition is illustrated below.

1) *Functional Decomposition (Level 0)* : In the highest level of the SMSS, the black box diagram of the system is comprised of three inputs and three outputs. The inputs of the device are: 120VAC from a power outlet, a single or series of syringes, and user requests from the GUI. The outputs are: linear force placed on the plunger of the syringe, a GUI display for the user, and output flow rate data.

The user will load a syringe into the device, supply power and interact with the GUI by enter the requested data from the SMSS on a keyboard. The requested data for each pump includes the volumetric flow rate, radius of the syringe, capacity of the syringe, and the duration of the device. Upon starting, the device will run the motors and supply enough linear force to produce the requested flow rate. Depending on the user selection, a stop can be time sensitive or triggered manually.

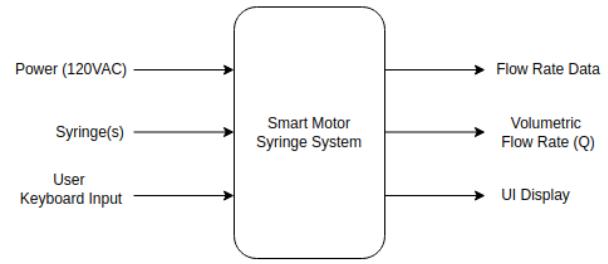


Fig. 7: Level 0 of the SMSS

TABLE V: Design Level 0

Market	SMSS
Inputs	<ul style="list-style-type: none"> • Power, 120 VAC • Syringe $\in [1\text{mL}, 30\text{mL}]$ • User input from keyboard
Outputs	<ul style="list-style-type: none"> • Volumetric flow rate (Q) from syringe • Flow rate data • User interface display
Functionality	After supplying power and water-filled syringe(s), the user will interact with the GUI and enter hardware metrics (on a keyboard) for each pump. After starting the machine, it will produce linear force on each syringe along with measured flow rate data from each device. The GUI will visually displayed for the user on a monitor.

2) *Functional Decomposition (Level 1)* : The next level of hierarchy, Figure 8, further breaks down the system into individual sections. The motor power supply (VMOT), digital power (VDIG), and syringes are split respectively for simplicity. The hub controller is a GUI driven controller designed to send commands to each of the pump controllers. It is separated from the other pump controllers to highlight its higher hierarchical place as well as its interaction with user inputs/output devices. It also acts as a main device

with the pump controllers assuming the sub role for I2C communication protocol; the hub controller also supplies VDIG to all of the individual pump controllers. Lastly, the pump controllers are the electrical and mechanical hardware used to translate electrical quantities into linear force. This is a large set of equipment that demands an additional level of decomposition to break down cleanly; this is seen in Figure 9.

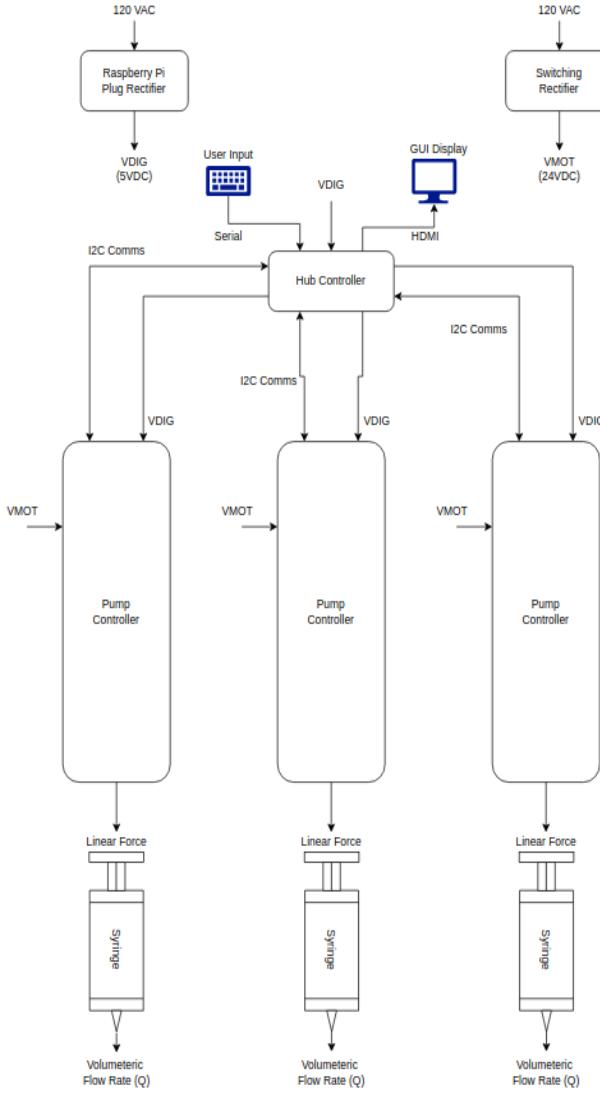


Fig. 8: Level 1 of the SMSS

TABLE VI: Switching Power Supply in Level 1

Module	Switching Rectifier
Input	120 VAC
Output	24VDC (VMOT)
Function	Convert the 120 VAC wall source and provide the stepper motors with 24VDC and +1A continuous current.

TABLE VII: Digital Power Supply in Level 1

Module	Plug Rectifier
Input	120 VAC
Output	5 VDC (VDIG)
Function	Convert the 120 VAC wall source and provide the Raspberry Pi with 5VDC and +1A continuous current.

TABLE VIII: GUI Display in Level 1

Module	Display Monitor
Input	HDMI from hub controller
Output	Visual information for user
Function	The monitor displays the GUI for the user

TABLE IX: User Input in Level 1

Module	Keyboard
Input	User key press
Output	Serial signal for controller
Function	Translates user key presses into digital signals

TABLE X: Hub Controller in Level 1

Module	Hub Controller
Input	
Output	
Function	Directs the state of each motor controller unit, translates user requests into usable data, and stores measured flow rate data from each unit.

TABLE XI: Syringe in Level 1

Module	Syringe
Input	
Output	Volumetric Flow (Q)
Function	Translates linear force into flow rate

TABLE XII: Pump Controller in Level 1

Module	Pump Controller
Input	<ul style="list-style-type: none"> •5 VDC (VDIG) •24VDC (VMOT) •I2C commands from Hub
Output	<ul style="list-style-type: none"> •Linear force on syringe •I2C response to Hub •Measured flow rate to Hub
Function	The motor controller and plant system are responsible for translating the hub commands into a linear force as well as supply the hub with response information and flow rate data.

3) *Functional Decomposition (Level 2):* Level 2 of the functional decomposition focuses on an individual pump controller and the electrical/mechanical components inside. The top block in Figure 9 is the motor controller. For this version of the SMSS, the motor controller is an Arduino Nano that is connected to the hub controller with I2C and digital power VDIG. The motor controller controls the local logic flow and adjusts the setting/performance of the stepper motor (indirectly through the stepper driver). The stepper driver translates the low power signals from the motor controller into a high-power signal usable by the stepper motor. Lastly, the ball screw translates the rotation of the stepper motor into linear velocity.

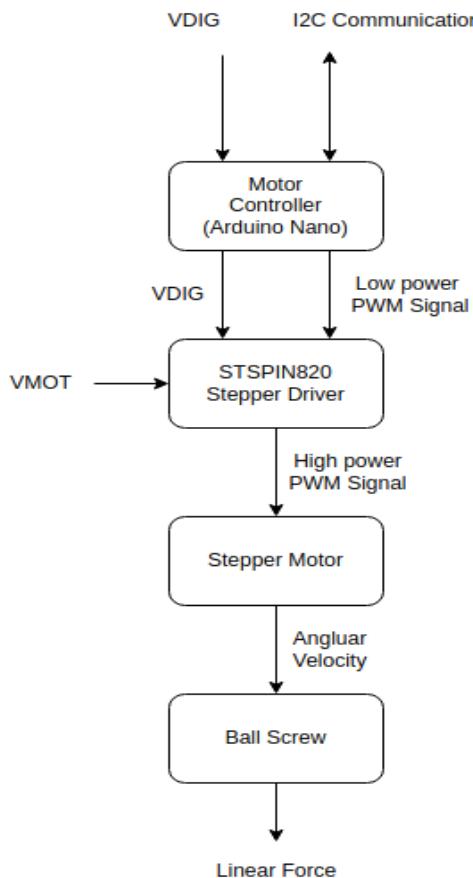


Fig. 9: Level 2 of the Pump Controller

TABLE XIII: Motor Controller in Pump Controller Level 2

Module	Motor Controller
Input	<ul style="list-style-type: none"> •5 VDC (VDIG) •Bi-directional I2C communication
Output	<ul style="list-style-type: none"> •5 VDC (VDIG) •Low power PWM signal
Function	The motor controller converts hub controller requests into a PWM signal for the stepper driver. The motor controller collects flow rate data and can act as a standalone device should the user only need one pump.

TABLE XIV: Stepper Driver in Pump Controller Level 2

Module	Stepper Driver
Input	<ul style="list-style-type: none"> •5 VDC (VDIG) •24 VDC (VMOT) •5v logic level PWM signal
Output	24v logic level PWM signal
Function	The stepper driver takes in a relatively low voltage PWM signal, modulates that signal with VMOT, and outputs a higher power PWM signal with options for step division (micro stepping).

TABLE XV: Stepper Motor in Pump Controller Level 2

Module	Stepper Motor
Input	High power PWM
Output	Angular velocity ω
Function	Converts the PWM pulses into rotational movement.

TABLE XVI: Ball Screw in Pump Controller Level 2

Module	Ball Screw
Input	Rotational Velocity
Output	Linear force (and velocity)
Function	Convert the rotation of the motor into linear force/velocity.

B. Hub Controller

This subsection discusses the hub controller, covering the GUI interface as well as the logic flow. The user interface will be a step-by-step example walk through that covers all of the interactions the user will come across. The logic flow will be presented with flow diagrams and outlines.

1) *User Interface:* The user interface uses a terminal-based library called Curses or NCurses. This library is already installed on most OS and is usually seen in BIOS menus or other applications requiring a lightweight GUI. It should be noted that curses operates in the terminal, often forgoing the use of a mouse and relying solely on the keyboard for user input. Curses does offer mouse integration but due to time constraints, the keyboard will be the only input device for the hub controller while in the terminal. It is still recommended to use a mouse to interact with the raspberry pi.

Upon powering the hub controller (or raspberry pi for this version of SMSS), a yellow LED will light up. This light is attached to the STANDBY state of the motor controller (Arduino Nano). Table XVII pairs the corresponding color combination of LED to the motor controllers current state.

TABLE XVII: Device State per LED Combination

LED	State
Yellow	Standby
Blue	Data collection ("Datapull") for timed run mode.
Blue + White	Data collection ("Datapull") for jog mode.
Green	Motor running in timed run mode.
Green + White	Motor running in jog mode.

Figure 10 shows the starting condition of three syringe pumps and their yellow STANDBY LEDs. The LEDs located on the three Nano boards do not represent the controller states; only the three rows of LEDs off of the boards are status LEDs. It should be noted that device one is the right-most device, progressing numerically to the left (the left-most device is number 3).

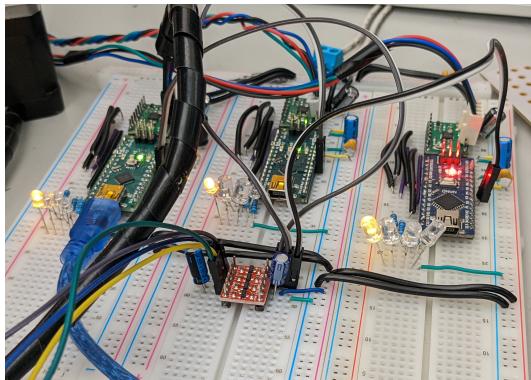


Fig. 10: LED Indicators on Start

Figure 11 shows the command lines to navigate the user to the GUI. The first line, `cd Desktop/SMSS22`, changes the terminal directory to the location of the hub controller's python file. the next line, `python3 Hub.py` runs the python script. It is recommended to resize the terminal window to a larger one before pressing enter.

** The user can press CTRL+z to exit the GUI. **

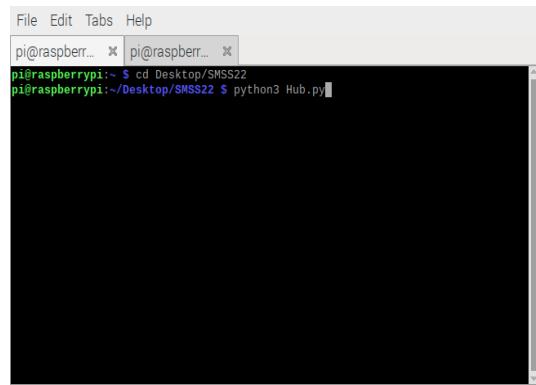


Fig. 11: Running Hub.py in the Terminal

Figure 12 represents the controls for selecting the device run states as well as the direction of the motors in a later menu. The WASD keys are used to move the highlighted cursor left/right with W acting as the enter key.

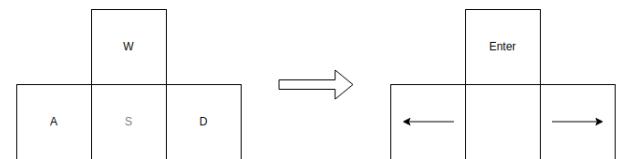


Fig. 12: Controls for Preset Options

Figure 13 shows the start screen of the GUI after selecting how each device should be run. Notice each device window has its own mode and the white block cursor waiting on bottom near the CONFIRM button. Pressing W here will advance the GUI to data collection. Figure 14 shows the color pattern of the LEDs after selecting the run modes. Notice device 1 has both blue and white LEDs while device two only has blue on. Device three still has yellow on and is put in standby or "OFF" until another cycle.

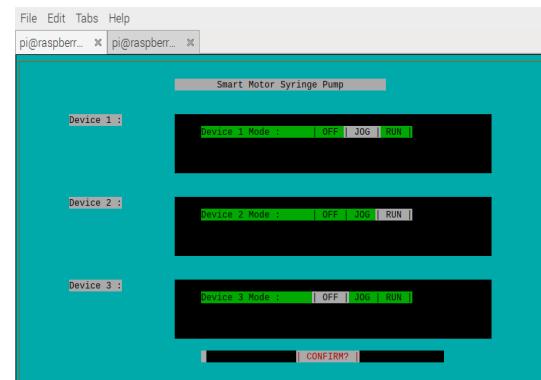


Fig. 13: Selecting Each Run State on the Different Devices

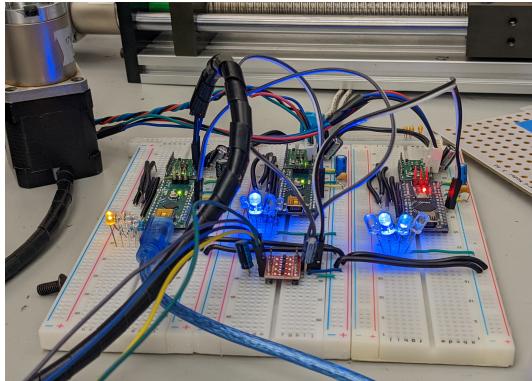


Fig. 14: LED Indicators for Device/State

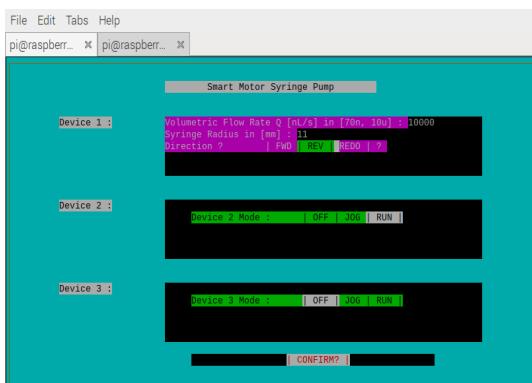


Fig. 15: Entering User Data into the Hub Controller

Figure 15 gives an idea on what it would be like entering information into the hub. The GUI will prompt the user for nominal values and the user will enter them using the number keys and enter. Figure 16 illustrates the keys used when entering parameters. It should be noted that the direction of the motor is still selected using WASD. the option REDO is used to revert back to STANDBY mode for that particular device in order to reassign a run mode. Figure 17 shows example data entered into the GUI with the CONFIRM below awaiting a W for confirmation.

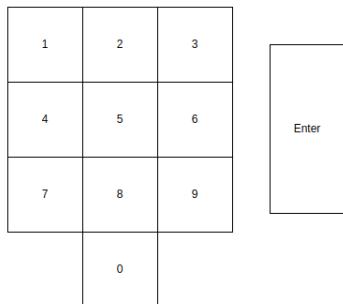


Fig. 16: Controls for Numerical Options

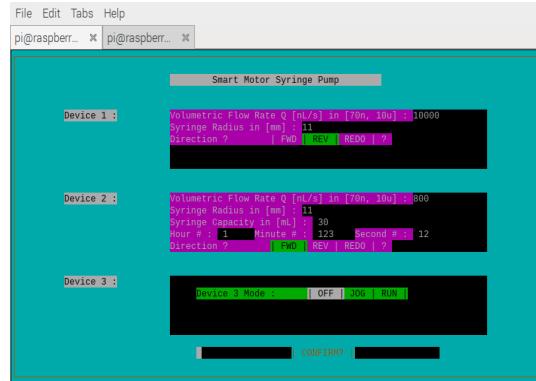


Fig. 17: Concluding Data Pull from User

After confirming the data pull state, the devices will enter the RUN state (Figure 18). The GUI will display the parameters that are being output and the time elapsed if running in run mode. Figure 19 shows the LEDs for each device. Device 1 is in jog and shows green and white lights, device 2 is in run mode and only uses a green LED, and device 3 is still showing a yellow LED because it is still in standby mode.

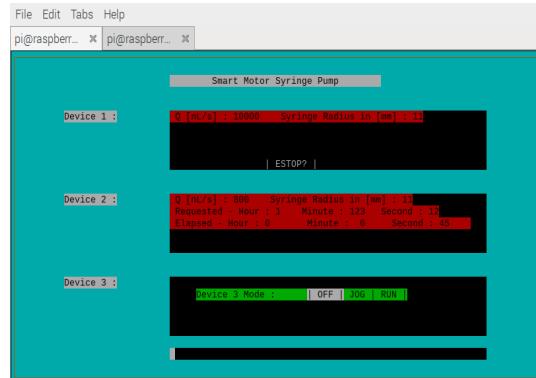


Fig. 18: GUI Run State

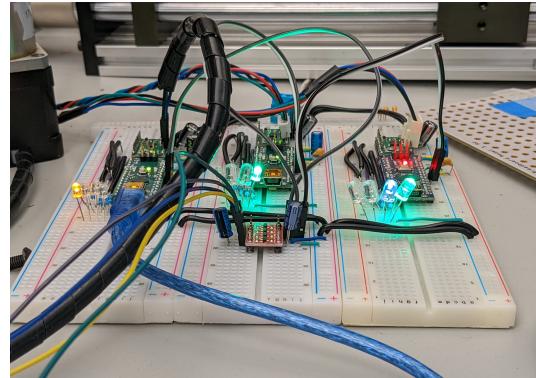


Fig. 19: LED Indicators during Run/Jog State

Figure 18/20 also give information on the estop cursor. Figure 18 shows the estop cursor on device one as "| ESTOP |" while Figure 20 has its estop cursor at the bottom (| ESTOP ALL? |). The first figure will estop the device 1 only while the second figure will stop all of the devices if pressed.

Figure 21 shows the estop controls; they are using WASD but in a different order than the mode selection.

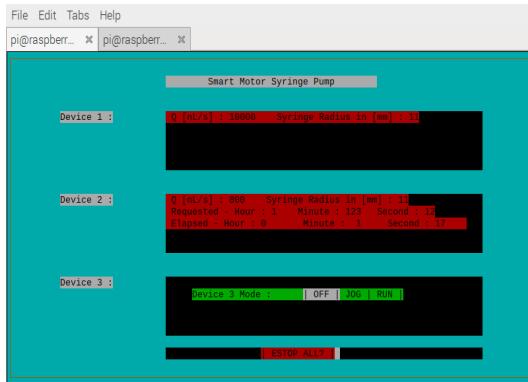


Fig. 20: Run Screen : E-Stop All Devices

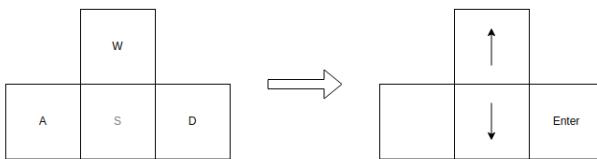


Fig. 21: Controls for E-Stop

Figure 22 shows an example of an estop event. The estop cursor is replaced with a | STOP'D | mark, the motor halted, and the motor controller returns to the STANDBY state as seen in Figure 23

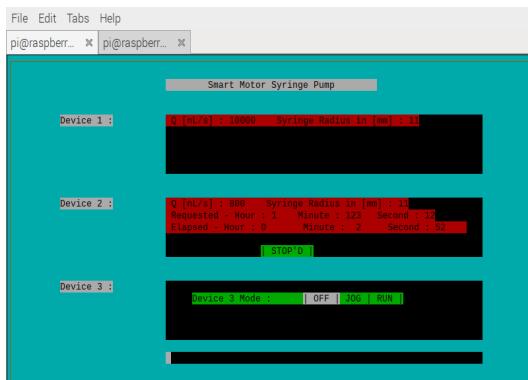


Fig. 22: Run Screen : E-Stop Device #2

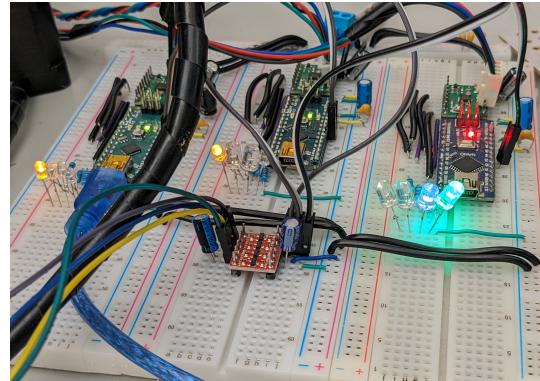


Fig. 23: LED Indicator with E-Stop Event

The hub controller communicates with each motor controller after the user has entered in the requested parameters. No motor controller hardware is the same and as such, each user request is checked by the motor controller to see if the hardware can produce the correct output. If the requested volumetric flow rate is too large for the hardware to output, an error message will appear (Figure 24) and the user will be prompted to enter new parameters. Other error codes exist: Figure 25 is too small a Q value and Figure 26 refers to the run time exceeding the fluid capacity of the syringe.

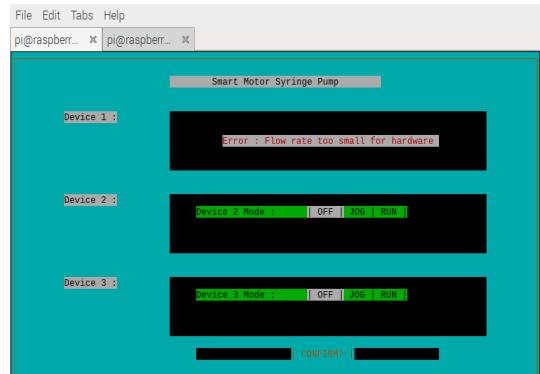


Fig. 24: Error Code : Too Large for Hardware to Output

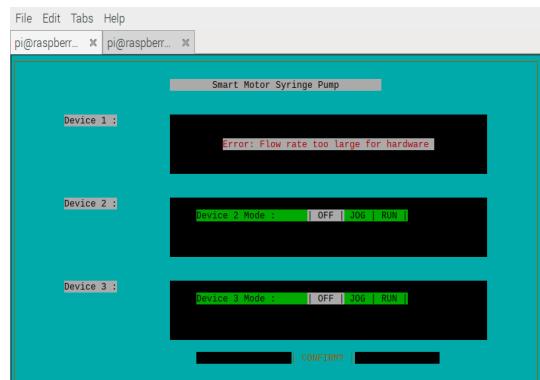


Fig. 25: Error Code : Too Small for Hardware to Output

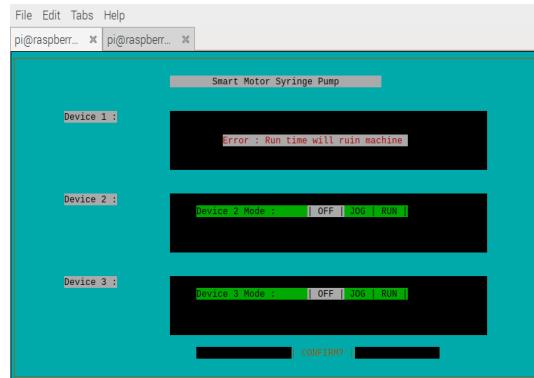


Fig. 26: Error Code : Too Long for Hardware to Run

After all devices have halted, the GUI screen will return to polling for the device run mode and all of the indicator LEDs will revert back to yellow.

2) Logic Flow: This subsection discusses the logic flow of the hub controller. The python code of this controller can be seen in Appendix C.

Looking at Figure 27, the hub controller has four states that it operates in: START, DATAPULL, RUN, and RESET. The START state is written to do a few items: note which devices are active on the I2C bus, reset the motor controllers to their STANDBY state, and ask the user which run mode is proper for each device. After all active devices have selected a run mode, the hub transitions to DATAPULL. DATAPULL will prompt the user for data parameters (volumetric flow rate Q , syringe radius, syringe capacity, duration, and motor direction) and send that data to the individual motor controllers. Should the user request be outside the operating range of the hardware, the motor controller will send back an error code that will be displayed on the GUI. The program will loop until all devices are error free and the system is ready to progress into the next state. The RUN state will turn on the stepper motors and poll for a stop condition. Should the user select an estop in the GUI or the motor controller reach its max time duration, the motor controller will shut off the motor and the hub will reset for that device. After all devices have reached a stop condition, the hub will proceed to the RESET state to clear all user parameters in the hub/motor controllers. The hub will return to the START state and STANDBY in the motor controllers.

It should be noted that the GUI referenced in the previous subsection is the same here and can be referenced as such.

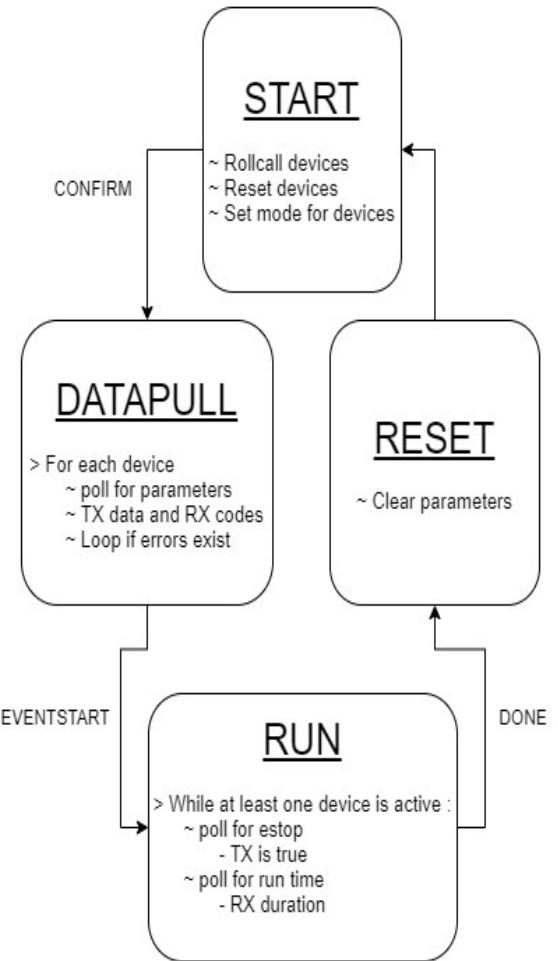


Fig. 27: Top Level Flow Chart of Hub Controller

C. Motor Controller

This subsection discusses an individual motor controller in the SMSS in three different sub-sub-sections. The first will discuss the characterization of the system, the second will map the control flow, and lastly there is a section about the I2C communication between the hub and motor controllers.

1) Plant Characterization: Starting with the syringe, the mechanical conversion that takes place in the syringe is as follows in Figure 29. The volume of fluid displaced by the syringe plunger is proportional to the speed at which the plunger is moving. This plunger speed (V) is applied across the plunger's surface area, producing a volumetric flow $\frac{m^3}{sec}$. This unit can be converted to $\frac{L}{sec}$ with unit conversion.

