Pumpy – Design and Manufacture of a Syringe Pump

Student: Daniel Woolsey

Supervisor: Dr Nick Zakhleniuk

Second Assessor: Dr Vishuu Mohan

BSc Computer Science

University of Essex

2021

Acknowledgements

I would like to thank my family and friends for their invaluable support throughout this difficult year.

A big thank you to my supervisor Dr Nick Zakhleniuk whose weekly meetings and consistent guidance helped bring this project to where it is now.

Finally, I'd like to thank the open source community, who have provided resources for anyone to learn and utilise new and interesting technology.

Abstract

Due to COVID-19, medical equipment and PPE have been in short supply and high demand. For the first time hospitals relied on individuals to 3D print face masks to handle the growing supply concerns they faced. This has the potential to be applied to other forms of medical equipment such as syringe pumps, which provide continuous, hands-free drug delivery and can cost more than £1500 per unit.

In this project we designed, 3D printed and built 2 medical-use syringe pumps based on Joshua Pearce's Open Source Syringe Pump. We programmed both a touchscreen interface for a Raspberry Pi computer and an Android application for remote pump operation via Bluetooth.

We have demonstrated that medical-use syringe pumps can be built for less than 10% of their market value. This open-source approach to manufacturing equipment can allow hospitals to handle supply issues internally and can help to provide inexpensive medical equipment to poor areas with limited modern technology.

Table of Contents

1. Introduction

Medical equipment is always in demand yet remains in a status quo – startups with new ideas must work with large corporations to make them a reality. This is due to the difficulty of creating equipment that meets with the approval of regulatory boards worldwide. Large corporations can monopolize specific parts of the market and charge extortionate premiums to the healthcare system that needs it. For example, Johnson and Johnson (JNJ) acquired Auris Health for their Monarch endoscopy robotic system for $3.4 billion [1]. Such a device will become commonplace in hospitals over the next decade but was only able to reach FDA approval after the JNJ acquisition.

Modern equipment like Monarch is a true vision of the future but ignores a huge problem the medical equipment sector faces – the mismatch problem. Outlined in a WHO report from 2010, it explains that the medical equipment industry invests more in R&D for equipment that will only be used in 1st world countries rather than creating accessible, affordable and appropriate equipment for use in low-resource settings [2]. The report hypothesizes that a countervailing force must focus the medical device market towards public health considerations at all stages of the device lifecycle. I believe that open source hardware can be that force.

During the COVID-19 pandemic we saw waves of medical equipment shortages in hospitals worldwide. This led to several open source initiatives allowing individuals to help provide PPE to medical staff such as 3DCrowd [3] and the NIH 3D Print Exchange [4]. Coupled with research into the adoption of open source hardware for diagnostic tools and ventilators [5], we saw hospitals begin to produce their own reusable PPE in-house [6] to handle the supply chain issues they faced. Having seen the benefits of 3D printing and open source hardware in hospitals, there is a chance that open source can be a new norm in the medical industry.

The open source ecosystem is, by nature, decentralized and democratic [7], which goes against many business practices where ideas are patented, and their inner workings are hidden trade secrets. The switch to open source brings product development to the people, with democratic decision making leading to rapid innovation. We've seen this with software like Linux, Android, and Arduino – all of them open source projects used in commercial products and provide a better experience for both developers and consumers.

When applying the open source methodology to more complex medical equipment than PPE we are faced with a big issue – regulation. Medical authorities such as the FDA provide resources for manufacturers to follow when attempting to get a device approved for use [8]. Applying this approval process to open source where change and innovation are prioritized is a challenge, since any revisions will have to be re-approved. To overcome this, any open source medical device requires a medically approved baseline with changes approved by committee in regular intervals.

In this paper I want to analyze and build upon an already existing open source baseline for a complex medical device – a syringe pump. Syringe Pumps (also known as Infusion Pumps or

Syringe Drivers) are motorized linear actuators that deliver medicine to a patient at a constant rate over a 24-hour period [9]. They are commonly used for patients who have trouble taking medication orally or are in end of life care.

The main goals of this paper are as follows:

- To analyze, build and test a current open source Syringe Pump
- Modify and improve the hardware
- Design an electronics system for user friendly operation
- Create a mobile application for remote pump operation
- Explore the viability of a cheap and open source medical Syringe Pump solution

In Section 2 I explore the background of the main areas the project concerns: Syringe Pumps (both commercial and open source), Open Source technology, 3D Printing and the medical uses of Syringe Pumps along with their issues. Section 3 shows my technical documentation for the project hosted on Gitlab. Section 4 explains how I approached each stage of hardware and why I chose the solutions I did. Section 5 describes my approach to software – the chosen programming language, libraries I decided upon and how they were utilized. Section 6 details the process of my work from building the first prototype to creating a final solution. Finally, in Section 7 I discuss the results and performance of my Syringe Pump system, the project planning and development strategies I used, the benefits and weaknesses of open source development and the future viability of open source medical devices.

2. Literature Review

This project builds upon previous Syringe Pump projects with a focus on using open source hardware and modern manufacturing processes to bring cheaper medical equipment to the healthcare industry. Below I discuss the main aspects of my project: Syringe Pumps, Open Source, 3D printing and the history of medical Syringe Pumps.

2a. Syringe Pumps

Syringe Pumps are used in a variety of contexts. In medicine, they are used to deliver medicine at a constant rate under the skin, helping to manage symptoms such as pain, nausea, and vomiting [9]. In laboratories, they are used to automate routine life science experiments that are taxing to perform by hand [10] or for microfluidics experiments where precise flow rate control is necessary [11]. Syringe Pump prices range between £300 and £2800 [12] for what appears to be a simple piece of equipment. Medical equipment businesses can charge a high price for 2 reasons:

- Cost of getting a medical device approved by regulatory boards such as the FDA and MHRA
- Lack of market competition

For laboratory work, this price point was too high to warrant buying many of them. To solve this problem, scientists have turned to open source hardware to build extensible Syringe Pumps tailored to the work being done

Back in 2014, a group from Michigan Tech University released the Open-Source Syringe Pump Library (OSSP). They provided a proof of concept for a 3D printed Syringe Pump that costs £70 that performs better than most commercial Syringe Pumps in terms of speed and force but is not as accurate as more expensive models [12]. The 3D printed files, bill of materials and build guide were made available for users to build and modify on Appropedia [13] and has been downloaded almost 9000 times [14]. The pump itself is a proof of concept, so members of the open source community have designed modifications to improve on the original design. One such project is the Lynch Pump Modifications that provide a new carriage and syringe holder that are vastly superior to the original designs [15].

There are 2 interesting papers that have built upon the OSSP for specialized use cases in laboratories: The Programmable Dual Syringe Pump (PDSP) and the Poseidon Syringe Pump and microscope.

The PDSP fuses together 2 OSSPs to automate laboratory work in life science applications. More specifically, infected blood cells are infused from one pump to another via a homogenizer. The homogenizer breaks down these cells into their liquid protein constituents with each infusion through it. This process is repeatedly done by hand and allows scientists to analyze the structure of blood cells. Doing this process by hand led to user-dependent variation in the results and damage to the equipment as pressure was applied unevenly [10]. Using the OSSP with a Raspberry Pi controller and a touchscreen interface, they designed a GUI in Python with Tkinter. This system is fully customizable for up to 16 pumps working concurrently and can be used in other areas of lab work.

The Poseidon Syringe Pump is a modified OSSP that incorporates a microscope to perform experiments on microfluidic devices where liquid is infused into systems as small as 10 microns wide [11]. These experiments require a flexible device that can provide precise flow rate control and allow the user to customize the operation software for their use case. Commercial syringe pumps are inadequate for this task since their software is closed source and limited. The pump itself is driven by an Arduino microcontroller that receives software calls from a Raspberry Pi with a touchscreen interface. The GUI is written in Python and uses the PyQt library.

Systems such as the 2 described show that there is a need for a cheap and customizable alternative to commercial syringe pumps. There exist some trivial projects that incorporate syringe pumps into robotic mechanisms such as the 8ARM Precision Pump. This mechatronics system acts as a robot bartender capable of mixing drinks to a high degree of accuracy, around 1 +- 0.05 cL. Although this is an interesting concept there are many environmental variables that the system cannot account for [16].

2b. Open Source

Open Source is a blanket term for many different criteria that can be applied to anything people can modify and share because its design is publicly accessible. These criteria include free redistribution, access to source code and allowance for modification and derivative works [17]. Use of the term open source began in software around 1990, where development shifted from large scale and rigid waterfall models to iterative development styles like Agile. This change brought about more democratic software development, with project such as Linux and GitHub being developed by hundreds of individuals who weren’t motivated by money, but by seeing what they could build together. This idea became a philosophy for many programmers and led to the creation of systems such as Arduino [18], Android [19] and Apache [20] which continue to grow to this day.

