Arduino Based Automated Flow Control System

by
Tehreem Naeem

BE (Mechatronics), NUST, 2005

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering

in the
School of Engineering Science
Faculty of Applied Sciences

Simon Fraser University
Fall 2021

Declaration of Committee

Name:	**Tehreem Naeem**
Degree:	Master of Engineering
Title:	Arduino Based Automated Flow Control System
Committee:	Chair: Anita Tino Lecturer, Engineering Science
	Michael Adachi Supervisor Assistant Professor, Engineering Science
	Balbir (Bob) Gill Committee Member Sessional Instructor, Engineering Science

Abstract

At present when precise, accurate and time efficient systems have become a challenge, automated techniques are being replaced by manual practices. As a need of the hour, we introduce an automated, cost effective, yet reliable and efficient flow control system for fluids. Our prototype system can dispense varying amounts of fluids in milliliters (maximum 1L) as per demand of the user using peristaltic pump and solenoid valves. It uses the principle of time-based fluid dispensing achieved using PWM technology. To satisfy the principle used and verify the system's accuracy, fluids in different amount were dispensed for different amount of time and monitored. The experimental results of the fluid control system when tested showed linear relationship between the dispensing time and desired volumes of fluids. The added feature of our dispensing system is GUI for operating the Arduino. A screen comes up with buttons to toggle timers for individual valves as well as a button to run a sequence of commands in a specific order if necessary. GUI works in conjunction with the Arduino so the commands it sends must be formatted properly for the Arduino to actuate correctly. This system can be used in pharmaceutical and beverage industries as well as in different laboratories for dispensing and filling different volumes of fluids. [1]

Keywords: PWM; MOSFET; GUI; DC Voltage; Arduino; Peristaltic; Solenoid valve dispensing system; flow control

Acknowledgements

It is a great privilege for me to express my profound gratitude to my respected teacher Dr. Michael Adachi, Senior Professor, Engineering Sciences Department SFU, for his constant guidance, valuable suggestions, supervision and inspiration throughout my work without which it would have been difficult to complete this project within scheduled time.

I would like to take this opportunity to thank all the respected teachers of this department for being a perennial source of inspiration and showing the right path at the time of necessity. I would like to express my gratitude towards Engineering Sciences Department, to run the experimentation in the Electronics Laboratory at Burnaby Campus. I appreciate everyone who has directly or indirectly contributed towards the project.

Finally, to my supportive husband, Naeem, my deepest gratitude. Your encouragement when the times got rough are much appreciated and duly noted. It was a great relief to know that you were willing to provide management of our household activities while I completed my work. My heartful thanks.

Table of Contents

List of Tables

List of Figures

List of Acronyms

GUI	Graphical User Interface
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
DC	Direct Current
PWM	Pulse Width Modulation
IDE	Integrated Development Environment
SRAM	Static Random-Access Memory
D9	Digital pin 9 on Arduino UNO board
D10	Digital pin 10 on Arduino UNO Board
D11	Digital pin 11 on Arduino UNO board

Chapter 1.

Introduction

1.1. About the System

In the past many years, automated flow control system has been in high demand, largely because of its ability to drive efficiency, reduce error rates, and improve the uncertainties of general flow operations of liquids. The high production and safety concerns of today have resulted in an increased need for automotive approaches to substitute earlier manual techniques. [2]

Automated systems supersede manual effort. We have sophisticated and advanced machines nearly in every field to enhance throughput, efficiency and accuracy. Our pharmaceutical industries, clinical laboratories, environmental, chemical, food and beverage companies are also included in this category. Sample preparation process in all the mentioned industries is of critical importance especially with respect to pharmaceutical and food industry. The quality of the end products highly depends on the correct and reliable dispensing of needed volumes for the ingredients being mixed. [3]

For example, it is essential to mix exact volumes of syrup with carbonated fluids in food and beverage companies to obtain the proper taste. The correct concentrations of drugs or injections in pharmaceutical companies is more important as incorrect measurements can lead to life threatening results. [4]

To attain this level of precision, use of automation techniques for fluid measurement and dispensing is a need in place of manual methods which inherit errors. [1]

Flow control is a mechanism that regulates the flow or pressure of liquid. To be able to effectively manage flow control, flow control valves are used. The flow control of liquids is regulated as these respond to signals that are produced by independent devices like pump or motors etc. [5]

Our Flow control system delivers a precise amount of liquid in the milliliter range into a tube. Microcontroller (Atmega 328p Arduino Uno Board) is used as controller to automate the operation of flow of liquids. Arduino Microcontroller is selected as the controller because it is easier to implement, and the compact size of Arduino Uno board makes it easier to mount it on any system. The machine is also easy to operate and user friendly, where simple steps are needed to operate the machine. In this project microcontroller is used as the core of the system. [6]

In our microcontroller-based flow control system, the Arduino runs a program that parses serial input data of a specific format in order to extract commands from the graphical user interface (GUI) running on a computer. Data must be sent in a specific form to be interpreted properly and is configured within the code. A screen comes up with buttons to toggle timers for individual valves as well as a button to run a sequence of commands in a specific order if necessary. The Processing code works in conjunction with the Arduino so the commands it sends must be formatted properly for the Arduino to actuate correctly. The GUI builds a command string based on the user input by taking which channel should be actuated (“Liquid 1”, “Liquid 2”, or “Liquid 3”) and the amount of time set (1000ms, 2000ms etc) and sends that value out to the serial monitor to be read by the Arduino.

The user is requested to press the buttons on GUI to control the dispensing of fluids. This information is processed in the dispensing units and is used to dispense liquid according to the milliseconds selected. According to the allowance of time limit, selected liquid is discharged to the tube. Solenoid valves manipulate a flowing fluid using Peristaltic pump and control the flow of the liquid from the container electrically. [7]

In our project we are using a special type of pump known as PERISTALTIC PUMP. Peristaltic pumps are one of the most common items that we find in our house premises; a peristaltic pump is one of the displacement pumps which are positive in nature and is used to pump various kinds of fluids

A peristaltic pump is a pump, operated by a motor, that is able to uptake a liquid through one tube and drip it out through another tube. We can easily design a liquid dispensing system by using this peristaltic pump.

Chapter 2.

Background Knowledge

A literature review provided us an insight of automated techniques used for fluid dispensing along with several designs of dispensing systems proposed by various inventors. Hickerson [8], Ceccarelli et al [9] and Awada et al. [10] patented different devices but they all lacked the features of a pump and the ability for the user to control the quantity of liquid being dispensed.

2.1. Classification of Dispensing System

An improved liquid measuring and dispensing device was then proposed by Hanson in connection to cooking oil measurement and dispensing overcoming the deficiencies of the devices developed by aforementioned inventors [11].

A general purpose precise volume fluid dispenser was invented by Takacs to be used in pharmaceutical, beverage, concrete mixing and other industries. It consisted of pressure level sensor, bubbler tube, and a controller that controls the inflow and outflow of the precise volume of liquids from the entire system [12].

Butler et al. described a chemical storage and injection system for automatic dispensing of desired volumes of chemicals for industrial applications. His method included the utilization of sensors, valves and a controller for controlling the dual chamber filling and metered dispensing of the chemicals into a vessel [3].

Few well known and widely used fluid dispensing techniques are time-pressure dispensing and volumetric fluid dispensing using either pinch valves or positive displacement piston pumps [13].

2.2. Time Pressure Dispensing

Time pressure dispensing involves application of air pressure to the top of a container holding the fluid, forcing the material to be dispensed from its outlet. The amount

of fluid to be dispensed is directly linked to the length of time the pressure is applied along with other factors. These systems are widely used but are difficult to control [14].

The evolution of time pressure dispensing into automated equipment was a logical next step. However, implementing a smart system to compensate for the numerous dispensing variables has proved to be a difficult task. Time pressure dispensing is the most widely used dispensing method, but an increasing number of automated dispensing systems now utilize more reliable methods such as Archimedes metering pumps or piston positive displacement pumps. [15]

2.3. Volumetric Fluid Dispensing

The second technique is volumetric fluid dispensing using pinch valves. This includes a tube or cylinder whose volume is already known, employed between a fluid reservoir and a filling vessel. When this tube or cylinder is filled with the desired volume often determined by optical sensors, the premeasured fluid can be dispensed using pinch valves [9]. The volumetric positive displacement piston pump utilizes a tube of desired volume enclosed by pistons from either end. The sequence of downward and upward stroke of the piston governs the filling and dispensing of the predetermined amount of fluid. This method has been found advantageous due to its precise dispensing capability irrespective of the material's viscosity [16].

Our work presented here is about the design and implementation of a cost effective, reliable and flexible prototype of an automated adjustable fluid dispensing system. Time based dispensing technique was employed and the designed system can be used in clinical labs and pharmaceutical industry. The salient feature of our proposed system is the automated controlling of the entire process of dispensing a precise and user defined quantity of fluid into tubes from beakers [1].

Techniques	Equipment	Manufacturer
Time Pressure Dispensing	Time pressure liquid dispensing Pump	GPD Global
Volumetric Fluid dispensing	Volumetric Piston Dispenser for Volumetric dispensing Volumetric Dosing Dispenser	Scheugenpflug Nordson

Table 1. Summary of technologies of flow control mechanisms

Above table shows the summary of Existing Flow control mechanisms and technologies employed in them.

Chapter 3.