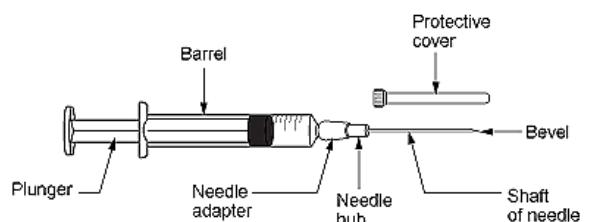


Fig. 28: Components of a Syringe

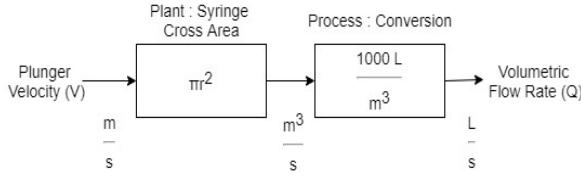


Fig. 29: Plant Conversion of Syringe

After understanding how the linear velocity V is translated into a volumetric flow rate Q , let's discuss how the gearbox's RPM is converted into a linear velocity. Figure 30 illustrates how the RPM from the gearbox is translated to a linear velocity on the syringe plunger. The gearbox shaft is coupled to the screw rod at a 1 to 1 ratio; this segment does not require further analysis as no slip was witnessed. The interaction of the screw rod and the pillow block does however: the pillow block is attached to the frame of the ball screw kit and must be engaged with the screw rod to move. The helical spiral in Figure 31 represents the screw rod that engages with the threaded bearing in the pillow block. The linear speed of the pillow block is dependant on the rotational speed as well as a few factors. The position of a single particle on the helix is represented as $r(t)$, with respect to time.

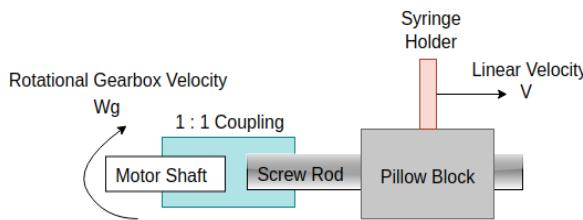


Fig. 30: Translation of Angular Velocity to Linear Velocity

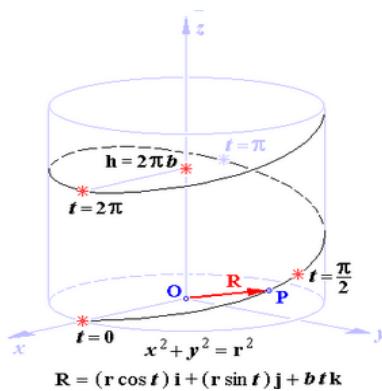


Fig. 31: Cylindrical Helix Model for Ball Screw

$$\text{Let } r(t) = \rho \cos(\bar{\omega}_g t) \hat{A}_x + \rho \sin(\bar{\omega}_g t) \hat{A}_y + \gamma \bar{\omega}_g t \hat{A}_z \quad [m] \quad (1)$$

Where ρ is the screw rod radius and γ is the rod's pitch. The rod's pitch is the z-axis distance between points of the same xy angle.

To find the velocity of the particle with respect to time, the derivative of $r(t)$ is taken.

$$\vec{v}(t) = \frac{d}{dt} r(t) \left[\frac{m}{s} \right] \quad (2)$$

$$\vec{v}(t) = -\rho \bar{\omega}_g \sin(\bar{\omega}_g t) \hat{A}_x + \rho \bar{\omega}_g \cos(\bar{\omega}_g t) \hat{A}_y + \gamma \bar{\omega}_g \hat{A}_z \left[\frac{m}{s} \right] \quad (3)$$

Focusing attention on the Z-axis movement :

$$\begin{aligned} \vec{v}_z(t) &= \gamma \bar{\omega}_g \left[\frac{m}{s} \right] \\ \bar{\omega}_g &= \frac{\vec{v}_z(t)}{\gamma} \left[\frac{rad}{s} \right] \\ \overrightarrow{RPM}_{gearbox}(t) &= \frac{2\pi \vec{v}_z(t)}{\gamma} \left[\frac{Rev}{s} \right] \end{aligned} \quad (4)$$

The final equation relates the rotational speed of the gearbox to the linear velocity of the pillow block. Figure 32 is the plant of the system from this point. From here, relating individual steps of the stepper motor to the output RPS is the final step in characterizing the system. The STSPIN820 motor driver can output up to 256 microsteps, allowing the driver to further divide the input PWM freq at the cost of motor torque [9]. The stepper motor itself markets 200 steps per revolution; both motor step and the driver's microstep will alter the motor's speed. Figure 33 depicts the addition of both step properties as well as the final representation of the hardware system.

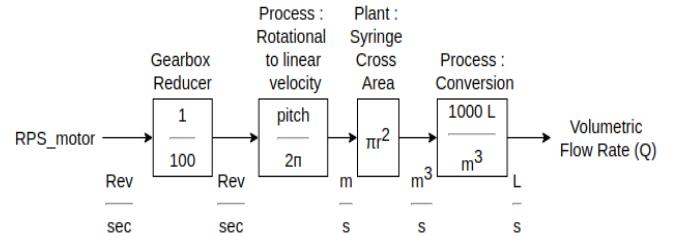


Fig. 32: Translation of Motor Speed to Flow Rate

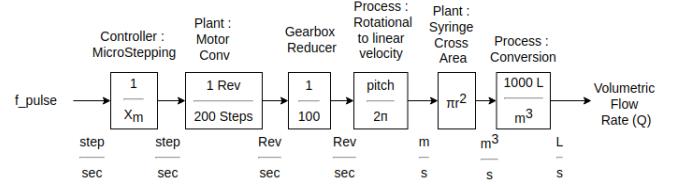


Fig. 33: Complete Hardware Characterization of Pump

Working backward to find the appropriate frequency, f_{user} can be defined as :

$$\begin{aligned} f_{user} &= X_m * \frac{400\pi N Q}{1000\gamma\pi r^2} \\ &= X_m * \frac{2NQ}{5\gamma r^2} \end{aligned} \quad (5)$$

where N is the gear ratio, γ is the pitch of the ball screw, X_m is the microstep divisor, f_{user} is the frequency derived from the user requested Q , and r is the radius of the syringe plunger.

From experimentation, it was found that the minimum frequency the stepper motor should operate at or above is 120 Hz ($f_{min} = 120\text{Hz}$). With this standard, finding the proper microstep size is a simple equation :

$$\begin{aligned} f_{user} &\geq f_{min} \\ X_m * \frac{2NQ}{5\gamma r^2} &\geq f_{min} \\ X_m &\geq \frac{f_{min}}{\frac{2NQ}{5\gamma r^2}} \\ \therefore X_m &\geq \frac{5\gamma r^2 f_{min}}{2NQ} \end{aligned} \quad (6)$$

The range of Q is limited to the frequency range of the stepper motor and was found experimentally as well. Both maximum and minimum Q values would stall the stepper motor. It was also noticed that frequencies above and below the range could move the motor but not reliably. The reliable range is :

$$Q \in [10n, 16\mu] \left[\frac{L}{sec} \right]$$

2) Control Flow: The individual pump/motor controllers have the same code structure but can have different hardware parameters (syringe radius, ball screw radius, etc). Moreover, each controller can be placed into a I2C mode - the hub controller will dictate the flow of user info, or in serial mode - the Arduino IDE serial terminal will prompt the user for information. It should be noted that the serial method is device specific and wont control more than one pump controller. The serial mode is also referred as manual mode because it is easy to tune parameters for an individual pump without the use of an additional controller. To use either mode, comment out the unused setting as in Figure 34.

Both modes of the motor controller use different communication protocols but are similar in state structure. Figure 35 represents the highest level of flow for the motor controller : 6 states that will continue to loop until power is cycled. The controller will start and stay in STANDBY until the controller has received a START/JOG request. After leaving, the controller will go in to either DATAPULL_S (for timed duration) or DATAPULL_J (for jog) to retrieve the hardware parameters from the user. This data can be from I2C or serial depending on which portion of code had been enabled. After checking for errors, the device will proceed to the appropriate RUN state. Once the device has received an ESTOP or time-dependent FINISH command, the device will reset all variables in ESTOP and proceed to wait in STANDBY. This top level is almost identical to the hub controller but has many lower level differences.

Lower level flow charting can be found in Appendix D.

```
// Test Variables (COMMENT OUT TO DESELECT) -- DONT FORGET
#define TESTING // Comment to cancel testing
#ifndef SERIAL_COMM // Comment to cancel serial
#define I2C_COMM // Comment to cancel I2C comms
#ifndef STSPIN220 // Comment out to NOT use timer 2
#define STSPIN820 // Comment out to NOT use the
```

Fig. 34: Tool selection for the Motor Controller

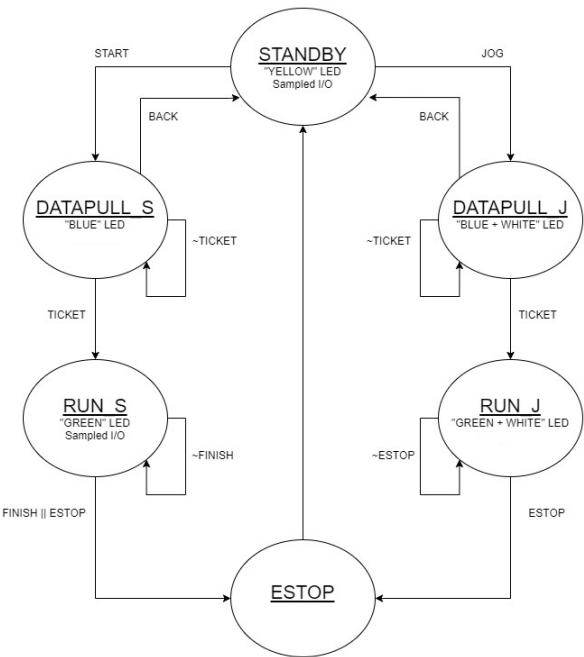


Fig. 35: Top Level Flow Chart of Motor Controller

3) I2C Communication Between Controllers: The hub controller assumes the position of the main device with the pump controllers acting as sub devices. The sub devices don't communicate with each other; only the hub controller sends a receive (MOSI) or request (MISO) command. The hub controller is a raspberry pi, which has a logic high of 3.3V, and the pump controller, arduino nano, has a logic high of 5v. With a difference in logic level voltage, a bi-directional logic converter was used in order for the devices to communicate to each other without hardware damage. For simplicity, a Sparkfun MOSFET converter was used as seen in the Appendix D schematics.

The data sent and requested by the main controller was often over the single packet 255 bit limit. To overcome this, data packets are processed to handle numbers less than 16,777,215. This is done by sending factors that are multiplied by 256 as well as remainders that are summed with the total number. The trade off for this solution is the extra time it takes to transmit three extra packets of information.

If the number is large enough to fill the second factor,

$$\text{Number} = 256 * (256\text{Fact}_2 + \text{Remain}_2) + \text{Remain}_1$$

else :

$$\text{Number} = 256\text{Fact}_1 + \text{Remain}_1$$

where each variable above represents a single packet.

V. TESTING

This section outlines the process for verifying the engineering requirements (Table IV) of the SMSS. The financial requirement, keeping the SMSS under \$500 USD is verifiable by viewing the Bill of Materials in Appendix D Figure 50. This requirement is met for this generation of the SMSS because the cost was under \$500 USD.

The second and third requirement will be validated in the next section below; the fourth requirement can be verified by noting the libraries and hardware used. The code (Appendix C) that was used is open source online and the hardware is not locked by physical/digital means. Both Arduino and Raspberry Pi devices are designed to be altered down to the copper or the kernel.

The fifth requirement was verified visually by inserting different syringes into the housing. The recorded range of syringes was 1mL to 30mL, going beyond the requirement.

Requirement six, concerning 3 pump controllers, is also verified in the subsection below. That requirement requests an analysis of three separate devices and their output flow rate.

Requirement seven is the GUI requirement and it is met by viewing the hub controller section earlier on in the paper.

Requirement eight, requesting that the drive train and pump housing be 3D printed, has fallen short of its goal. The drive train is a metallic ball screw that must be metal for friction and wear purposes. 3D printing this piece would require additional work for a lesser component. The housing of the entire system has also yet to be 3D printed because the footprint of the SMSS will continue to decrease. It was not fit to enclose the entire system at this stage of prototyping. Printing a piece that hold the ball screw and syringe together has proved to be a value component in this stage of the SMSS.

The last requirement was also short of its goal. This generation of the SMSS was unable to integrate viable feedback in the system. Different feedback methods (potentiometers, resolvers, pressure sensors) were considered but were left out due to budget and time constraints. Without feedback, output data couldn't be produced and sent to the hub controller.

A. Verification Method for Hardware

The output flow rate of the SMSS was measured with a LabSmith uProcess kit. This kit measures the pressure inside a CapTile cavity and was the only method used. To use this device properly, the user must setup the device as shown in Figure 36 and note physical measurements of the differential tubing. This tube's inner diameter and length are important in translating the pressure measurement into a measured volumetric flow rate. While the SMSS is applying pressure on the plunger of the syringe, water is flowing through the filter and into the three-way CapTile. The uPOS senor then measures the pressure at its highest potential at the intersection in reference to the lowest potential point at the other end of the differential tube. The pressure across

the differential tube is the measured result ΔP [kPa]. The change in pressure can be translated into a volumetric flow rate with the following equation. It should be noted that the GUI has Q in terms of nL/sec, hence the conversion to nL/s here.

$$\begin{aligned}\Delta P &= P_{meas} - P_{atm} \quad [\text{kPa}] \\ &\approx P_{meas} - 1 \quad [\text{kPa}]\end{aligned}\quad (7)$$

$$\begin{aligned}Q_{deriv} &= \frac{\pi (\Delta P) D^4}{128\mu L} \left[\frac{m^3}{\text{sec}} \right] \\ &= \frac{\pi (\Delta P) D^4}{128\mu L} * \frac{10^3 L}{m^3} * \frac{10^9 nL}{L} * \frac{10^3 Pa}{kPa} \left[\frac{nL}{\text{sec}} \right] \\ &\therefore = \frac{\pi (\Delta P) D^4}{128\mu L} * 10^{15} \left[\frac{nL}{\text{sec}} \right]\end{aligned}\quad (8)$$

where

- ΔP is the change in pressure measured by the uPOS (kPa)
- D is the inner diameter of the differential tube (m)
- μ is the dynamic fluid viscosity of water at 21 C (Pa * sec)
- L is the length of the differential tube for which the pressure change takes place (m).
- Q_{deriv} is the volumetric flow rate derived from the change in pressure measurement (nL/sec). This variable will be compared with the Q value entered by the user in the GUI.

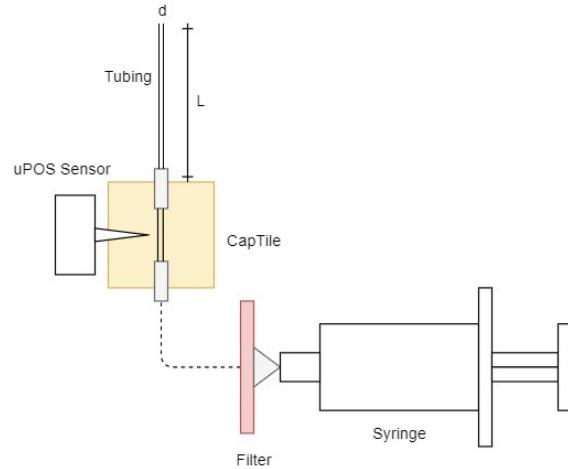


Fig. 36: Verification Model

B. Results

Using the setup in the sub-section above, a variety of nominal values were sampled. This method was used to experimentally find the operating range of the SMSS.

1) *Ceiling of Q Range:* The ceiling of an individual syringe pump was initially verified visually by witnessing the spin of the motor while coupled with a syringe. Later in the project, this was properly verified using the uProcess kit.

All other values above $16\mu L/sec$ were not repeatable and ended up stalling the motor. It should be noted that this real max value of $16\mu L/sec$ is far below the set engineering requirement; below both the maximum and minimum requirement. Here are the results :

TABLE XVIII: Parameters of Ceiling Test

Variable	Value	Unit
Syringe #	S1	-
Motor #	M1	-
Arduino #	A1	-
Motor Controller #	STSPIN820	-
Sensor	uPOS250-T116	-
Pipe Length L	0.0820	m
Pipe Inner Dia D	0.15	mm
Dynamic Fluid Viscosity μ	0.000975 @ 20°C	Pa*sec
Syringe Radius(GUI)	11	mm
Motor Direction	Forward	-

Measured Q vs. Time

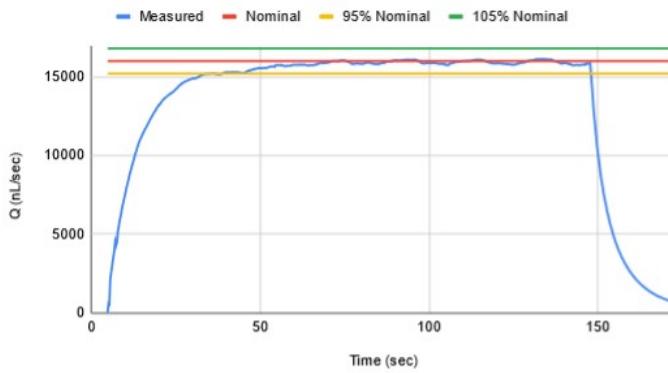


Fig. 37: Coarse Measurement for Nominal Q : 16000 nL/sec

Measured Q vs. Time

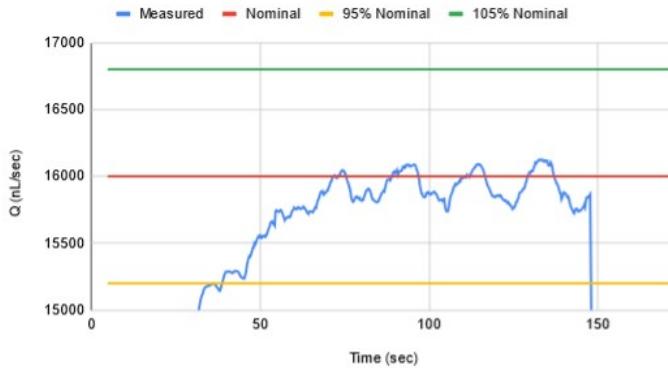


Fig. 38: Fine Measurement for Nominal Q : 16000 nL/sec

Figure 37 is the entire capture of processed data. There is an exponential rise to 16000 nL/sec and it holds around that number until the pump is turned off, near 150 seconds. The coarse figure appears to hit and stay on the nominal value

however, Figure 38 shows differently. The same value lines are present but the measured result is oscillating around the nominal value. It should be noted that the water flowing out of the differential tube is jet-like and not droplets.

2) *Floor of Q Range*: The smallest output was verified using the uProcess kit as well. The minimum engineering requirement ($100\mu L/sec$) is higher than the experimental minimum ($10nL/sec$).

TABLE XIX: Parameters of Floor Test

Variable	Value	Unit
Syringe #	S1	-
Motor #	M1	-
Arduino #	A1	-
Motor Controller #	STSPIN820	-
Sensor	uPOS250-T116	-
Pipe Length L	0.0820	m
Pipe Inner Dia D	0.15	mm
Dynamic Fluid Viscosity μ	0.000975 @ 20°C	Pa*sec
Syringe Radius(GUI)	11	mm
Motor Direction	Forward	-

Measured Q vs Time

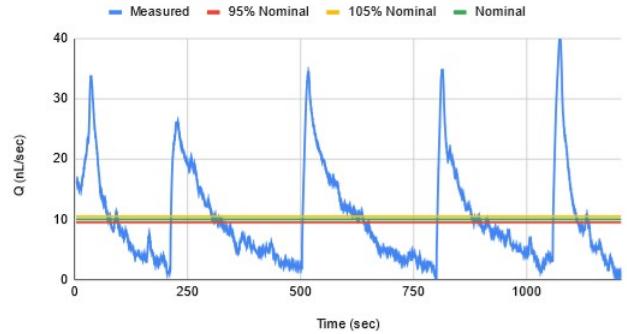


Fig. 39: Measurement for Nominal Q : 10 nL/sec

Measured Q vs. Time

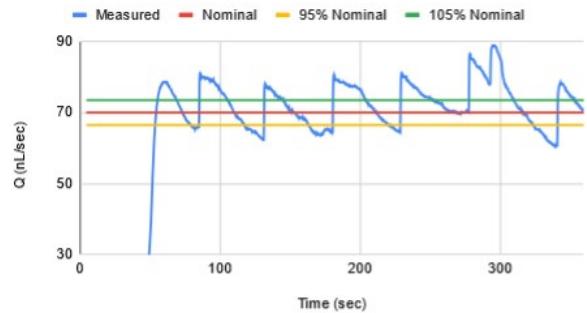


Fig. 40: Measurement for Nominal Q : 70 nL/sec

Figure 39 illustrates the volumetric flow output when the nominal Q value is 10 nL/sec . The large pressure spikes seen aligned with droplets falling from the differential tube. When the droplet was building in size on the tip of the tube,

the pressure would decrease. When the droplet fell, there was a large spike in pressure. It is difficult to say whether the pump could hit this nominal value accurately.

Figure 40 has similar spikes to the previous figure but are less pronounced. The pressure spikes are related to the physical droplets as well; the syringe pump can hit this Q value reliably and largely stay within $\pm 10\%$ untuned.

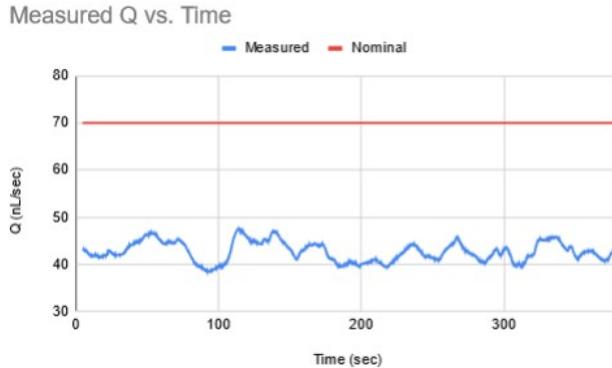


Fig. 41: Special Measurement for Nominal Q : 70 nL/sec

From this point, the data suggested that the exiting droplets were having an impact on the low Q data. Figure 41 shows the same nominal flow request but with the differential tube under water in the collection dish. The measured data is far from the nominal value, using the assumption from equation (7), but the large spikes are smoother than the previous test.

3) Additional Data within Range: Additional tests were made within the Q range described above. These tests are useful in understanding unforeseen problems in the system.

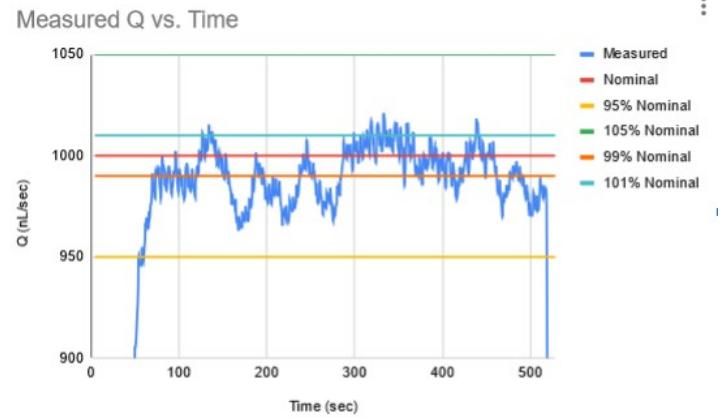


Fig. 42: Measurement for Nominal Q : 1000 nL/sec

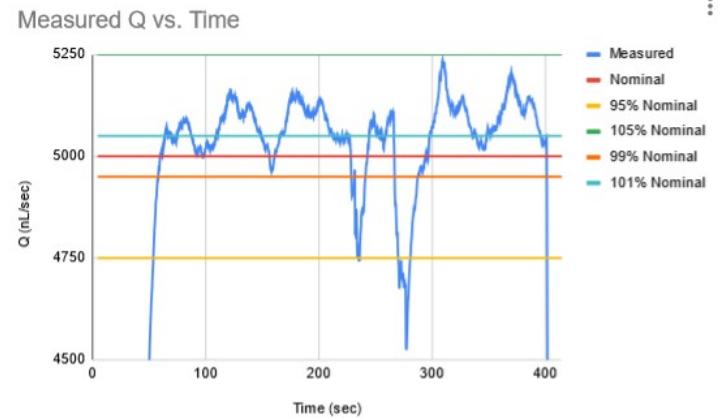


Fig. 43: Special Measurement for Nominal Q : 5000 nL/sec

Figure 42 is additional data within the Q range. There are inconsistencies between 200 and 300 seconds while running the system naturally. Figure 43 is another example of inconsistencies : the large dips between 200 - 300 seconds are the outcome motor failure. For one reason or another, the motor was stalling during testing, emitting a large hum and vibration. The cause for this is still unknown and output returns to the typical pattern after this event.

C. Analysis

After viewing the data, there is confidence that the pump system can achieve nominal rates within 70-16000 nL/sec, but a few problems have to be addressed. The measurement method used needs further investigation into the droplet phenomenon: does surface tension or falling droplets have an effect on low Q output? One correction could be to record data with the exit tubing under water; further testing is needed.

Other issues came from the 3D printed parts attached to the ball screw assembly: the amount of flex in these parts gave inaccurate readings for the initial round of tests. There were re-prints of existing parts with higher infill percentages as well as newly drafted parts that were over engineered to avoid flexing under pressure. The original set of prints did

TABLE XX: Parameters of Additional Data

Variable	Value	Unit
Syringe #	S1	-
Motor #	M1	-
Arduino #	A1	-
Motor Controller #	STSPIN820	-
Senor	uPOS250-T116	-
Pipe Length L	0.0820	m
Pipe Inner Dia D	0.15	mm
Dynamic Fluid Viscosity μ	0.000975 @ 20°C	Pa*sec
Syringe Radius(GUI)	11	mm
Motor Direction	Forward	-

end up breaking over time and were recorded deafening the pressure data as their internal structure began to fail. It would be recommended to replace the existing 3D printed parts with materials that has little flex.

Another issue came from the motor or motor controller: there were periods where the motor struggled to output the correct frequency. This can be seen in the data in Figure 43 as a large trough in the blue measured data. More testing is required to identify the root of this problem.

The idea of fine tuning the motor controllers was conceived early in the planning phase of the project but ended up not happening. It was decided that digitally tuning the controller before addressing the other mechanical / data validation problems would not be the best course of action. The error measured in the data was also above nominal in the lower Q values while below in higher values, making the tuning process more than a simple offset of frequency.

It should be noted that the range of Q dictated by the engineering requirements was not met : the individual pumps that comprise the SMSS are capable of nano-liter or micro-liter flow rates but not milliliter. A future re-assessment of the engineering requirements is needed in order to confirm that the smaller range is beneficial. Fortunately customer Dr Hawkins has expressed that the finer Q range is beneficial.

VI. CONCLUSION

This generation of the Smart Motor Syringe System demonstrated several new additions to this ongoing project while getting closer to a smooth, nominal volumetric flow rate. The new additions include but are not limited to : the introduction of a hub controller, a simple GUI, and multiple configuration setting for three complete motor controllers.

The hub controller and GUI were successful in directing the user's requests into viable data across all three devices. The user can control up to three devices, run each device manually or timed, get real time error codes, and emergency stop each device individually or all together. While the code has not been exhausted of all edge case bugs, there are still some underlying issues to address. For future endeavors, the python code written can be improved with more modular-based function scripting and the removal of most global variables in the code. Both of these items would improve the readability of the code and potentially increase the performance time. Communication between devices was also witnessed to be slower than initially thought, taking up to 100ms per transmission. Adjusting the I2C speed on both hub and motor controller unfortunately introduced more errors.

Looking at the individual motor controllers, each device performed close to the nominal requested Q. Most of the measured volumetric flow rates came within a 5% tolerance and demonstrated a finer accuracy than previously prescribed in the initial engineering requirements. That being said, there were considerable items that needed addressing. There were unforeseen motor stalls mid-test as well as large pressure spikes when measuring small Q values. This observation suggests that there should be a different testing

setup that prevents droplets from forming on the differential tubing. Lastly, the use of 3D printed parts became a problem and contributed to a number of accuracy errors early on. It is strongly recommendation to replace all of the printed parts with metal or a material that doesn't flex. Only after designing parts that were either over-engineered and/or over-filled was there little dampening/delay in the system.