Economists would have you believe that innovation comes solely from competition within markets: businesses competing to gain the largest profits by creating the best product. This point is antithetical to the philosophy of open-source development [17]. Open-source still allows for businesses to profit from open-source work, depending on the license type a project falls under, but the driving factor is innovation first and foremost – innovation coming from any individual/party with an interest in the research. With the internet allowing information to spread faster than ever before, open-source has the means to fully democratise information and prove that Isaac Newton’s famous quote: “If I have seen further it is by standing on the shoulders of giants” is as true today as it was in his day. [21]

For the context of our project, we are going to look at Free and Open Source Hardware (FOSH). This has become a popular topic in the last decade, with 3D printing as a driving force to find cost-effective, reproducible solutions to expensive commercial equipment. Centralized repositories containing FOSH projects include the Open Source Repository [22], the NIH 3D Print Exchange [4] and journals such as *HardwareX* [23] and the *Journal of Open Hardware* [24]. A boundary to progress for FOSH projects comes from how it is licensed. There are 2 main classes of open source licenses:

- Copyleft – Users can redistribute the original work or any derivative work so long as it uses the same license as the original work
- Permissive – Users can redistribute the original work or any derivative work under any license they choose

Copyleft licenses preserve the original works license through any derivative work – that is work that modifies the original. On the other hand, Permissive licenses allow redistribution of work in any setting: open source or commercial. Permissive licenses allow for large scale implementation of derivative work since the project can be turned into a business while still preserving its open source roots. The best example of this is the Android operating system, allowing users to distribute applications under any license while the OS remains open source.

FOSH requires a self-sustaining community to provide new points of innovation and collaboration. We see this with websites such as Thingiverse – a huge repository of 3D printing projects and Just One Giant Lab (JOGL) – a place where volunteers can get involved in current open source projects. Individuals such as Linus Torvalds and Richard Stallman have been pivotal in the emergence of open source, and for FOSH one such individual is Joshua Pearce. He works at Michigan Tech and has worked in the FOSH community for over a decade. His work includes the Open Source Syringe Pump Library [12] and the RepRapable Recyclebot [25] – a system to turn household plastics into printable 3D filament. He has also written 2 books explaining open source to the layman through DIY projects and many articles explaining the benefits of adopting FOSH in scientific fields such as nanotechnology [26]. His research has been cited over 15,000 times and has inspired many derivative works such as this project. Without him the FOSH community and my own project would be very different.

2c. 3D Printing

3D Printing as we know it today began in 2005 at the University of Bath with the RepRap project. It intended to create a cheap 3D printer that could print its own parts, eventually becoming a self-replicating machine. Their first machine cost £430 to produce which then printed parts for the second machine, costing £300. In the first year and a half more than 2500 RepRap machines were built [27]. Before this, 3D printers would cost over £25,000 and used a technique known as Fused Deposition Modelling (FDM) whose patent wouldn't expire until 2009 [28]. This patent, along with several others filed by Stratasys Inc, allowed the company to monopolize the 3D printing market, charging extortionate amounts for their products. The expiration of the FDM patent brought about a new age for 3D printing, with companies such as Ultimaker and Creality bring the cost of such printers down to as low as £200.

FDM is how the majority of modern 3D printers work: they deposit filaments of thermal plastics along a pre-defined path to create an object. The most used plastics are PLA, ABS and TPU, each with their own benefits and drawbacks. There are 2 configurations of 3D printer currently found on the market:

- Cartesian Printers
    - 3 axis system of rails and belts that can move the print head or print bed in the X, Y and Z direction one at a time
    - Movement paths are done one axis at a time using cartesian equations (x and y terms only)
    - Moving parts are heavy so print speed tends to be slower (around 60 – 80 mm/s)
    - Such printers include the Creality Ender printers and MakerBot's Ultimaker printers
- Delta Printers
    - 3 independent arms on rails that move just the print head, the print bed is motionless
    - Movement paths are calculated using trigonometric functions between the arms for multi-axis movement
    - Circular print bed allows for a smaller form factor
    - Lighter print head, allowing for faster print speed (around 70 – 120 mm/s)
    - Example printers include the Anet A4 and the FLSUN Q5

To make use of a 3D printer, you need a 3D model file (usually as a '.stl' or '.obj') and some slicing software (such as Cura or PrusaSlicer). 3D models can be built yourself using a program like Blender or Fusion 360 or found on sites like Thingiverse that host thousands of models. The slicer will take your model file and printer configuration to generate a '.gcode' file. This file is a list of hardware instructions telling your printer when to heat up, where to move and when to extrude plastic. Slicers contain a wealth of configurable settings to help calibrate your printer – a process required on most new printers.

With the 3D printing market valued at £9.7 billion in 2020 with an average growth rate of 14.6% annually [29], the future impact of this technology on the environment must be considered. Waste plastics currently account for 5 million tons of waste every year in the UK alone [30] and a new wave of plastic manufacturing could add to that. Fortunately, one of the most used 3D printing plastics, PLA, is biologically sourced from fermented corn and can biodegrade in industrial composting procedures. These conditions require a constant temperature of 40-50 degrees for between 47 to 90 days, at which point PLA breaks down into water and carbon dioxide [31].

There are also upcycling initiatives in the 3D printing community using RepRap Recyclebots [25]. These devices allow users to take household plastics such as milk, water and detergent bottles, shred them and extrude them into usable 3D printer filament with a diameter margin of error of <5%. This brings the cost of 1kg of filament from £20 to less than 2p. The device itself costs around £540 to make, with the return on investment being a big adoption factor.

The future of 3D printing hinges on the community of makers working together to show the benefits of this technology to consumers. Thingiverse was created in 2008 alongside the RepRap project so users could make and share designs, and currently hosts over 2 million models. The website Reddit has several friendly communities dedicated to 3D printing such as r/fixmyprint, r/functionalprint and r/3dprinting along with communities for all common brands of printer. Having bought an Ender 5 Pro for this project, the maker community has been an excellent resource for learning what these machines can do.

2d. History of Medical Syringe Pumps

When medical syringe pumps were first adopted in the 1980's, there was a single company who manufactured and distributed them – Graseby Medical. [32] Other companies attempted to enter the market such as Baxter International, but never gained wide adoption since Graseby beat them to market. Problems then arose in 1994 when hazard notices from the British Medical Devices Association (MDA) came out regarding the confusion of 2 Graseby pump models [33]. The MS16 model infused at millimeters per hour, whereas the MS26 infused at millimeters per 24 hours. The similarity in design meant medical staff would occasionally confuse the 2 pumps, leading to a rise in inappropriate infusion rate errors and patient deaths. The MDA's notices required users to use 'enhanced labelling' to ensure the 2 types of pump were not confused. At the time, few alternatives existed on the market and the cost of new equipment and retraining staff was too high.

These notices did very little to mitigate the initial problem, and in 2010 a report from the National Patient Safety Agency (NPSA) explained the need to remove these problematic systems from use [33]. It reported that between 1st January 2005 and 30th June 2010 there were 175 incidents involving Graesby syringe pumps, 8 of which resulted in patient deaths. These incidents were caused by 'incorrect rate setting' due to human error and 'unexplained fast infusion' caused by device failure. This report highlighted the need for new equipment and better staff training: a need that had been ignored over 15 years prior.

Following this report, a transition plan was put in place for the next 5 years. This would involve slowly phasing out Graesby syringe pumps and retraining staff on new equipment from McKinley Medical that infused in mL per hour. This process had to be a balance between the cost of replacement programs and increments in patient safety afforded by new technology. Phasing out the older syringe pumps had to be slow to ensure patients who required the equipment were able to continue their treatment. The NHS Purchasing and Supply Agency knew this back in December 2008 but were unwilling to make the change, stating: “currently there are safer alternatives to the MS26/MS16 pumps on the UK market… adopting safer syringe drivers would require modification of established working protocols and practices in UK palliative care, and a significant amount of retraining for staff”. This lack of action led to 4 deaths in 2009 caused by incorrect use of the Graesby pumps.

A report in November 2018 from the Department of Health and Social Care reviewed the action taken following the NPSAs 2010 report [34]. It showed that action had slowed by 2013, with a further NPSA report reinforcing the need to switch from Graesby pumps by 2015. They reported a further 23 incidents of harm involving these pumps between 2011 and 2012, although none were fatal. Reports from the media eventually led to the full removal of these pumps from use.

In hindsight we can see that the NPSA were aware of the problem for 15 years before action was properly taken, the process of adopting new equipment was one they were unwilling to make at the time. This was due to several reasons: lack of market competition, cost of equipment and retraining, balancing the use of potentially dangerous equipment with its ability to save lives. They also believed the initial problem could be fixed by solving the human error, a solution that didn’t work. This shows us that giving greater control to the hardware and minimizing human interaction could be a safer alternative, although it puts the manufacturer at greater risk when problems arise.

From this we can predict how an open source medical syringe pump could be implemented for use across the world and the important factors to consider: routine assessment of device safety, transition plan for both phasing the device in and out, constant analysis of the devices obsolescence. These are all iterative processes, plans that will change over time, and are suited to an open source approach. The real issue with bringing open source systems into the medical industry is regulation – how can we ensure consistent device quality when we aren’t manufacturing each device ourselves. One such solution is found with the charity Glia [35]. They have designed and built open source medical equipment such as stethoscopes and tourniquets, with their stethoscope becoming the first open source medical device to be clinically validated and widely available [36]. The work they’re doing shows that open source alternatives to medical equipment are viable and that over time, an open source approach to medical equipment and regulation could lead to safer equipment use.

3. Technical Documentation

This covers the series of files stored on the Gitlab repository for this project. The repository contains the files and build guides for both versions of the Syringe Pump, pictures, and schematics of the electronics setup along with documentation for the Pump operation code, GUI and Android application.

This documentation is for users who wish to replicate and build upon the current state of the Pumpy Syringe Pump project without delving too deep into the academic side of the project.