Methodology

The methodology of the project is;

i. Designing a circuit using peristaltic pump, solenoid valves, manual valve, beakers and tubes.

ii. Arduino Uno board is connected to the circuit for monitoring and command giving purpose.

iii. We are using three liquid to monitor their flow with respect to time.

iv. The inlet of the peristaltic pump is connected to the T- joint which in turn is connected to two Solenoid valves to intake fluid1 and fluid 2 and at pump outlet, a pipe is connected which runs to the waste beaker. Liquid3 is injected in manual valve through dropper. We use dropper because we want to check the flow of small volume as well.

v. A three-way manual valve is placed between valves and pump input.

vi. Inlet of two solenoid valves are connected to beakers filled with fluid1 and fluid 2 through pipes and the outlet is connected to pump through manual valve.

vii. Serial Communication to connecting the Arduino uno with the computer is done.

viii. Solenoid Valves and pumps are connected to Arduino through MOSFETS.

ix. MOSFETS are used as a switch in the circuit.

x. 12 Volts DC power supply is given to the circuit through MOSFETS.

xi. Now we upload the program into Arduino and click on the “VERIFY” key of the simulator.

xii. Pump runs by the code written in Arduino IDE and we can control the process by the GUI developed in a software called “Processing”

Proposed Design

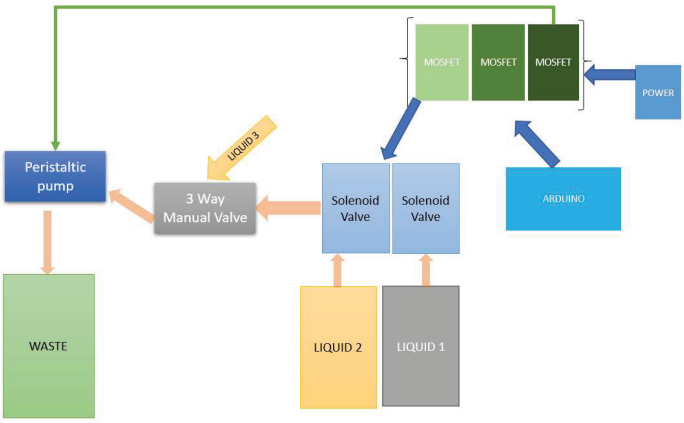

Figure 1. Block diagram of Proposed Design

Figure 1 shows a proposed design in the form of a block diagram. It describes the principal parts and their connections. Main components are represented by blocks connected by lines that show the relationships of blocks. This is main step towards our electronic design and process flow diagram.

Chapter 4.

Prototyping of the system

Our project requires two parts, HARDWARE and SOFTWARE.

4.1.1. Hardware requirement

i. Arduino Uno R3

ii. MOSFETS

iii. Power supply

iv. 12V peristaltic pump

v. Beakers

vi. Solenoid valves

vii. Three-way manual valve

viii. Jump wires

ix. Pipe

x. Wooden boxes.

4.1.2. Software requirement

i. Arduino IDE.

ii. Processing for Graphical User Interface (GUI)

4.2. Hardware Requirements

4.2.1. Arduino UNO R3

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board.

The Arduino platform has become quite popular with people just starting out with electronics, and for good reason. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.

The Arduino is a microcontroller board based on the ATmega8. It has 14 digital - input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.

i. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter. Revision 2 of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode. Revision of the board has the following new features:

ii. pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible with both the board that uses the AVR, which operates with 5V and with the Arduino Due that operates with 3.3V. The second one is a not connected pin, that is reserved for future purposes.

iii. Stronger RESET circuit.

iv. ATmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform.

Figure 2. Arduino UNO board
Source: https://www.distrelec.biz/Web/WebShopImages/landscape_large/9-/01/arduino-a000066.jpg

- Here are some parameters of Arduino and the descriptions.
- Microcontroller - ATmega328
- Operating voltage - 5V
- Input voltage (recommended) - 7-12V
- Input voltage (limits) - 6-20V

- Digital I/O pins - 14 (D0-D13 are digital I/O; D3, D5, D6, D9, D10, D11 are PWM).
- Analog I/P pins - 6.
- Flash memory - 32KB (ATmega328).
- SRAM - 2 KB (ATmega328).
- EEPROM - 1KB (ATmega328).
- Clock speed - 1.6 MHz
- Length - 68.6 mm.
- Width - 53.4 mm.
- Weight - 25 g.

4.2.2. Dual Channel MOSFETS

We are using three NPN MOSFETS as switches. Two for solenoid valves and one for liquid that is injected through manual valve. The Arduino can not drive the flow switches and pump directly from a digital pin. The MOSFET connects the flow switch or pump to the power supply with a switch using much less current from the Arduino.

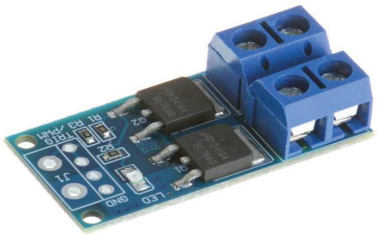

Figure 3. Dual Channel MOSFET
Source: https://www.electan.com/images/7325415412.jpg

The MOSFET (Metal Oxide Semiconductor Field Effect Transistor) transistor is a semiconductor device that is widely used for switching purposes and for the amplification of electronic signals in electronic devices. A MOSFET is either a core or integrated circuit

where it is designed and fabricated in a single chip because the device is available in very small sizes.

A MOSFET is a four-terminal device having source(S), gate (G), drain (D) and body (B) terminals. In general, the body of the MOSFET is in connection with the source terminal thus forming a three-terminal device such as a field-effect transistor. MOSFET is generally considered as a transistor and employed in both the analog and digital circuits. This is the basic introduction to MOSFET [17].

We are using Dual Channel MOSFET with the following Parameters.

Parameter

- Operating Voltage: DC 5V - 36V
- Maximum current up to 30Auous current: 15A
- High Reliability
- Power: 400W.
- Perfect support PWM

4.2.3. Applications

It can be used to control the output of power equipment, motors, light bulbs, LED lights, DC motors, micro-pumps, solenoid valves, etc., can input PWM, control motor speed, lamp brightness and so on.

12V PERISTALTIC PUMP

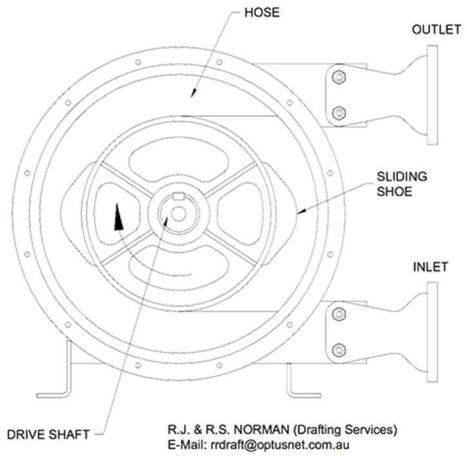

Figure 4. 2V DC peristaltic pump
Source: https://accendoreliability.com/wp-content/uploads/2017/11/V2E6-5-1.png

We are using Peristaltic pump. The name “Peristaltic pump” was derived the word “peristalsis” which is its pumping principle. It consists of a flexible tube, rotor with rollers, shoes or wipers, pump casing etc. The fluid is kept inside the flexible tube or hose which compress and relaxes making the inward and outward flow of the fluid. For that rollers attached with the rotor, shoe and wipers are attached to the external surface of the tube which initiates the compression and relaxation. When the rotor gets started, an under-compression tube part closes compelling the liquid to move through. A cam is passed then to make the tube regain its original shape and the fluid flow is induced back to the pump, this process is called restitution.

The peristaltic pump was first patented in the United States by Rufus Porter and J.D. Bradley in 1855 (U.S. Patent number 12753) as a well pump, and later by Eugene Allen in 1881 (U.S. Patent number 249285) for blood transfusions. It was developed by heart surgeon Dr. Michael DeBakey for blood transfusions while he was a medical student in 1932 and later used by him for cardiopulmonary bypasssystems. A specialized non-occlusive roller pump (US Patent 5222880) using soft flat tubing was developed in 1992 for cardiopulmonary bypass systems.

Here we are using a 12V DC peristaltic pump. The specifications of the peristaltic pump are as follows: -

- Voltage rating: - 12 V DC.
- Current rating: - 0.75 amps.
- Wattage rating: - 9 watt.
- Working condition temperature: - 0-40oC. V.
- Relative humidity: - < 80%.
- Flow rate: - 0-100 ml/min.
- Rotate speed: - 0.1-5000 rpm.
- Driver size: - diameter 27.6 X height 37.9 (mm).
- Head size: - diameter 31.7 X height 20.1 (mm).
- Equipped with pump tube (ID x OD): - 2.5 mm inner diameter X 4.7 mm.
- Suitable for medical, chemical experiments, environmental protection.

4.2.4. Solenoid Valves

We are using two solenoid valves for liquid1 and liquid2. Solenoid valve works by controlling the flow of liquids or gases in a positive, fully closed or fully open mode. They are often used to replace manual valves or for remote control. Solenoid valve function involves either opening or closing an orifice in a valve body, which either allows or prevents flow through the valve. A plunger opens or closes the orifice by raising or lowering within a sleeve tube by energising the coil.

Image redacted due to Copyright.

Figure 5. Solenoid valve
Source: https://images-na.ssl-images-amazon.com/images/I/41w4%2BMEORuL._SX342_.jpg

Solenoid valves consist of a coil, plunger and sleeve assembly. In normally closed valves, a plunger return spring holds the plunger against the orifice and prevents flow. Once the solenoid coil is energised, the resultant magnetic field raises the plunger, enabling flow. When the solenoid coil is energised in a normally open valve, the plunger seals off the orifice, which in turn prevents flow.

In our project, it is necessary to start or stop the flow in the circuit to control the fluids in the system. An electronically operated solenoid valve is usually used for this purpose. By being solenoid actuated, solenoid valves are positioned in the hardware in a way that they are conveniently controlled by simple electrical switches.

Solenoid valves are the most frequently used control elements in fluidics. They are commonly used to shut off, release, dose, distribute or mix fluids. For that reason, they are found in many application areas. Solenoids generally offer fast and safe switching, long service life, high reliability, low control power and compact design [18].

4.2.5. Three-way Manual Valve

Figure redacted due to Copyright considerations.

Figure 6. 3 Way Ball Valve 1/8" NCH O/D Tube
Source: https://m.media-amazon.com/images/I/516lzFWQyWL._SL1000_.jpg

Figure 6 shows three-way stainless-steel manual valve. It has three ports or openings that are connected to pipes or tubes for fluid flow. These ports are described as inlets and outlets. We are using two ports as inlets and one as outlet.