Beyond the existing technology presented in this report, there are some short and long term endeavors. Better power connectors for the motor supply are an immediate upgrade that would reduce some uncertainty in troubleshooting the stepper motors. A feedback loop was a short term goal and engineering requirement that did not take off due to time and financial constraints. A feedback loop for each syringe pump would allow better control of the motor controllers while providing logging data to the hub controller. Additional output signals were a short term goal that was cut as well. Instead of the single step function output, Dr. Hawkins requested a ramping output with the potential for other wave forms. This item was eventually cut due to time constraints. A Long term goal for this project includes a PCB design to reduce the project's footprint.

APPENDIX A IMPACT ANALYSIS OF SENIOR PROJECT

Project Title : Smart Motor Syringe System

Student Names : Connor Wilson

Advisor's Name : Ben Hawkins

A. Summary of Functional Requirements

The Smart Motor Syringe System is a syringe pump that aims to autonomously output multiple flow rates and provide the user with research-grade log data. It will be using three modular syringe pumps inside the system to administer three separated fluids. This system will be cost effective by utilizing open source hardware and software while providing medical grade flow rate tolerances.

B. Primary Constraints

The primary constraints of the SMSS include an absence of feedback, proprietary equipment, and budget. The absence of a feedback loop places a large risk on the system should part of the system change. The lack of custom equipment is also a constraint because the user can introduce a syringe or controller that is not well adapted to the existing SMSS system. Calibration would be required. Lastly, budget is a constraint because there cannot be high quality parts or application specific components without going over budget.

C. Economic

- **Human Capital :** The SMSS will create jobs in engineering, manufacturing, sales, and marketing. The SMSS could have an international presence should the ideal company see growth opportunities overseas.

- Financial Capital : The SMSS is designed to perform a highly complex task for under market value. The open-source technology in this device will provide a large margin of overhead and profit while saving the customer a portion as well. The financial capital is also aimed at potential investors as the product grows into a company.
- Natural Capital : While the SMSS is made from non-renewable parts that cannot be viably broken down into raw resources, the SMSS components can be repurposed very easily. Most of the components inside the SMSS are stand alone devices, like the Raspberry Pi or the Arduino Nano, that can be used for other projects. A 3D printed housing case can also be broken down in a PLA/ABS foundry much easier than a complex polyethylene injection mold.
- Cost and Timing : The cost of the SMSS is broken down into assembly and validation. Since most of the manufacturing is done by the component sellers (Raspberry and Arduino), most of the time and cost spent in manufacturing is assembly. Validation on the other hand will require large sets of unit testing and calibration. This portion can easily be \$100 per product. Overall, as more testing and assembly turn to automation, the less spent on cost and time.

TABLE XXI: Expected Cost/Revenue if Manufactured on a Large Scale

If Manufactured	Units	Reasoning
Units per year	200	This is the initial market before appealing to the average consumer
Manufacturing Cost	\$400	Number includes material, parts, labor, and re-occurring validation
Purchase Price	\$500	Minimum Viable Product
Profit per year	\$10,000	Estimated
Operating cost for user	\$200 USD for power \$ 40 for annual care	Device requires power and requires regular service for lubricated parts.

D. Manufacturing

The SMSS will require a separate system for hardware assembly and validation along with software. Beyond the physical, validation testing and calibration will need to be performed on each unit as part of strict quality control. If the unit is sold as a kit, the open source hardware and software will be provided and the quality assurance will be left up to the user.

E. Environmental Impact

The SMSS will have a notable impact on the environment including significant power consumption, material consumption, and end-of-life disposal. The power consumed

by this unit will add further demand for electricity and will not have a power saving feature due to its critical performance features being the minimum viable product. 3D printed shells will also require a lot of power from the grid.

Beyond power consumption, the SMSS will require large refinement of materials in order to make the components. This includes silicon for chips as well as precious metals that are in high demand. Chip shortages brought on by COVID-19 are also brought into play as manufacturers seek out unsavory alternatives for refined materials.

This refinement also makes this product hard to recycle and reclaim. Stripping this project back into its separated forms will require a large amount of processing and possibly toxic chemicals that can cause damage to organic life. This includes the ABS printing filament.

F Sustainability

- **Issues maintaining the complete device or system**
The SMSS relies on the functionality of multiple microcontrollers, stepper motors, ball screws, and power converters. Fluids in the syringe could cause potential harm to the electrical components. Physical damage or build-up of particles along the drive train could cause potential strain and damage to the system, shortening the lifespan of the product.

The SMSS is also an open source project, allowing the user to replace commercial hardware components and reprogram them using an online database (Github) of code.

- **How the project impacts the sustainable use of resources** The SMSS does use electronic hardware that is hard to recycle but is easy to replace or reuse. It uses open source hardware/software and can be purchased, maintained, or re-purposed should the user no longer use the SMSS as intended.

- **Upgrades that would improve the design of the project** Calibration kits and negative feedback are two endeavors that would greatly improve the performance and reliability of the SMSS. The calibration kits would allow the user to ensure the precision/accuracy rates are correct as the system ages. Tuning software parameters with the calibration kit could potentially save the user money. Feedback compensation is another idea that would allow the system to perform better and hold tighter tolerances.

- **Challenges associated with upgrading the design** When upgrading the SMSS, it is important to take into account the customer base. If the customer is fluent in open source code/hardware, then applying updates through Git is likely the course of travel. Should the SMSS reach a launch date through the progress of others, it could potentially be up-gradable without pulling code requests.

The hardware will be easy to upgrade in the early prototype phases until a release date. These upgrades are not limited to those in the conclusion.

G. Ethical

The SMSS is an open source syringe pump that has a few concerns. First, having a medical device with potential security risks can cause a few issues and benefits. The security risks involved with open source are up for debate as some open source systems, such as Linux, are stronger because of their large user community involvement. That being said, the Raspberry and Arduino systems have security risks that could cause unforeseen consequences.

Additional concerns arise if the user decides against using medical grade parts such as precision stepper motors or syringes. This option leaves the SMSS functionality up to the user should they void.

Using this technology to intentionally cause harm goes against my personal ethics. Misusing or misdiagnosing concentrations or flow rates into an organism is also not safeguarded by this device and responsibility must be placed on the user.

H. Health and Safety

The SMSS is a medical device that could have grave consequences if system or user errors occur. Beyond validation, the user needs to fully understand the risk involved with injecting an organism with a particular fluid and concentration. Improper use of this equipment could also result in dire consequences. This could be a valid reason why this technology is not readily advertised to the public as it comes with a steep set of instructions for proper operation. Overall, this system could result in injury, serious harm, or even death if not used properly.

I. Social and Political

The political/social impact of the SMSS involves the right-to-repair. As companies turn away from patients, they deny consumers the right to repair their own products. By keeping proprietary technology secret, consumers are forced to seek third parties that specialize in reverse engineering. While the SMSS is open source, it does have a high operating standard. Should the SMSS become a viable and successful product, the open source angle of marketing could cause consumers to demand more legal freedoms with their products. That being said, consumers should have access to proper calibration to meet the SMSS's bare engineering requirements.

There are conflicts with having an option to repair the SMSS. As a prebuilt system, the SMSS will run as intended just as other devices on the market. However, repairing and altering this product should also be viable.

APPENDIX B
SCHEMATICS

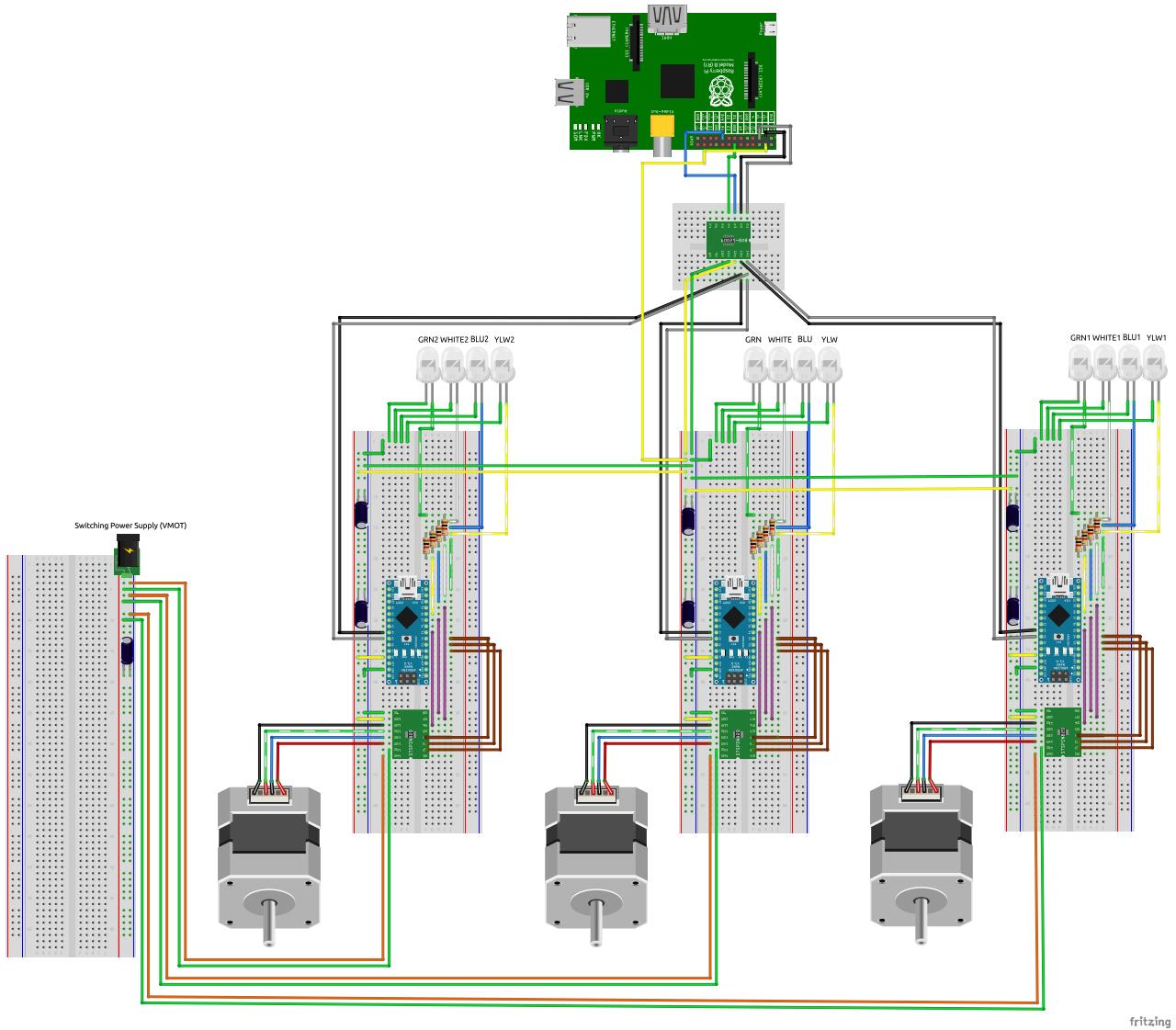


Fig. 44: Hardware Layout of the SMSS

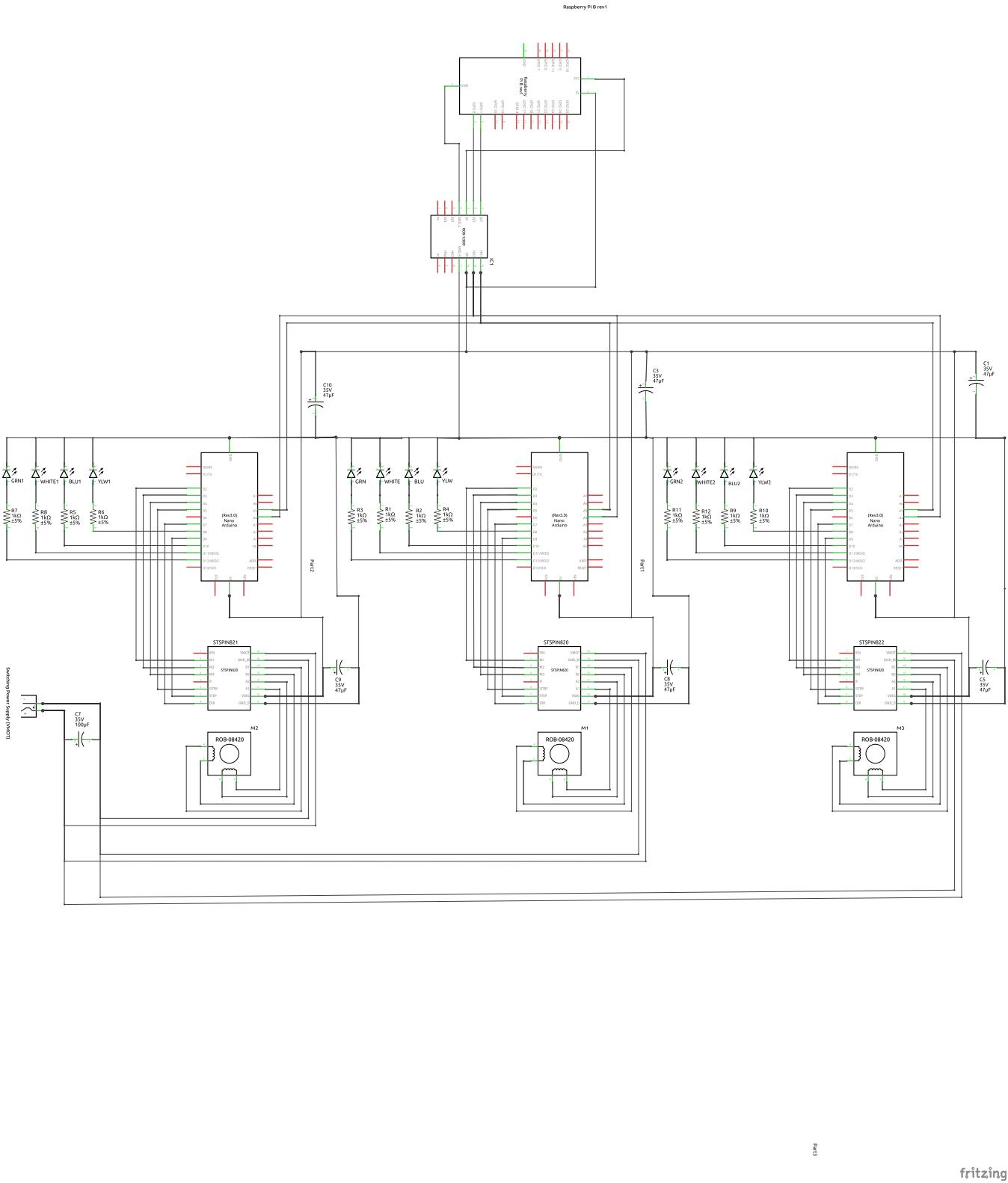


Fig. 45: Schematic of the SMSS

fritzing

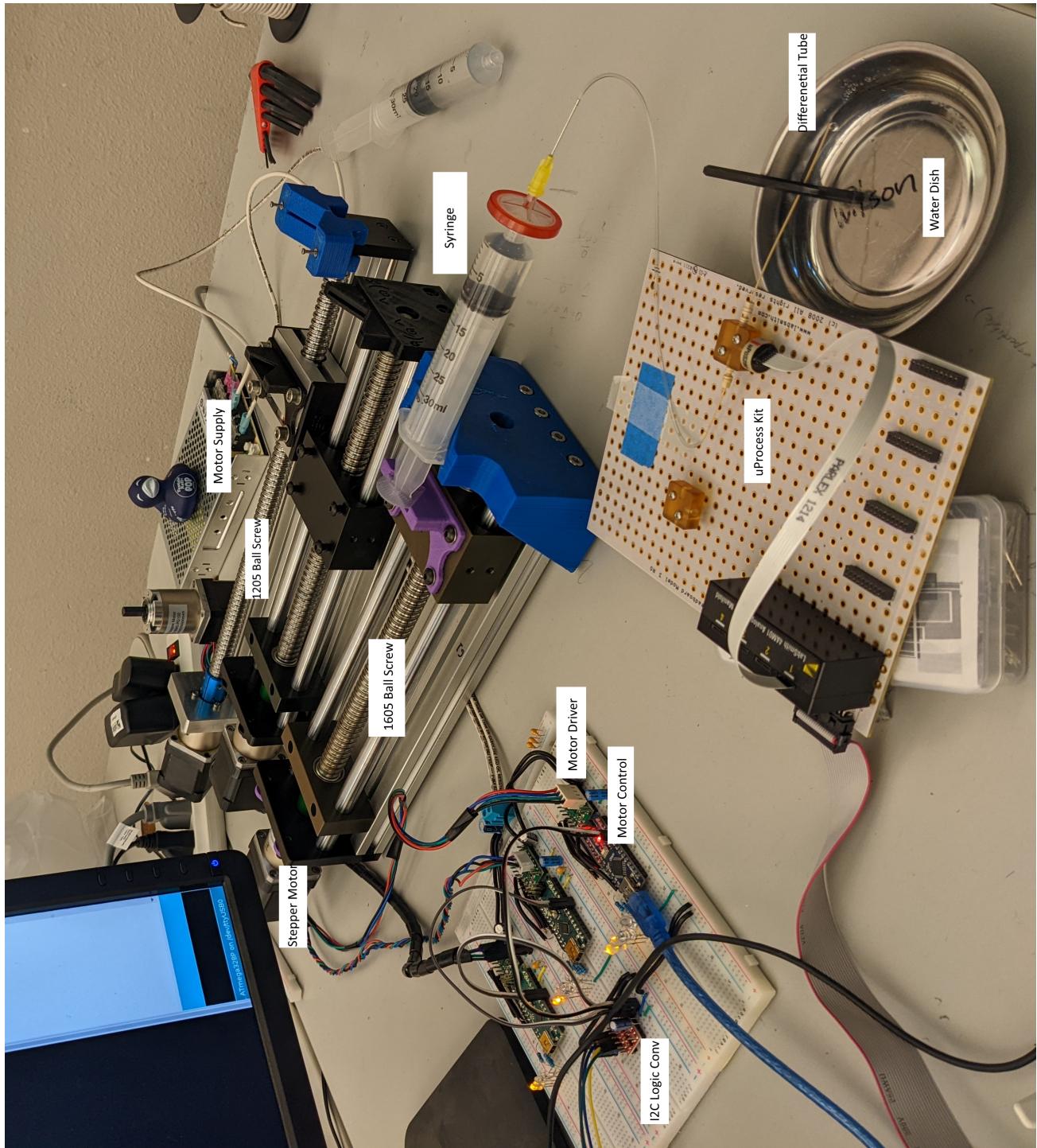


Fig. 46: Component Locations in the SMSS

APPENDIX C

CODE

A. Motor Controller

```

1 // Connor Wilson
2 // Winter 2022
3 // SMS Motor Controller V1.0
4
5 #include <TimerOne.h>           // Custom PWM freq ( ONLY WORKS WITH MEGA328 - NO MICRO BOARDS) - PWM
6     only for pins(9, 10) on nano
7 #include <Wire.h>                // I2C
8
9
10 // Test Variables (COMMENT OUT TO DESELECT) -- DONT FORGET TO CHANGE ARD ADDRESS (i2c_address) BELOW
11 #define TESTING                  // Comment to cancel testing
12 // #define SERIAL_COMM斯          // Comment to cancel serial comms [manual testing]
13 #define I2C_COMM斯              // Comment to cancel I2C comms
14 // #define STSPIN220             // Comment out to NOT use the STSPIN220 Pololu low-volt motor controller
15 #define STSPIN820               // Comment out to NOT use the STSPIN820 Pololu high-volt motor controller
16
17 // State machine
18 enum State_enum {STANDBY, DATAPULL_S, DATAPULL_J,
19     RUN_S, RUN_J, STOP}; // States of the state machine
20 unsigned int state = STANDBY; // default state
21
22 #define START 'S'
23 #define JOG 'J'
24 #define BACK 'B'
25 #define ESTOP 'E'
26 bool TICKET = false; // true if user supplied data is valid with hardware
27 bool FINISH = false; // true if user requested duration is matched by hardware
28
29
30 #ifdef STSPIN220 // Low voltage motor controller library
31 // Pin Connections
32 #define M1 2                  // microstep config
33     bit 1
34 #define M2 3                  // microstep config
35     bit 2
36 #define STBY 5                // Controller's standby enable (both EN and STBY must be low for low power mode)
37 #define STEP 9                // PWM signal for motor
38 #define DIR 7                 // Direction of motor
39 // NOTE : STEP, DIR, M2, M1 are all used to set the microstep in the init latching
40 // but only STEP and DIR are available after init
41
42 // LED State Indicators
43 #define YELLOW 8
44 #define BLUE 10
45 #define WHITE 11
46 #define GREEN 12
47
48
49
50 #define S_16      0xF
51 #define S_32      0x4
52 #define S_64      0xD
53 #define S_128     0x8
54 #define S_256     0x6
55 #define S_FULL   0x0
56 #define S_HALF   0x4
57 #define S_QTR    0x2
58 #define S_8       0x6
59 #define S_16      0x1
60 #define S_32      0x5
61 char step_config = S_FULL; // Set default to full step
62
63
64 #ifdef STSPIN820 // High voltage motor controller library
65 // Pin Connections
66 #define M1 2                  // microstep config
67     bit 1
68 #define M2 3                  // microstep config
69     bit 2
70 #define M3 4                  // microstep config
71     bit 3
72 #define STBY 5                // Controller's standby enable (both EN and STBY must be low for low power mode)
73 #define STEP 9                // PWM signal for motor
74 #define DIR 7                 // Direction of motor
75 // NOTE : M1-3 are used for microstep latching (ie STEP/DIR are not used for latching like the STSPIN220)
76
77 // LED State Indicators
78 #define YELLOW 8
79 #define BLUE 10
80 #define WHITE 11
81 #define GREEN 12
82
83 // Microstep settings for md39a[ M1, M2, STEP, DIR ]
84 #define S_FULL   0x0
85 #define S_HALF   0x4
86 #define S_QTR    0x2
87 #define S_8       0x6
88 #define S_16      0x1
89 #define S_32      0x5
90 // #define S_64      no 64 bit
91 #define S_128     0x3
92 #define S_256     0x7
93 char step_config = S_FULL; // Set default to full step
94
95
96 // Motor States
97 #define FORWARD LOW // motor direction for pin logic
98 #define REVERSE HIGH // motor direction for pin logic
99
100 bool DIRECTION = FORWARD; // motor direction
101 bool RUNNING = false; // true if motor is running
102
103
104
105
106
107 // Pulse/Freq Variables
108 #define DC 0.8 * 1024 // Duty Cycle of PWM
109 // [UNSURE WHAT RATIO IS BEST]
110 #define MTR_FREQ_MAX 875 // Max freq motor can handle [Hz]

```

```

110 #define MTR_RES_MIN 120 // Min 156 int motor_dir = FORWARD; // [ (1) -
    freq for smooth performance [Hz] // Min 157 REV, (0) - FWD, (2) - *BACK to STANDBY] *some
111 #define MTR_FREQ_MIN 0 // Min 158 cases(change 18r)
    freq motor can handle [Hz]
        [UNKNOWN]

112
113
114 // Physical Variables
115 #define INTERNAL_STEP 200 // gear
    Internal step size of the motor
116 #define GEAR_RATIO 99.05 // gear
    ratio of the planetary gear
117 #define THREAD_PITCH 0.005 // gear
    thread pitch of the actuator's screw [m]
118 float syr_capacity = 30; // gear
    capacity of syringe in mL
119 float syr_radius_user = 11; // gear
    RADIUS of the 30ml BD syringe [m]
120 float q_user = 0; // User
    volumetric flow rate as requested by user [L/s]
121 float freq = 200; // [sec]
    step
122 unsigned long sec_user = 0; // User
    requested sec duration
123 unsigned long min_user = 0; // User
    requested min duration
124 unsigned long hour_user = 0; // User
    " hour "
125 unsigned long duration = 0; // Run
    time {msec}
126 unsigned long time_stamp_start = 0; // time
    keeping
127 unsigned long time_stamp_end = 0; // "
    keeping

128
129
130 // Serial Variables
131 #ifdef SERIAL_COMM
132 char receivedChar; // pool
    for char data
133 boolean newData_char = false; // true
    if new char data is RX
134 boolean newData_str = false; // true
    if new str data is RX
135 const byte numChars = 32; // str
    length when TX
136 char receivedChars[numChars]; // pool
    for str data
137 bool displayed = false; // true
    if graphic is displayed
138#endif

139
140 // I2C Variables
141 #ifdef I2C_COMM
142
143 // I2C Addresses
144 #define PI_ADD 0x01 // Pi
    address
145 #define ARD_ADD_1 0x14 // 187
    Arduino #1
146 #define ARD_ADD_2 0x28 // 188
    Arduino #2
147 #define ARD_ADD_3 0x42 // 189
    Arduino #3
148 int i2c_address = ARD_ADD_3; // EDIT
    THIS TO CHANGE ADDRESS
149
150 // I2C Variables
151 int poll_user = 0; // run
    state of the ard [OFF(0), JOG(1), START(2) ]
152 int recData = 0; // storage
    for dictation codes
153 char strRX[2000]; // RX
    storage
154 char dur_mark = 0; // marker
    to aid recv/req track incoming hour, min, sec
155 bool datapull_flag = false; // 190
    completed datapull from I2C
156 int motor_dir = FORWARD; // [ (1) -
    REV, (0) - FWD, (2) - *BACK to STANDBY] *some
157 cases(change 18r)
158 // Pi's Dictating (main) codes
159 #define POLldata 1 // Ard
    will recv[OFF, JOG, START]
160 #define Q_DATA 2 // Ard
    will recv[q_user]
161 #define R_DATA 3 // Ard
    will recv[syr_radius_user]
162 #define CAP_DATA 4 // Ard
    will recv[syr_capacity]
163 #define DUR_DATA 5 // Ard
    will recv[hour_user, min_user, sec_user]
164 #define DIR_DATA 6 // Ard
    will recv[DIRECTION]
165 #define EVENT_DATA 7 // Ard
    will proceed to RUN
166 #define ESTOP_DATA 8 // Ard
    will proceed to STOP
167 #define REDO_DATA 9 // Ard
    will reset to STANDBY from DATAPULL
168 #define BLOCK_DATA 10 // User
    // Arduino's Requested (Sub) codes
169 #define LARGE_ERR 0x3 // User
    req too large (Q) for hardware constraints
170 #define SMALL_ERR 0x4 // User
    req too small (Q) for hardware constraints
171 #define DUR_ERR 0x5 // User
    req too long (time) for hardware constraints
172 #define LD_ERR 0x6 // User
    too large and long
173 #define SD_ERR 0x7 // user
    too small and long
174 #define DICT_ERR 0x8 // wrong
    dictating codes
175 #define NO_ERR 0x10 // No
    error to report
176 int error_code = NO_ERR; // User
177 #define WAIT_CODE 0x15 // Ard
    needs time to pull data and validate
178 #define FINISH_CODE 0x16 // Ard's
    duration has been reached and finished RUN_S
179 #define TIME_CODE 0x17 // Ard is
    not done with RUN_S and will return time(h,m,s)
    if prompted
180 bool redo_flag = false; // User
181 bool time_flag = false; // User
    help with req time code [ true - sent TIME_CODE
    already]
182 char block_recv = 0; // User
183 unsigned long block[6] = {0, 0, 0, 0, 0, 0}; // [
    BLOCK_DATA, dict_code, fac1, rem1, fac2, rem2]
    for data RX (I2c only transmitts 256 bit)
184
185
186
187
188
189 // Arduino checkpoints
190 bool dp_checkpoint = false; // true -
    ard has finished processing TICKET
191 bool eventstart = false; // true -
    Pi has req to RUN the DATAPULL'd ard
192 bool estop_user = false; // true -
    Pi req arduino to stop
193
194#endif
195
196
197 void setup() {
198
199 #ifdef SERIAL_COMM
200     Serial.begin(9600); // User
201     Serial comms init
202#endif