Below I describe the table of contents for the technical documentation along with the link to it in the repository:

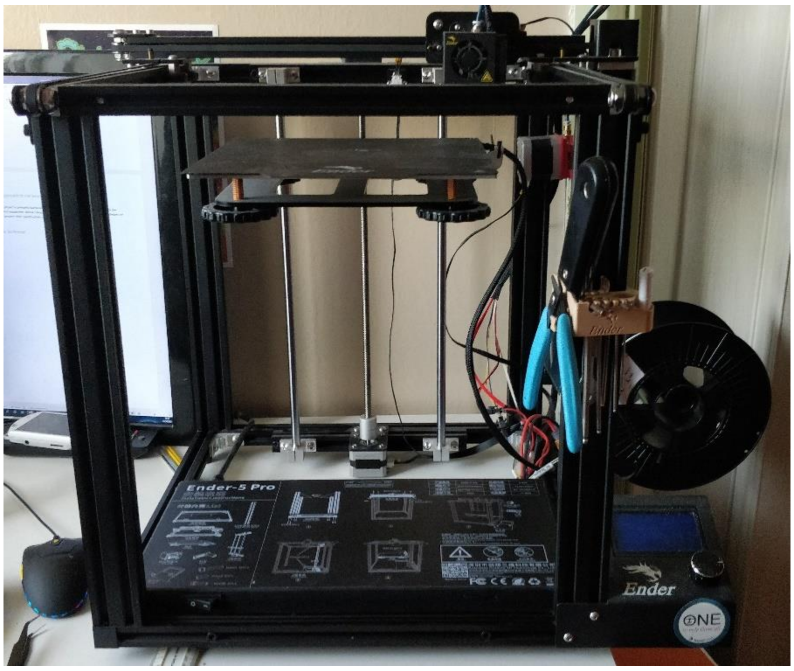
https://cseegit.essex.ac.uk/ce301_2020/ce301_woolsey_daniel.git

TABLE OF CONTENTS

4. Approach to Hardware

This project is primarily hardware focused and choices have had to be made to pick the correct equipment. Below I describe the electronics and syringe pump hardware chosen for this project: their specifications and the reasoning behind choosing them.

4a. 3D Printer



For a 3D Printer I went with the Creality Ender 5 Pro: a cartesian gantry consumer grade printer. It makes use of FDM (Fused Deposition Modelling) to extrude a variety of plastics onto the heated bed. This printer retails at a cost of £294.64 from the manufacturer but can be found cheaper at other online vendors [37].

I chose this printer over other similar models for several reasons:

- Upgraded motherboard for a quiet and more precise printing performance
- Large build plate (220x220x300mm) for printing bigger models
- Bowden tubing to accommodate printing different plastics such as flexible filament

All the parts for both syringe pumps are printed using 1.75mm diameter PLA filament from 3DJake and Geeetech. I chose PLA due to its wide availability, tensile strength, and biodegradability [27]. The cost for 1kg of generic brand PLA is between £19 and £25 from Amazon.

4b. Raspberry Pi

The microcontroller is the most important component of this project: it handles all aspects of pump operation and control. Commercial pumps make use of bespoke embedded software chips from companies such as CME Medical. Other Open Source pumps make use of 3D printer driver boards such as RAMPS or Arduino [11]. The biggest issue with these is their minimal operating systems that provide the bare minimum of utilities for a developer.

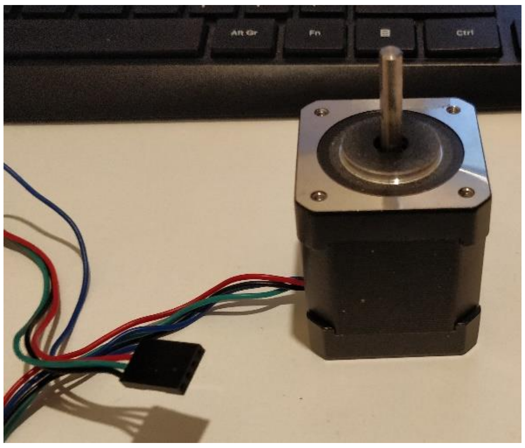
With that in mind I had to decide on a microcontroller that fit the specifications of my project:

- Handles touchscreen display for easy user interaction
- Large amount of digital I/O for required peripherals and extensibility
- OS that can run a programming language with libraries for a GUI
- Communication with mobile devices either by WiFi or Bluetooth
- IoT (Internet of Things) compliant for future updates

The Raspberry Pi fits all my requirements for a microcontroller: it runs a flavor of Linux called Raspbian with great developer support, 40 GPIO (general purpose input/output) pins for external peripherals, HDMI output for a video display, WiFi and Bluetooth built in for communication and external updates.

For a microcontroller this is an expensive option at £35 from eBay, whereas an Arduino based microcontroller (Arduino Uno) costs around £5 from eBay. The extra features and Linux-based OS we get from the Raspberry Pi makes it much easier to build the software we want.

4c. NEMA 17 Motor



Stepper motors can be bought with various specifications: voltages up to 24V, current between 0.2A and 2A, and face plate sizes from 0.8 inches up to 4.2 inches. Previous open source pumps have made use of both NEMA 11 and NEMA 17 motors [12] and I chose to use a NEMA 17 (17 meaning a 1.7-inch faceplate). They are commonly found in 3D printers and CNC machines and require a motor driver for the Raspberry Pi to run them.

The version I chose runs on 12V at 2A and can handle up to 24V but that isn't necessary for our application – pushing a syringe with liquid inside doesn't require a force that large and would create unnecessary vibrations during operation.

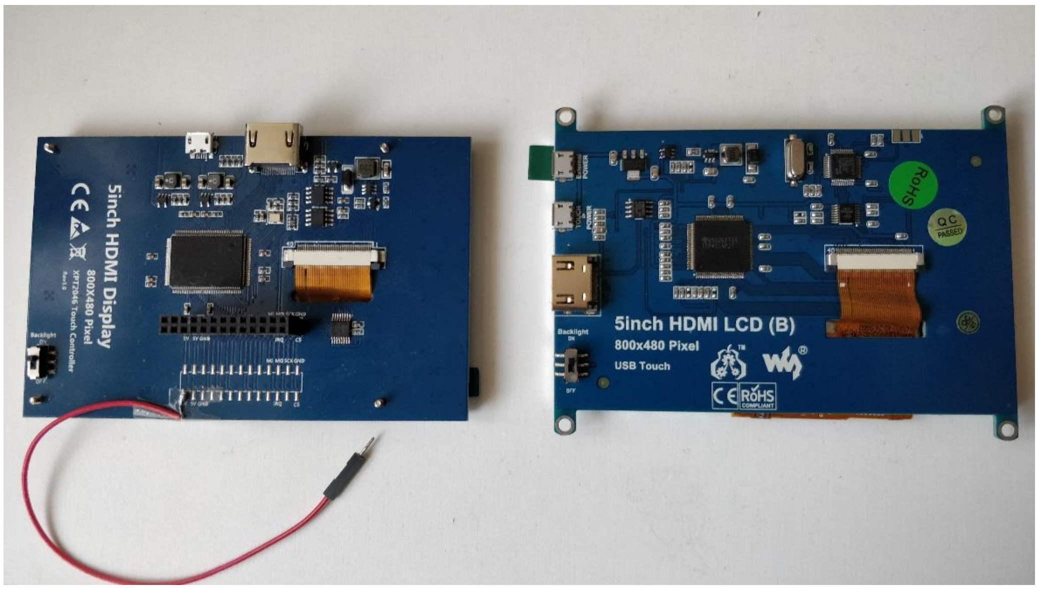
Smaller versions of the motor could be used to reduce the build size if a user wanted but would involve modifying the current 3D printed files that are designed with a NEMA 17 motor in mind.

4d. A4988 Motor Driver

The A4988 is the most used driver for stepper motors and can be found in most 3D printers. Its prevalence means it's cheaply available at roughly £3 from eBay. I could have gone with a DRV8825 motor driver which is built to handle higher voltage and current, but the NEMA 17 motor I'm using can handle 24V at 2A which is well within the tolerance of the A4988. This also has a current limiting potentiometer on the board which is used to prevent damage being done to the motor by the power supply. Since our power supply matches the specification of the motor, we don't need to worry about adjusting it.

It also has 5 micro-stepping modes ranging from 200 steps per revolution up to 3200. There is negligible difference in positional accuracy of the stepper motor as we increase our steps per revolution, but we do benefit from reduced vibration [38]. For our Syringe Pump it is likely to be operating for up to 24 hours continuously, so a greater number of steps per revolution can provide smaller dosages more regularly for greater control of liquid infused.