4.2.6. Jump wires

We are using jump wires to connect Arduino UNO to MOSFETS and valves by inserting their end connectors into the slots provided at Arduino UNO board.

Figure redacted due to Copyright considerations.

Figure 7. Jump wires
Source: https://cdn.sparkfun.com//assets/parts/7/8/9/2/11709-01.jpg

Figure 7 shows jump wires which we are using to interconnect components of our system. They are also called electrical wires and have connector or pin at each end.

4.2.7. Beakers

Total 3 no of beakers are used in our project. Two are of 500ml and one of 125ml. Two beakers are filled with liquid1 and liquid2 and one is used for waste. A pipe carrying waste runs from outlet of pump to the waste beaker.

Figure redacted due to Copyright considerations.

Figure 8. 500ml and 125ml beakers

Figure 8 shows plastic beakers we are using for our system. One of 500ml beaker is used for liquid1, and other 500ml beaker is used for waste. We are using 125ml beaker for liquid2.

4.2.8. Pipes

Pipes are used here in the inlets and the outlets of the peristaltic pump, solenoid valves and manual valve.

Figure redacted due to Copyright considerations.

Figure 9. Tygon 2375 Plastic PVC Tubing, 1/16" ID, 1/8" OD, 1/32" Wall, 50' Length, Clear

Source: https://www.usplastic.com/catalog/images/products/Misc/sku/400/57544p.jpg

Figure 9 shows the pipes we are using for our flow control system. This tubing is formulated to transfer a wide variety of liquids. This is made up of PVC based material and the clarity allows for easy visual monitoring.

4.3. Software Requirements

As explained earlier our project consists of hardware and software. Hardware parts are explained above and software requires as follows:

4.3.1. Arduino IDE (Integrated Development Environment)

The open-source Arduino Software IDE (Integrated Development Environment) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-

source software. This software can be used with any Arduino board. The Arduino development environment contains a text editor for writing code, a message area, a

text console, a toolbar with buttons for common functions, and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.

Software written using Arduino are called sketches. These sketches are written in the text editor. Sketches are saved with the file extension. ino. It has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino environment including complete error messages and other information. The bottom right-hand comes of the window displays the current board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

Figure 10. ARDUINO IDE
Source: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQLRq18VXW3sJ3TAw9qX0Zl3sk_H64jKDzgFg&usqp=CAU

Figure 10 shows Arduino Integrated Development Environment.

Chapter 5.

Experimental setup

The low-cost fluid Automated Flow Control System was developed using “Arduino UNO”. The microcontroller of Arduino Uno was programmed in IDE. The principle utilized for dispensing precise, accurate and user defined volumes of fluids was” time based dispensing‟ i.e. the amount of fluid to be dispensed will be determined by the time taken by that fluid to dispense completely [14].

We have developed a Graphical User Interface (GUI) which has four separate press buttons. Three buttons are used to control flow three liquids, one for each liquid separately. To automate the system, we have developed another press button on GUI which run the program in a sequence, and automatically calls the routine for each liquid and pump runs for a specific amount of time for each liquid. Each liquid is dispensed in a certain amount for a definite amount of time in an automated sequence. This time is loaded in the microcontroller’s timer through programming to control the opening and closing of solenoid valve along with the working of the pump. We aimed to investigate the working accuracy of the fluid dispenser by dispensing three different fluids in a definite order. A separate time factor was measured for each type of fluid to be used in programming [1].

5.1. Conceptual Design

Our conceptual design is given below:

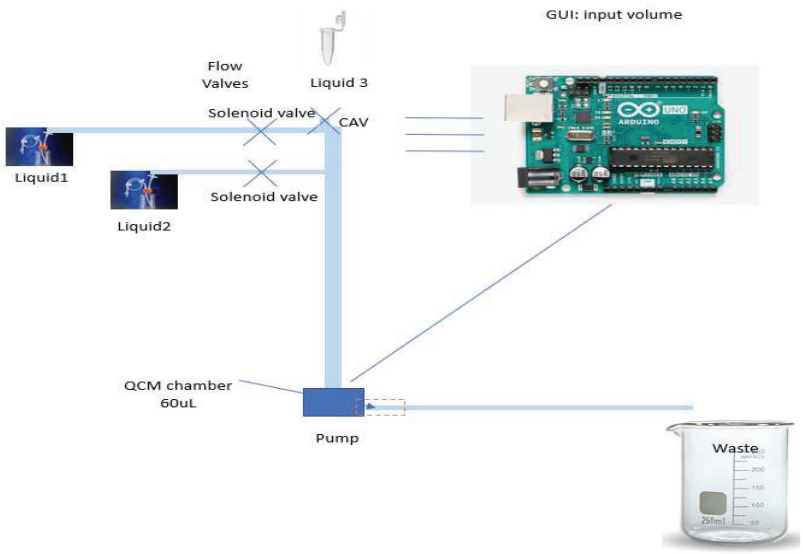

Figure 11. Conceptual design

Figure 11 shows main components of the hardware design to describe the main idea of the circuit. We used symbol for solenoid and manual valve. Image for beaker is used to show waste.

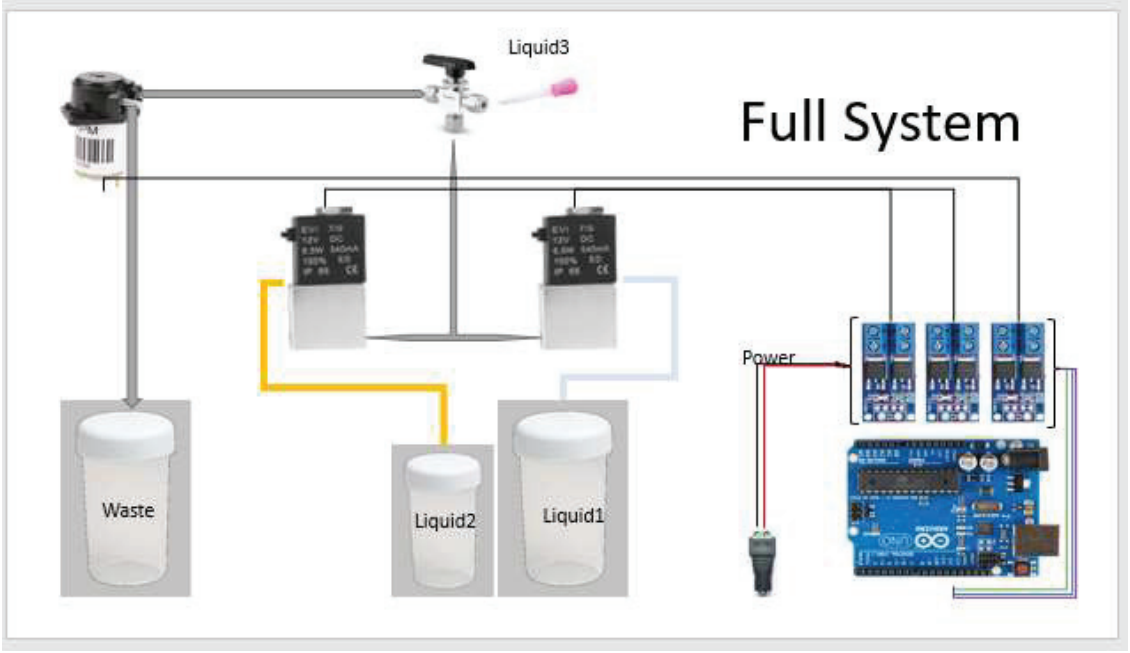

Figure 12. Conceptual design with connections and components

Figure 12 shows full system with components and connections. Above schematic uses real images of the components and shows connections between them according to our electric circuit.

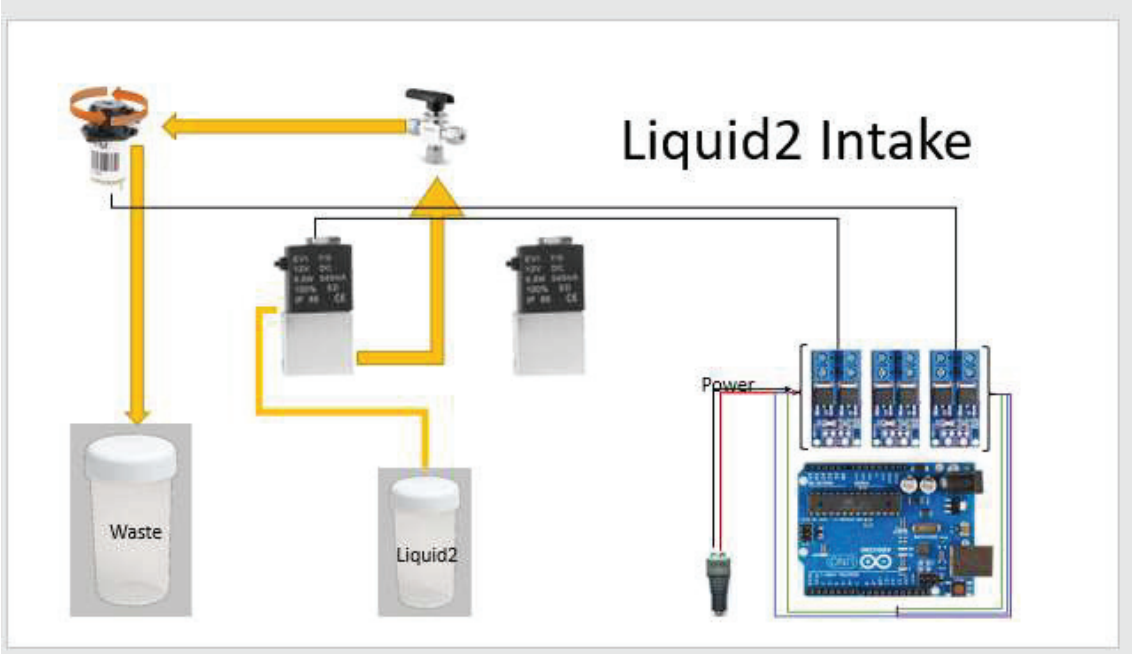

Figure 13. Conceptual design showing Liquid2 Intake

Figure 13 shows liquid2 intake process when one valve is open and other is closed. Figure clearly shows that one solenoid valve is connected and other one is inactive.