```

```

203
204 #ifdef I2C_COMMS
205
206 #ifdef TESTING
207   Serial.begin(9600);
208 #endif
209
210   Wire.begin(i2c_address); // I2C bus logon with sub address
211   Wire.onReceive(recvEvent); // run recvEvent on a main write to read <-
212   Wire.onRequest(reqEvent); // run reqEvent on a main read to write ->
213 #endif
214
215   pinMode(STEP, OUTPUT); // Setup controller pins
216   pinMode(DIR, OUTPUT);
217   pinMode(STBY, OUTPUT);
218   pinMode(M1, OUTPUT);
219   pinMode(M2, OUTPUT);
220
221 #ifdef STSPIN820
222   pinMode(M3, OUTPUT);
223 #endif
224
225   pinMode(BLUE, OUTPUT); // Indicator LED pin setup
226   pinMode(YELLOW, OUTPUT);
227   pinMode(GREEN, OUTPUT);
228   pinMode(WHITE, OUTPUT);
229   digitalWrite(DIR, REVERSE); // Direction of motor
230
231 }
232
233
234 void loop() {
235
236 #ifdef SERIAL_COMM
237   serial_state_machine();
238 #endif
239
240 #ifdef I2C_COMM
241   i2c_state_machine();
242 #endif
243 }
244
245
246 #ifdef STSPIN220
247 /*~~~~~ MtrCtrl_init_220()
248 ~~~~~
249 Purpose : initialize or change the low voltage
250 md39a controllers step size
251 Input : 0x4'b [ M1, M2, STEP, DIR ] as
252 calculated by the requested step setting
253 Outut : digital write to pins
254 Notes : ~EN pin is held low
255 */
256 void MtrCtrl_init_220(char pin_config) {
257
258 #ifdef TESTING
259   Serial.print("pin_config : ");
260   Serial.println(pin_config, HEX);
261 #endif
262
263   //The recommended power-up sequence is following:
264
265   //1. Power-up the device applying the VS supply
266   // voltage but keeping both STBY and EN/FAULT
267   // inputs low.
268   digitalWrite(STBY, LOW);
269
270   //2. Set the MODEx inputs according to the target
271   // step resolution (see Table 1).
272   digitalWrite(M3, pin_config & 1);
273   digitalWrite(M2, (pin_config >> 1) & 1);
274   digitalWrite(M1, (pin_config >> 2) & 1);
275
276 #ifdef TESTING
277   Serial.println(pin_config & 1);
278   Serial.println((pin_config >> 1) & 1);
279   Serial.println((pin_config >> 2) & 1);
280 #endif
281
282   //3. Wait for at least 1 microsec (minimum tMODEsu
283   // time).
284   delayMicroseconds(1);
285
286   //4. Set the STBY high. The MODEx configuration is
287   // now latched inside the device.
288   digitalWrite(STBY, HIGH);
289
290   //5. Wait for at least 100 microsec (minimum
291   // tMODEho time).
292   delayMicroseconds(100);
293
294   //6. Enable the power stage releasing the EN/FAULT
295   // input and start the operation.
296 }
297
298 #endif
299
300 #ifdef STSPIN820
301 /*~~~~~ MtrCtrl_init_820()
302 ~~~~~
303 Purpose : initialize or change the high voltage
304 md37a controllers step size
305 Input : 0x4'b [ M1, M2, M3 ] as calculated by
306 the requested step setting
307 Outut : digital write to pins
308 Notes : ~EN pin is held low
309 */
310 void MtrCtrl_init_820(char pin_config) {
311
312 #ifdef TESTING
313   Serial.print("pin_config : ");
314   Serial.println(pin_config, HEX);
315 #endif
316
317   //The recommended power-up sequence is following:
318
319   //1. Power-up the device applying the VS supply
320   // voltage but keeping both STBY and EN/FAULT
321   // inputs low.
322   digitalWrite(STBY, LOW);
323
324   //2. Set the MODEx inputs according to the target
325   // step resolution (see Table 1).
326   digitalWrite(M3, pin_config & 1);
327   digitalWrite(M2, (pin_config >> 1) & 1);
328   digitalWrite(M1, (pin_config >> 2) & 1);
329
330 #ifdef TESTING
331   Serial.println(pin_config & 1);
332   Serial.println((pin_config >> 1) & 1);
333   Serial.println((pin_config >> 2) & 1);
334 #endif
335
336   //3. Wait for at least 1 microsec (minimum tMODEsu
337   // time).
338   delayMicroseconds(1);
339
340   //4. Set the STBY high. The MODEx configuration is
341   // now latched inside the device.
342   digitalWrite(STBY, HIGH);
343
344   //5. Wait for at least 100 microsec (minimum
345   // tMODEho time).
346   delayMicroseconds(100);
347
348   //6. Enable the power stage releasing the EN/FAULT
349   // input and start the operation.
350 }
```

```

331 }
332
333 #endif
334
335 /*~~~~~ Q_to_Freq()
336 ~~~~~ Purpose : Convert the requested Q rate into a
337 frequency for the stepper motor
338 Input : q_user, syr_radius, Physical quantities
339 Outut : freq
340 */
341 float Q_to_Freq( float Q, float Radius) {
342     float freq = ( Q * INTERNAL_STEP * GEAR_RATIO) /
343         (500 * Radius * Radius * THREAD_PITCH);
344     return freq;
345 }
346
347 /*~~~~~ MicroCali()
348 ~~~~~ Purpose : Recalibrate the md39a motor controller
349 's microstep setting to accomidate the min freq
350 recommendation
351     Max freq will result in Full step at
352 max freq + warning
353 Input : frequency, [global] step_config, [global]
354 ] MTR_FREQ_MAX, [global] MTR_RES_MIN
355 Output : frequency, [global] step_config, [
356 global] TICKET
357 Notes : Removes decimal point to save memory
358 */
359 double MicroCali( float frequency ) {
360
361     // If greater than the max frequency
362     if ( frequency > MTR_FREQ_MAX) {
363
364         step_config = S_FULL;
365             // Set controller to full
366         step
367         TICKET = false;
368             // Mark ticket invalid
369
370 #ifdef SERIAL_COMMs
371     Serial.println(" Requested Q value exceeds
372 hardware's MAX RPS"); // Inform user
373 #endif
374
375 #ifdef I2C_COMMs
376     error_code = LARGE_ERR;
377 #endif
378
379     return MTR_FREQ_MAX;
380             // Set freq to max
381 }
382
383 // Or if less than the min freq
384 else if (frequency < MTR_RES_MIN) {
385
386     double microstep = MTR_RES_MIN / frequency;
387             // Microstep size
388
389     if ( microstep <= 2 ) {
390             // Half step
391     step_config = S_HALF;
392             // Set controller to half
393     step
394     TICKET = true;
395             // Approved ticket
396     return 2 * frequency;
397             // double freq
398 }
399
400 else if ( microstep <= 4 ) {
401     step_config = S_QTR;
402             // Set controller to half
403 }
404
405 step
406 TICKET = true;
407             // Approved ticket
408 return 4 * frequency;
409             // double freq
410
411 else if ( microstep <= 8 ) {
412     step_config = S_8;
413     TICKET = true;
414             // Approved ticket
415     return 8 * frequency;
416 }
417
418 else if ( microstep <= 16 ) {
419     step_config = S_16;
420     TICKET = true;
421             // Approved ticket
422     return 16 * frequency;
423 }
424
425 else if ( microstep <= 32 ) {
426     step_config = S_32;
427     TICKET = true;
428             // Approved ticket
429     return 32 * frequency;
430 }
431
432 else if ( microstep <= 64 ) {
433     // the STSPIN820 doesnt have 64 bit -> make it
434     128 bit instead
435 #ifdef STSPIN220
436     step_config = S_64;
437     TICKET = true;
438             // Approved ticket
439     return 64 * frequency;
440 #else
441     step_config = S_128;
442     TICKET = true;
443             // Approved ticket
444     return 128 * frequency;
445 #endif
446 }
447
448 else if ( microstep <= 128 ) {
449     step_config = S_128;
450     TICKET = true;
451             // Approved ticket
452     return 128 * frequency;
453 }
454
455 else if ( microstep <= 256 ) {
456     step_config = S_256;
457     TICKET = true;
458             // Approved ticket
459     return 256 * frequency;
460 }
461
462 else {
463     step_config = S_256;
464             // Set controller to
465     smallest step
466     TICKET = false;
467
468 #ifdef SERIAL_COMMs
469     Serial.println(" Requested Q value is below
470 hardware's MIN RPS"); // Inform user
471 #endif
472
473 #ifdef I2C_COMMs
474     error_code = SMALL_ERR;
475             // report error to pi
476 #endif
477
478 return MTR_RES_MIN;
479 }

```

```

        // Set freq to min
518
519     if (displayed == false) {
520         // dont print unless its the
521         // first
522         Serial.println(" ~~~~~Standby Mode
523 ~~~~~~ ");
524         Serial.println(" Press S for START mode or J
525 for JOG (No line ending) \n\n\n");
526         displayed = true;
527     }
528
529     recvOneChar();
530
531     if (receivedChar == START) {
532         // User selects START for normal use
533 #ifdef SERIAL_COMMs
534         displayed = false;
535         //graphic already displayed...no
536         more pls
537 #endif
538
539         digitalWrite(YELLOW, LOW);
540         // LED indicator
541         state = DATAPULL_S;
542     }
543
544     else if (receivedChar == JOG) {
545         // User selects JOG to jog motor
546 #ifdef SERIAL_COMMs
547         displayed = false;
548 #endif
549
550         digitalWrite(YELLOW, LOW);
551         // LED indicator
552         state = DATAPULL_J;
553
554     else {
555         state = STANDBY; // Or wait until user
556         chooses
557     }
558
559     break;
560
561     case DATAPULL_S:
562
563         digitalWrite(BLUE, HIGH);
564         // LED indicator
565
566         Serial.println(" ~~~~~DATAPULL_S~~~~~
567         ");
568         Serial.println(" What is the fluid capacity of
569         the syringe[mL]? (newline)");
570         while (newData_str == false) {
571             recvWithEndMarker();
572         }
573         syr_capacity = userString2Float() / 1000;
574
575         Serial.println(" What is the radius of the
576         syringe plunger[um]? (newline)");
577         while (newData_str == false) {
578             recvWithEndMarker();
579         }
580         syr_radius_user = userString2Float() /
581         1000000;
582
583         Serial.println(" What is the requested
584         volumetric flow rate Q [nL/s] ? Range [69n , 10u
585         ] (newline)");
586         while (newData_str == false) {
587             recvWithEndMarker();
588         }
589     }
590
591     void serial_state_machine() {
592
593         switch (state) {
594
595             case STANDBY:
596
597                 digitalWrite(YELLOW, HIGH); // LED
598
599             case JOG:
600
601                 digitalWrite(YELLOW, HIGH); // LED
602
603             case START:
604
605                 digitalWrite(YELLOW, HIGH); // LED
606
607             default:
608
609                 digitalWrite(YELLOW, LOW); // LED
610
611         }
612
613     }
614
615     // Set freq to min
616
617     if (displayed == false) {
618         // dont print unless its the
619         // first
620         Serial.println(" ~~~~~Standby Mode
621 ~~~~~~ ");
622         Serial.println(" Press S for START mode or J
623 for JOG (No line ending) \n\n\n");
624         displayed = true;
625     }
626
627     recvOneChar();
628
629     if (receivedChar == START) {
630         // User selects START for normal use
631 #ifdef SERIAL_COMMs
632         displayed = false;
633         //graphic already displayed...no
634         more pls
635 #endif
636
637         digitalWrite(YELLOW, LOW);
638         // LED indicator
639         state = DATAPULL_S;
640     }
641
642     else if (receivedChar == JOG) {
643         // User selects JOG to jog motor
644 #ifdef SERIAL_COMMs
645         displayed = false;
646 #endif
647
648         digitalWrite(YELLOW, LOW);
649         // LED indicator
650         state = DATAPULL_J;
651
652     else {
653         state = STANDBY; // Or wait until user
654         chooses
655     }
656
657     break;
658
659     case DATAPULL_S:
660
661         digitalWrite(BLUE, HIGH);
662         // LED indicator
663
664         Serial.println(" ~~~~~DATAPULL_S~~~~~
665         ");
666         Serial.println(" What is the fluid capacity of
667         the syringe[mL]? (newline)");
668         while (newData_str == false) {
669             recvWithEndMarker();
670         }
671         syr_capacity = userString2Float() / 1000;
672
673         Serial.println(" What is the radius of the
674         syringe plunger[um]? (newline)");
675         while (newData_str == false) {
676             recvWithEndMarker();
677         }
678         syr_radius_user = userString2Float() /
679         1000000;
680
681         Serial.println(" What is the requested
682         volumetric flow rate Q [nL/s] ? Range [69n , 10u
683         ] (newline)");
684         while (newData_str == false) {
685             recvWithEndMarker();
686         }
687     }
688
689     void serial_state_machine() {
690
691         switch (state) {
692
693             case STANDBY:
694
695                 digitalWrite(YELLOW, HIGH); // LED
696
697             case JOG:
698
699                 digitalWrite(YELLOW, HIGH); // LED
700
701             case START:
702
703                 digitalWrite(YELLOW, HIGH); // LED
704
705             default:
706
707                 digitalWrite(YELLOW, LOW); // LED
708
709         }
710
711     }
712
713     // Set freq to min
714
715     if (displayed == false) {
716         // dont print unless its the
717         // first
718         Serial.println(" ~~~~~Standby Mode
719 ~~~~~~ ");
720         Serial.println(" Press S for START mode or J
721 for JOG (No line ending) \n\n\n");
722         displayed = true;
723     }
724
725     recvOneChar();
726
727     if (receivedChar == START) {
728         // User selects START for normal use
729 #ifdef SERIAL_COMMs
730         displayed = false;
731         //graphic already displayed...no
732         more pls
733 #endif
734
735         digitalWrite(YELLOW, LOW);
736         // LED indicator
737         state = DATAPULL_S;
738     }
739
740     else if (receivedChar == JOG) {
741         // User selects JOG to jog motor
742 #ifdef SERIAL_COMMs
743         displayed = false;
744 #endif
745
746         digitalWrite(YELLOW, LOW);
747         // LED indicator
748         state = DATAPULL_J;
749
750     else {
751         state = STANDBY; // Or wait until user
752         chooses
753     }
754
755     break;
756
757     case DATAPULL_S:
758
759         digitalWrite(BLUE, HIGH);
760         // LED indicator
761
762         Serial.println(" ~~~~~DATAPULL_S~~~~~
763         ");
764         Serial.println(" What is the fluid capacity of
765         the syringe[mL]? (newline)");
766         while (newData_str == false) {
767             recvWithEndMarker();
768         }
769         syr_capacity = userString2Float() / 1000;
770
771         Serial.println(" What is the radius of the
772         syringe plunger[um]? (newline)");
773         while (newData_str == false) {
774             recvWithEndMarker();
775         }
776         syr_radius_user = userString2Float() /
777         1000000;
778
779         Serial.println(" What is the requested
780         volumetric flow rate Q [nL/s] ? Range [69n , 10u
781         ] (newline)");
782         while (newData_str == false) {
783             recvWithEndMarker();
784         }
785     }
786
787     void serial_state_machine() {
788
789         switch (state) {
790
791             case STANDBY:
792
793                 digitalWrite(YELLOW, HIGH); // LED
794
795             case JOG:
796
797                 digitalWrite(YELLOW, HIGH); // LED
798
799             case START:
800
801                 digitalWrite(YELLOW, HIGH); // LED
802
803             default:
804
805                 digitalWrite(YELLOW, LOW); // LED
806
807         }
808
809     }
810
811     // Set freq to min
812
813     if (displayed == false) {
814         // dont print unless its the
815         // first
816         Serial.println(" ~~~~~Standby Mode
817 ~~~~~~ ");
818         Serial.println(" Press S for START mode or J
819 for JOG (No line ending) \n\n\n");
820         displayed = true;
821     }
822
823     recvOneChar();
824
825     if (receivedChar == START) {
826         // User selects START for normal use
827 #ifdef SERIAL_COMMs
828         displayed = false;
829         //graphic already displayed...no
830         more pls
831 #endif
832
833         digitalWrite(YELLOW, LOW);
834         // LED indicator
835         state = DATAPULL_S;
836     }
837
838     else if (receivedChar == JOG) {
839         // User selects JOG to jog motor
840 #ifdef SERIAL_COMMs
841         displayed = false;
842 #endif
843
844         digitalWrite(YELLOW, LOW);
845         // LED indicator
846         state = DATAPULL_J;
847
848     else {
849         state = STANDBY; // Or wait until user
850         chooses
851     }
852
853     break;
854
855     case DATAPULL_S:
856
857         digitalWrite(BLUE, HIGH);
858         // LED indicator
859
860         Serial.println(" ~~~~~DATAPULL_S~~~~~
861         ");
861         Serial.println(" What is the fluid capacity of
862         the syringe[mL]? (newline)");
863         while (newData_str == false) {
864             recvWithEndMarker();
865         }
866         syr_capacity = userString2Float() / 1000;
867
868         Serial.println(" What is the radius of the
869         syringe plunger[um]? (newline)");
870         while (newData_str == false) {
871             recvWithEndMarker();
872         }
873         syr_radius_user = userString2Float() /
874         1000000;
875
876         Serial.println(" What is the requested
877         volumetric flow rate Q [nL/s] ? Range [69n , 10u
878         ] (newline)");
879         while (newData_str == false) {
880             recvWithEndMarker();
881         }
882     }
883
884     void serial_state_machine() {
885
886         switch (state) {
887
888             case STANDBY:
889
890                 digitalWrite(YELLOW, HIGH); // LED
891
892             case JOG:
893
894                 digitalWrite(YELLOW, HIGH); // LED
895
896             case START:
897
898                 digitalWrite(YELLOW, HIGH); // LED
899
900             default:
901
902                 digitalWrite(YELLOW, LOW); // LED
903
904         }
905
906     }
907
908     // Set freq to min
909
910     if (displayed == false) {
911         // dont print unless its the
912         // first
913         Serial.println(" ~~~~~Standby Mode
914 ~~~~~~ ");
915         Serial.println(" Press S for START mode or J
916 for JOG (No line ending) \n\n\n");
917         displayed = true;
918     }
919
920     recvOneChar();
921
922     if (receivedChar == START) {
923         // User selects START for normal use
924 #ifdef SERIAL_COMMs
925         displayed = false;
926         //graphic already displayed...no
927         more pls
928 #endif
929
930         digitalWrite(YELLOW, LOW);
931         // LED indicator
932         state = DATAPULL_S;
933     }
934
935     else if (receivedChar == JOG) {
936         // User selects JOG to jog motor
937 #ifdef SERIAL_COMMs
938         displayed = false;
939 #endif
940
941         digitalWrite(YELLOW, LOW);
942         // LED indicator
943         state = DATAPULL_J;
944
945     else {
946         state = STANDBY; // Or wait until user
947         chooses
948     }
949
950     break;
951
952     case DATAPULL_S:
953
954         digitalWrite(BLUE, HIGH);
955         // LED indicator
956
957         Serial.println(" ~~~~~DATAPULL_S~~~~~
958         ");
959         Serial.println(" What is the fluid capacity of
960         the syringe[mL]? (newline)");
961         while (newData_str == false) {
962             recvWithEndMarker();
963         }
964         syr_capacity = userString2Float() / 1000;
965
966         Serial.println(" What is the radius of the
967         syringe plunger[um]? (newline)");
968         while (newData_str == false) {
969             recvWithEndMarker();
970         }
971         syr_radius_user = userString2Float() /
972         1000000;
973
974         Serial.println(" What is the requested
975         volumetric flow rate Q [nL/s] ? Range [69n , 10u
976         ] (newline)");
977         while (newData_str == false) {
978             recvWithEndMarker();
979         }
980     }
981
982     void serial_state_machine() {
983
984         switch (state) {
985
986             case STANDBY:
987
988                 digitalWrite(YELLOW, HIGH); // LED
989
990             case JOG:
991
992                 digitalWrite(YELLOW, HIGH); // LED
993
994             case START:
995
996                 digitalWrite(YELLOW, HIGH); // LED
997
998             default:
999
1000                 digitalWrite(YELLOW, LOW); // LED
1001
1002         }
1003
1004     }
1005
1006     // Set freq to min
1007
1008     if (displayed == false) {
1009         // dont print unless its the
1010         // first
1011         Serial.println(" ~~~~~Standby Mode
1012 ~~~~~~ ");
1013         Serial.println(" Press S for START mode or J
1014 for JOG (No line ending) \n\n\n");
1015         displayed = true;
1016     }
1017
1018     recvOneChar();
1019
1020     if (receivedChar == START) {
1021         // User selects START for normal use
1022 #ifdef SERIAL_COMMs
1023         displayed = false;
1024         //graphic already displayed...no
1025         more pls
1026 #endif
1027
1028         digitalWrite(YELLOW, LOW);
1029         // LED indicator
1030         state = DATAPULL_S;
1031     }
1032
1033     else if (receivedChar == JOG) {
1034         // User selects JOG to jog motor
1035 #ifdef SERIAL_COMMs
1036         displayed = false;
1037 #endif
1038
1039         digitalWrite(YELLOW, LOW);
1040         // LED indicator
1041         state = DATAPULL_J;
1042
1043     else {
1044         state = STANDBY; // Or wait until user
1045         chooses
1046     }
1047
1048     break;
1049
1050     case DATAPULL_S:
1051
1052         digitalWrite(BLUE, HIGH);
1053         // LED indicator
1054
1055         Serial.println(" ~~~~~DATAPULL_S~~~~~
1056         ");
1056         Serial.println(" What is the fluid capacity of
1057         the syringe[mL]? (newline)");
1058         while (newData_str == false) {
1059             recvWithEndMarker();
1060         }
1061         syr_capacity = userString2Float() / 1000;
1062
1063         Serial.println(" What is the radius of the
1064         syringe plunger[um]? (newline)");
1065         while (newData_str == false) {
1066             recvWithEndMarker();
1067         }
1068         syr_radius_user = userString2Float() /
1069         1000000;
1070
1071         Serial.println(" What is the requested
1072         volumetric flow rate Q [nL/s] ? Range [69n , 10u
1073         ] (newline)");
1074         while (newData_str == false) {
1075             recvWithEndMarker();
1076         }
1077     }
1078
1079     void serial_state_machine() {
1080
1081         switch (state) {
1082
1083             case STANDBY:
1084
1085                 digitalWrite(YELLOW, HIGH); // LED
1086
1087             case JOG:
1088
1089                 digitalWrite(YELLOW, HIGH); // LED
1090
1091             case START:
1092
1093                 digitalWrite(YELLOW, HIGH); // LED
1094
1095             default:
1096
1097                 digitalWrite(YELLOW, LOW); // LED
1098
1099         }
1100
1101     }
1102
1103     // Set freq to min
1104
1105     if (displayed == false) {
1106         // dont print unless its the
1107         // first
1108         Serial.println(" ~~~~~Standby Mode
1109 ~~~~~~ ");
1110         Serial.println(" Press S for START mode or J
1111 for JOG (No line ending) \n\n\n");
1112         displayed = true;
1113     }
1114
1115     recvOneChar();
1116
1117     if (receivedChar == START) {
1118         // User selects START for normal use
1119 #ifdef SERIAL_COMMs
1120         displayed = false;
1121         //graphic already displayed...no
1122         more pls
1123 #endif
1124
1125         digitalWrite(YELLOW, LOW);
1126         // LED indicator
1127         state = DATAPULL_S;
1128     }
1129
1130     else if (receivedChar == JOG) {
1131         // User selects JOG to jog motor
1132 #ifdef SERIAL_COMMs
1133         displayed = false;
1134 #endif
1135
1136         digitalWrite(YELLOW, LOW);
1137         // LED indicator
1138         state = DATAPULL_J;
1139
1140     else {
1141         state = STANDBY; // Or wait until user
1142         chooses
1143     }
1144
1145     break;
1146
1147     case DATAPULL_S:
1148
1149         digitalWrite(BLUE, HIGH);
1150         // LED indicator
1151
1152         Serial.println(" ~~~~~DATAPULL_S~~~~~
1153         ");
1153         Serial.println(" What is the fluid capacity of
1154         the syringe[mL]? (newline)");
1155         while (newData_str == false) {
1156             recvWithEndMarker();
1157         }
1158         syr_capacity = userString2Float() / 1000;
1159
1160         Serial.println(" What is the radius of the
1161         syringe plunger[um]? (newline)");
1162         while (newData_str == false) {
1163             recvWithEndMarker();
1164         }
1165         syr_radius_user = userString2Float() /
1166         1000000;
1167
1168         Serial.println(" What is the requested
1169         volumetric flow rate Q [nL/s] ? Range [69n , 10u
1170         ] (newline)");
1171         while (newData_str == false) {
1172             recvWithEndMarker();
1173         }
1174     }
1175
1176     void serial_state_machine() {
1177
1178         switch (state) {
1179
1180             case STANDBY:
1181
1182                 digitalWrite(YELLOW, HIGH); // LED
1183
1184             case JOG:
1185
1186                 digitalWrite(YELLOW, HIGH); // LED
1187
1188             case START:
1189
1190                 digitalWrite(YELLOW, HIGH); // LED
1191
1192             default:
1193
1194                 digitalWrite(YELLOW, LOW); // LED
1195
1196         }
1197
1198     }
1199
1200     // Set freq to min
1201
1202     if (displayed == false) {
1203         // dont print unless its the
1204         // first
1205         Serial.println(" ~~~~~Standby Mode
1206 ~~~~~~ ");
1206         Serial.println(" Press S for START mode or J
1207 for JOG (No line ending) \n\n\n");
1208         displayed = true;
1209     }
1210
1211     recvOneChar();
1212
1213     if (receivedChar == START) {
1214         // User selects START for normal use
1215 #ifdef SERIAL_COMMs
1216         displayed = false;
1217         //graphic already displayed...no
1218         more pls
1219 #endif
1220
1221         digitalWrite(YELLOW, LOW);
1222         // LED indicator
1223         state = DATAPULL_S;
1224     }
1225
1226     else if (receivedChar == JOG) {
1227         // User selects JOG to jog motor
1228 #ifdef SERIAL_COMMs
1229         displayed = false;
1230 #endif
1231
1232         digitalWrite(YELLOW, LOW);
1233         // LED indicator
1234         state = DATAPULL_J;
1235
1236     else {
1237         state = STANDBY; // Or wait until user
1238         chooses
1239     }
1240
1241     break;
1242
1243     case DATAPULL_S:
1244
1245         digitalWrite(BLUE, HIGH);
1246         // LED indicator
1247
1248         Serial.println(" ~~~~~DATAPULL_S~~~~~
1249         ");
1250         Serial.println(" What is the fluid capacity of
1251         the syringe[mL]? (newline)");
1252         while (newData_str == false) {
1253             recvWithEndMarker();
1254         }
1255         syr_capacity = userString2Float() / 1000;
1256
1257         Serial.println(" What is the radius of the
1258         syringe plunger[um]? (newline)");
1259         while (newData_str == false) {
1260             recvWithEndMarker();
1261         }
1262         syr_radius_user = userString2Float() /
1263         1000000;
1264
1265         Serial.println(" What is the requested
1266         volumetric flow rate Q [nL/s] ? Range [69n , 10u
1267         ] (newline)");
1268         while (newData_str == false) {
1269             recvWithEndMarker();
1270         }
1271     }
1272
1273     void serial_state_machine() {
1274
1275         switch (state) {
1276
1277             case STANDBY:
1278
1279                 digitalWrite(YELLOW, HIGH); // LED
1280
1281             case JOG:
1282
1283                 digitalWrite(YELLOW, HIGH); // LED
1284
1285             case START:
1286
1287                 digitalWrite(YELLOW, HIGH); // LED
1288
1289             default:
1290
1291                 digitalWrite(YELLOW, LOW); // LED
1292
1293         }
1294
1295     }
1296
1297     // Set freq to min
1298
1299     if (displayed == false) {
1300         // dont print unless its the
1301         // first
1302         Serial.println(" ~~~~~Standby Mode
1303 ~~~~~~ ");
1303         Serial.println(" Press S for START mode or J
1304 for JOG (No line ending) \n\n\n");
1305         displayed = true;
1306     }
1307
1308     recvOneChar();
1309
1310     if (receivedChar == START) {
1311         // User selects START for normal use
1312 #ifdef SERIAL_COMMs
1313         displayed = false;
1314         //graphic already displayed...no
1315         more pls
1316 #endif
1317
1318         digitalWrite(YELLOW, LOW);
1319         // LED indicator
1320         state = DATAPULL_S;
1321     }
1322
1323     else if (receivedChar == JOG) {
1324         // User selects JOG to jog motor
1325 #ifdef SERIAL_COMMs
1326         displayed = false;
1327 #endif
1328
1329         digitalWrite(YELLOW, LOW);
1330         // LED indicator
1331         state = DATAPULL_J;
1332
1333     else {
1334         state = STANDBY; // Or wait until user
1335         chooses
1336     }
1337
1338     break;
1339
1340     case DATAPULL_S:
1341
1342         digitalWrite(BLUE, HIGH);
1343         // LED indicator
1344
1345         Serial.println(" ~~~~~DATAPULL_S~~~~~
1346         ");
1346         Serial.println(" What is the fluid capacity of
1347         the syringe[mL]? (newline)");
1348         while (newData_str == false) {
1349             recvWithEndMarker();
1350         }
1351         syr_capacity = userString2Float() / 1000;
1352
1353         Serial.println(" What is the radius of the
1354         syringe plunger[um]? (newline)");
1355         while (newData_str == false) {
1356             recvWithEndMarker();
1357         }
1358         syr_radius_user = userString2Float() /
1359         1000000;
1360
1361         Serial.println(" What is the requested
1362         volumetric flow rate Q [nL/s] ? Range [69n , 10u
1363         ] (newline)");
1364         while (newData_str == false) {
1365             recvWithEndMarker();
1366         }
1367     }
1368
1369     void serial_state_machine() {
1370
1371         switch (state) {
1372
1373             case STANDBY:
1374
1375                 digitalWrite(YELLOW, HIGH); // LED
1376
1377             case JOG:
1378
1379                 digitalWrite(YELLOW, HIGH); // LED
1380
1381             case START:
1382
1383                 digitalWrite(YELLOW, HIGH); // LED
1384
1385             default:
1386
1387                 digitalWrite(YELLOW, LOW); // LED
1388
1389         }
1390
1391     }
1392
1393     // Set freq to min
1394
1395     if (displayed == false) {
1396         // dont print unless its the
1397         // first
1398         Serial.println(" ~~~~~Standby Mode
1399 ~~~~~~ ");
1400         Serial.println(" Press S for START mode or J
1401 for JOG (No line ending) \n\n\n");
1401         displayed = true;
1402     }
1403
1404     recvOneChar();
1405
1406     if (receivedChar == START) {
1407         // User selects START for normal use
1408 #ifdef SERIAL_COMMs
1409         displayed = false;
1410         //graphic already displayed...no
1411         more pls
1412 #endif
1413
1414         digitalWrite(YELLOW, LOW);
1415         // LED indicator
1416         state = DATAPULL_S;
1417     }
1418
1419     else if (receivedChar == JOG) {
1420         // User selects JOG to jog motor
1421 #ifdef SERIAL_COMMs
1422         displayed = false;
1423 #endif
1424
1425         digitalWrite(YELLOW, LOW);
1426         // LED indicator
1427         state = DATAPULL_J;
1428
1429     else {
1430         state = STANDBY; // Or wait until user
1431         chooses
1432     }
1433
1434     break;
1435
1436     case DATAPULL_S:
1437
1438         digitalWrite(BLUE, HIGH);
1439         // LED indicator
1440
1441         Serial.println(" ~~~~~DATAPULL_S~~~~~
1442         ");
1442         Serial.println(" What is the fluid capacity of
1443         the syringe[mL]? (newline)");
1444         while (newData_str == false) {
1445             recvWithEndMarker();
1446         }
1447         syr_capacity = userString2Float() / 1000;
1448
1449         Serial.println(" What is the radius of the
1450         syringe plunger[um]? (newline)");
1451         while (newData_str == false) {
1452             recvWithEndMarker();
1453         }
1454         syr_radius_user = userString2Float() /
1455         1000000;
1456
1457         Serial.println(" What is the requested
1458         volumetric flow rate Q [nL/s] ? Range [69n , 10u
1459         ] (newline)");
1460         while (newData_str == false) {
1461             recvWithEndMarker();
1462         }
1463     }
1464
1465     void serial_state_machine() {
1466
1467         switch (state) {
1468
1469             case STANDBY:
1470
1471                 digitalWrite(YELLOW, HIGH); // LED
1472
1473             case JOG:
1474
1475                 digitalWrite(YELLOW, HIGH); // LED
1476
1477             case START:
1478
1479                 digitalWrite(YELLOW, HIGH); // LED
1480
1481             default:
1482
1483                 digitalWrite(YELLOW, LOW); // LED
1484
1485         }
1486
1487     }
1488
1489     // Set freq to min
1490
1491     if (displayed == false) {
1492         // dont print unless its the
1493         // first
1494         Serial.println(" ~~~~~Standby Mode
1495 ~~~~~~ ");
1495         Serial.println(" Press S for START mode or J
1496 for JOG (No line ending) \n\n\n");
1497         displayed = true;
1498     }
1499
1500     recvOneChar();
1501
1502     if (receivedChar == START) {
1503         // User selects START for normal use
1504 #ifdef SERIAL_COMMs
1505         displayed = false;
1506         //graphic already displayed...no
1507         more pls
1508 #endif
1509
1510         digitalWrite(YELLOW, LOW);
1511         // LED indicator
1512         state = DATAPULL_S;
1513     }
1514
1515     else if (receivedChar == JOG) {
1516         // User selects JOG to jog motor
1517 #ifdef SERIAL_COMMs
1518         displayed = false;
1519 #endif
1520
1521         digitalWrite(YELLOW, LOW);
1522         // LED indicator
1523         state = DATAPULL_J;
1524
1525     else {
1526         state = STANDBY; // Or wait until user
1527         chooses
1528     }
1529
1530     break;
1531
1532     case DATAPULL_S:
1533
1534         digitalWrite(BLUE, HIGH);
1535         // LED indicator
1536
1537         Serial.println(" ~~~~~DATAPULL_S~~~~~
1538         ");
1538         Serial.println(" What is the fluid capacity of
1539         the syringe[mL]? (newline)");
1540         while (newData_str == false) {
1541             recvWithEndMarker();
1542         }
1543         syr_capacity = userString2Float() / 1000;
1544
1545         Serial.println(" What is the radius of the
1546         syringe plunger[um]? (newline)");
1547         while (newData_str == false) {
1548             recvWithEndMarker();
1549         }
1550         syr_radius_user = userString2Float() /
1551         1000000;
1552
1553         Serial.println(" What is the requested
1554         volumetric flow rate Q [nL/s] ? Range [69n , 10u
1555         ] (newline)");
1556         while (newData_str == false) {
1557             recvWithEndMarker();
1558         }
1559     }
1560
1561     void serial_state_machine() {
1562
1563         switch (state) {
1564
1565             case STANDBY:
1566
1567                 digitalWrite(YELLOW, HIGH); // LED
1568
1569             case JOG:
1570
1571                 digitalWrite(YELLOW, HIGH); // LED
1572
1573             case START:
1574
1575                 digitalWrite(YELLOW, HIGH); // LED
1576
1577             default:
1578
1579                 digitalWrite(YELLOW, LOW); // LED
1580
1581         }
1582
1583     }
1584
1585     // Set freq to min
1586
1587     if (displayed == false) {
1588         // dont print unless its the
1589         // first
1590         Serial.println(" ~~~~~Standby Mode
1591 ~~~~~~ ");
1591         Serial.println(" Press S for START mode or J
1592 for JOG (No line ending) \n\n\n");
1593         displayed = true;
1594     }
1595
1596     recvOneChar();
1597
1598     if (receivedChar == START) {
1599         // User selects START for normal use
1600 #ifdef SERIAL_COMMs
1601         displayed = false;
1602         //graphic already displayed...no
1603         more pls
1604 #endif
1605
1606         digitalWrite(YELLOW, LOW);
1607         // LED indicator
1608         state = DATAPULL_S;
1609     }
1610
1611     else if (receivedChar == JOG) {
1612         // User selects JOG to jog motor
1613 #ifdef SERIAL_COMMs
1614         displayed = false;
1615 #endif
1616
1617         digitalWrite(YELLOW, LOW);
1618         // LED indicator
1619         state = DATAPULL_J;
1620
1621     else {
1622         state = STANDBY; // Or wait until user
1623         chooses
1624     }
1625
1626     break;
1627
1628     case DATAPULL_S:
1629
1630         digitalWrite(BLUE, HIGH);
1631         // LED indicator
1632
1633         Serial.println(" ~~~~~DATAPULL_S~~~~~
1634         ");
1634         Serial.println(" What is the fluid capacity of
1635         the syringe[mL]? (newline)");
1636         while (newData_str == false) {
1637             recvWithEndMarker();
1638         }
1639         syr_capacity = userString2Float() / 1000;
1640
1641         Serial.println(" What is the radius of the
1642         syringe plunger[um]? (newline)");
1643         while (newData_str == false) {
1644             recvWithEndMarker();
1645         }
1646         syr_radius_user = userString2Float() /
1647         1000000;
1648
1649         Serial.println(" What is the requested
1650         volumetric flow rate Q [nL/s] ? Range [69n , 10u
1651         ] (newline)");
1652         while (newData_str == false) {
1653             recvWithEndMarker();
1654         }
1655     }
1656
1657     void serial_state_machine() {
1658
1659         switch (state) {
1660
1661             case STANDBY:
1662
1663                 digitalWrite(YELLOW, HIGH); // LED
1664
1665             case JOG:
1666
1667                 digitalWrite(YELLOW, HIGH); // LED
1668
1669             case START:
1670
1671                 digitalWrite(YELLOW, HIGH); // LED
1672
1673             default:
1674
1675                 digitalWrite(YELLOW, LOW); // LED
1676
1677         }
1678
1679     }
1680
1681     // Set freq to min
1682
1683     if (displayed == false) {
1684         // dont print unless its the
1685         // first
1686         Serial.println(" ~~~~~Standby Mode
1687 ~~~~~~ ");
1687         Serial.println(" Press S for START mode or J
1688 for JOG (No line ending) \n\n\n");
1689         displayed = true;
1690     }
1691
1692     recvOneChar();
1693
1694     if (receivedChar == START) {
1695         // User selects START for normal use
1696 #ifdef SERIAL_COMMs
1697         displayed = false;
1698         //graphic already displayed...no
1699         more pls
1700 #endif
1701
1702         digitalWrite(YELLOW, LOW);
1703         // LED indicator
1704         state = DATAPULL_S;
1705     }
1706
1707     else if (receivedChar == JOG) {
1708         // User selects JOG to jog motor
1709 #ifdef SERIAL_COMMs
1710         displayed = false;
1711 #endif
1712
1713         digitalWrite(YELLOW, LOW);
1714         // LED indicator
1715         state = DATAPULL_J;
1716
1717     else {
1718         state = STANDBY; // Or wait until user
1719         chooses
1720     }
1721
1722     break;
1723
1724     case DATAPULL_S:
1725
1726         digitalWrite(BLUE, HIGH);
1727         // LED indicator
1728
1729         Serial.println(" ~~~~~DATAPULL_S~~~~~
1730         ");
1730         Serial.println(" What is the fluid capacity of
1731         the syringe[mL]? (newline)");
1732         while (newData_str == false) {
1733             recvWithEndMarker();
1734         }
1735         syr_capacity = userString2Float() / 1000;
1736
1737         Serial.println(" What is the radius of the
1738         syringe plunger[um]? (newline)");
1739         while (newData_str == false) {
1740             recvWithEndMarker();
1741         }
1742         syr_radius_user = userString2Float() /
1743         1000000;
1744
1745         Serial.println(" What is the requested
1746         volum
```

```

576     q_user = userString2Float() / 1000000000;           645     freq = Q_to_Freq(q_user, syr_radius_user);          // Get period from Q
577     Serial.println(" Duration hours? (newline)");      646     freq = MicroCalib(freq);                          // Calibrate for
578     while (newData_str == false) {                      647     microstepping + TICKET
579         recvWithEndMarker();                           648
580     }                                                 649
581     hour_user = userString2Float();                   650
582
583     Serial.println(" Duration minutes? (newline)");   651     if (duration == 0) {
584 ;                                              652         TICKET = false;
585     while (newData_str == false) {                     653     }
586         recvWithEndMarker();                           654     else if (TICKET == true) {
587     }                                                 655         validDur_Start(q_user, syr_capacity,
588     min_user = userString2Float();                   656         duration); // final TICKET confirm duration
589
590     Serial.println(" Duration seconds? (newline)");  657     }
591 ;                                              658
592     while (newData_str == false) {                     659     if (receivedChar == BACK) {
593         recvWithEndMarker();                           660         digitalWrite(BLUE, LOW);
594     }                                                 661             // LED indicator
595     sec_user = userString2Float();                   662         state = STANDBY;
596
597     Serial.println(" Direction? F for forward, R    663             // User requests to go back
598 for reverse, or B to back to Standby (No line 664     else if (TICKET) {
599 end)");                                         665             // Must lie within Q
600     newData_char = false;                           666     and duration range
601     while (newData_char == false) {                 667         digitalWrite(BLUE, LOW);
602         recvOneChar();                            668             // LED indicator
603     }                                                 669         state = RUN_S;
604     if (receivedChar == 'R') {                      670             // Valid data, move forward
605         DIRECTION = REVERSE;                      671     else {
606
607 #ifdef TESTING                                672         state = DATAPULL_S; // Repeat if the data
608     Serial.println(" Capacity, raduis, Q, hour, 673     wasnt RX/right
609     min, sec, Dir ");                           674     }
610     Serial.print(syr_capacity);                  675
611     Serial.print(", ");                         676     case DATAPULL_J:
612     Serial.print(syr_radius_user, 8);            677         digitalWrite(BLUE, HIGH);
613     Serial.print(", ");                         678             // LED indicator
614     Serial.print(q_user, 10);                   679         digitalWrite(WHITE, HIGH);
615     Serial.print(", ");                         680             // LED indicator
616     Serial.print(hour_user);                  681
617     Serial.print(", ");                         682     Serial.println("\n\n\n\n~~~~~");
618     Serial.print(min_user);                    683     DATAPULL_JOG~~~~~ ");
619     Serial.print(sec_user);                    684     Serial.println(" What is the radius of the
620     Serial.print(", ");                      685     syringe plunger [um]? (newline)");
621     Serial.println("Reverse\n\n");               686     while (newData_str == false) {
622
623 #endif                                         687         recvWithEndMarker();
624     }                                              688         }
625     else if (receivedChar == 'F') {              689     syr_radius_user = userString2Float() / 1000000;
626         DIRECTION = FORWARD;                   690
627
628 #ifdef TESTING                                691     Serial.println(" What is the requested
629     Serial.println(" Capacity, raduis, Q, hour, 692     volumetric flow rate Q [nL/s] ? Range [69n , 10u
630     min, sec, Dir ");                           693     ] (newline)");
631     Serial.print(syr_capacity);                  694     while (newData_str == false) {
632     Serial.print(", ");                         695         recvWithEndMarker();
633     Serial.print(syr_radius_user, 8);            696     }
634     Serial.print(q_user, 10);                   697     q_user = userString2Float() / 1000000000;
635     Serial.print(", ");                         698
636     Serial.print(hour_user);                  699     Serial.println(" Direction? F for forward, R
637     Serial.print(", ");                      700     for reverse, or B to back to Standby (No line
638     Serial.print(min_user);                    ending)");
639 #endif                                         701     newData_char = false;
640
641     duration = ( (3600 * hour_user) + (60 * 702     while (newData_char == false) {
642         min_user) + sec_user); // entire duration in 703         recvOneChar();
643         // msec for easy compare later           704     }
644