4e. Touchscreen



When choosing a touchscreen display for the Raspberry Pi I had to decide on the best input method from the following options:

- Ribbon cable directly inserted into the Raspberry Pi
- GPIO and HDMI inserted into the board
- USB and HDMI inserted into the board

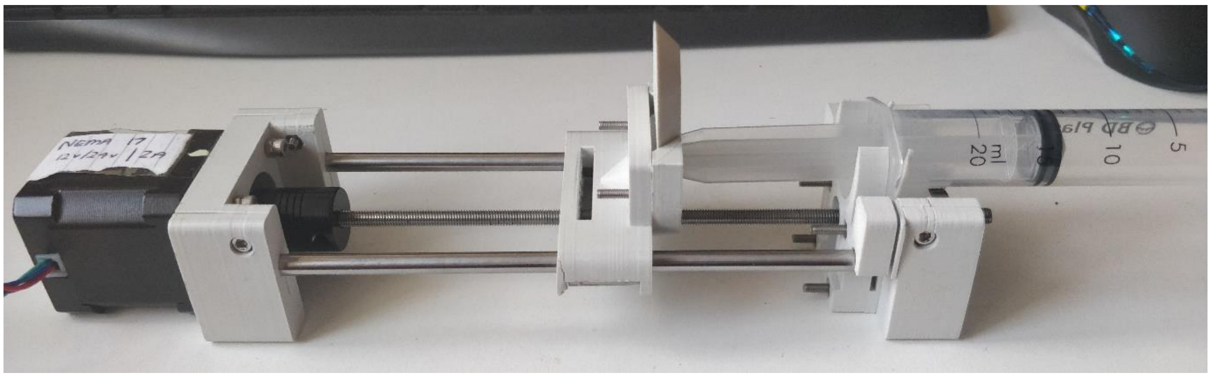
The first option can only be found on touchscreens built by the original Raspberry Pi manufacturers – these are expensive at £60 from The Pi Hut and you can get the same performance from a 3rd party touchscreen for £30 to £40, albeit without the ribbon cable interface.

The second option can be seen on the left-hand side image above, an Elecrow 5-inch TFT Display (£40). This requires 26 GPIO pins and an HDMI connector which is unhelpful for our build. It uses all the 5V power pins on the board so I had to solder a jumper cable to power the motor driver and other components – if done incorrectly this could damage the display. The screen also sits above the rest of the GPIO pins making it awkward to setup other I/O and as such I chose a different display.

Finally, I found the 3rd option seen on the right-hand side in the image above. This is a Waveshare LCD B and costs around £43 from Amazon. It interfaces with the Raspberry Pi using USB and HDMI, leaving the GPIO available for any extra peripherals we may want to use. Since we need space for a breakout board to attach our motor driver to, this display was the best option.

The Waveshare LCD comes with a CNC cut case and I have provided 3D printed files for attachable legs as a case stand. The touchscreen works right away with the Raspberry Pi and can be calibrated easily in the settings menu.

4f. Syringe Pump v1



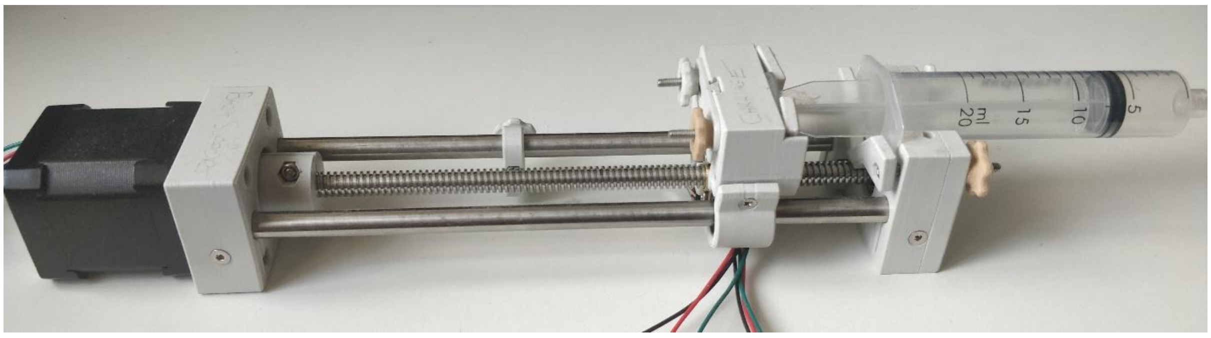
This first iteration of the Syringe Pump is a modified version of the Open Source Syringe Pump [12]. The main printed sections of the pump are as follows:

- Motor End – Holds the motor and steel rods together
- Carriage – Contains the linear bearings and plunger holder, moves along the rails and threaded rod
- Plunger Holder – Made up of 2 parts, the Body Holder attaches to the carriage for the plunger to slide into and the tab stops the plunger from moving
- Idler End – Holds the steel rods and ball bearing to the Syringe Holders
- Syringe Holders – 2 of these are attached to the Idler End with space between to insert and tighten the syringe body

The biggest issue I found with this design is the replacement of a syringe using the Syringe Holders. They require you to loosen the screws on the Idler End to remove it and then retighten the screws after. This issue is dealt with in the next iteration of the pump.

The last important piece of the pump is the threaded rod coupled to the motor that runs through the carriage. This allows the carriage to run up and down the pump when the motor is activated. The pitch (distance between 2 threads) of the rod is essential to calculating the distance travelled when the motor moves the coupler. The diameter of the rod is linked to the stability of the pump: the greater the diameter of the rod the less the pump will vibrate. This model uses a 5mm diameter rod, something I've chosen to upgrade for v2.

4g. Syringe Pump v2



The image above of the Syringe Pump v2 is similar in design to the v1 pump, with some modifications based on issues with v1 and from another Syringe Pump project [39]:

- Carriage – Redesigned with hand knobs for easy syringe replacement and space underneath for limit switches to prevent over-infusion and damage to both the Carriage and Idler End
- Limit Stoppers – 2 of these are placed on one of the steel rods during assembly and can be moved by loosening the hand knobs on the side for setting up different sized syringes
- Syringe and Plunger Holders – Both use a similar design, also incorporating hand knobs, to allow for easy syringe replacement
- Coupler – I replaced the metal coupler with a 3D printed one since it was tighter, stronger, and cheaper

For the threaded rod, I raised the diameter from 5mm to 8mm which helped deal with the vibration issues I had with v1. Choosing a different sized threaded rod meant that I had to redo my calculations for distance travelled by the carriage – both versions of these calculations can be found in Section 5c.

The hand knobs are designed such that no screwdriver is needed when replacing the syringe. Occasionally the bolts can come loose from the hand knobs so using super glue on the hand knobs is advised. In the future I plan to adjust their design so that this is no longer an issue.

5. Approach to Software

Having chosen the hardware and built both versions of the Syringe Pump, I had to decide on the appropriate software tools. This involved testing libraries, building user interfaces, and designing a mobile application.

5a. Choosing a Language

Since I had picked the Raspberry Pi as my microcontroller I was able to develop my software on Raspbian, a flavor of Linux. This gave me a wide variety of languages and libraries to choose from when developing each aspect of the software.

For hardware level programming, the Raspberry Pi gives us libraries [40] for 4 languages: C, C++, Python and Processing3. The choice of language here will persist over almost all aspects of development so I had to choose something I was both comfortable with and could build GUIs with easily.

Processing3 is more of a teaching tool and IDE built on top of other languages, with variants existing for Java, JavaScript, and Python [41]. It allows beginner programmers to work on basic frontend projects before exploring other languages. Building a GUI with Processing3 could be easy, but I was unfamiliar with the language and libraries it provided.

C and C++ both interact with the Raspberry Pi using the same set of libraries – either libgpiod or pigpio. As they are both compiled languages, they can provide a more sophisticated development experience, with speed and efficiency greater than scripting languages such as Python. I have a small amount of experience with both these languages and knew that building a GUI along with a Bluetooth handler could lead to development issues that would halt the progress of the project. As such, I decided to go with the last option – Python.

According to the PYPL (PopularitY of Programming Languages) the most popular language used for tutorials via Google Trends is Python [42]. It's a widely used language with many libraries for all aspects of development. This helps massively with combining the hardware code with GUI code as well as implementing remote control via Bluetooth. For these reasons, along with my own history with Python, I chose this for my language.

b. Pump Operation

To get our motor driver working we must use Pythons `RPi.GPIO` library. This library allows us to initialize the GPIO pins we're working with:

- DIR – Pin 20 – Tells the motor which direction to move in (clockwise/anti-clockwise)
- STEP – Pin 21 – Tells the motor to move a single step
- MODE – Pins 5,6,13 – Tells the motor what level of micro-stepping we want

For setting up our MODE output, we have a dictionary mapping the micro-stepping level to a tuple containing the appropriate configuration.

From here we can make the motor do a full revolution at any given micro-stepping mode. We do this by setting our DIR pin to clockwise (integer value 1) and MODE pins to the appropriate dictionary entry. Then we iterate through the number of steps per revolution our MODE is set to, activating the STEP pin, delaying for a short time, then deactivating the STEP pin. We require a delay between STEP pin writes to compensate for our code working faster than the motor can handle. For testing purposes, we delay for 0.005s but for our Syringe Pump we will calculate this delay based on how long the user wishes to use the Pump for.

5c. Infusion Calculation

Now that our Pump can move forwards and backwards, we must write some code to allow it to act as a Syringe Pump should. This means we must take some information from the user about the syringe and how long the infusion will take – then ensure the Pump infuses the full syringe in the given time frame.

To make this as easy as possible, we will generate 2 lookup tables – one for each Pump version – that can tell us how many steps (at a given micro-stepping value) it will take to fully infuse a specific size syringe.

To do this we have to calculate 3 values:

- Travel (mm) – Distance the carriage has moved along the threaded rod
- Conversion Ratio (mL/mm) – This will convert the travel distance in mm to the volume of liquid infused in mL
- Volume Ejected (mL) – The total volume of liquid infused by the Pump

<u>TRAVEL</u>

$$Travel\ (T) = pitch * \frac{steps\ taken}{steps\ per\ revolution}$$

Pitch is the distance between 2 threads on a threaded rod

For Syringe Pump v1, our pitch is 0.8mm and for Syringe Pump v2 it’s 2mm.