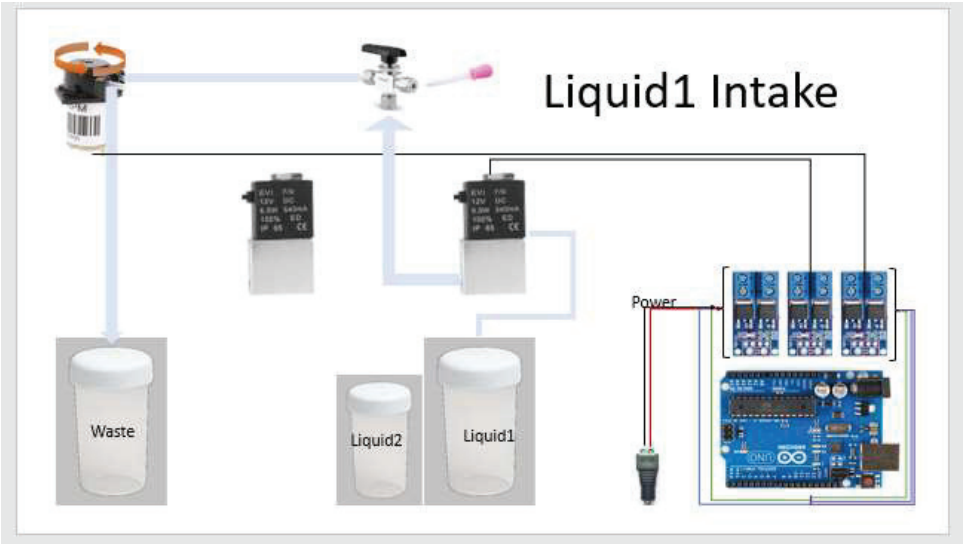

Figure 14. Conceptual design showing Liquid1 Intake

Figure 14 shows liquid1 intake process. In this schematic we can see that Solenoid valve1 is connected to the pump and solenoid valve2 is inactive. When command is sent from GUI after pressing button for liquid1, pump runs and liquid1 is taken.

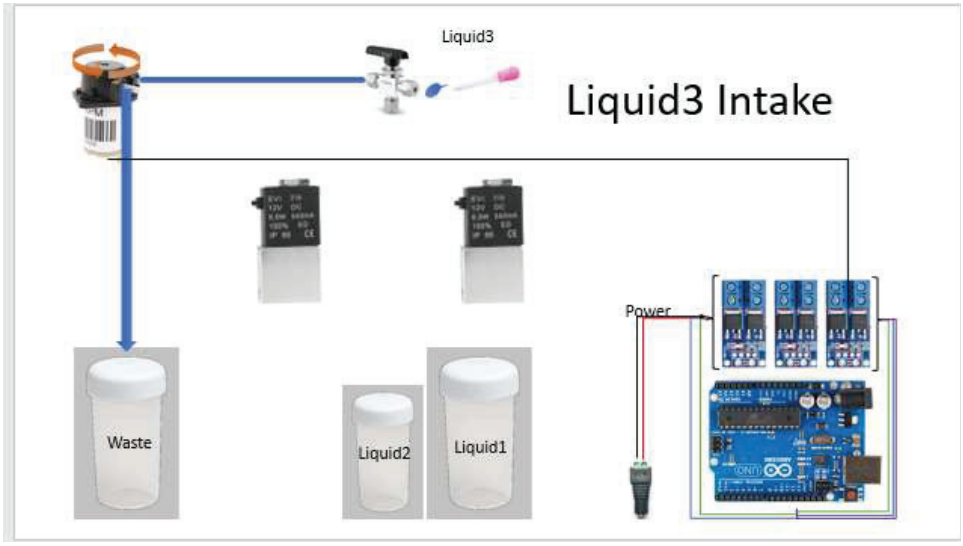

Figure 15. Conceptual design showing Liquid3 Intake

Figure 15 shows liquid3 intake process through a dropper when solenoid valves are not in use. In this case, only pump runs. In our automatic sequence which runs by pressing an individual button

on GUI, process pauses and ask for manual injection of liquid3 by dropper. And then when valve is closed, process is resumed.

5.2. In Home Flow Control Setup

Setup compromising of Arduino Uno, peristaltic pump, two solenoid valves and one three-way manual valve was tested for the purpose of understanding and testing the flow of liquids of three liquids. The 12 V AC adapter was used for power supply to the system. Jumper wires were soldered to connect the components. The picture shown below is an example setup made for the testing.

5.3. Initial Design

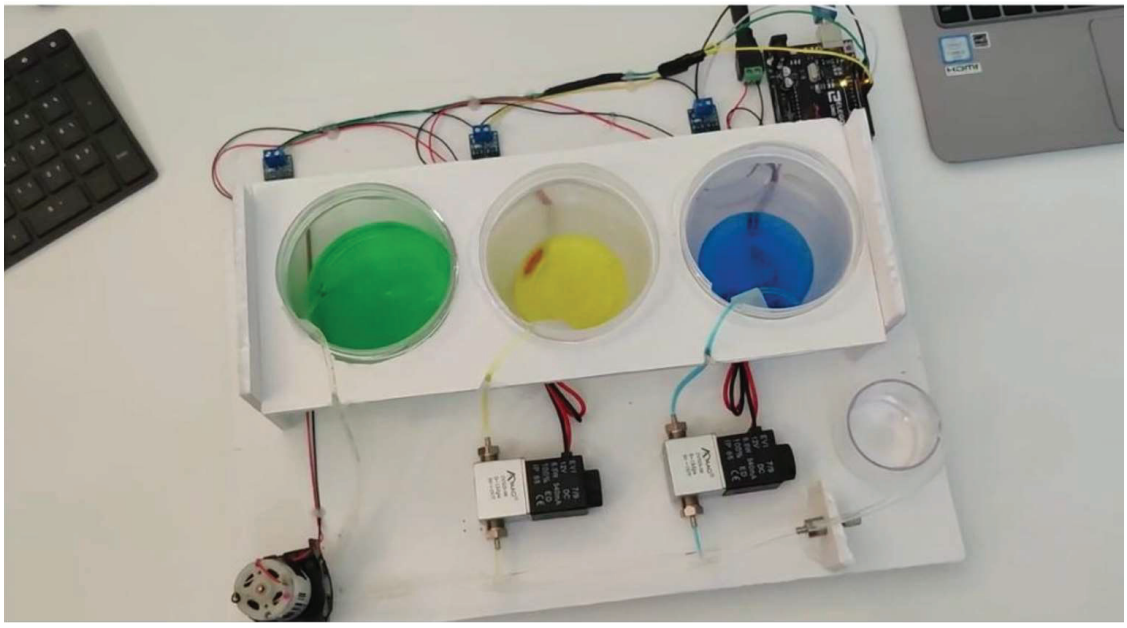

Figure 16. Initial setup

Figure 16 shows our initial setup. We made the initial setup with Styrofoam sheets to test our experiment. We used sheets to make stand for liquids as an upper layer and on lower layer, we placed our components which are Peristaltic pump, solenoid valves, MOSFETS and Arduino UNO board.

5.4. Final Design

Our Initial setup was taking up much volume, to reduce size and make our design more compact, we used wooden boxes and wooden sheets to mount our system on, which turned out to be more durable and reliable.

Module 1 Setup

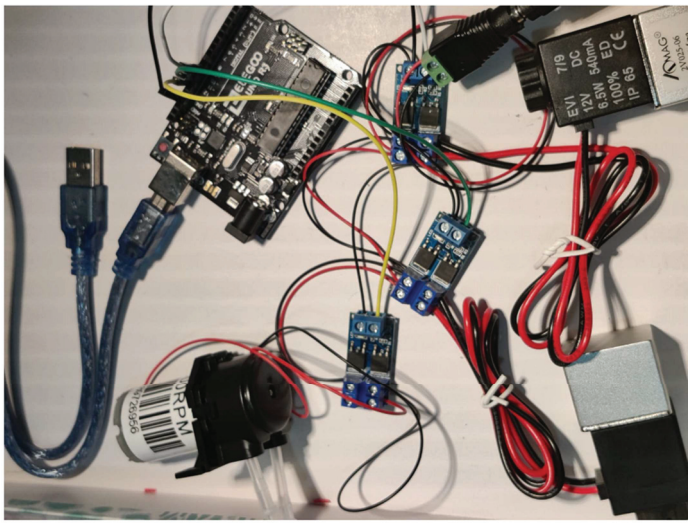

Figure 17. Module 1 Setup

Figure 17 shows the Module1 Setup to run the code and make sure the connections are correctly done. We connected components by soldering jump wires. We run the code and made sure the components are correctly connected according to our circuit diagram.

Module 2 Setup

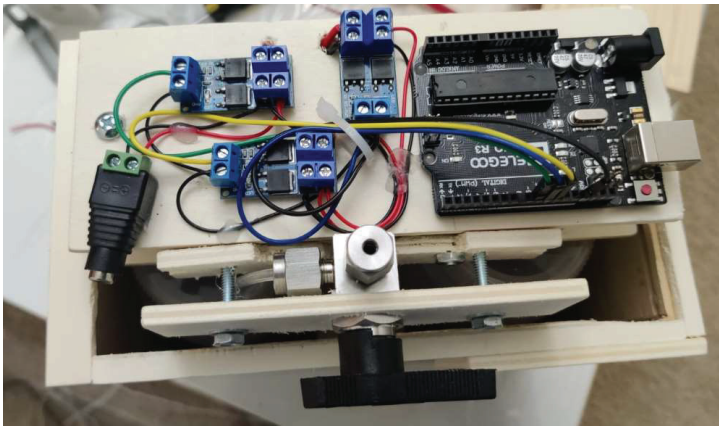

Figure 18. Module 2 Setup

Fig 18 shows hardware mounted on wooden sheets in two layers to save space and to make system more compact. On lower layer, we placed MOSFETS and Pump and on upper layer, we fixed rest of components like MOSFETS, Arduino Board, and Power Supply. We used two wooden boxes for our hardware. In one wooden box, we placed our electronics and in other wooden box, we placed beakers. We mounted boxes over each other to reduce size of our system. Manual valve is fixed outside of the box so that we can open and close it easily without opening the box.