```

```

701 #ifdef TESTING
702   Serial.println("radius, Q, freq ");
703   Serial.print(syr_radius_user, 8);
704   Serial.print(",");
705   Serial.print(q_user, 10);
706   Serial.print(",");
707   Serial.println(freq);
708 #endif
709
710   freq = MicroCalib(freq);           // Calibrate
711   // microstepping & TICKET validation
712
713 #ifdef TESTING
714   Serial.println("radius, Q, freq ");
715   Serial.print(syr_radius_user, 8);
716   Serial.print(",");
717   Serial.print(q_user, 10);
718   Serial.print(",");
719   Serial.println(freq);
720 #endif
721
722   if (receivedChar == BACK) {        // User requests to go
723     back
724     //reset metrics and go back
725     freq = 0;
726     digitalWrite(BLUE, LOW);          // LED indicator
727     digitalWrite(WHITE, LOW);         // LED indicator
728     state = STANDBY;
729   }
730
731   else if (TICKET == true) {
732     digitalWrite(BLUE, LOW);          // LED indicator
733     digitalWrite(WHITE, LOW);         // LED indicator
734     state = RUN_J;
735   }
736
737   else {
738     state = DATAPULL_J; // Repeat if the data
739     wasnt RX/right
740   }
741
742   break;
743
744 case RUN_J:
745
746   digitalWrite(GREEN, HIGH);          // LED indicator
747   digitalWrite(WHITE, HIGH);          // LED indicator
748
749   // Allowing arduino to roam through cycles and
750   // not wait for user input while running
751   if (displayed == false) {
752     Serial.println(" ~~~~~ Run_JOG
753     ~~~~~ ");
754     Serial.println(" Press E for ESTOP (No line
755     ending)");
756     displayed = true;
757   }
758
759   recvOneChar();
760
761   if (receivedChar == ESTOP) {
762     digitalWrite(GREEN, LOW);          // LED indicator
763     digitalWrite(WHITE, LOW);          // LED indicator
764     state = STOP;
765   }
766
767   else if (RUNNING == false) {
768
769 #ifdef STSPIN220
770     MtrCtrl_init_220(step_config);
771     // setup motor controller
772 #else
773     MtrCtrl_init_820(step_config);
774     // setup motor controller
775 #endif
776
777     digitalWrite(DIR, DIRECTION);
778     // rewrite DIR pin for direction
779     Timer1.initialize((1 / freq) * 1000000);
780     // Init PWM freq[microsec] (mult to
781     conv to usec)
782     Timer1.pwm(STEP, DC);
783     // Start motor
784     RUNNING = true;
785     state = RUN_J;
786   }
787
788   else {
789     state = RUN_J;
790   }
791
792   break;
793
794 case RUN_S:
795
796   digitalWrite(GREEN, HIGH);          // LED indicator
797
798   if (displayed == false) {
799     Serial.println(" ~~~~~ Run_START
800     ~~~~~ ");
801     Serial.println(" Press E for ESTOP (No line
802     ending)\n\n\n");
803     displayed = true;
804   }
805
806   recvOneChar();
807
808   if (receivedChar == ESTOP || FINISH == true) {
809     displayed = false;
810     digitalWrite(GREEN, LOW);          // LED indicator
811     state = STOP;
812   }
813
814   // init run of motor
815   else if (RUNNING == false) {
816
817 #ifdef STSPIN220
818     MtrCtrl_init_220(step_config);
819     // setup motor controller
820 #else
821     MtrCtrl_init_820(step_config);
822     // setup motor controller
823 #endif
824
825     digitalWrite(DIR, DIRECTION);
826     // rewrite DIR pin for direction
827     Timer1.initialize((1 / freq) * 1000000);
828     // Init PWM freq[microsec] (mult to
829     conv to usec)
830     time_stamp_start = millis();
831     // Start recording time
832     Timer1.pwm(STEP, DC);
833     // Start motor
834     RUNNING = true;
835     state = RUN_S;
836   }
837
838 }
```

```

// Running until final time
else {
    time_stamp_end = millis();
    time_stamp_end = (time_stamp_end -
time_stamp_start); // total elapsed time
    if (time_stamp_end / 1000 >= duration) {
        FINISH = true; // if time had reached user
's request
    }
    state = RUN_S;
}

break;

case STOP:
    digitalWrite(STBY, LOW);
    // Place motor controller
    in standby
    Timer1.disablePwm(STEP);
    // turn off motor
    RUNNING = false;
    TICKET = false;
    displayed = false;
    FINISH = false;
    q_user = 0;
    // clear user variables
    syr_radius_user = 0;
    freq = 0;
    duration = 0;
    hour_user = 0;
    min_user = 0;
    sec_user = 0;
    state = STANDBY;
    break;

default:
    state = STOP;
    break;
}

}

/*~~~~~ recvOneChar()
~~~~~ Purpose : Receive one char from the user
Input : serial
Output : [global] newData
*/
void recvOneChar() {
    if (Serial.available() > 0) {
        receivedChar = Serial.read();
        newData_char = true;
    }
}

/*~~~~~ recvWithEndMarker()
~~~~~ Purpose : Receive set of char (up to 32) from the
user
Input : serial
Output : [global] receivedChars[]
*/
void recvWithEndMarker() {
    static byte ndx = 0;
    char endMarker = '\n';
    char rc;

    while (Serial.available() > 0 && newData_str ==
false) {
        // Running until final time
        else {
            time_stamp_end = millis();
            time_stamp_end = (time_stamp_end -
time_stamp_start); // total elapsed time
            if (time_stamp_end / 1000 >= duration) {
                FINISH = true; // if time had reached user
's request
            }
            state = RUN_S;
        }

        break;

    case STOP:
        digitalWrite(STBY, LOW);
        // Place motor controller
        in standby
        Timer1.disablePwm(STEP);
        // turn off motor
        RUNNING = false;
        TICKET = false;
        displayed = false;
        FINISH = false;
        q_user = 0;
        // clear user variables
        syr_radius_user = 0;
        freq = 0;
        duration = 0;
        hour_user = 0;
        min_user = 0;
        sec_user = 0;
        state = STANDBY;
        break;

    default:
        state = STOP;
        break;
    }

}

/*~~~~~ recvOneChar()
~~~~~ Purpose : Receive one char from the user
Input : serial
Output : [global] newData
*/
void recvOneChar() {
    if (Serial.available() > 0) {
        receivedChar = Serial.read();
        newData_char = true;
    }
}

/*~~~~~ recvWithEndMarker()
~~~~~ Purpose : Receive set of char (up to 32) from the
user
Input : serial
Output : [global] receivedChars[]
*/
void recvWithEndMarker() {
    static byte ndx = 0;
    char endMarker = '\n';
    char rc;

    while (Serial.available() > 0 && newData_str ==
false) {
        rc = Serial.read();

        if (rc != endMarker) {
            receivedChars[ndx] = rc;
            ndx++;
            if (ndx >= numChars) {
                ndx = numChars - 1;
            }
        } else {
            receivedChars[ndx] = '\0'; // terminate the
string
            ndx = 0;
            newData_str = true;
        }
    }
}

/*~~~~~ userString2Float()
~~~~~ Purpose : convert recvWithEndMarker's output
array of char receivedChars[ndx] to float &
continues data flow
Input : [global] char receivedChars[]
Output : float
*/
float userString2Float() {
    if (newData_str == true) {
        String user_str;
        user_str = String(receivedChars);
        newData_str = false;
        return user_str.toFloat();
    }
}

#endif

#endif

/*~~~~~ i2c_state_machine()
~~~~~ Purpose : Control the flow of the motor
controller
Input : I2C user input signals
Output :
*/
void i2c_state_machine() {
    switch (state) {

    case STANDBY:

        digitalWrite(YELLOW, HIGH);
        // LED indicator

        if (estop_user == true) {
            state = STOP;
        }

        else if (poll_user == 2) {
            // User selects START for
normal use
            digitalWrite(YELLOW, LOW);
            // LED indicator
            state = DATAPULL_S;
        }
    }
}