<u>CONVERSION RATIO</u>

$$Conversion\ Ratio\ (CR) = \frac{volume\ of\ syringe}{length\ of\ syringe}$$

The length of the syringe may vary between brands, so these calculations are configured to the BD Plastipak brand of syringes. Most commercial Syringe Pumps will require either a specific brand of syringe or will allow the user to choose which brand they have and will configure themselves accordingly.

Our system is configured for 20mL and 60mL BD Plastipak syringes which are of length 70mm and 109mm respectively.

<u>VOLUME EJECTED</u>

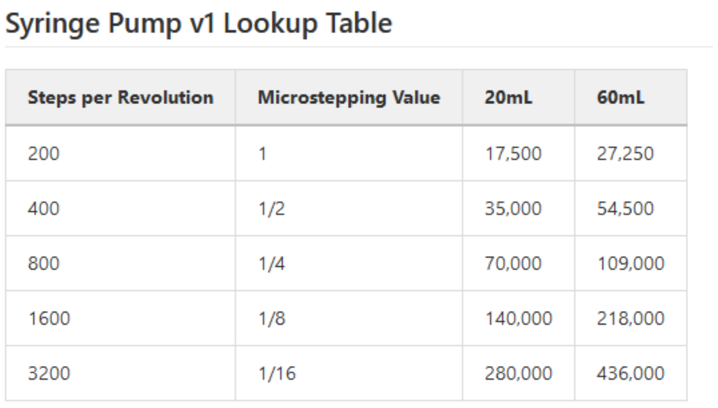
$$Volume\ Ejected\ (V) = \left(\frac{volume\ of\ syringe}{length\ of\ syringe}\right) * \left(pitch * \frac{steps\ taken}{steps\ per\ revolution}\right)$$

V = CR * T

This formula allows us to calculate the volume of liquid infused by the Syringe Pump based on the syringe itself and the steps the motor has taken. From this we can calculate the number of steps taken to infuse a full syringe at a given micro-stepping rate.

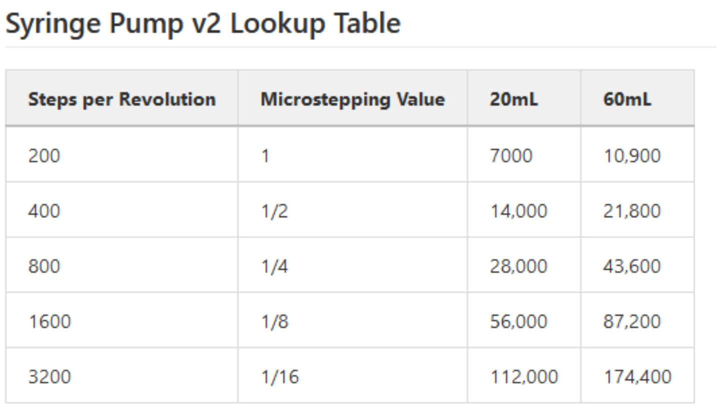
For each micro-stepping rate, we double the steps per revolution value, so these values will be 200, 400, 800, 1600 and 3200.

Doing this for both pumps and each micro-stepping value, I had the values for my lookup tables shown below.



Syringe Pump v1 Lookup Table

Steps per Revolution	Microstepping Value	20mL	60mL
200	1	17,500	27,250
400	1/2	35,000	54,500
800	1/4	70,000	109,000
1600	1/8	140,000	218,000
3200	1/16	280,000	436,000



Syringe Pump v2 Lookup Table

Steps per Revolution	Microstepping Value	20mL	60mL
200	1	7000	10,900
400	1/2	14,000	21,800
800	1/4	28,000	43,600
1600	1/8	56,000	87,200
3200	1/16	112,000	174,400

5d. Testing Suite

With the code written for basic pump operation and lookup tables for syringe infusion calculated, it was time to bring them together. The lookup tables are implemented as dictionaries mapping syringe size values to a dictionary of micro-stepping values to their respective step counts. These tables are calculated manually for now, although I would like to add a software component that allows users to add custom syringes that are selectable in the main GUI.

Next, I had to use these lookup table values to calculate the required delay between motor steps. For consistency I decided to keep all time variables to seconds, including the user input time. In the future I would allow for user to select a unit of time, either seconds, minutes or hours, and the code would perform the conversion to seconds if required. Then we can divide the users time input by the total number of steps from the lookup table to get the delay per step.

Our previous pump operation example uses 2 delay calls: one after activating the STEP pin and one after deactivation. This means we divide our delay value by 2 to get our actual delay between pin writes. Combining these steps together we can write functions to infuse a full syringe.

To finish up the testing suite, I implemented functionality for continuous withdrawal and infusion at a user defined micro-stepping rate. The delay for these functions is hardcoded into a dictionary that divides the initial delay value (0.005s) by 2 for each level of micro-stepping. This way no matter what the micro-stepping value is the continuous operations will always take the same amount of time. This is better shown in bullet points:

- MS Value – 1 – Delay = 0.005
- MS Value – 1/2 - Delay = 0.005/2
- MS Value – 1/4 - Delay = 0.005/4
- MS Value – 1/8 – Delay = 0.005/8
- MS Value – 1/16 – Delay = 0.005/16

The full documentation for the testing suite can be found in the GitLab documentation outlined in Section 3.

5e. Pumpy Class

With all the functionality tested and working correctly, I had to properly encapsulate this data into a class. Doing this improves the readability of the code and allows us to import the functionality into any other Python file. Since I planned to build a GUI and Bluetooth handler, keeping operation functionality in its own class was the best way to implement the system.

This class, known as Pumpy, involved having pin values be customizable in the constructor in case of future electronics updates, creating a pump method that combined together all the elements from my testing suite, implementing a continuous operation mode and bringing in limit switch support for the Syringe Pump v2. These act as while conditions for both continuous and pump operations, ensuring the pump stops if the appropriate switch is pressed.

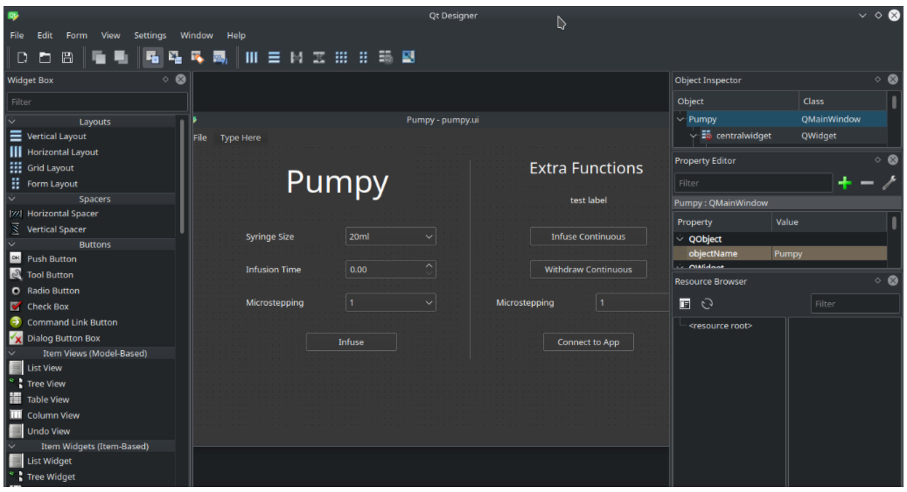
5f. GUI

With all the pump operation handled, it was time to decide on an appropriate GUI library. Pythons popularity means that a large number of well documented libraries exist for building GUIs, including Tkinter, PyQT and Kivy. These libraries can all register touchscreen input as a mouse event, meaning no extra work was needed to make the GUI touchscreen capable.

Tkinter was the first library I looked at since it's built directly into the Python standard library. Several great tutorials exist to walk the user through since most Python users will try Tkinter as their first GUI library, so I tried the Coders Legacy tutorial series [43]. Designing a GUI is done programmatically, so elements are either managed by a preset layout that determines their location on screen or the user hardcodes their locations in a fixed window size. Both methods were unsuitable for my initial GUI design, so I looked for another library.

Kivy was the next library I tried since it has a different method for designing GUIs. This involves using a separate '.kv' file to define the structure of your GUI which can be imported into Python to add functionality. The pong game tutorial on the Kivy website gives a great introduction to the Kv file syntax and building a functional GUI [44]. Once I began working on my own GUI, I encountered the same issues I found with Tkiner: properly formatting layouts to display elements where I wanted them was a challenge. I decided to switch libraries again to one that allowed me to design my GUI with external software.

This is where I began working with PyQt5. A program called Qt Designer is installed alongside the PyQt library and allows us to build a GUI in a much more intuitive manner. Once your design is completed you can export it as a '.ui' file which can be imported into Python. There you can implement your functionality using the PyQt5 library. This was a much better way of building a GUI and is how I decided to build my application.



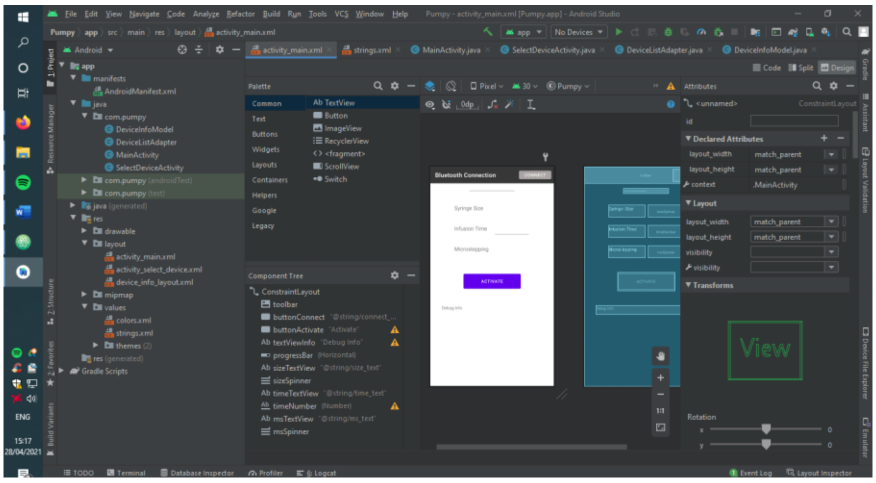
5g. Mobile Application

With a GUI in place I had to decide how I would operate the system remotely. This required picking a mobile platform and a communication protocol to send data over.