Setup for beakers

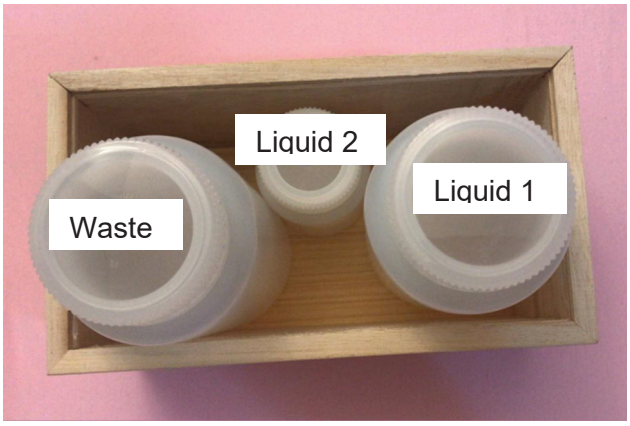

Figure 19. Shows setup for beakers

Fig 19 shows setup after fixing the beakers inside a wooden box. Two beakers are used for two kinds of liquids and one is for waste. We are injecting liquid 3 manually by a dropper through a manual valve.

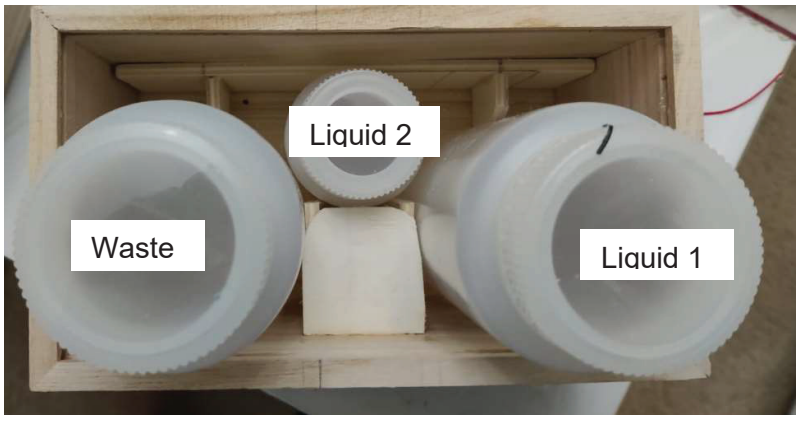

Figure 20. Beakers after fixing in the box

Fig 20 shows setup after fixing the beakers inside a wooden box. We put some wooden fixtures to hold beakers in their place.

Final setup after fabrication

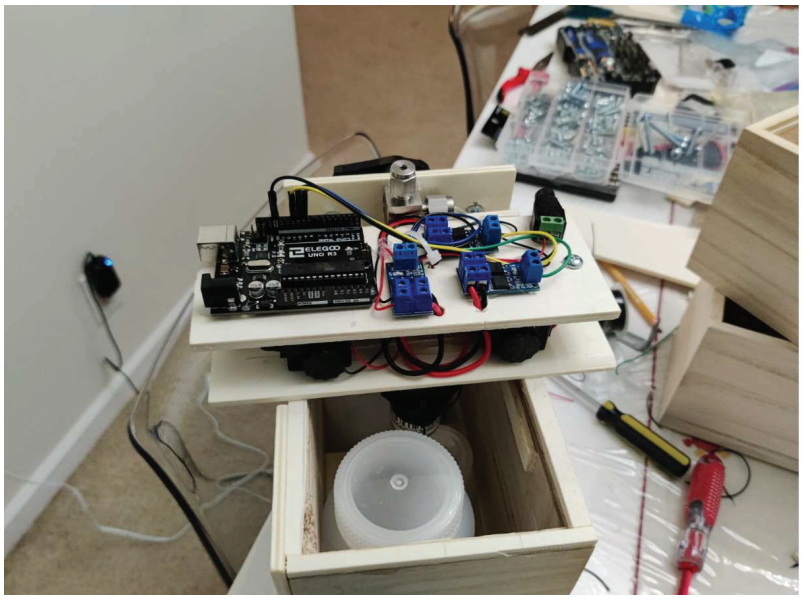

Figure 21. Final Hardware setup

Fig 21 shows clear image of two layered hardware and box for beakers.

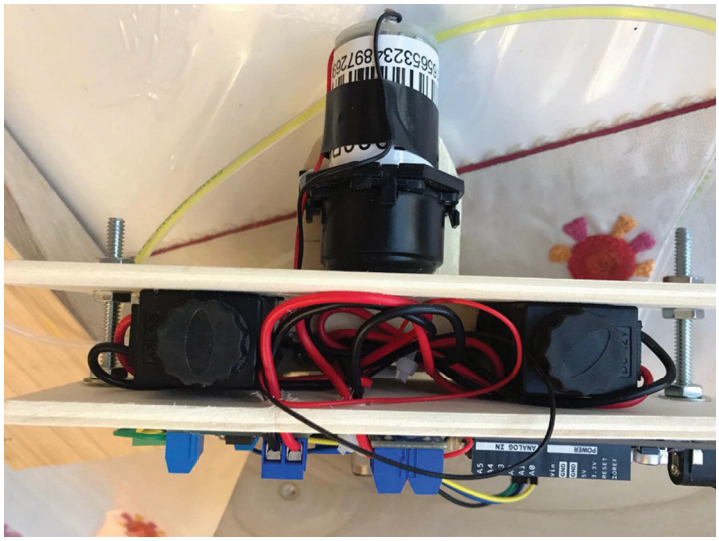

Figure 22. Hardware in two layers

Fig 22 shows detailed image of two-layer system showing pump fixed underside, two solenoid valves are on first layer, MOSFETS, Power jack and Arduino board are fixed on second layer.

Finished Design

Figure 23. Finished Design of Flow Control System

Figure 23 shows our finished product with lid on it. We have mounted two boxes over each other and fixed them with screws. We have fixed lid over boxes with hinges.

5.5. Implementation

5.5.1. Hardware Implementation

We connected digital pins i.e., D9, D10 PWM/SS and D11 of ARDUINO UNO to three MOSFETS. Arduino turns the MOSFETS on. Pump is controlled by Pulse Width Modulation Signal. Instead of just turning on and staying on, it is turned on and off very quickly. The amount of time it is turned on for compared to off for determines how fast the pump runs. The PWM value written in a normal Arduino is 1 byte in size. the max value of a byte is 255. 127 would be 50% duty cycle. DC motors have pretty linear power curves so it would be roughly half. A value of 127 (meaning 127/255) would be exactly 50% of the time on and 50% off. Since ours is 80, it spends (80/255=.313) 31% of the time on and 69% off. This makes it look like its spinning slower which gives us more control over the amount of liquid pumped. The 80 is picked arbitrarily and it can be changed if we would

like the motor to spin faster or slower. Two Solenoid valves are used for two different liquids and one three-way manual valve is used for taking a very small amount of liquid.

Pinout Diagram

Figure redacted due to Copyright considerations.

Figure 24 Pinout diagram
Source: https://robu.in/wp-content/uploads/2020/07/1-3.jpg

Fig 24 shows Pinout Diagram for Arduino UNO. We used digital pin 9, pin10 and pin 11 and named them D9, D10 and D11. D9 is connected to Solenoid Valve1, D10 is connected to solenoid valve2 and D11 is connected to Peristaltic pump.

5.5.2. Software Implementation

For software implementation we require a software namely Arduino IDE. This software enables us to load the program in Arduino board. ARDUINO UNO is a microcontroller board based on the “ATmega328P” or “ATmega328” IC. It has 14 digital I/O pins (of which 6 can be used as PWM O/O pins), 6 analog I/P pins, a 16 MHz quartz crystal, a USB connecting jack, a power jack, an ICSP header and a reset button. The O/P of the UNO is 5V, so we have to connect an external voltage source to the MOSFET. Next we have our setup() function, which makes the motor an O/P device, since we are writing to the motor. After this we have our loop() function.

This is how a peristaltic pump can be controlled by an ARDUINO microcontroller IC and can be used for a water dispensing system. The code for running the peristaltic pump is written in C++ language in a computer software called “ARDUINO IDE”.

5.5.3. Graphical User Interface (GUI)

The code in Processing serves as the GUI for operating the Arduino. The Arduino handles reading the input commands and actuating the valves and pump. It is the job of the GUI to send text-based commands to the Arduino to take action. The buttons on the GUI represent different text commands to send and are dynamic, updating as the user changes the time delay for the system to run. The commands are sent over a virtual com port that the Arduino is monitoring. The library used in Processing is the Serial library. Processing and the Arduino are communicating on the same serial port, so the Arduino is constantly monitoring and waiting for input. Once the GUI sends a command, the Arduino sees that data is available on the serial port and reads it in to parse for the correct action to take. A simple function is used from the Serial library to send the command to the Arduino. The function is myPort.write(String); which the GUI uses to send the commands over as text for the Arduino to read. The GUI waits to send a command until the previous one is complete. Since the GUI sent the command, it knows how long the Arduino will take to run it before the GUI should send anything else. The GUI builds a command string based on the user input by taking which channel should be actuated (“liquid1”, “liquid2”, or “liquid3”) and the amount of time set (1000ms, 2000ms, etc) and sends that value out to the serial monitor of the form to be read by the Arduino. The sequence function runs a little differently by automatically sending multiple signals in a predefined order to run a process with one click. It sends the commands for the Arduino to perform before a required pause. It allows a user to insert a solution manually while waiting for another click on the program to resume the rest of the sequential process in the following order.