```

```

955 }
956
957 else if (poll_user == 1) {
958     // User selects JOG to jog motor
959     digitalWrite(YELLOW, LOW);
960     // LED indicator
961     state = DATAPULL_J;
962 }
963
964 else {
965     state = STANDBY; // Or wait until user
966     chooses
967 }
968
969 break;
970
971 case DATAPULL_S:
972
973     digitalWrite(BLUE, HIGH);
974         // LED indicator
975
976     if (redo_flag == true || estop_user == true)
977         state = STOP;
978     }
979
980     // if all data has been pulled from Pi,
981     process and find errors (if exist)
982     else if (dp_checkpoint == false &&
983     datapull_flag == true) {
984
985         // Process variables for valid TICKET and
986         check for errors
987         duration = (3600 * hour_user) + (60 *
988         min_user) + sec_user; // entire duration in
989         msec for easy compare later
990
991 #ifdef TESTING
992     Serial.print("hour_user : ");
993     Serial.println(3600 * hour_user);
994     Serial.print("min_user : ");
995     Serial.println(60 * min_user);
996     Serial.print("sec_user : ");
997     Serial.println(sec_user);
998     Serial.print("Duration : ");
999     Serial.println(duration, 10);
1000 #endif
1001
1002     freq = Q_to_Freq(q_user, syr_radius_user);
1003         // Get period from Q
1004
1005 #ifdef TESTING
1006     Serial.print("Freq : ");
1007     Serial.println(freq);
1008
1009     freq = MicroCalib(freq);
1010         // Calibrate for
1011         microstepping + error check (TICKET)
1012
1013 #ifdef TESTING
1014     Serial.print("Freq: ");
1015     Serial.println(freq);
1016 #endif
1017
1018     if (duration == 0) {
1019         TICKET = false;
1020     }
1021     else if (TICKET == true) {
1022         validDur_Start(q_user, syr_capacity,
1023         duration); // final TICKET confirm duration +
1024         error
1025     }
1026
1027     // Go to STANDBY is user sel to go back
1028     if (motor_dir == 2) {
1029         poll_user = 0; // reset user
1030
1031         selection
1032             digitalWrite(BLUE, LOW);
1033             state = STANDBY;
1034         }
1035
1036         // Proceed with current data
1037         else {
1038             // ready to give feedback to Pi
1039             dp_checkpoint = true;
1040             state = DATAPULL_S;
1041         }
1042
1043         // if all pulled data is valid, Pi has valid
1044         TICKET, and is ready to run
1045         else if (TICKET == true && eventstart == true)
1046             { // Must lie within Q
1047                 and duration range
1048                 digitalWrite(BLUE, LOW);
1049                     // LED indicator
1050                 datapull_flag = false;
1051                 eventstart = false;
1052                 TICKET = false;
1053                 state = RUN_S;
1054             }
1055
1056             // Not all data has been pulled/ Pi not ready
1057             to RUN / Errors exist
1058             else {
1059                 state = DATAPULL_S;
1060             }
1061
1062             break;
1063
1064 case DATAPULL_J:
1065     digitalWrite(BLUE, HIGH);
1066         // LED indicator
1067     digitalWrite(WHITE, HIGH);
1068         // LED indicator
1069
1070     if (redo_flag == true || estop_user == true) {
1071         state = STOP;
1072     }
1073
1074     // if all data has been pulled from Pi,
1075     process and find errors (if exist)
1076     else if (dp_checkpoint == false &&
1077     datapull_flag == true) {
1078         freq = Q_to_Freq(q_user, syr_radius_user);
1079             // Get period from Q
1080         freq = MicroCalib(freq);
1081             // Calibrate
1082             microstepping & TICKET validation
1083
1084             // Go to STANDBY is user sel to go back
1085             if (motor_dir == 2) {
1086                 digitalWrite(BLUE, LOW);
1087                 state = STANDBY;
1088             }
1089
1090             // Proceed with current data
1091             else {
1092                 // ready to give feedback to Pi
1093                 dp_checkpoint = true;
1094                 state = DATAPULL_J;
1095             }
1096
1097             // if all pulled data is valid, Pi has valid
1098             TICKET, and is ready to run
1099             else if (TICKET == true && eventstart == true)
1100                 {
1101                     digitalWrite(BLUE, LOW);
1102                         // LED indicator
1103                 }
1104
1105             }
1106
1107             }
1108
1109             }
1110
1111             }
1112
1113             }
1114
1115             }
1116
1117             }
1118
1119             }
1120
1121             }
1122
1123             }
1124
1125             }
1126
1127             }
1128
1129             }
1130
1131             }
1132
1133             }
1134
1135             }
1136
1137             }
1138
1139             }
1140
1141             }
1142
1143             }
1144
1145             }
1146
1147             }
1148
1149             }
1150
1151             }
1152
1153             }
1154
1155             }
1156
1157             }
1158
1159             }
1160
1161             }
1162
1163             }
1164
1165             }
1166
1167             }
1168
1169             }
1170
1171             }
1172
1173             }
1174
1175             }
1176
1177             }
1178
1179             }
1180
1181             }
1182
1183             }
1184
1185             }
1186
1187             }
1188
1189             }
1190
1191             }
1192
1193             }
1194
1195             }
1196
1197             }
1198
1199             }
1200
1201             }
1202
1203             }
1204
1205             }
1206
1207             }
1208
1209             }
1210
1211             }
1212
1213             }
1214
1215             }
1216
1217             }
1218
1219             }
1220
1221             }
1222
1223             }
1224
1225             }
1226
1227             }
1228
1229             }
1230
1231             }
1232
1233             }
1234
1235             }
1236
1237             }
1238
1239             }
1240
1241             }
1242
1243             }
1244
1245             }
1246
1247             }
1248
1249             }
1250
1251             }
1252
1253             }
1254
1255             }
1256
1257             }
1258
1259             }
1260
1261             }
1262
1263             }
1264
1265             }
1266
1267             }
1268
1269             }
1270
1271             }
1272
1273             }
1274
1275             }
1276
1277             }
1278
1279             }
1280
1281             }
1282
1283             }
1284
1285             }
1286
1287             }
1288
1289             }
1290
1291             }
1292
1293             }
1294
1295             }
1296
1297             }
1298
1299             }
1300
1301             }
1302
1303             }
1304
1305             }
1306
1307             }
1308
1309             }
1310
1311             }
1312
1313             }
1314
1315             }
1316
1317             }
1318
1319             }
1320
1321             }
1322
1323             }
1324
1325             }
1326
1327             }
1328
1329             }
1330
1331             }
1332
1333             }
1334
1335             }
1336
1337             }
1338
1339             }
1340
1341             }
1342
1343             }
1344
1345             }
1346
1347             }
1348
1349             }
1350
1351             }
1352
1353             }
1354
1355             }
1356
1357             }
1358
1359             }
1360
1361             }
1362
1363             }
1364
1365             }
1366
1367             }
1368
1369             }
1370
1371             }
1372
1373             }
1374
1375             }
1376
1377             }
1378
1379             }
1380
1381             }
1382
1383             }
1384
1385             }
1386
1387             }
1388
1389             }
1390
1391             }
1392
1393             }
1394
1395             }
1396
1397             }
1398
1399             }
1400
1401             }
1402
1403             }
1404
1405             }
1406
1407             }
1408
1409             }
1410
1411             }
1412
1413             }
1414
1415             }
1416
1417             }
1418
1419             }
1420
1421             }
1422
1423             }
1424
1425             }
1426
1427             }
1428
1429             }
1430
1431             }
1432
1433             }
1434
1435             }
1436
1437             }
1438
1439             }
1440
1441             }
1442
1443             }
1444
1445             }
1446
1447             }
1448
1449             }
1450
1451             }
1452
1453             }
1454
1455             }
1456
1457             }
1458
1459             }
1460
1461             }
1462
1463             }
1464
1465             }
1466
1467             }
1468
1469             }
1470
1471             }
1472
1473             }
1474
1475             }
1476
1477             }
1478
1479             }
1480
1481             }
1482
1483             }
1484
1485             }
1486
1487             }
1488
1489             }
1490
1491             }
1492
1493             }
1494
1495             }
1496
1497             }
1498
1499             }
1500
1501             }
1502
1503             }
1504
1505             }
1506
1507             }
1508
1509             }
1510
1511             }
1512
1513             }
1514
1515             }
1516
1517             }
1518
1519             }
1520
1521             }
1522
1523             }
1524
1525             }
1526
1527             }
1528
1529             }
1530
1531             }
1532
1533             }
1534
1535             }
1536
1537             }
1538
1539             }
1540
1541             }
1542
1543             }
1544
1545             }
1546
1547             }
1548
1549             }
1550
1551             }
1552
1553             }
1554
1555             }
1556
1557             }
1558
1559             }
1560
1561             }
1562
1563             }
1564
1565             }
1566
1567             }
1568
1569             }
1570
1571             }
1572
1573             }
1574
1575             }
1576
1577             }
1578
1579             }
1580
1581             }
1582
1583             }
1584
1585             }
1586
1587             }
1588
1589             }
1590
1591             }
1592
1593             }
1594
1595             }
1596
1597             }
1598
1599             }
1600
1601             }
1602
1603             }
1604
1605             }
1606
1607             }
1608
1609             }
1610
1611             }
1612
1613             }
1614
1615             }
1616
1617             }
1618
1619             }
1620
1621             }
1622
1623             }
1624
1625             }
1626
1627             }
1628
1629             }
1630
1631             }
1632
1633             }
1634
1635             }
1636
1637             }
1638
1639             }
1640
1641             }
1642
1643             }
1644
1645             }
1646
1647             }
1648
1649             }
1650
1651             }
1652
1653             }
1654
1655             }
1656
1657             }
1658
1659             }
1660
1661             }
1662
1663             }
1664
1665             }
1666
1667             }
1668
1669             }
1670
1671             }
1672
1673             }
1674
1675             }
1676
1677             }
1678
1679             }
1680
1681             }
1682
1683             }
1684
1685             }
1686
1687             }
1688
1689             }
1690
1691             }
1692
1693             }
1694
1695             }
1696
1697             }
1698
1699             }
1700
1701             }
1702
1703             }
1704
1705             }
1706
1707             }
1708
1709             }
1710
1711             }
1712
1713             }
1714
1715             }
1716
1717             }
1718
1719             }
1720
1721             }
1722
1723             }
1724
1725             }
1726
1727             }
1728
1729             }
1730
1731             }
1732
1733             }
1734
1735             }
1736
1737             }
1738
1739             }
1740
1741             }
1742
1743             }
1744
1745             }
1746
1747             }
1748
1749             }
1750
1751             }
1752
1753             }
1754
1755             }
1756
1757             }
1758
1759             }
1760
1761             }
1762
1763             }
1764
1765             }
1766
1767             }
1768
1769             }
1770
1771             }
1772
1773             }
1774
1775             }
1776
1777             }
1778
1779             }
1780
1781             }
1782
1783             }
1784
1785             }
1786
1787             }
1788
1789             }
1790
1791             }
1792
1793             }
1794
1795             }
1796
1797             }
1798
1799             }
1800
1801             }
1802
1803             }
1804
1805             }
1806
1807             }
1808
1809             }
1810
1811             }
1812
1813             }
1814
1815             }
1816
1817             }
1818
1819             }
1820
1821             }
1822
1823             }
1824
1825             }
1826
1827             }
1828
1829             }
1830
1831             }
1832
1833             }
1834
1835             }
1836
1837             }
1838
1839             }
1840
1841             }
1842
1843             }
1844
1845             }
1846
1847             }
1848
1849             }
1850
1851             }
1852
1853             }
1854
1855             }
1856
1857             }
1858
1859             }
1860
1861             }
1862
1863             }
1864
1865             }
1866
1867             }
1868
1869             }
1870
1871             }
1872
1873             }
1874
1875             }
1876
1877             }
1878
1879             }
1880
1881             }
1882
1883             }
1884
1885             }
1886
1887             }
1888
1889             }
1890
1891             }
1892
1893             }
1894
1895             }
1896
1897             }
1898
1899             }
1900
1901             }
1902
1903             }
1904
1905             }
1906
1907             }
1908
1909             }
1910
1911             }
1912
1913             }
1914
1915             }
1916
1917             }
1918
1919             }
1920
1921             }
1922
1923             }
1924
1925             }
1926
1927             }
1928
1929             }
1930
1931             }
1932
1933             }
1934
1935             }
1936
1937             }
1938
1939             }
1940
1941             }
1942
1943             }
1944
1945             }
1946
1947             }
1948
1949             }
1950
1951             }
1952
1953             }
1954
1955             }
1956
1957             }
1958
1959             }
1960
1961             }
1962
1963             }
1964
1965             }
1966
1967             }
1968
1969             }
1970
1971             }
1972
1973             }
1974
1975             }
1976
1977             }
1978
1979             }
1980
1981             }
1982
1983             }
1984
1985             }
1986
1987             }
1988
1989             }
1990
1991             }
1992
1993             }
1994
1995             }
1996
1997             }
1998
1999             }
1999
2000             }
2000
2001             }
2001
2002             }
2002
2003             }
2003
2004             }
2004
2005             }
2005
2006             }
2006
2007             }
2007
2008             }
2008
2009             }
2009
2010             }
2010
2011             }
2011
2012             }
2012
2013             }
2013
2014             }
2014
2015             }
2015
2016             }
2016
2017             }
2017
2018             }
2018
2019             }
2019
2020             }
2020
2021             }
2021
2022             }
2022
2023             }
2023
2024             }
2024
2025             }
2025
2026             }
2026
2027             }
2027
2028             }
2028
2029             }
2029
2030             }
2030
2031             }
2031
2032             }
2032
2033             }
2033
2034             }
2034
2035             }
2035
2036             }
2036
2037             }
2037
2038             }
2038
2039             }
2039
2040             }
2040
2041             }
2041
2042             }
2042
2043             }
2043
2044             }
2044
2045             }
2045
2046             }
2046
2047             }
2047
2048             }
2048
2049             }
2049
2050             }
2050
2051             }
2051
2052             }
2052
2053             }
2053
2054             }
2054
2055             }
2055
2056             }
2056
2057             }
2057
2058             }
2058
2059             }
2059
2060             }
2060
2061             }
2061
2062             }
2062
2063             }
2063
2064             }
2064
2065             }
2065
2066             }
2066
2067             }
2067
2068             }
2068
2069             }
2069
2070             }
2070
2071             }
2071
2072             }
2072
2073             }
2073
2074             }
2074
2075             }
2075
2076             }
2076
2077             }
2077
2078             }
2078
2079             }
2079
2080             }
2080
2081             }
2081
2082             }
2082
2083             }
2083
2084             }
2084
2085             }
2085
2086             }
2086
2087             }
2087
2088             }
2088
2089             }
2089
2090             }
2090
2091             }
2091
2092             }
2092
2093             }
2093
2094             }
2094
2095             }
2095
2096             }
2096
2097             }
2097
2098             }
2098
2099             }
2099
2100             }
2100
2101             }
2101
2102             }
2102
2103             }
2103
2104             }
2104
2105             }
2105
2106             }
2106
2107             }
2107
2108             }
2108
2109             }
2109
2110             }
2110
2111             }
2111
2112             }
2112
2113             }
2113
2114             }
2114
2115             }
2115
2116             }
2116
2117             }
2117
2118             }
2118
2119             }
2119
2120             }
2120
2121             }
2121
2122             }
2122
2123             }
2123
2124             }
2124
2125             }
2125
2126             }
2126
2127             }
2127
2128             }
2128
2129             }
2129
2130             }
2130
2131             }
2131
2132             }
2132
2133             }
2133
2134             }
2134
2135             }
2135
2136             }
2136
2137             }
2137
2138             }
2138
2139             }
2139
2140             }
2140
2141             }
2141
2142             }
2142
2143             }
2143
2144             }
2144
2145             }
2145
2146             }
2146
2147             }
2147
2148             }
2148
2149             }
2149
2150             }
2150
2151             }
2151
2152             }
2152
2153             }
2153
2154             }
2154
2155             }
2155
2156             }
2156
2157             }
2157
2158             }
2158
2159             }
2159
2160             }
2160
2161             }
2161
2162             }
2162
2163             }
2163
2164             }
2164
2165             }
2165
2166             }
2166
2167             }
2167
2168             }
2168
2169             }
2169
2170             }
2170
2171             }
2171
2172             }
2172
2173             }
2173
2174             }
2174
2175             }
2175
2176             }
2176
2177             }
2177
2178             }
2178
2179             }
2179
2180             }
2180
2181             }
2181
2182             }
2182
2183             }
2183
2184             }
2184
2185             }
2185
2186             }
2186
2187             }
2187
2188             }
2188
2189             }
2189
2190             }
2190
2191             }
2191
2192             }
2192
2193             }
2193
2194             }
2194
2195             }
2195
2196             }
2196
2197             }
2197
2198             }
2198
2199             }
2199
2200             }
2200
2201             }
2201
2202             }
2202
2203             }
2203
2204             }
2204
2205             }
2205
2206             }
2206
2207             }
2207
2208             }
2208
2209             }
2209
2210             }
2210
2211             }
2211
2212             }
2212
2213             }
2213
2214             }
2214
2215             }
2215
2216             }
2216
2217             }
2217
2218             }
2218
2219             }
2219
2220             }
2220
2221             }
2221
2222             }
2222
2223             }
2223
2224             }
2224
2225             }
2225
2226             }
2226
2227             }
2227
2228             }
2228
2229             }
2229
2230             }
2230
2231             }
2231
2232             }
2232
2233             }
2233
2234             }
2234
2235             }
2235
2236             }
2236
2237             }
2237
2238             }
2238
2239             }
2239
2240             }
2240
2241             }
2241
2242             }
2242
2243             }
2243
2244             }
2244
2245             }
2245
2246             }
2246
2247             }
2247
2248             }
2248
2249             }
2249
2250             }
2250
2251             }
2251
2252             }
2252
2253             }
2253
2254             }
2254
2255             }
2255
2256             }
2256
2257             }
2257
2258             }
2258
2259             }
2259
2260             }
2260
2261             }
2261
2262             }
2262
2263             }
2263
2264             }
2264
2265             }
2265
2266             }
2266
2267             }
2267
2268             }
2268
2269             }
2269
2270             }
2270
2271             }
2271
2272             }
2272
2273             }
2273
2274             }
2274
2275             }
2275
2276             }
2276
2277             }
2277
2278             }
2278
2279             }
2279
2280             }
2280
2281             }
2281
2282             }
2282
2283             }
2283
2284             }
2284
2285             }
2285
2286             }
2286
2287             }
2287
2288             }
2288
2289             }
2289
2290             }
2290
2291             }
2291
2292             }
2292
2293             }
2293
2294             }
2294
2295             }
2295
2296             }
2296
2297             }
2297
2298             }
2298
2299             }
2299
2300             }
2300
2301             }
2301
2302             }
2302
2303             }
2303
2304             }
2304
2305             }
2305
2306             }
2306
2307             }
2307
2308             }
2308
2309             }
2309
2310             }
2310
2311             }
2311
2312             }
2312
2313             }
2313
2314             }
2314
2315             }
2315
2316             }
2316
2317             }
2317
2318             }
2318
2319             }
2319
2320             }
2320
2321             }
2321
2322             }
2322
2323             }
2323
2324             }
2324
2325             }
2325
2326             }
2326
2327             }
2327
2328             }
2328
2329             }
2329
2330             }
2330
2331             }
2331
2332             }
2332
2333             }
2333
2334             }
2334
2335             }
2335
2336             }
2336
2337             }
2337
2338             }
2338
2339             }
2339
2340             }
2340
2341             }
2341
2342             }
2342
2343             }
2343
2344             }
2344
2345             }
2345
2346             }
2346
2347             }
2347
2348             }
2348
2349             }
2349
2350             }
2350
2351             }
2351
2352             }
2352
2353             }
2353
2354             }
2354
2355             }
2355
2356             }
2356
2357             }
2357
2358             }
2358
2359             }
2359
2360             }
2360
2361             }
2361
2362             }
2362
2363             }
2363
2364             }
2364
2365             }
2365
2366             }
2366
2367             }
2367
2368             }
2368
2369             }
2369
2370             }
2370
2371             }
2371
2372             }
2372
2373             }
2373
2374             }
2374
2375             }
2375
2376             }
2376
2377             }
2377
2378             }
2378
2379             }
2379
2380             }
2380
2381             }
2381
2382             }
2382
2383             }
2383
2384             }
2384
2385             }
2385
2386             }
2386
2387             }
2387
2388             }
2388
2389             }
2389
2390             }
2390
2391             }
2391
2392             }
2392
2393             }
2393
2394             }
2394
2395             }
2395
2396             }
2396
2397             }
2397
2398             }
2398
2399             }
2399
2400             }
2400
2401             }
2401
2402             }
2402
2403             }
2403
2404             }
2404
2405             }
2405
2406             }
2406
2407             }
2407
2408             }
2408
2409             }
2409
2410             }
2410
2411             }
2411
2412             }
2412
2413             }
2413
2414             }
2414
2415             }
2415
2416             }
2416
2417             }
2417
2418             }
2418
2419             }
2419
2420             }
2420
2421             }
2421
2422             }
2422
2423             }
2423
2424             }
2424
2425             }
2425
2426             }
2426
2427             }
2427
2428             }
2428
2429             }
2429
2430             }
2430
2431             }
2431
2432             }
2432
2433             }
2433
2434             }
2434
2435             }
2435
2436             }
2436
2437             }
2437
2438             }
2438
2439             }
2439
2440             }
2440
2441             }
2441
2442             }
2442
2443             }
2443
2444             }
2444
2445             }
2445
2446             }
2446
```

```

1079     digitalWrite(WHITE, LOW);           // LED indicator
1080     datapull_flag = false;
1081     eventstart = false;
1082     TICKET = false;
1083     state = RUN_J;
1084 }
1085
1086 // Not all data has been pulled/ Pi not ready
1087 // to RUN / Errors exist
1088 else {
1089     state = DATAPULL_J;
1090 }
1091
1092 break;
1093
1094 case RUN_J:
1095
1096 // Device order delay
1097 if (i2c_address == ARD_ADD_1) {
1098     delay(1800);
1099 }
1100 else if (i2c_address == ARD_ADD_2) {
1101     delay(900);
1102 }
1103
1104 digitalWrite(GREEN, HIGH);          // LED indicator
1105 digitalWrite(WHITE, HIGH);          // LED indicator
1106
1107
1108 if (estop_user == true) {
1109     digitalWrite(GREEN, LOW);         // LED indicator
1110     digitalWrite(WHITE, LOW);         // LED indicator
1111     state = STOP;
1112 }
1113
1114 else if (RUNNING == false) {
1115
1116 #ifdef STSPIN220
1117     MtrCtrl_init_220(step_config);   // setup motor controller
1118 #else
1119     MtrCtrl_init_820(step_config);   // setup motor controller
1120 #endif
1121
1122     if (motor_dir == 1) {
1123         digitalWrite(DIR, LOW);
1124     }
1125
1126     else {
1127         digitalWrite(DIR, HIGH);
1128     }
1129
1130
1131 #ifdef TESTING
1132     Serial.print("freq");
1133     Serial.println(freq);
1134 #endif
1135
1136     Timer1.initialize((1 / freq) * 1000000);    // Init PWM freq[microsec] (mult to
1137     // conv to usec)
1138     Timer1.pwm(STEP, DC);                      // Start motor
1139     RUNNING = true;
1140     state = RUN_J;
1141 }
1142
1143 else {
1144     state = RUN_J;
1145 }
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3259
3260
3261
3262
3263
3264
```

```

1210
1211     case STOP:
1212
1213         digitalWrite(STBY, LOW);
1214             // Place motor controller in standby
1215         Timer1.disablePwm(STEP);
1216             // turn off pwm
1217
1218         digitalWrite(GREEN, LOW);
1219             // LED indicator
1220         digitalWrite(WHITE, LOW);
1221             // LED indicator
1222         digitalWrite(YELLOW, LOW);
1223             // LED indicator
1224         digitalWrite(BLUE, LOW);
1225
1226         q_user = 0;
1227         syr_radius_user = 0;
1228         freq = 0;
1229         duration = 0;
1230         hour_user = 0;
1231         min_user = 0;
1232         sec_user = 0;
1233         motor_dir = 0;
1234         step_config = 0;
1235
1236         estop_user = false;
1237         dp_checkpoint = false;
1238         eventstart = false;
1239         RUNNING = false;
1240         TICKET = false;
1241         FINISH = false;
1242         datapull_flag = false;
1243         time_flag = false;
1244         redo_flag = false;
1245
1246         poll_user = 0;
1247         error_code = NO_ERR;
1248         recData = 0;
1249         strRX[200] = {0};
1250         dur_mark = 0;
1251
1252         state = STANDBY;
1253         break;
1254
1255     default:
1256         state = STOP;
1257         break;
1258
1259     }
1260
1261 void recvEvent(int numBytes) {
1262
1263     unsigned long temp;
1264
1265     recData = Wire.read();
1266
1267     // Ignore data unless recv BLOCK_CODE
1268     if (block_recv == 0) {
1269         //Ignore data is not a block (its rollcall or i2cdetect instead)
1270         if (recData == BLOCK_CODE) {
1271             //block[0] = recData;
1272             block_recv++;
1273         }
1274
1275     // Recv first data block
1276     else if (block_recv == 1) {
1277         block[1] = recData;
1278         block_recv++;
1279
1280
1281     #ifdef TESTING
1282         Serial.print("block[1] : ");
1283         Serial.println(block[1]);
1284     #endif
1285     }
1286
1287     // Recv second data block
1288     else if (block_recv == 2) {
1289         block[2] = recData;
1290         block_recv++;
1291
1292     #ifdef TESTING
1293         Serial.print("block[2] : ");
1294         Serial.println(block[2]);
1295     #endif
1296     }
1297
1298     // Recv third data block
1299     else if (block_recv == 3) {
1300         block[3] = recData;
1301         block_recv++;
1302
1303     #ifdef TESTING
1304         Serial.print("block[3] : ");
1305         Serial.println(block[3]);
1306     #endif
1307     }
1308
1309     // Recv fourth data block
1310     else if (block_recv == 4) {
1311         block[4] = recData;
1312         block_recv++;
1313
1314     #ifdef TESTING
1315         Serial.print("block[4] : ");
1316         Serial.println(block[4]);
1317     #endif
1318     }
1319
1320     // Recv fifth data block
1321     else if (block_recv == 5) {
1322         block[5] = recData;
1323         block_recv++;
1324
1325     #ifdef TESTING
1326         Serial.print("block[5] : ");
1327         Serial.println(block[5]);
1328     #endif
1329     }
1330
1331
1332
1333     // If entire block has been recv, process data
1334     if (block_recv == 6) {
1335
1336         //if second factor is non-zero, data uses fac2,
1337         rem2
1338         if (block[4] != 0) {
1339             temp = 256 * (256 * block[4] + block[5]) +
1340             block[3];
1341
1342             // Only one factor of 256
1343             else if (block[2] != 0) {
1344                 temp = 256 * block[2] + block[3];
1345             }
1346
1347             // Recv data is below 256
1348             else {
1349                 temp = block[3];
1350             }
1351
1352     #ifdef TESTING
1353         Serial.print("temp : ");
1354         Serial.println(temp);
1355     #endif
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
333
```

```

1356 //reset index
1357 block_recv = 0;
1358
1359
1360 // Read Dict Codes to store in right variable
1361 switch (block[1]) {
1362
1363     case POLLDATA:
1364         poll_user = temp;
1365         break;
1366
1367     case Q_DATA:
1368         q_user = temp;
1369         q_user /= 1000000000;
1370         break;
1371
1372     case R_DATA:
1373         syr_radius_user = temp;
1374         syr_radius_user /= 1000;
1375         break;
1376
1377     case CAP_DATA:
1378         syr_capacity = temp;
1379         syr_capacity /= 1000;
1380         break;
1381
1382     case DUR_DATA:
1383         if (dur_mark == 0) {
1384             hour_user = temp;
1385             dur_mark++;
1386         }
1387         else if (dur_mark == 1) {
1388             min_user = temp;
1389             dur_mark++;
1390         }
1391         else if (dur_mark == 2) {
1392             sec_user = temp;
1393             dur_mark = 0;
1394         }
1395     }
1396
1397     break;
1398
1399     case DIR_DATA:
1400         motor_dir = temp;
1401         datapull_flag = true;
1402
1403 #ifdef TESTING
1404     Serial.println(poll_user);
1405     Serial.println(q_user, 10);
1406     Serial.println(syr_radius_user, 8);
1407     Serial.println(syr_capacity);
1408     Serial.println(hour_user);
1409     Serial.println(min_user);
1410     Serial.println(sec_user);
1411     Serial.println(motor_dir);
1412 #endif
1413
1414     break;
1415
1416     case EVENT_DATA:
1417         eventstart = true;
1418         break;
1419
1420     case REDO_DATA:
1421         redo_flag = temp;
1422         Serial.println("STOPPED");
1423         state = STOP;
1424         break;
1425
1426     case ESTOP_DATA:
1427         estop_user = true;
1428         break;
1429
1430     default:
1431         error_code = DICT_ERR;
1432         break;
1433     }
1434 }
1435
1436 }
1437
1438 */
1439 /* ~~~~~ reqEvent()
1440 Purpose : Write data to main at its request
1441 Input : I2C request
1442 Output : TX corresponding error code
1443 */
1444 void reqEvent() {
1445
1446     // Tell Pi RUN_S has finished
1447     if (FINISH == true) {
1448         Wire.write(FINISH_CODE);
1449         estop_user = true;
1450     }
1451
1452
1453     // If in RUN_S but enough time hasn't elapsed,
1454     // send time_code for next TX
1455     else if (state == RUN_S && time_flag == false) {
1456         Wire.write(TIME_CODE);
1457         time_flag = true;
1458     }
1459
1460
1461     // Once Pi has recv time_code, send the time
1462     // metrics
1463     else if (state == RUN_S && time_flag == true) {
1464
1465         // convert time_end to sec from ms
1466         unsigned long time_end = time_stamp_end / 1000;
1467
1468         #ifdef TESTING
1469             Serial.print("time_end : ");
1470             Serial.println(time_end);
1471         #endif
1472
1473         if (dur_mark == 0) {
1474             hour_user = int(time_end / 3600);
1475             Wire.write(hour_user);
1476             dur_mark++;
1477
1478         #ifdef TESTING
1479             Serial.print("hour_user : ");
1480             Serial.println(hour_user);
1481         #endif
1482
1483     }
1484
1485         else if (dur_mark == 1) {
1486             min_user = int((time_end - (3600 * hour_user)) / 60);
1487             Wire.write(min_user);
1488             dur_mark++;
1489
1490         #ifdef TESTING
1491             Serial.print("min_user : ");
1492             Serial.println(min_user);
1493         #endif
1494
1495     }
1496
1497         else if (dur_mark == 2) {
1498             sec_user = int( time_end - (3600 * hour_user) - (60 * min_user));
1499             Wire.write(sec_user);
1500             dur_mark = 0;
1501             time_flag = false;
1502
1503         #ifdef TESTING
1504             Serial.print("sec_user : ");
1505             Serial.println(sec_user);
1506         #endif
1507
1508     }
1509
1510     }
1511
1512     }
1513
1514 }
```

```

1505     Serial.print("sec_user : ");
1506     Serial.println(sec_user);
1507 #endif
1508
1509 }
1510 }
1511
1512
1513
1514 // Tell Pi to wait if ard is still calculating/
1515 // validating TICKET
1516 else if (dp_checkpoint == false && state != RUN_S)
1517 {
1518     Wire.write(WAIT_CODE);
1519     // Ard hasnt had time to validate TICKET
1520     yet
1521 }
1522
1523
1524 // Return error (or no error) in DATAPULL
1525 else {
1526     if (error_code != NO_ERR) {
1527         // If errors exist, prepare for another
1528         datapull
1529         dp_checkpoint = false;
1530         datapull_flag = false;
1531     }
1532     Wire.write(error_code);
1533     // TX error
1534 #ifdef TESTING
1535     Serial.print("Error code : ");
1536     Serial.println(error_code, DEC);
1537 #endif
1538
1539     error_code = NO_ERR;
1540 }
1541
1542 }
1543
1544 #endif

```

Listing 1: Motor Controller Code

B. Hub Controller

```

1 # Connor Wilson
2 # Winter 2022
3 # SMSS Hub Controller V1.0
4
5
6 from time import sleep          # python
7     delay
8 from smbus2 import SMBus         # state
9 from transitions import Machine
10    machine
11 import curses                  # GUI
12 from curses import wrapper
13 from curses.textpad import Textbox
14 import random
15
16 class HubMachine(object):
17
18     states = ['START', 'DATAPULL', 'RUN', 'RESET']
19
20     def __init__(self, name):
21         self.name = name
22
23         # States and their corresponding triggers/
24         # transitions
25         self.machine = Machine(model= self, states =
26             HubMachine.states, initial = 'START')
27         self.machine.add_transition(trigger = 'CONFIRM', source = 'START', dest = 'DATAPULL')
28         self.machine.add_transition(trigger = 'EVENTSTART', source = 'DATAPULL', dest = 'RUN')
29         self.machine.add_transition(trigger = 'DONE', source = 'RUN', dest = 'RESET')
30         self.machine.add_transition(trigger = 'RESTART', source = 'RESET', dest = 'START')
31
32         # States for GUI
33         global FRAME, POLL, DATA, RUN, ERROR, USER_CONFIRM,
34             USER_ESTOP, reqGUI
35         FRAME = 0x0
36         POLL = 0x1
37         DATA = 0x2
38         ERROR = 0x3
39         RUN = 0x4
40         USER_CONFIRM = 0x6
41         USER_ESTOP = 0x7
42         reqGUI = 0x0          # Working var for GUI states
43
44         # Index for device spacing in the GUI
45         global column
46         column = 0
47
48         # Data between user-curses-arduino
49         global rollcall, poll_user, q_user, radius_user,
50             cap_user
51         global hour_user, min_user, sec_user, dir_user,
52             back_flag
53         global estop_flag, ard_duration
54         rollcall = [0, 0, 0]
55         poll_user = [0, 0, 0]
56         q_user = [0, 0, 0]
57         radius_user = [0, 0, 0]
58         cap_user = [0, 0, 0]
59         hour_user = [0, 0, 0]
60         min_user = [0, 0, 0]
61         sec_user = [0, 0, 0]
62         dir_user = [0, 0, 0]
63
64         # 1 - REV, 0 - FWD
65         back_flag = [False, False, False]
66             # User sel to go back
67         estop_flag = [False, False, False]
68             # User sel for device estop