For a mobile platform I had 2 choices: iOS or Android. As of March 2021, Android is present on over 70% of mobile devices with iOS trailing at 27% [45]. On top of this, iOS charge $99 a year as a developer fee to access advanced tools and upload your application to the App Store [46]. All of this, along with the fact that Android began as an open source project, pushed me to develop my application on Android.

Developing for a new platform meant working with a new programming language. Android applications can be written in one of 2 languages: Java and Kotlin. Kotlin is a variant of Java meant to make development easier and is used primarily for Android development. Learning Kotlin for building this application could prove to be time consuming, and since I have used Java throughout my degree I decided to work with Java.

Google provide the Android Studio IDE for free which contains tools for designing an interface, incorporating external libraries, and testing the application with an Android Emulator. By activating developer mode on an Android device, you can also upload your application directly to the device for further testing.



Android also has libraries for directly interfacing with a devices Bluetooth adapter. This meant I could search for and connect to external devices over Bluetooth for transmitting and receiving data. Since the Raspberry Pi has built in Bluetooth, I could build a Bluetooth handler using Pythons ‘pybluez’ library to accept a connection from the mobile application. From here I can wait to receive operation information or send information back to the mobile application.

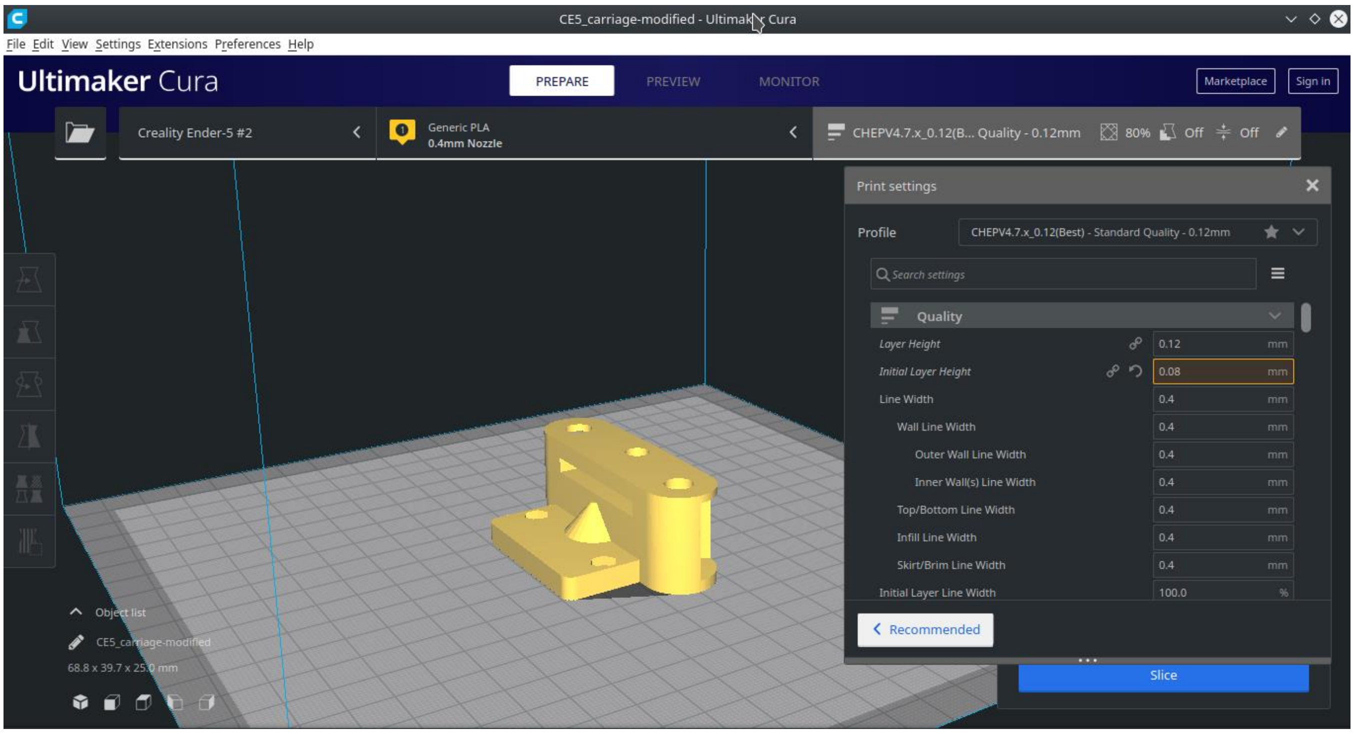
Bringing all these features together gives us the Pumpy software pipeline: hardware operation, touchscreen GUI and mobile application.

6. Development, Design and Manufacturing

Here I will go through the steps I took to bring together the hardware and software elements from Sections 4 and 5 into my final product.

6a. Setting up a 3D Printer

Once I had my 3D printer built and setup, I had to choose some slicer software. This allows the user to import 3D models, tune various printer settings and generate a '.gcode' file that the printer can run. For this I chose Ultimaker Cura: it's the most popular slicer since it's free, open source and has many advanced settings.



Print settings

Profile: CHEPV4.7.x_0.12(Best) - Standard Quality - 0.12mm

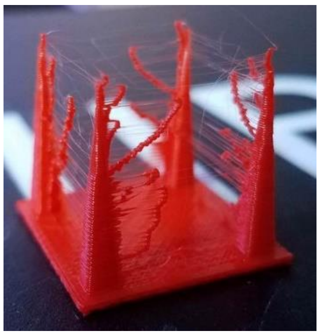
Quality

Layer Height	0.12 mm
Initial Layer Height	0.08 mm
Line Width	0.4 mm
Wall Line Width	0.4 mm
Outer Wall Line Width	0.4 mm
Inner Wall(s) Line Width	0.4 mm
Top/Bottom Line Width	0.4 mm
Infill Line Width	0.4 mm
Skirt/Brim Line Width	0.4 mm
Initial Layer Line Width	100.0 %

With the software in place, I had to adjust some settings to ensure my prints were of a high quality:

- Initial Layer Height – This ensures the first layer printed adheres to the print bed, this is important because if the first layer fails then the rest of the print will warp as a result. Making the initial layer height shorter than the normal layer height allows the plastic to squish flatter against the print bed. Based on the printers design, there are so called 'magic numbers' that work best for layer height. For my printer these are increments of 0.04mm, so I use 0.08 and 0.12mm as my layer heights.
- Print Speed – Since I'm using a cartesian printer I ensure consistent quality prints by using a speed between 60 and 80mm/s. Going higher than 80mm/s creates gaps in the print layers since the printer extrudes plastic slower than the print head moves.

- Retraction – There are 2 main retraction settings: Speed and Distance. When the print head must move over a gap without extruding it performs a retraction: the filament is pulled a given distance out of the nozzle at a given speed. This is used to avoid stringing, where filament still in the nozzle is pulled along the print creating thin and unwanted lines over gaps. An example image is provided to the right.



Having completed several test prints such as the retraction tower shown above, I got to work printing the 3D parts for the Syringe Pump v1.

6b. Designing a Syringe Pump

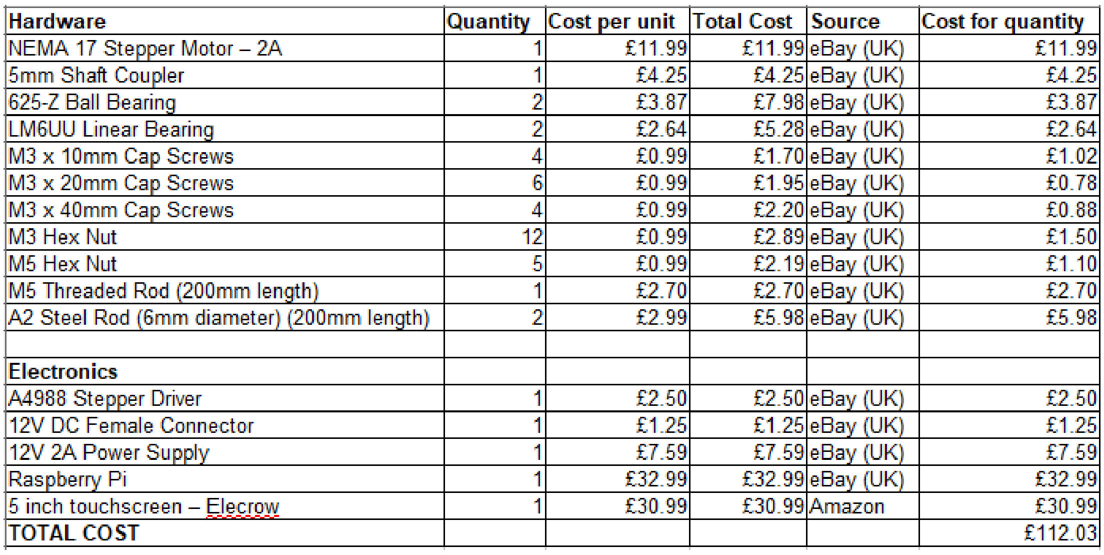
The original design of my pump makes use of the files provided in the Open-Source Syringe Pump Library [12]. My aim was to build and test this design, noting modifications I wanted to make as I went along. Using these, I could improve on an initial working design before adding my own features to the pump.