1. CLICK BUTTON
2. Liquid1 for 5000ms
3. Pause for injecting liquid3 through dropper
4. click the button again to continue dispensing liquid1 for 1000ms

5. Liquid2 for 5000ms

6. Yellow water for 5000ms

7. Liquid1 for 5000ms

Once complete, the program waits for another command to take action again.

GUI Pop-up Menu

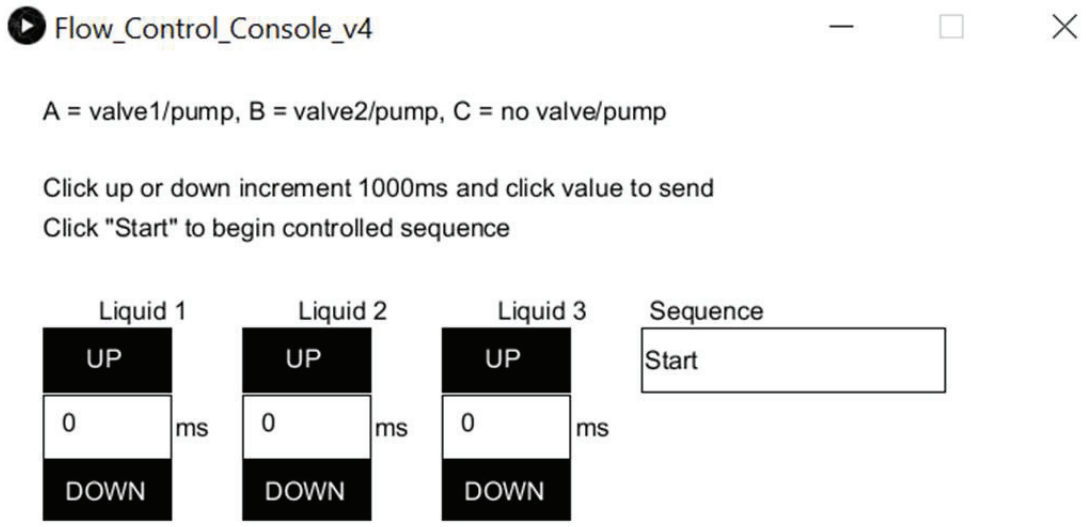

Figure 25. GUI Pop-up Menu

Chapter 6.

Results and Discussion

The testing was performed on the developed equipment to determine the relationship between the volume of fluid and the time taken by it to dispense the required volume into a graduated container. For this purpose, the experimentations were carried out iteratively on three kinds of fluids for the user defined volume fillings in a series of marked vessels. Different volumes of each kind of fluid were taken. We call it run1, run2 and run3 for different amount of time like 4 seconds, 8seconds, 12 seconds and 16 seconds. Table 2 shows data obtained during experiment for volumes of three kinds of liquids vs time. We observed a linear relationship between the volume of the fluids and the time taken to dispense those fluids as represented in Figure 26. The same figure also describes that for each liquid, rate of dispense is same. The system was also checked for its smooth operability. It demonstrated satisfied and accurate working of the prototype dispensing system.

Table 3 shows rate correlation. Figure 27 shows Individual runs for three liquids and describes a strong correlation between liquid dispensed for the same amount of time.

An automated fluid dispensing setup provides with flexibility to the user in terms of time selection. At the same time, the time-based dispensing method reduced the hardware cost of the dispensing system which would otherwise increase if volumetric positive displacement piston pump or time-pressure dispensing techniques were employed.

	Motor speed 100/255 for all tests @ 12V			
	Time	Volume (mL)		
		Run 1	Run 2	Run 3
Start	0	0	0	0
End	4	1.8	1.85	2
Rate		0.450	0.463	0.500
Average Rate	0.471			
Start	0	0	0	0
End	8	3.7	3.75	3.8
Rate		0.463	0.469	0.475
Average Rate	0.469			
Start	0	0	0	0
End	12	5.8	5.6	5.5
Rate		0.483	0.467	0.458
Average Rate	0.469			
Start	0	0	0	0.000
End	16	7.5	7.6	7.6
Rate		0.46875	0.475	0.475
Average Rate	0.473			

Table 2. Data showing volume of liquid vs time

Rate for liquid flow is very consistent for various duration of motor run time.

Rate for liquid flow is very consistent for various duration of motor run time.

Rate Correlation	
4s	0.471
8s	0.469
12s	0.469
16s	0.473

Table 3. Data showing Average rate vs time

Average rate is the rate at which liquid is dispensed for a certain amount of time.

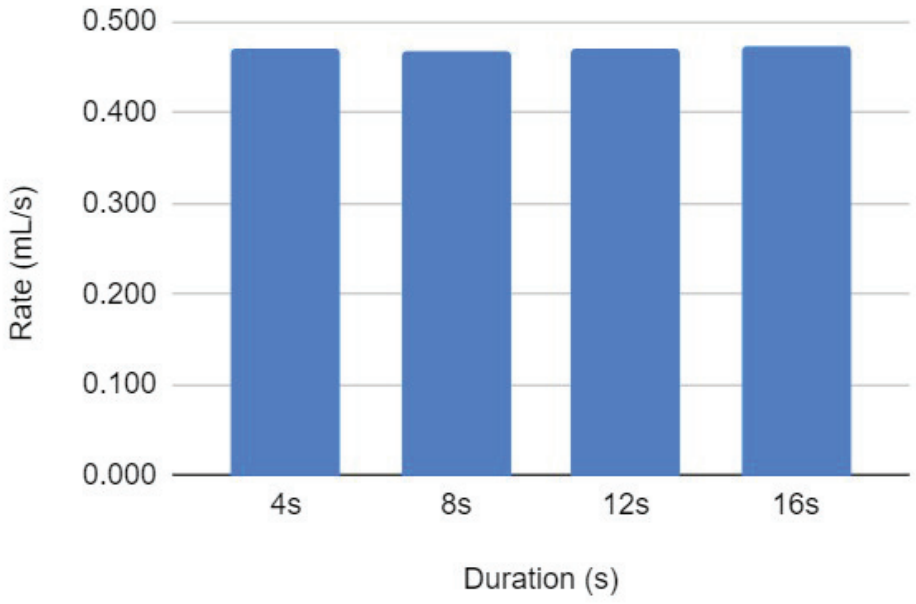

Figure 26. Chart showing rate vs time

Figure 26 shows relation of rate(mL/s) vs time.

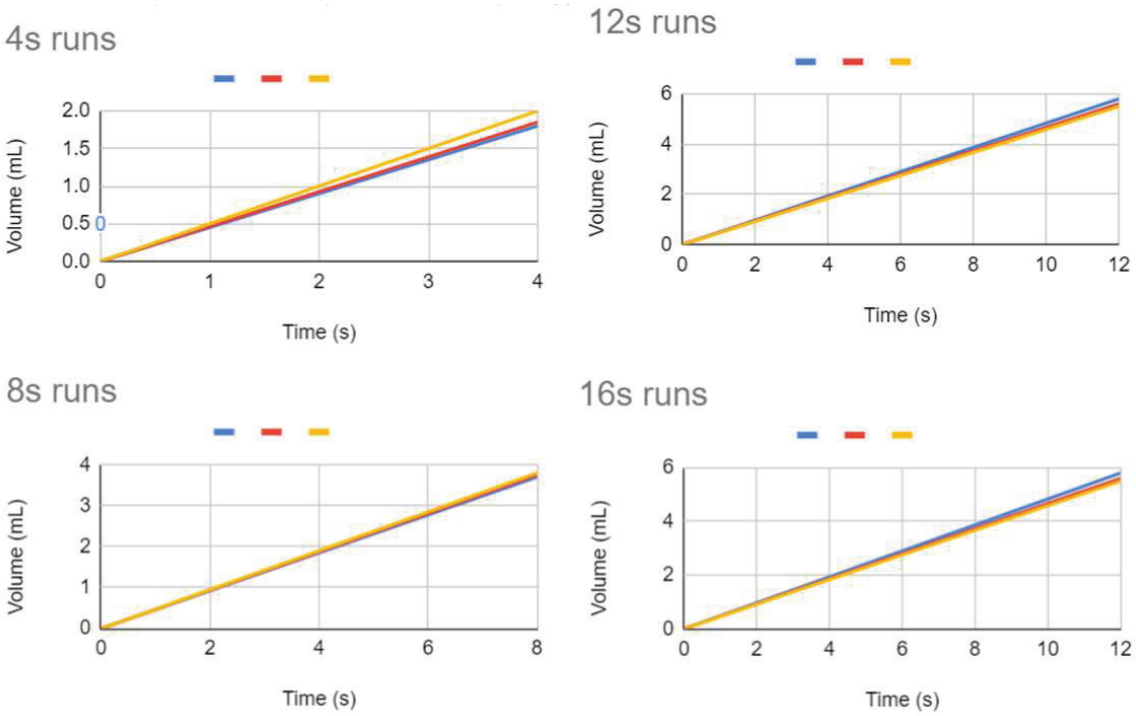

Figure 27. Individual runs for volume vs time

Figure 27 shows Individual runs which describe a strong correlation between liquid dispensed for the same amount of time.

Chapter 7.

Conclusion and Future Work

In this project we can conclude that three liquids of different volumes can be dispensed for different amount of time individually and automatically in a definite sequence after one another by using time-based approach with Arduino uno board. The speed of the motor can be controlled by using PWM technique.

An automated fluid dispensing system was successfully developed in connection to a prospective usage in the pharmaceutical, food and beverage companies and clinical labs for time-based dispensing of different fluids with varying viscosities. It allows its operator to control the working of the system from few buttons on Graphical User Interface. Future modifications include wireless and distant control of such systems by distance enhancement and dispensing lesser fluid volumes ranging below 1 ml. The decrease in pump size and tubing system will allow micro liter fluid dispensing desired for micro level applications. Prospective studies will also emphasize on enhancing system accuracy by removing air bubbles. Fluid flow sensing technique using fluid flow meter can be employed rather than time-based technique and concentration measurement circuit can be incorporated for concentration-based fillings.