```

```

61     ard_duration = [[0,0,0], [0,0,0], [0,0,0]]           # run time from ard
62
63     # Arduino's Requested (Sub) codes
64     global LARGE_ERR, SMALL_ERR, DUR_ERR, LD_ERR, SD_ERR
65         , DICT_ERR, NO_ERR
66     global WAIT_CODE, FINISH_CODE, TIME_CODE
67     global user_code
68
69     LARGE_ERR = 0x3                                # User req too
70         large (Q) for hardware constraints
71     SMALL_ERR = 0x4                                # User req too
72         small (Q) for hardware constraints
73     DUR_ERR = 0x5                                # User req too
74         long (time) for hardware constraints
75     LD_ERR = 0x6                                # User too large
76         and long
77     SD_ERR = 0x7                                # User too small
78         and long
79     DICT_ERR = 0x8                                # Wrong
80         dictating codes
81     NO_ERR = 0x10                                 # No error to
82         report
83     WAIT_CODE = 0x15                             # Ard needs time
84         to pull data and validate
85     FINISH_CODE= 0x16                            # Ard's duration
86         has been reached and finished RUN_S
87     TIME_CODE = 0x17                            # Ard is not
88         done with RUN_S and will return time(h,m,s) if
89         prompted
90
91     user_code = 0                                # working
92         variable
93
94     global estop_index
95     estop_index = 0
96
97     def main():
98
99         # Working Variables that pass through main and
100        # GUI
101        global rollcall, poll_user, q_user, radius_user,
102            cap_user
103        global hour_user, min_user, sec_user, dir_user,
104            back_flag
105        global estop_flag, ard_duration, reqGUI,
106            user_code
107
108        # I2C Addresses
109        PI_ADD = 0x01
110        ARD_ADD_1 = 0x14
111        ARD_ADD_2 = 0x28
112        ARD_ADD_3 = 0x42
113        address = [ARD_ADD_1, ARD_ADD_2, ARD_ADD_3]
114
115        # Pi's Dictating (main) codes
116        POLLDATA = 1                                # Ard will
117        recv[OFF, JOG, START]
118        Q_DATA = 2                                # Ard will
119        recv[q_user]
120        R_DATA = 3                                # Ard will
121        recv[syr_radius_user]
122        CAP_DATA = 4                                # Ard will
123        recv[syr_capacity]
124        DUR_DATA = 5                                # Ard will
125        recv[hour_user, min_user, sec_user]
126        DIR_DATA = 6                                # Ard will
127        recv[DIRECTION]
128        EVENT_DATA = 7                            # Ard will
129        proceed RUN
130        ESTOP_DATA = 8                            # Ard will
131        proceed to STOP
132        REDO_DATA = 9                            # Ard will
133        revert back to STANDBY from DATAPULL
134        BLOCK_DATA = 10                           # Ard will
135        recv a block (4) of data

```



```

349             #reset var
350             time_saver[i] = 0
351
352         elif time_saver[i] > 300 and
353             poll_user[i] == 1:
354             # Print time data
355             reqGUI = RUN
356             column = COL_CONST * i +
357             COL_OFFSET
358             curs(stdscr)
359             time_saver[i] = 0
360
361             time_saver[i] += 1
362
363         #Proceed if all devices are finished
364         Hub.DONE()
365
366
367     #reset variables
368     elif Hub.state == 'RESET' :
369         rollcall = [0, 0, 0]
370         poll_user = [0, 0, 0]
371         q_user = [0,0,0]
372         radius_user = [0,0,0]
373         cap_user = [0,0,0]
374         hour_user = [0,0,0]
375         min_user = [0,0,0]
376         sec_user = [0,0,0]
377         dir_user = [0,0,0]
378         back_flag = [False, False, False]
379         estop_flag = [False, False, False]
380         ard_duration = [[0,0,0], [0,0,0],
381         [0,0,0]]
382         user_code = 0
383         global estop_index
384         estop_index = 0
385
386         Hub.RESTART()
387
388
389 # Error_2Str()
390 # SIGNATURE : int -> String
391 # PURPOSE : Takes in Error code and returns a string
392 #           that aids the user in correcting their
393 #           data
394 #           requests
395 def Error_2Str(error):
396
397     global LARGE_ERR, SMALL_ERR, DUR_ERR, LD_ERR,
398     SD_ERR, DICT_ERR, NO_ERR
399
400     if error == LARGE_ERR:
401         return "Error: Flow rate too large for
402 hardware"
403     elif error == SMALL_ERR:
404         return "Error : Flow rate too small for
405 hardware"
406     elif error == DUR_ERR:
407         return "Error : Run time will ruin machine"
408     elif error == LD_ERR:
409         return "Error : Too large flow rate + time
410 overflow"
411     elif error == SD_ERR:
412         return "Error : Too small of Q + time
413 overflow"
414     elif error == DICT_ERR:
415         return "Error : Ard got lost...GULP"
416     else:
417         return f"Your error code is unique... : {error}"
418
419 # transmit block()
420
421 # SIGNATURE : int, int, int ->
422 # PURPOSE : Transmits a dictating code +  data to an
423 #           address
424 #
425 def transmit_block(addr, dict_code, int_data):
426
427     # Code for ard to prep for a block
428     BLOCK_DATA = 10
429
430     # [block, dict_code, factor1, remainder1,
431     # factor2, remainder2]
432     block = [BLOCK_DATA, dict_code, 0, int_data, 0,
433     0]
434
435     # (can only handle 256 ^3)...increase for larger
436     # data set
437     i = 2
438
439     while int_data > 255:
440         block[i] = int(min(int_data / 256, 256))
441         # store factor of 256
442         block[i+1] = int(int_data % 256)
443         # store remainder of 256
444         int_data /= 256
445         # break down data
446         i += 2
447
448     #transmit data
449     with SMBus(1) as bus:
450         for i in range(6):
451             bus.write_byte(addr, block[i])
452             sleep(0.15)
453
454     # deviceRollcall()
455     # SIGNATURE : int_array -> int_array
456     # Purpose : Sends byte to each device using I2C and
457     #           Stores 1 in array for a sucessful
458     #           transmission (device active)
459     #           Stores 0 in array for unsuccessful
460     #           transmission (device not active)
461     def deviceRollcall(myaddress):
462
463         dev_status = [0, 0, 0]
464
465         with SMBus(1) as bus: # write to BUS
466
467             for i in range(3):
468
469                 # Try to contact each device
470                 try:
471                     bus.write_byte(myaddress[i], 0)
472                     dev_status[i] = 1
473                     #print(f"Device {i} on bus")
474                 except:
475                     #print(f"Device {i} not on bus")
476                     dev_status[i] = 0
477                     #pass
478
479             sleep(0.1)
480             return dev_status
481
482
483     # GUI
484     def curs(stdscr):
485
486         global FRAME, POLL, DATA, RUN, ERROR,
487         USER_CONFIRM, USER_ESTOP
488         global reqGUI
489         global rollcall, poll_user, q_user, radius_user,
490             cap_user
491         global hour_user, min_user, sec_user, dir_user
492         global estop_index, user_code
493
494
495         # Color combinations (ID, foreground, background
496         )

```

```

481 curses.init_pair(1, curses.COLOR_RED, curses.
482 COLOR_WHITE) 551
483 RED_AND_WHITE = curses.color_pair(1) 552
484
485 curses.init_pair(2, curses.COLOR_BLACK, curses.
486 COLOR_WHITE) 553
487 BLACK_AND_WHITE = curses.color_pair(2) 554
488
489 curses.init_pair(3, curses.COLOR_BLACK, curses.
490 COLOR_GREEN) 555
491 BLACK_AND_GREEN = curses.color_pair(3) 556
492
493 curses.init_pair(4, curses.COLOR_YELLOW, curses.
494 COLOR_CYAN) 557
495 YELLOW_AND_CYAN = curses.color_pair(4) 558
496
497 curses.init_pair(5, curses.COLOR_WHITE, curses.
498 COLOR_MAGENTA) 559
499 WHITE_AND_MAGENTA = curses.color_pair(5) 560
500
501 curses.init_pair(6, curses.COLOR_BLACK, curses.
502 COLOR_RED) 561
503 BLACK_AND_RED = curses.color_pair(6) 562
504
505 # Display background frame
506 if reqGUI == FRAME:
507
508     stdscr.clear() 563
509     stdscr.attron(YELLOW_AND_CYAN) 564
510     stdscr.resize(28,100) 565
511     stdscr.move(0,0) 566
512
513     # Fill square main in
514     for y in range(1,28):
515         for x in range(1, 100-1):
516             stdscr.addch(" ")
517             stdscr.move(y,1)
518     stdscr.border() 567
519
520     #Title block
521     stdscr.move(2,30)
522     stdscr.addstr("          Smart Motor Syringe
523 Pump      ", BLACK_AND_WHITE) 568
524
525     stdscr.attroff(YELLOW_AND_CYAN)
526     stdscr.attron(BLACK_AND_WHITE)
527
528     #Device one block
529     stdscr.move(5,10)
530     stdscr.addstr("Device 1 :")
531     stdscr.move(5,30)
532
533     for y in range(5,11):
534         for x in range(30, 90-1):
535             stdscr.addch(" ")
536             stdscr.move(y,30)
537
538     # Device two block
539     stdscr.move(12,10)
540     stdscr.addstr("Device 2 :")
541     stdscr.move(12,30)
542
543     for y in range(12,18):
544         for x in range(30, 90-1):
545             stdscr.addch(" ")
546             stdscr.move(y,30)
547
548     # Device three block
549     stdscr.move(19,10)
550     stdscr.addstr("Device 3 :")
551     stdscr.move(19,30)
552
553     for y in range(19,25):
554
555         for x in range(30, 90-1):
556             stdscr.addch(" ")
557             stdscr.move(y,30)
558
559         # Poll user for device run modes
560         elif reqGUI == POLL:
561
562             global poll_user
563             #avoiding a global index here people...
564             #forgive me coding gods
565             index_gui = int((column-5)/7)
566
567             if rollcall[index_gui] > 0:
568                 # display window
569                 winPol11 = curses.newwin(5, 60, column,
570                                         30)
571
572                 curses.noecho()
573                 winPol11.nodelay(True)
574                 winPol11.clear()
575                 winPol11.attron(BLACK_AND_GREEN)
576                 winPol11.addstr(1, 5,f"Device {index_gui
577 + 1} Mode :    | OFF | JOG | RUN |")
578
579                 winPol11.refresh()
580
581                 # collect data
582                 q = 0
583                 key = ""
584
585                 while key != 'w':
586
587                     winPol11.attroff(BLACK_AND_WHITE)
588                     winPol11.attron(BLACK_AND_GREEN)
589                     winPol11.addstr(1, 5, f"Device {
590 index_gui + 1} Mode :    | OFF | JOG | RUN |")
591
592                     try:
593                         key = winPol11.getkey()
594                     except:
595                         key = None
596
597                     if key == 'a':
598                         if q > 0:
599                             q -= 1
600
601                     elif key == 'd':
602                         if q < 2:
603                             q += 1
604
605                     if q == 0:
606                         winPol11.attroff(BLACK_AND_GREEN)
607
608                         winPol11.attron(BLACK_AND_WHITE)
609                         winPol11.addstr(1, 26, " | OFF | ")
610
611                         poll_user[index_gui] = 0
612
613                         # disable the device
614
615                     elif q == 1:
616                         winPol11.attroff(BLACK_AND_GREEN)
617
618                         winPol11.attron(BLACK_AND_WHITE)
619                         winPol11.addstr(1, 32, " | JOG | ")
620
621                         poll_user[index_gui] = 1
622
623                         # jog the device

```

```

617         elif q == 2:
618             winPoll1.attroff(BLACK_AND_GREEN)
619         )
620             winPoll1.attroff(BLACK_AND_WHITE)
621             winPoll1.addstr(1, 38, "| RUN |")
622         )
623             poll_user[index_gui] = 2
624             # run the device
625
626             winPoll1.refresh()
627             sleep(0.5)
628
629 # Simple enter button
630 elif reqGUI == USER_CONFIRM:
631
632     # display window
633     winPoll4 = curses.newwin(1, 46, 25, 35)
634     curses.noecho()
635     # dont repeat user input
636     winPoll4.nodelay(True)
637     # dont wait user input
638     winPoll4.clear()
639     winPoll4.attron(curses.color_pair(random.randint(1,6))) #randomly change color
640     winPoll4.addstr(0, 18, "| CONFIRM? |")
641     winPoll4.refresh()
642
643     key = ""
644
645     while key != 'w':
646
647         try:
648             key = winPoll4.getkey()
649         except:
650             key = None
651
652         winPoll4.refresh()
653         sleep(0.5)
654
655 # Poll metrics for running in START
656 elif reqGUI == DATA:
657
658     #avoiding a global index here people...
659     forgive_me_coding_gods
660     index_gui = int((column-5)/7)
661
662     # User wants to disable the device (leave in standby - no update)
663     if poll_user[index_gui] == 0:
664         pass
665
666     # User selected JOG function
667     elif poll_user[index_gui] == 1:
668
669         winData1 = curses.newwin(5, 60,
670         column, 30)
671         winData1.attron(WHITE_AND_MAGENTA)
672
673
674         winData1.addstr(0, 0, "Volumetric
675 Flow Rate Q [nL/s] in [70n, 10u] : ")
676         winData1.refresh()
677         win_data_ret = curses.newwin(1, 10,
678         column, 76)
679         curses.echo()
680         box = Textbox(win_data_ret)
681         box.edit()
682         q_user[index_gui] = box.gather()
683         strip().replace("\n", "")
684
685         winData1.addstr(1, 0, "Syringe
686 Radius in [mm] : ")
687         win_data_ret.clear()
688         win_data_ret.mvwin(column + 1, 55)
689
690         winData1.refresh()
691         sleep(0.5)
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2
```

```

746     box = Textbox(win_data_ret)
747     box.edit()
748     q_user[index_gui] = box.gather().strip().replace("\n", "")
749
750     winData1.addstr(1, 0, "Syringe")
751     Radius_in [mm] : "
752         win_data_ret.clear()
753         win_data_ret.mvwin(column + 1, 55)
754         winData1.refresh()
755         box.edit()
756         radius_user[index_gui] = box.gather().strip().replace("\n", "")
757
758         winData1.addstr(2, 0, "Syringe")
759         Capacity_in [mL] : "
760         win_data_ret.clear()
761         win_data_ret.mvwin(column + 2, 58)
762         winData1.refresh()
763         box.edit()
764         cap_user[index_gui] = box.gather().strip().replace("\n", "")
765
766         winData1.addstr(3, 0, "Hour # : ")
767         win_data_ret.clear()
768         win_data_ret.mvwin(column + 3, 40)
769         winData1.refresh()
770         box.edit()
771         hour_user[index_gui] = box.gather().strip().replace("\n", "")
772
773         winData1.addstr(3, 16, "Minute # : ")
774
775         win_data_ret.clear()
776         win_data_ret.mvwin(column + 3, 58)
777         winData1.refresh()
778         box.edit()
779         min_user[index_gui] = box.gather().strip().replace("\n", "")
780
781         winData1.addstr(3, 35, "Second # : ")
782
783         win_data_ret.clear()
784         win_data_ret.mvwin(column + 3, 77)
785         winData1.refresh()
786         box.edit()
787         sec_user[index_gui] = box.gather().strip().replace("\n", "")
788
789         winData1.addstr(4, 0, "Direction ? | FWD | REV | REDO | ? ")
790         win_data_ret.clear()
791         winData1.refresh()
792
793             winData1.attroff(WHITE_AND_MAGENTA)
794             winData1.attron(BLACK_AND_GREEN)
795
796                 # collect data
797                 curses.noecho()
798                 winData1.nodelay(True)
799                 q = 0
800                 key = ""
801                 while key != 'w':
802                     try:
803                         key = winData1.getkey()
804                     except:
805                         key = None
806
807                         if key == 'a':
808                             if q > 0:
809                                 q -= 1
810
811                         elif key == 'd':
812                             if q < 2:
813                                 q += 1
814
815                         winData1.attroff(BLACK_AND_GREEN)
816                         winData1.attron(WHITE_AND_MAGENTA)
817                         winData1.addstr(4, 0, "Direction ? | FWD | REV | REDO | ? ")
818                         winData1.attroff(WHITE_AND_MAGENTA)
819                         winData1.attron(BLACK_AND_GREEN)
820
821                         if q == 0:
822                             winData1.addstr(4, 19, "| FWD |")
823                             dir_user[index_gui] = 1
824                             back_flag[index_gui] = False
825
826                         elif q == 1:
827                             winData1.addstr(4, 25, "| REV |")
828                             dir_user[index_gui] = 0
829                             back_flag[index_gui] = False
830
831                         elif q == 2:
832                             winData1.addstr(4, 31, "| REDO |")
833                             back_flag[index_gui] = True
834
835                         winData1.refresh()
836                         sleep(0.5)
837
838
839             # JOG-RUN metrics
840             elif reqGUI == RUN:
841
842                 # avoiding a global index here people...
843                 forgive_me_coding_gods
844                 index_gui = int((column-5)/7)
845
846                 # User wants to disable the device (leave in standby - no update)
847                 if poll_user[index_gui] == 0:
848                     pass
849
850
851                 # User sel JOG function
852                 elif poll_user[index_gui] == 1:
853
854                     winRun1 = curses.newwin(5, 60,
855                     column, 30)
856                     winRun1.attron(BLACK_AND_RED)
857
858                     winRun1.addstr(0, 0, f" Q [nL/s] : {q_user[index_gui]} Syringe Radius in [mm] : {radius_user[index_gui]}")
859                     #winRun1.addstr(4, 18, "| ESTOP |")
860                     winRun1.refresh()
861
862
863                 # User selected RUN function
864                 elif poll_user[index_gui] == 2:
865                     winRun1 = curses.newwin(5, 60,
866                     column, 30)
867                     winRun1.attron(BLACK_AND_RED)
868
869                     winRun1.addstr(0, 0, f" Q [nL/s] : {q_user[index_gui]} Syringe Radius in [mm] : {radius_user[index_gui]}")
870                     winRun1.addstr(1, 0, f" Requested - Hour : {hour_user[index_gui]} Minute : {min_user[index_gui]} Second : {sec_user[index_gui]}")
871                     winRun1.addstr(2, 0, f" Elapsed - Hour : {ard_duration[index_gui][0]} Minute : {ard_duration[index_gui][1]} Second : {ard_duration[index_gui][2]} ")
872                     #winRun1.addstr(4, 18, "| ESTOP |")
873                     winRun1.refresh()

```

```

868         #sleep(1)
869
870
871     # Simple ESTOP button
872     elif reqGUI == USER_ESTOP:
873
874         COL_CONST = 7
875         COL_OFFSET = 5
876
877         global estop_flag
878
879         # display window
880         winStop = curses.newwin(1, 60, 25, 30)
881         curses.noecho()
882         winStop.nodelay(True)
883         winStop.clear()
884         winStop.refresh()
885
886         key = ""
887         try:
888             key = winStop.getkey()
889         except:
890             key = None
891
892         if key == 'w':
893             if estop_index > 0:
894                 estop_index -= 1
895
896         elif key == 's':
897             if estop_index < 3:
898                 estop_index += 1
899
900
901         if estop_index < 3 and poll_user[estop_index] > 0:
902             winStop.mvwin(estop_index * COL_CONST + 9, 30)
903             winStop.attron(curses.color_pair(estop_index))
904             winStop.addstr(0, 18, "| ESTOP? |")
905             winStop.refresh()
906
907         elif estop_index == 3:
908             winStop.mvwin(25, 30)
909             winStop.attron(BLACK_AND_RED)
910             winStop.addstr(0, 18, "| ESTOP ALL? |")
911             winStop.refresh()
912
913
914         if key == 'd':
915
916             # stop indiv device
917             if estop_index != 3:
918                 estop_flag[estop_index] = True
919                 winStop.attron(BLACK_AND_GREEN)
920                 winStop.addstr(0, 18, "| STOP'D |")
921                 winStop.refresh()
922
923             # stop all devices
924             elif estop_index == 3:
925                 estop_flag[0] = True
926                 estop_flag[1] = True
927                 estop_flag[2] = True
928                 estop_index = 0
929
930
931
932     # Presents error string in device window
933     elif reqGUI == ERROR:
934         #avoiding a global index here people...
935         forgive me coding gods
936         index_gui = int((column-5)/7)
937
938         # display window
939         winPoll4 = curses.newwin(5, 60, column, 30)
940         curses.noecho()
941         winPoll4.nodelay(True)
942
943         winPoll4.clear()
944         winPoll4.attron(RED_AND_WHITE)
945         winPoll4.addstr(2, 10, Error_2Str(user_code))
946
947         winPoll4.refresh()
948
949         sleep(8)
950
951     else:
952         print("Requested GUI Error")
953
954
955     wrapper(curs)
956
957
958
959
960 if __name__ == "__main__":
961     main()