6bi. Bill of Materials

Both pump versions use a similar set of parts that cannot be 3D printed. The 2 most important parts are the threaded rod and the motor. The specifications of the threaded rod and motor will dictate how I calculate the infusion operation as well as the sizes of all the 3D printed parts.

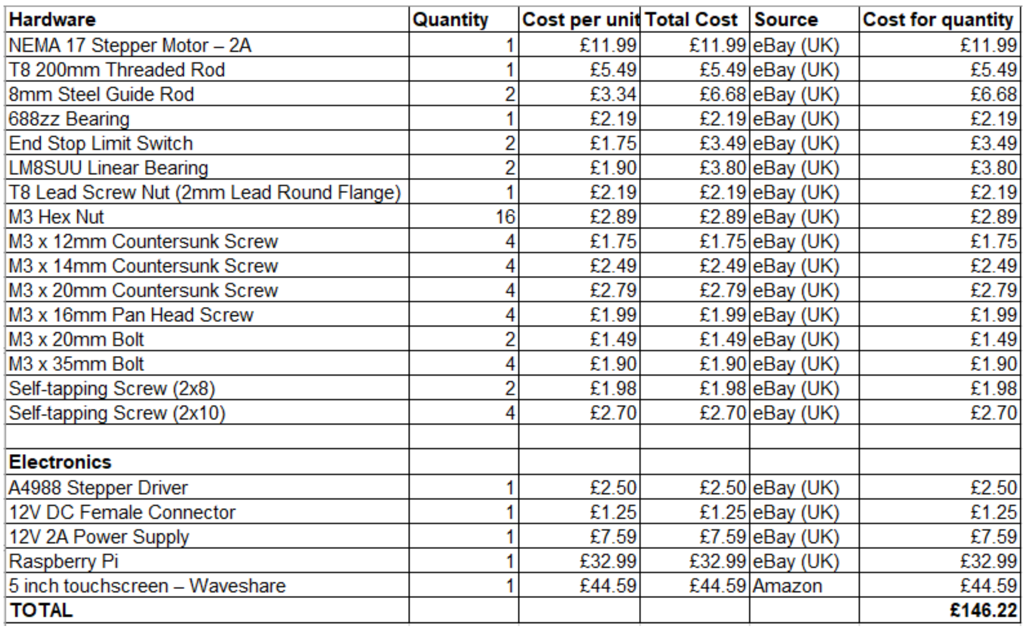
Below are both of the bill of materials tables for each version of the Syringe Pump along with the total cost (not including 3D printed parts):

SYRINGE PUMP v1



Hardware	**Quantity**	**Cost per unit**	**Total Cost**	**Source**	**Cost for quantity**
NEMA 17 Stepper Motor – 2A	1	£11.99	£11.99	eBay (UK)	£11.99
5mm Shaft Coupler	1	£4.25	£4.25	eBay (UK)	£4.25
625-Z Ball Bearing	2	£3.87	£7.98	eBay (UK)	£3.87
LM6UU Linear Bearing	2	£2.64	£5.28	eBay (UK)	£2.64
M3 x 10mm Cap Screws	4	£0.99	£1.70	eBay (UK)	£1.02
M3 x 20mm Cap Screws	6	£0.99	£1.95	eBay (UK)	£0.78
M3 x 40mm Cap Screws	4	£0.99	£2.20	eBay (UK)	£0.88
M3 Hex Nut	12	£0.99	£2.89	eBay (UK)	£1.50
M5 Hex Nut	5	£0.99	£2.19	eBay (UK)	£1.10
M5 Threaded Rod (200mm length)	1	£2.70	£2.70	eBay (UK)	£2.70
A2 Steel Rod (6mm diameter) (200mm length)	2	£2.99	£5.98	eBay (UK)	£5.98
Electronics					
A4988 Stepper Driver	1	£2.50	£2.50	eBay (UK)	£2.50
12V DC Female Connector	1	£1.25	£1.25	eBay (UK)	£1.25
12V 2A Power Supply	1	£7.59	£7.59	eBay (UK)	£7.59
Raspberry Pi	1	£32.99	£32.99	eBay (UK)	£32.99
5 inch touchscreen – Elecrow	1	£30.99	£30.99	Amazon	£30.99
TOTAL COST					£112.03

SYRINGE PUMP v2



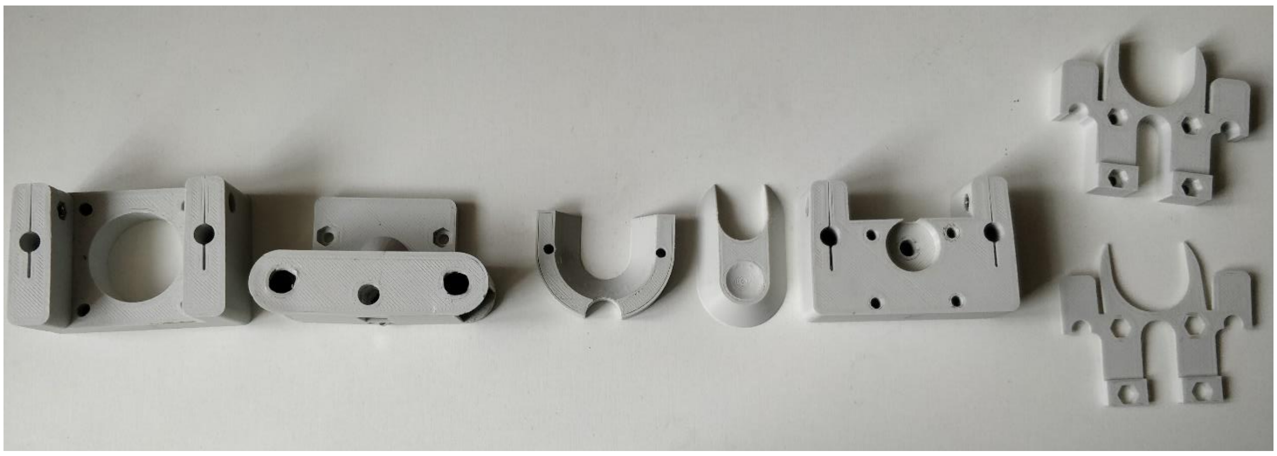
Hardware	Quantity	Cost per unit	Total Cost	Source	Cost for quantity
NEMA 17 Stepper Motor – 2A	1	£11.99	£11.99	eBay (UK)	£11.99
T8 200mm Threaded Rod	1	£5.49	£5.49	eBay (UK)	£5.49
8mm Steel Guide Rod	2	£3.34	£6.68	eBay (UK)	£6.68
688zz Bearing	1	£2.19	£2.19	eBay (UK)	£2.19
End Stop Limit Switch	2	£1.75	£3.49	eBay (UK)	£3.49
LM8SUU Linear Bearing	2	£1.90	£3.80	eBay (UK)	£3.80
T8 Lead Screw Nut (2mm Lead Round Flange)	1	£2.19	£2.19	eBay (UK)	£2.19
M3 Hex Nut	16	£2.89	£2.89	eBay (UK)	£2.89
M3 x 12mm Countersunk Screw	4	£1.75	£1.75	eBay (UK)	£1.75
M3 x 14mm Countersunk Screw	4	£2.49	£2.49	eBay (UK)	£2.49
M3 x 20mm Countersunk Screw	4	£2.79	£2.79	eBay (UK)	£2.79
M3 x 16mm Pan Head Screw	4	£1.99	£1.99	eBay (UK)	£1.99
M3 x 20mm Bolt	2	£1.49	£1.49	eBay (UK)	£1.49
M3 x 35mm Bolt	4	£1.90	£1.90	eBay (UK)	£1.90
Self-tapping Screw (2x8)	2	£1.98	£1.98	eBay (UK)	£1.98
Self-tapping Screw (2x10)	4	£2.70	£2.70	eBay (UK)	£2.70
Electronics					
A4988 Stepper Driver	1	£2.50	£2.50	eBay (UK)	£2.50
12V DC Female Connector	1	£1.25	£1.25	eBay (UK)	£1.25
12V 2A Power Supply	1	£7.59	£7.59	eBay (UK)	£7.59
Raspberry Pi	1	£32.99	£32.99	eBay (UK)	£32.99
5 inch touchscreen – Waveshare	1	£44.59	£44.59	Amazon	£44.59
TOTAL					**£146.22**

The majority of the materials were sourced from various UK based eBay sellers between 2020 and 2021, so prices could change, or other vendors may offer a cheaper alternative.