References

[1] N. S. M. H. Syed Muhammad Omair, "ResearchGate," [Online]. Available: https://www.researchgate.net/publication/282643162_Fluid_Dispenser_Prototype_A_Time_Based_Approach. [Accessed December 2020].

[2] J. X. Liu, New developments in robotics research, Nova Publishers, 2005.

[3] T. E. R. M. H. W. A. a. E. H. J. Butler, "Chemical Dispensing system and method," 2001.

[4] A. Godschalk Jr Louis, "Device for dispensing measured quantities of liquid, Google Patents," 1968.

[5] Corrosionpedia, 21 July 2014. [Online]. Available: https://www.corrosionpedia.com/definition/527/flow-control. [Accessed December 2020].

[6] P. A. V. V. G. BipinMashilkar, "AUTOMATED BOTTLE FILLING SYSTEM," IRJET e-ISSN: 2395 - 0056 , vol. 03, no. 04, April-2016.

[7] E. A. Solutions, CONTROL VALVE HANDBOOK, Fisher, 2005.

[8] F. R. Hickerson, "Liquid dispensing system". USA Patent 5,044,527, 03 September 1991.

[9] a. A. C. L. R. Ceccarelli, "Pre-measured liquid and powder dispenser with overflow lube". US Patent 5,323,938, 28 June 1994.

[10] a. K. A. H. Awada, "Reusable and accurately pre-measured liquid dispenser". US Patent 5,584,420, 17 December 1996.

[11] T. R. Hanson, "Liquid measuring and dispensing device". 2005.

[12] G. W. Takacs, "Precise volume fluid dispenser". 1996.

[13] J. R. R. J. a. D. E. Keyes, "Time Volumetric Fluid Dispensing Apparatus". 2011.

[14] D. Dixon, "Time pressure dispensing," White papers, Universal Instruments, [Online]. Available: http://www4.uic.com/wcms/WCMS2. nsf/index/Resources_58. html [Accessed December 2020].

[15] [Online]. Available: https://www3.uic.com/wcms/images2.nsf/(GraphicLib)/Time_Pressure_Dispensing.PDF/$File/Time_Pressure_Dispensing.PDF.

[16] P. Swanson, "Improving industrial dispensing with pneumatic dispensing valves," 2007.

[17] [Online]. Available: https://www.elprocus.com/mosfet-as-a-switch-circuit-diagram-free-circuits/.

[18] [Online]. Available: https://www.burkert.co.uk/en/Company-Career/What-s-New/Press/Media/Technical-Reports/Technical-Reports-additional-topics/What-is-a-solenoid-valve-and-how-does-it-work#:~:text=Solenoid%20valve%20function%20involves%20either,coil%2C%20plunger%20and%20sleeve.

Appendix A.

Arduino IDE Code

```
#include <SoftwareSerial.h> //library set up for debugging

SoftwareSerial mySerial(2, 3); // RX, TX (debugging only)

const byte numChars = 60; //array to store incoming commands

char receivedChars[numChars]; //sets array size for serial input

boolean newData = false; //variable to save if data has been received

//digital pin each mosfet is connected to

byte valve1 = 9;

byte valve2 = 10;

byte pump = 11;

void setup() {

 pinMode(valve1, OUTPUT); //sets up digital pins as outputs to drive mosfet controllers

 pinMode(valve2, OUTPUT);

 pinMode(pump, OUTPUT);

 Serial.begin(115200); //initializes serial port to receive commands from the computer

 mySerial.begin(9600); //initializes software serial port. (debugging only)

 Serial.println("<Arduino is ready>");

 mySerial.println("<Arduino is ready>"); //debugging
```

```
}

void loop() {

 recvWithStartEndMarkers(); //function that starts when new data is received with correct start and end markers (< >)

 showNewData(); //displays received data in serial monitor and acts on data

}

void recvWithStartEndMarkers() {

 static boolean recvInProgress = false; //stores if currently reading from serial

 static byte ndx = 0; //serial array read location

 char startMarker = '<'; //sets expected start character for a command

 char endMarker = '>'; //sets expected end character for a command

 char rc; //current serial read character

 while (Serial.available() > 0 && newData == false) { //wait for new serial to be available in the buffer

  rc = Serial.read(); //reads one character at a time

  if (rc == startMarker) { //set that a receive is in progress if a character is the start marker

   recvInProgress = true; //sets to continue reading data

  }

  if (recvInProgress == true) { //add each character to the incoming string as long as its not the end character

   if (rc != endMarker) { //checks to see if end of serial string is reached

    receivedChars[ndx] = rc; //adds current character into input array
```

```
            ndx++; //inrements to next input array location
            if (ndx >= numChars) { //does not increment if its reached the buffer size of for input string
             ndx = numChars - 1;
            }
           }
           else {
            receivedChars[ndx] = '\0'; // this character is necessary in C to terminate any string so it must be added when input is over
            recvInProgress = false; //reset variable values to wait for next command
            ndx = 0; //resets index value for next loop
            newData = true; //sets that new data is available for the next section of code
           }
          }
         }
        }
        void showNewData() {
         if (newData == true) { //once a message is done being received, we tell Arduino we have new data available
          Serial.println(receivedChars); //prints message to serial monitor (debugging only)
          mySerial.println(receivedChars); //(debugging only)
```

```
            processData(); //this command takes the string we just received and parses it to see which switches to turn on

            memset(receivedChars, 0, sizeof receivedChars); //this resets the saved command variable to be empty

            newData = false; //since we have to most recent command, this tells the code to wait for another message

             Serial.print("m"); //debugging

           }

          }

          void processData() {

           int index1 = 0;

           int index2 = 0;

           char subChar[30] = {0}; //allows 8 digit input with \0 terminator for the command input

           while (receivedChars[index1] != '\0') { //read through until end of string

            while (!isDigit(receivedChars[index1]) && receivedChars[index1] != '\0')index1++; //read until next digit

            if (receivedChars[index1 - 1] == 'A' || receivedChars[index1 - 1] == 'a') { //detects if 'a' command for valve1 is sent

             //commands are sent like <A1000> so it checks to see if the letter A

             //was sent then saves the number after this letter

             while (receivedChars[index1] != '\0' && isDigit(receivedChars[index1])) {

              //read while numbers are still incoming (not \0 end and not
```

```
        //another command like B1000 after A1000
        subChar[index2] = receivedChars[index1]; //command array stores current location of serial input array
        index1++; //increment array index location to save the next character
        index2++;
      }
      Serial.print("'A' Command: "); //print once number ends (debugging only)
      Serial.println(subChar); // prints the timefor command to run
      mySerial.print("'A' Command: "); //debugging
      mySerial.println(subChar); //debugging
      index2 = 0; //resets index value to receive a new command
      digitalWrite(valve1, HIGH); //'A' command opens valve1
      analogWrite(pump, 100); //turn on pump at 100/255 speed
      delay(atoi(subChar)); //wait for command(ms) delay
      digitalWrite(valve1, LOW); //close valve
      digitalWrite(pump, LOW); //turns off pump
      memset(subChar, 0, sizeof subChar); //resets variable storing command so a new value can be stored
    }
    //comments are the same for B and C commands so refer back to corresponding line from above
    else if (receivedChars[index1 - 1] == 'B' || receivedChars[index1 - 1] == 'b') {
```

```
 while (receivedChars[index1] != '\0' && isDigit(receivedChars[index1])) {
  subChar[index2] = receivedChars[index1];
  index1++;
  index2++;
 }
 Serial.print("'B' Command: "); //print once number ends (debugging only)
 Serial.println(subChar);
 mySerial.print("'B' Command: "); //debugging
 mySerial.println(subChar); //debugging
 index2 = 0;
 digitalWrite(valve2, HIGH); //'B' command turns on valve2
 analogWrite(pump, 100); //turn on pump at 100/255 speed
 delay(atoi(subChar)); //wait by command(ms) delay
 digitalWrite(valve2, LOW); //close valve
 digitalWrite(pump, LOW); //turn off pump
 memset(subChar, 0, sizeof subChar); //reset command value
}
else if (receivedChars[index1 - 1] == 'C' || receivedChars[index1 - 1] == 'c') {
 while (receivedChars[index1] != '\0' && isDigit(receivedChars[index1])) {
  subChar[index2] = receivedChars[index1];
  index1++;
```

```
    index2++;
   }
   Serial.print("'C' Command: "); //print once number ends (debugging only)
   Serial.println(subChar);
   mySerial.print("'C' Command: "); //debugging
   mySerial.println(subChar); //debugging
   index2 = 0;
   analogWrite(pump, 100); //turn on pump at 100/255 speed, no valves turned on
   delay(atoi(subChar)); //wait by command(ms) delay
   digitalWrite(pump, LOW); //run pump for a little longer then turn off
   memset(subChar, 0, sizeof subChar); //reset command value
  }
  index1++;
 }
}
```

Appendix B.