```

Listing 2: Hub Controller Code

APPENDIX D
ADDITIONAL DOCUMENTS

A. Business Model Graphic

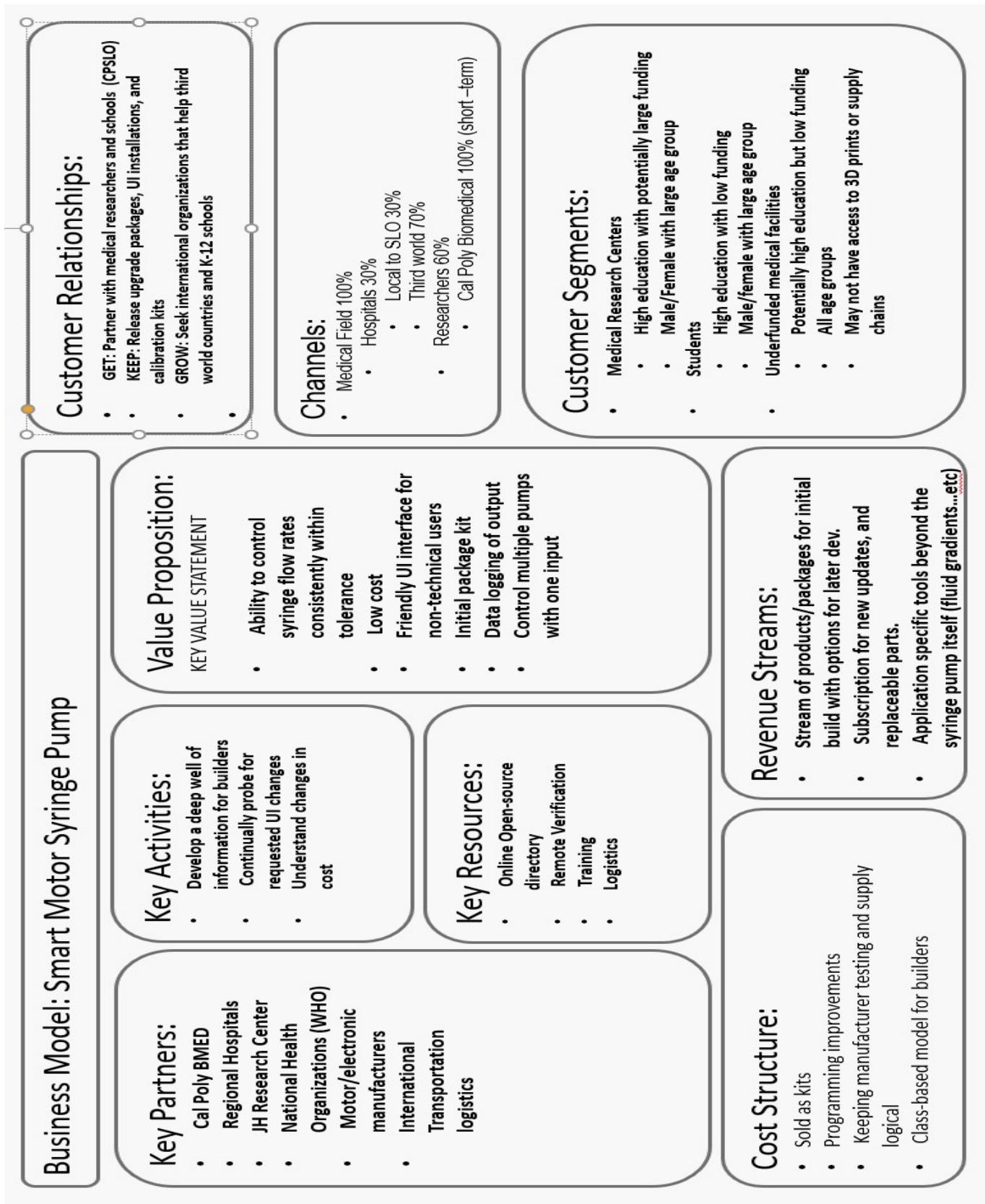


Fig. 47: Business Model Graphic

B. Marketing Data Graphic

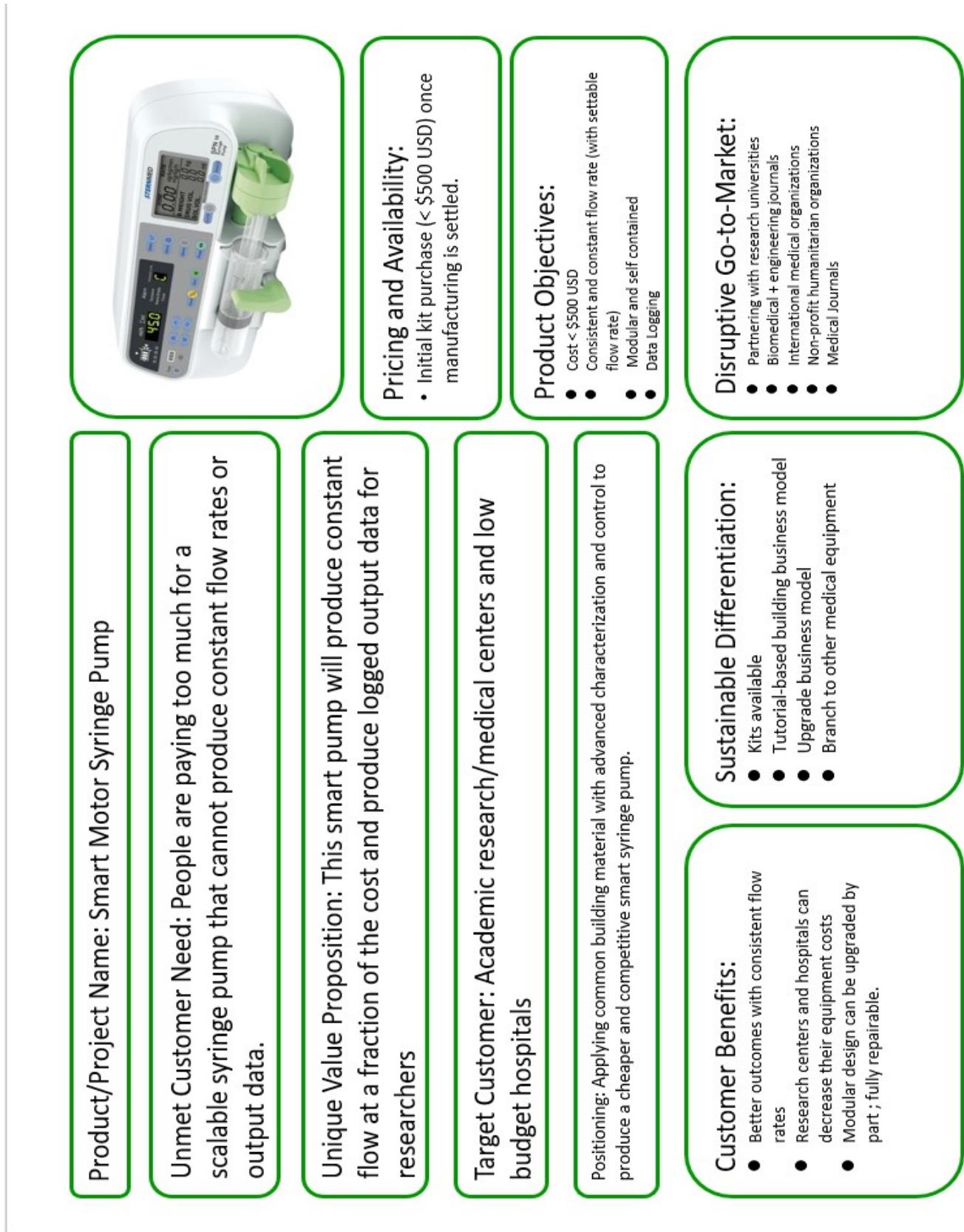


Fig. 48: Marketing Data Graphic

C. GANTT Chart

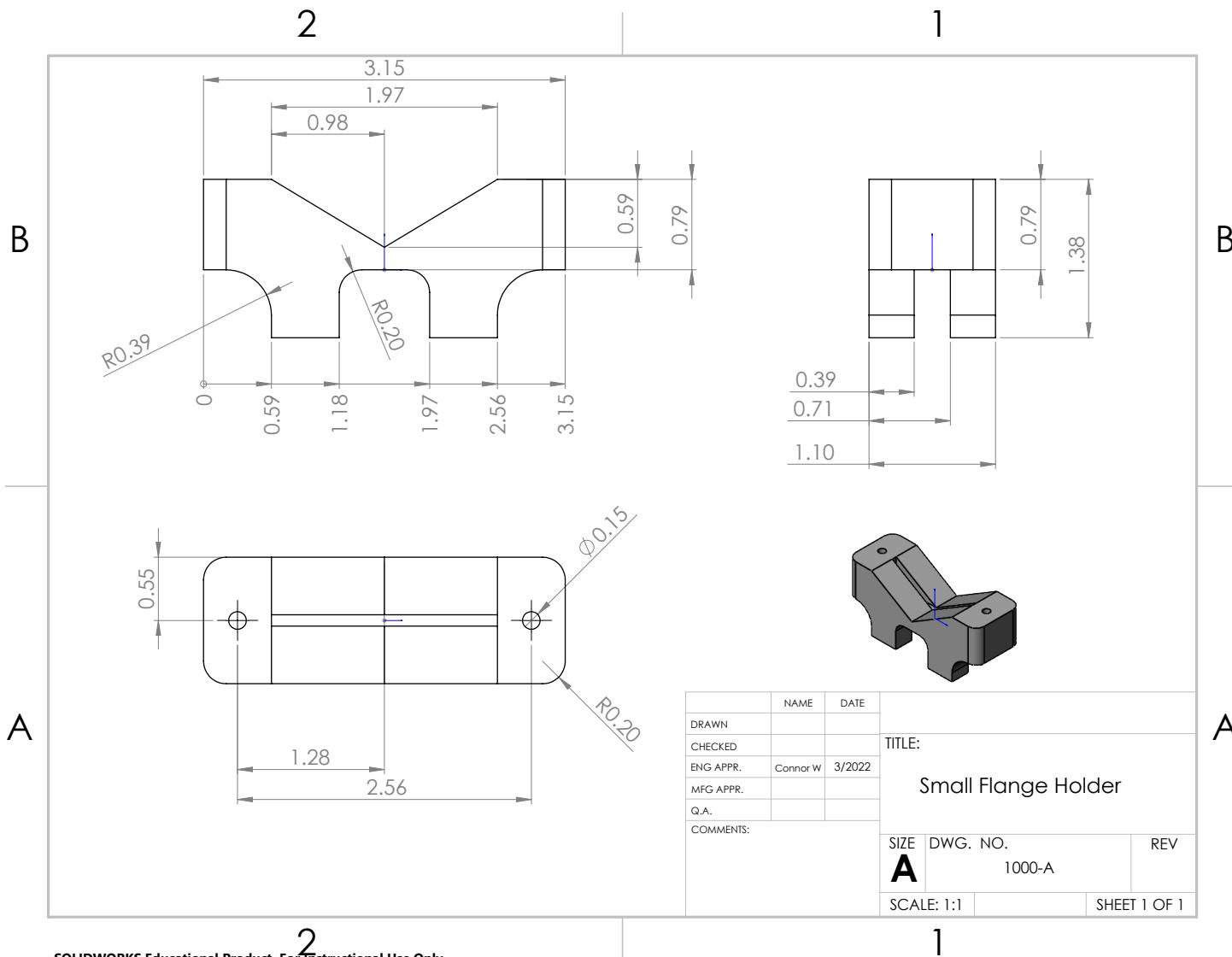


Fig. 49: Gantt Chart for Current Version of the SMSS

D. Bill of Materials

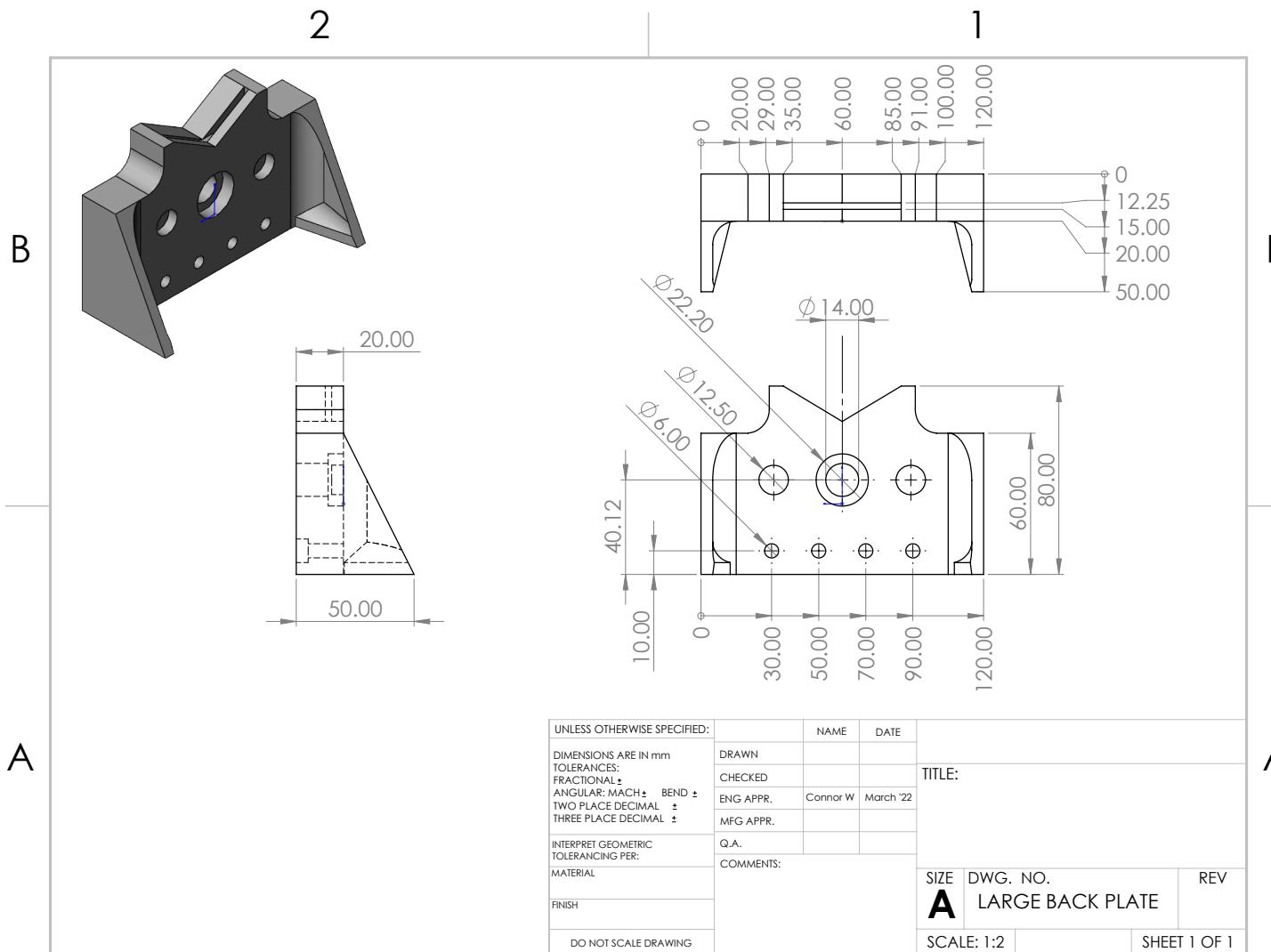
Fig. 50: Bill of Materials for Current Version of the SMSS

E. CAD Work



SOLIDWORKS Educational Product. For Instructional Use Only.

Fig. 51: Flange Holder for Smaller 1205 Ball Screw



SOLIDWORKS Educational Product. For Instructional Use Only.

Fig. 52: Flange Holder for Larger 1605 Ball Screw

F. Motor Controller Flow

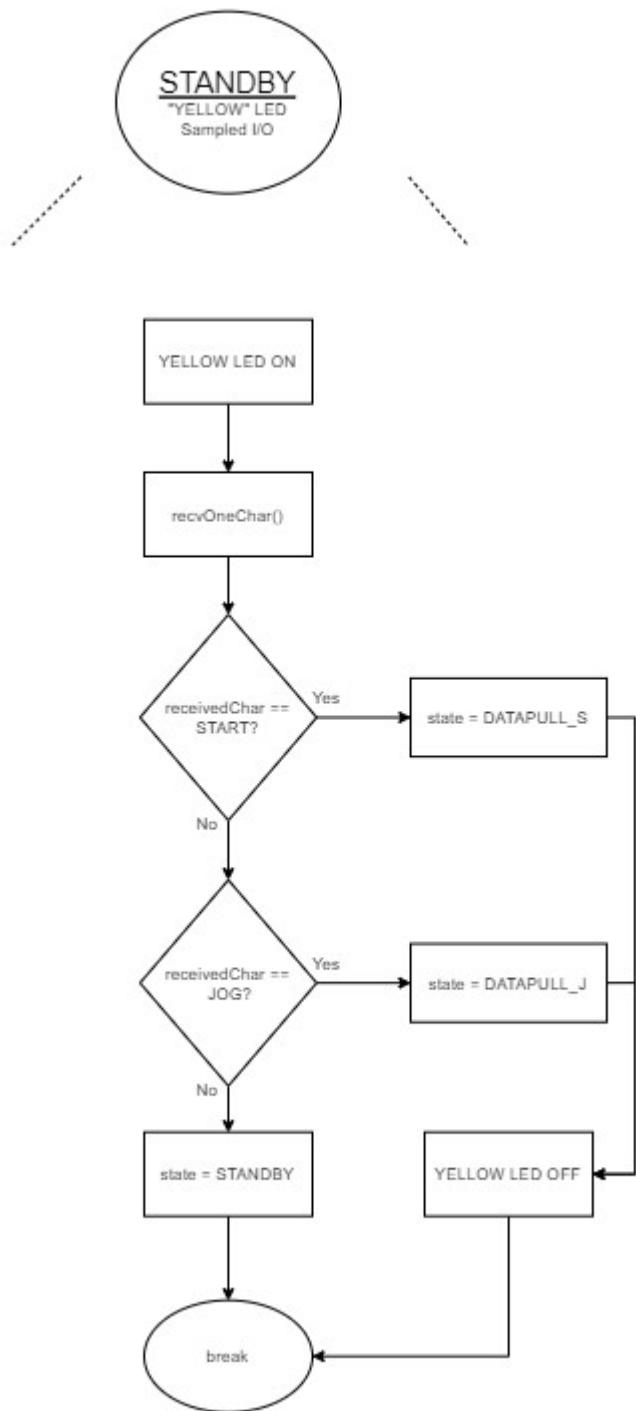


Fig. 53: STANDBY Logic Flow for Motor Controller

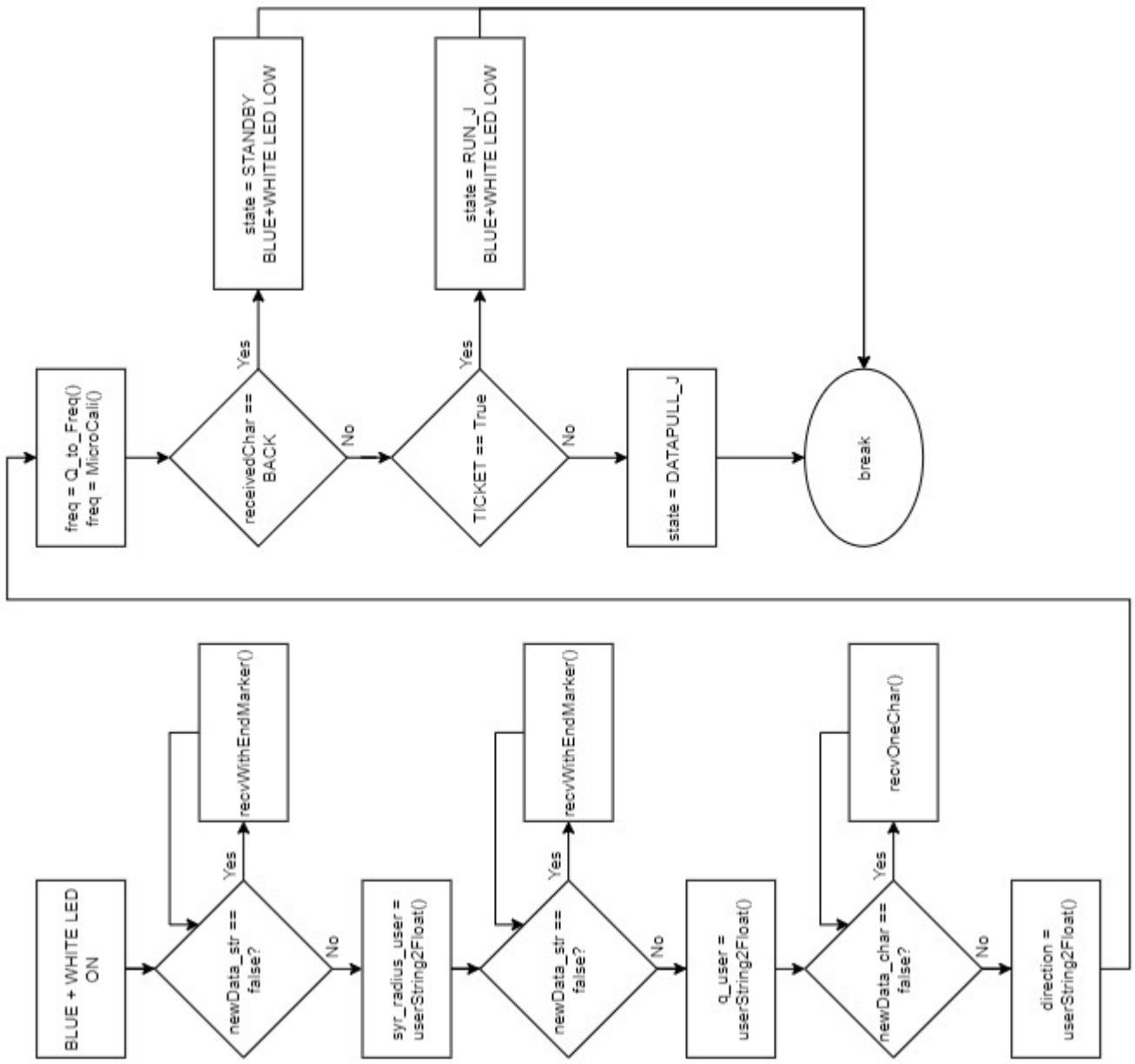


Fig. 54: DATAPULL_J Logic Flow for Motor Controller

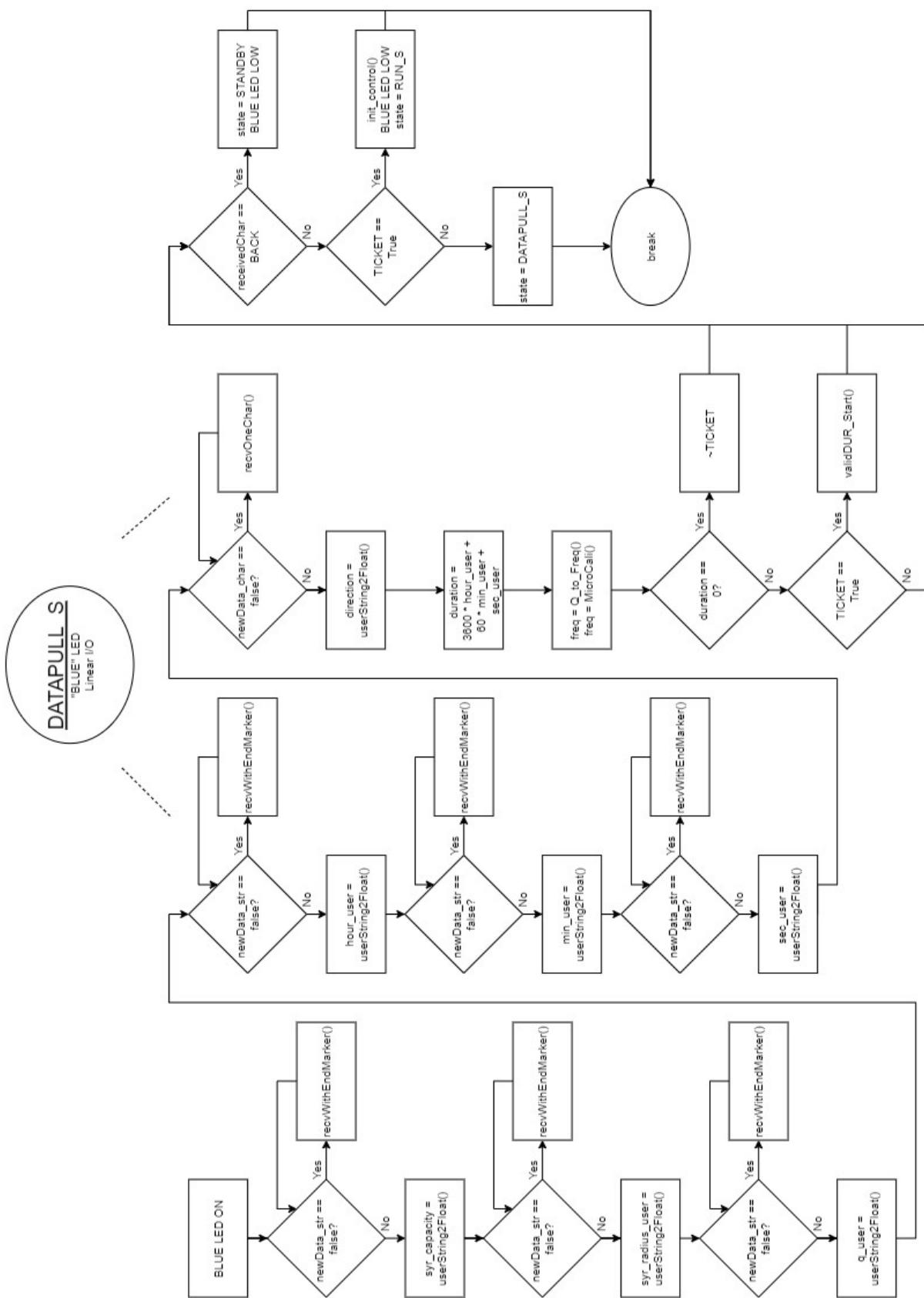


Fig. 55: DATAPULL_S Logic Flow for Motor Controller

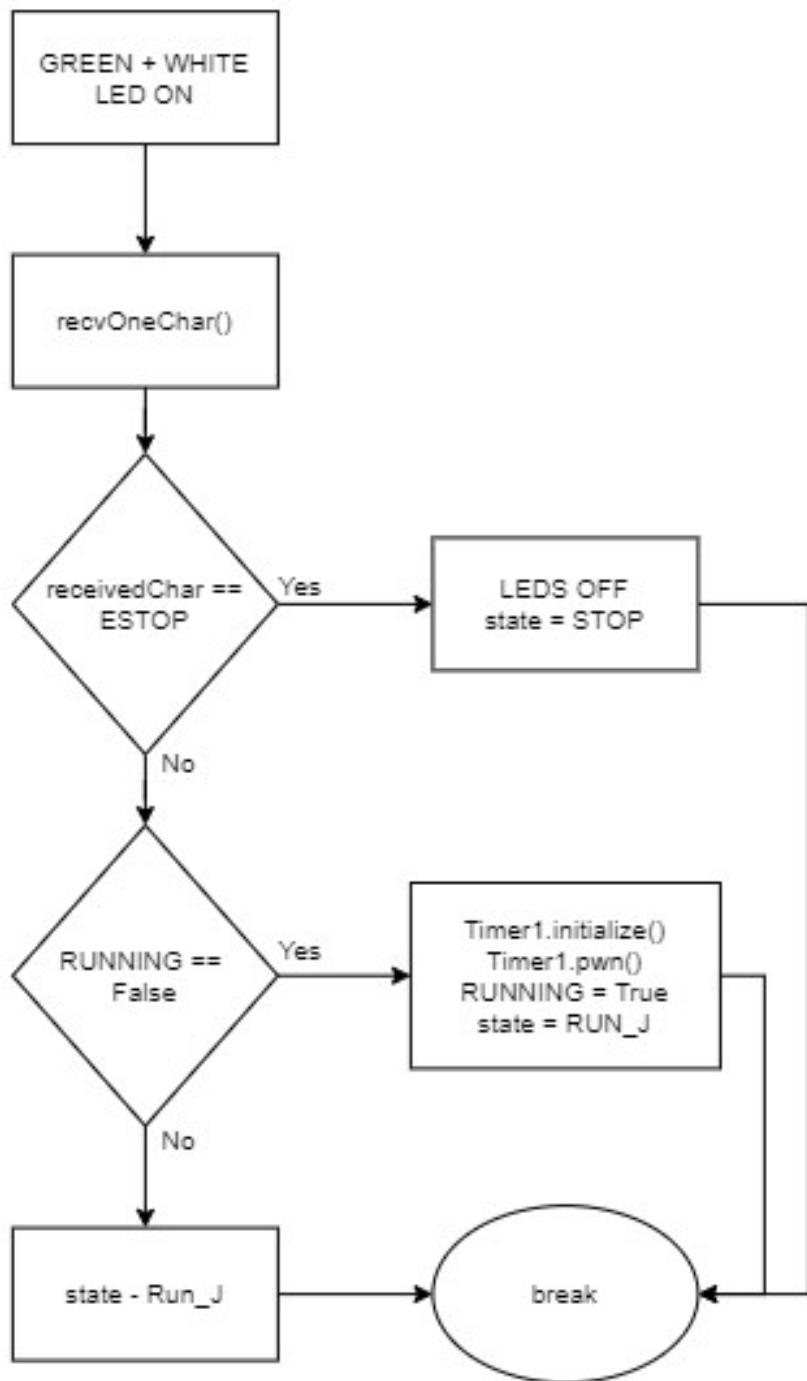


Fig. 56: RUN_J Logic Flow for Motor Controller

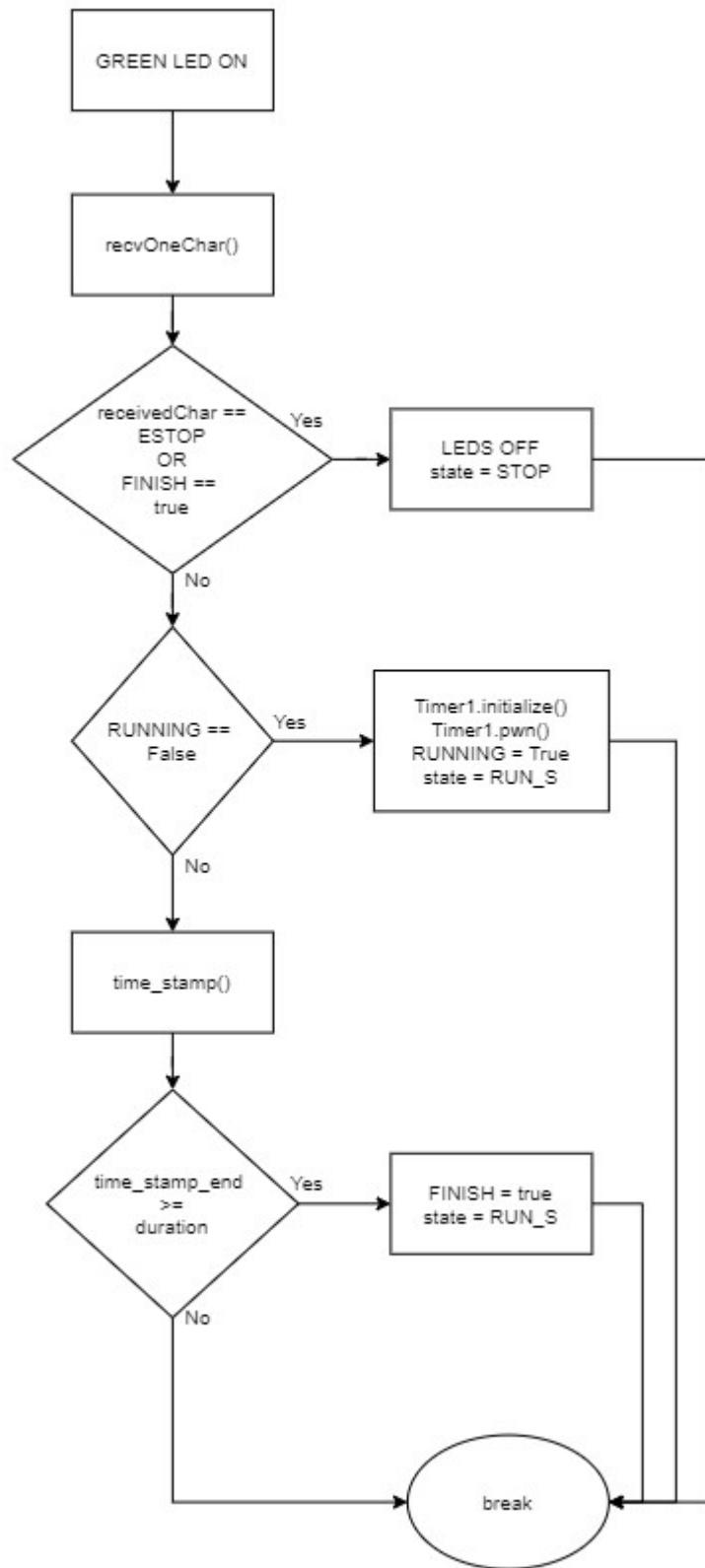


Fig. 57: RUN_S Logic Flow for Motor Controller

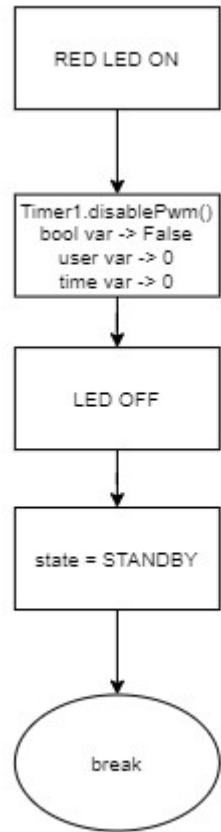


Fig. 58: STOP Logic Flow for Motor Controller

G. External Links

- Github: github.com/Cwilson-WhisperingRock/SMSS22

ACKNOWLEDGMENT

The author would like to thank

- Dr. Ben Hawkins for the guidance and knowledge to achieve success in a project such as this.
- Conrad Chan for putting forth amazing work for which this project stands on.
- My Fiancée for humouring me in my engineering rambles.

REFERENCES

- [1] Cision PR Newswire. *Global Intravenous Infusion Pump Market (2021 to 2030) - Featuring Baxter International, Terumo and Medtronic Among Others*. URL: <https://www.prnewswire.com/news-releases/global-intravenous-infusion-pump-market-2021-to-2030---featuring-baxter-international-terumo-and-medtronic-among-others-301416604.html>.
 - [2] Wikipedia. *Syringe Driver*. URL: https://en.wikipedia.org/wiki/Syringe_driver.
 - [3] AMIS Medical. *What is a Syringe Pump?* URL: <https://amismedical.com/what-is-syringe-pump/>.
- [4] American Institute of Physics. *Do teaching and communicating about microfluidics advances need improvement?* URL: <https://phys.org/news/2019-07-microfluidics-advances.html>.
 - [5] Conrad Chan and Jenny Chiao. *Smart Motor Syringe System*. Tech. rep. CA: California Polytechnic San Luis Obispo, Dec. 2020.
 - [6] CHEMYX. *What is a Syringe Pump?* URL: <https://www.chemyx.com/support/knowledge-base/application-reference-by-topic/what-is-a-syringe-pump-3/>.
 - [7] Google Trends. *Google Trend Search for (Syringe Pump)*. URL: <https://trends.google.com/trends/explore?q=syringe%5C20pump&geo=US>.
 - [8] MedicalExpo.com. *Medical Manufactures of Syringe Pumps*. URL: <https://www.medicalexpo.com/medical-manufacturer/1-channel-syringe-pump-2074.html>.
 - [9] Heyden Kirk Pittman. *Stepper Motor Theory*. URL: <https://www.haydonkerkpittman.com/learningzone/technicaldocuments/stepper-motor-theory>.