Processing Code for Graphical User Interface

```
import processing.serial.*;

PFont f; //font object to set up text

Serial myPort;

int sequence = 0; //stores whether code is doing the sequence or not

String sequence_value = "";

Switch valve1; //objects to control each MOSFET device

Switch valve2;

Switch pump1;

Sequence sequence1;

boolean down = false;

boolean down1 = false;

long sequencePreStartTime = millis();

long sequencePostStartTime = millis();

void setup() {

 size(700, 350);

 f = createFont("Arial", 16); //sets up font details for GUI

 valve1 = new Switch(25, 170); //creates GUI object for valve1

 valve2 = new Switch(150, 170); //creates GUI object for valve2
```

```
 pump1 = new Switch(275, 170); //creates GUI object for pump

 sequence1 = new Sequence(400, 170);

 myPort = new Serial(this, Serial.list()[0], 115200); //initializes serial on correct
com port

 printArray(Serial.list()); //prints all connected serial devices

}

void draw() {

 background(255); //sets GUI background to white

 textFont(f); //// Sets text font to "f" created before

 fill(0); //makes text black

 text("A = valve1/pump, B = valve2/pump, " +

  "C = no valve/pump\n\n"+

  "Click up or down increment 1000ms and click value to send\n" +

  "Click \"Start\" to begin controlled sequence", 25, 40); //prints this text at
coordinates 25, 40 of GUI

 text("Liquid 1", 60, 165); //prints switch identifier above each control

 text("Liquid 2", 185, 165);

 text("Liquid 3", 310, 165);

 text("Sequence", 405, 165);

 text("ms", 108, 238);

 text("ms", 233, 238);

 text("ms", 359, 238);
```

```
//text("\u03BCs", 484, 238);


//////////////////valve 1 controls//////////////////


if (valve1.onUp()) { //check if mouse is hovering over increment time box for valve1
 valve1.display(150, 0, 255); //changes highlight over box selected
 if (down && sequence == 0) { //checks if mouse was pressed down
  valve1.updateValue(true); // "1" indicates increment time value
  down = false; //resets variable until mouse pressed down again
 }
} else if (valve1.onDown()) { //check if mouse is hovering over decrement time box for valve1
 valve1.display(0, 150, 255); //correctly highlights boxes based on where the mouse is
 if (down && sequence == 0) { //check if mouse has been clicked
  valve1.updateValue(false); //"0" indicates decrement time value
  down = false; //reset down variable to only change once per click
 }
} else if (valve1.onData()) { //A send data box
 valve1.display(0, 0, 205); //correctly highlights boxes based on where the mouse is
 if (down && sequence == 0) { //check if mouse has been clicked
```

```
          valve1.sendData('a'); //sends data for valve1 through serial port to Arduino
          down = false; //reset down variable to only change once per click
         }
        } else {
         valve1.display(0, 0, 255); //correctly highlights boxes based on where the
mouse is
         //this is the default if mouse is not over any boxes
        }
        ////////////////valve 2 control////////////////////
        if (valve2.onUp()) {
         valve2.display(150, 0, 255);
         if (down && sequence == 0) {
          valve2.updateValue(true);
          down = false;
         }
        } else if (valve2.onDown()) {
         valve2.display(0, 150, 255);
         if (down && sequence == 0) {
          valve2.updateValue(false);
          down = false;
         }
```

```
} else if (valve2.onData()) {
 valve2.display(0, 0, 205);
 if (down && sequence == 0) {
  valve2.sendData('b');
  down = false;
 }
} else {
 valve2.display(0, 0, 255);
}
//////////////pump 1 controls//////////////////////
if (pump1.onUp()) { //pump increment box
 pump1.display(150, 0, 255);
 if (down && sequence == 0) {
  pump1.updateValue(true);
  down = false;
 }
} else if (pump1.onDown()) { //pump decrement box
 pump1.display(0, 150, 255);
 if (down && sequence == 0) {
  pump1.updateValue(false);
  down = false;
```

```
 }
} else if (pump1.onData()) { //pump send data box
 pump1.display(0, 0, 205);
 if (down && sequence == 0) {
  pump1.sendData('c');
  down = false;
 }
} else {
 pump1.display(0, 0, 255); //display default pump controls
}
//////////////sequence controls//////////////////////
if (sequence1.onData()) { //pump send data box
 sequence1.display(0, 0, 205, sequence_value);
 if (sequence1.onData() && down && sequence == 0) {
  sequence = 1;
  sequencePreStartTime = millis();//saves time that first section of
                  //sequence was started at
  down = false;
  sequence1.writeChannel("a", 1000);
  //sequence1.sendDataPre();//send commands for before manual pause
 } else if (sequence1.onData() && down && sequence == 2) {
```

```
  sequence = 3;
  sequence1.writeChannel("a", 10000);
  sequencePostStartTime = millis(); //saves time that second section of
                  //sequence was started at
  down = false;
  //sequence1.sendDataPost();//send commands for after manual pause
  //sequence = 0;
 }
} else {
 sequence1.display(0, 0, 255, sequence_value);//display default pump controls
}
if (sequence == 1 && millis() - sequencePreStartTime > 1000) {
 sequence = 2;
}
if (sequence == 5 && millis() - sequencePostStartTime > 20000) {
 sequence = 0;
}
else if (sequence == 4 && millis() - sequencePostStartTime > 15000) {
 sequence = 5;
 sequence1.writeChannel("a", 5000);
}
```

```
 else if (sequence == 3 && millis() - sequencePostStartTime > 10000) {
  sequence = 4;
  sequence1.writeChannel("b", 5000);
 }
 if (sequence == 0) sequence_value = "Start";
 if (sequence == 1) sequence_value = "Filling liquid 1";
 if (sequence == 2) sequence_value = "Waiting for liquid 3 input";
 if (sequence == 3) sequence_value = "Filling liquid 1";
 if (sequence == 4) sequence_value = "Filling liquid 2";
 if (sequence == 5) sequence_value = "Filling liquid 1";
}
void mousePressed() { //this function automatically runs in Processing if mouse has been pressed
 down = true; //if the mouse was pressed, we set "down" = true to update the time value then set it back
 //to false until the mouse has been clicked again
}
void mouseClicked() { //unused currently
 down1 = true;
}
class Sequence { //sequence object so we don't have to change the properties of valve1,valve2, and pump separately since
```

```
//they function very similarly

color c; //variables for contorlling the sequenced button

int xpos;

int ypos;

int aDispense;

int bDispense;

int cDispense;

int xwidth = 190;

int ywidth = 40;

String value = "Start";

// The Constructor is defined with arguments.

Sequence(int tempXpos, int tempYpos) { //this runs once when object is created to set up the x,y coordinates on the GUI

 //this runs once when object is created to set up the x,y coordinates on the GUI

 xpos = tempXpos;

 ypos = tempYpos;

}

void display(color supColor, color downColor, color dataColor, String sequence_value) { //displays buttons with correct colors based on mouse location

 //fill(upColor); //sets fill color of up button

 //rect(xpos, ypos, xwidth, ywidth); //draws up button

 //fill(downColor); //sets fill color of down button
```

```
  //rect(xpos, ypos+80, xwidth, ywidth); //draws down button
  //fill(255); //draws box that holds time value
  stroke(0);
  fill(dataColor);
  rect(xpos, ypos, xwidth, ywidth); //text box
  fill(0);
  text(sequence_value, xpos+2, ypos+26);
  fill(255);
 }
 boolean onData() {//checks to see if mouse is on time value button and changes its color to highlight
  if ((mouseX > xpos && mouseX < xpos+xwidth)
   && (mouseY > ypos && mouseY < ypos+ywidth)) {
   return true;
  } else {
   return false;
  }
 }
 void writeChannel(String channel, int timeDelay) {
  String result = "";
  result = "<"+channel+timeDelay+">"; //"<...>" is protocol the Arduino expects so it knows when a command starts and ends
```

```
println(result);
myPort.write(result); //writes the string after it has been properly encoded with
<>
//delay(timeDelay);
}
void sendDataPre() {//sends commands for control before pause for manual
entry
println("Filling clear water");
writeChannel("a", 1000); //fill line with buffer
//writeChannel("c", 1000); // fill with cytokine
println("Waiting for manual injection");
}
void sendDataPost() {//sends commands for control after pause for manual entry
println("Filling clear water");
writeChannel("a", 10000); //use buffer to move cytokine to QCM
println("Filling yellow water");
writeChannel("b", 5000); // fill with urea
println("Filling clear water");
writeChannel("a", 5000); //fill line with buffer
}
}
```

```
class Switch { //switch object so we don't have to change the properties of valve1,valve2, and pump separately since
 //they function very similarly
 color c;
 int xpos;
 int ypos;
 int xwidth = 80;
 int ywidth = 40;
 int value = 0;

 // The Constructor is defined with arguments.
 Switch(int tempXpos, int tempYpos) { //this runs once when object is created to set up the x,y coordinates on the GUI
  xpos = tempXpos;
  ypos = tempYpos;
 }
 void display(color upColor, color downColor, color dataColor) { //displays buttons with correct colors based on mouse location
  fill(upColor); //sets fill color of up button
  rect(xpos, ypos, xwidth, ywidth); //draws up button
  fill(downColor); //sets fill color of down button
  rect(xpos, ypos+80, xwidth, ywidth); //draws down button
```

```
  fill(255); //draws box that holds time value
  stroke(0);
  fill(dataColor);
  rect(xpos, ypos+42, xwidth, ywidth); //text box
  fill(0);
  text(value, xpos+12, ypos+65);
  fill(255);
  text("UP", xpos+27, ypos+25);
  text("DOWN", xpos+14, ypos+2*ywidth+28);
 }
 void updateValue(boolean val) { // runs when increment or decrement buttons are clicked to update value
  if (val == true) {
   value+=1000;
  } else if (val == false && value >=100) {
   value-=1000;
  }
 }
 boolean onUp() { //checks to see if mouse is on up button and changes its color to highlight
  if ((mouseX > xpos && mouseX < xpos+xwidth)
   && (mouseY > ypos && mouseY < ypos+ywidth)) {
```

```
 return true;
 } else {
 return false;
 }
}
boolean onDown() {//checks to see if mouse is on down button and changes its color to highlight
 if ((mouseX > xpos && mouseX < xpos+xwidth)
 && (mouseY > ypos+80 && mouseY < ypos+80+ywidth)) {
 return true;
 } else {
 return false;
 }
}
boolean onData() {//checks to see if mouse is on time value button and changes its color to highlight
 if ((mouseX > xpos && mouseX < xpos+xwidth)
 && (mouseY > ypos+ywidth && mouseY < ypos+2*ywidth)) {
 return true;
 } else {
 return false;
 }
```

```
}

void sendData(char channel) { //sends the correct time value over the serial port with an identifier character

 //so the Arduino knows which valves to activate

 String result = "<"+channel+value+">"; //"<...>" is protocol the Arduino expects so it knows when a command starts and ends

 myPort.write(result); //writes the string after it has been properly encoded with <>

} }
```