6bii. 3D Printed Parts

For the Syringe Pump v1 we make use of the following 3D printed parts which are shown in the image below from left to right:

- Motor End
- Carriage – Modified
- Plunger Holder Base
- Plunger Holder Tab
- Idler End
- Syringe Body Holder (x2) - Modified

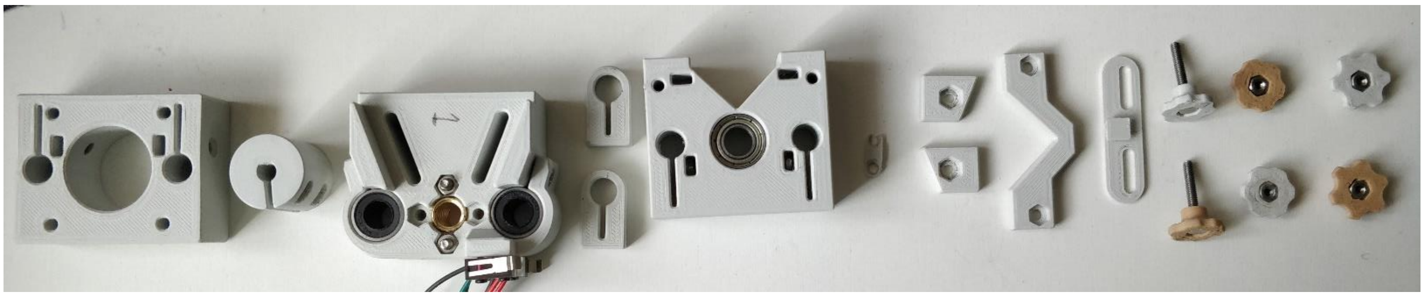


The total print time for all these parts was 17h2m and used up 82 grams of PLA filament.

The modifications made to the Carriage and Syringe Body Holders was done using Autodesk Meshmixer. This software allows users to import model files in the '.stl' or '.obj' format and modify them however they choose.

For the Syringe Pump v2 we use the following 3D printed parts, shown in the image below from left to right:

- Back Support
- Coupling
- Carriage
- Limit Stop (x2)
- Front Support
- Wire Holder
- Plunger Holders
- Side Syringe Holder
- Slider
- 7mm Hand Knobs
- 8mm Hand Knobs
- 9mm Hand Knobs



The total print time for all these parts was 16h58m and used up 79 grams of PLA filament.

All the model files in '.stl' format can be found in the GitLab repository provided in Section 3.

6c. Building a Syringe Pump

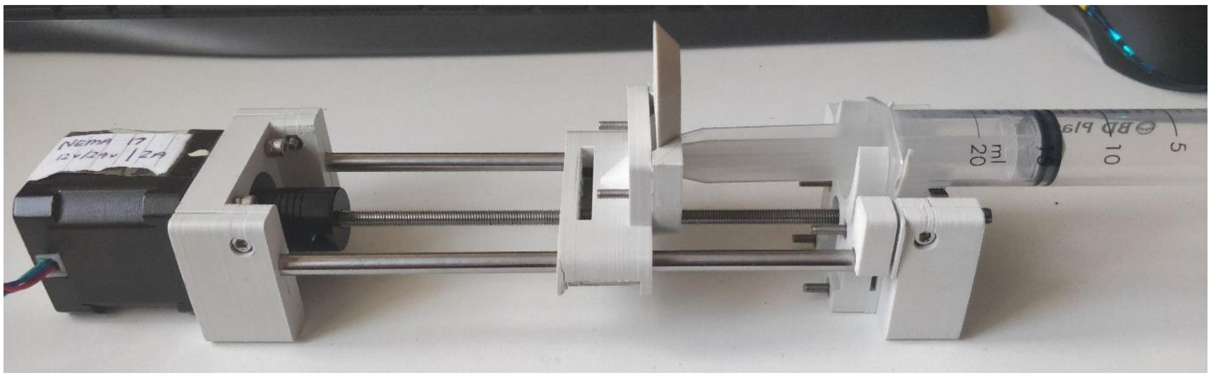
Here I will discuss the process I went through building the Syringe Pump v1, the modifications I chose to make and the process for building the Syringe Pump v2.

6ci. Syringe Pump v1

With all the 3D printed parts completed and the bill of materials ordered it was time to build. The Open Source Syringe Pump Library has a build guide on its Appropedia page [13] which was simple enough to follow but had some difficult steps. The carriage model they designed required you to cut holes for the linear bearings to be inserted. Instantly I realized how difficult this was in practice and made a note of it for modification.

When sliding the carriage onto the 3 rods I realized how loose it was without a syringe and the Plunger Holder Tab inserted. To compensate for this, I decided I would have to get rid of the small nut underneath the carriage that the threaded rod goes through and replace it with a more stable part such as a lead screw nut.

The Syringe Body holders were designed for use with 20mL syringes only: the top of the holder was a closed circle. For my project, I had discovered that medical syringe pumps make use of 20mL and 60mL syringes. This meant I had to either modify these or design new ones from scratch.



Once it was fully built, I wired up the motor to the A4988 motor driver, connected that to the Raspberry Pi and wrote the first part of my testing suite. Here I discovered that when calibrated at the micro-stepping values ‘1’ and ‘1/2’ the motor would vibrate strongly and slowly move across the table if left long enough. When tested at the 3 other micro-stepping values the effect was less noticeable. To deal with this I would have to either build a case to stop the pump from moving or increase the diameter of the rods to make the pump heavier and more stable.

Since I had a working syringe pump prototype, I was able to implement my infusion calculations described in Section 5c and ensure they functioned as expected.

From here I began to outline the modifications I wanted to make for a new iteration of the Syringe Pump.

6cii. Modifications

With a better understanding of how the Open Source Syringe Pump Library worked, I collated together my modification ideas along with one from the Lynch Open Source Syringe Pump modifications page on Appropedia [15]. These modifications are listed below:

- Carriage
    - Add spacing to the underside of the carriage where the linear bearings are inserted to avoid users having to cut a space open themselves
    - Add spacing through the bottom where the threaded rod slides through for the same reason as above
    - Replace the M5 hex nut that holds the threaded rod in place with an attachable lead screw nut to help prevent unloaded carriage movement
    - Inclusion of limit switches to prevent the carriage pushing against the motor and idler ends which can damage them
- Plunger Holder Body and Tab
    - Widen both to allow for 60mL syringes, but would require modifying the carriage to accommodate the larger body
    - Design a new system that incorporates this into the carriage without a top so any size syringe will fit
- Body Holders
    - Use the Lynch modified body holders that remove the top section and allow for easy syringe removal – but these still don't fit 60mL syringes
    - Use a single piece that pushes the body from underneath onto the idler end and tightens from the other side, ensuring any size syringe will fit and only requiring 2 screws to tighten
- Steel and Threaded Rods
    - Make all these thicker to help reduce vibration from the motor, would require modification to both ends and the carriage

These modifications led me to look at various syringe pump projects that people had uploaded to Thingiverse. Here I came across a design that incorporated many of the modifications I outlined above: limit switches on the carriage, improved syringe body and plunger holders and hand knobs for easy adjustment [39]. Using the OSSP as my template, I began incorporating elements of Andrey Samokhins design, along with my own ideas listed above, into the Syringe Pump v2.

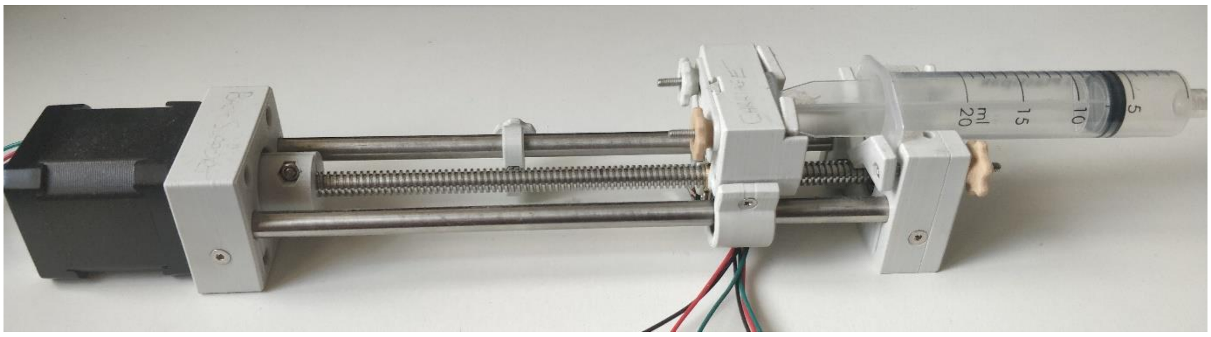
While testing out modifications on my Syringe Pump v1, I wrote up a new build guide that includes my modified files which can all be found in the Gitlab repository from Section 3.

6ciii. Syringe Pump v2

Once I had all my modified 3D printed files printed and ready, I put together and ordered the new bill of materials. This included 8mm rods instead of 5mm, a new touchscreen for a better electronics form factor, limit switches and a lead screw nut. The limit switches arrived attached to a breakout board that was too large for the carriage. This meant de-soldering the limit switch component from the board, which can be done with a soldering iron, solder and some tweezers.

During the build I noticed some issues with the hand knobs. The hex nut spacing inside of them was too large so the hand knob would slide away when in use. As a quick fix the user could superglue the hex nuts into the hand knobs, but I decided I would modify each hand knob size so the nut wouldn't slip out.

The new design also used moveable limit switch stoppers for the left side steel rod. These would activate the limit switches attached to the carriage and tell the pump to stop moving. Since they could move different sized syringes could make use of them, but the user would have to set them up before filling the syringe. To get around this I made markings on the rod to identify where the limit stopper should be placed for 20mL and 60mL syringes. At this point I am still trying to decide on a better way to deal with this issue.



As I built this new version, I wrote a build guide which is available on the Gitlab repository found in Section 3.

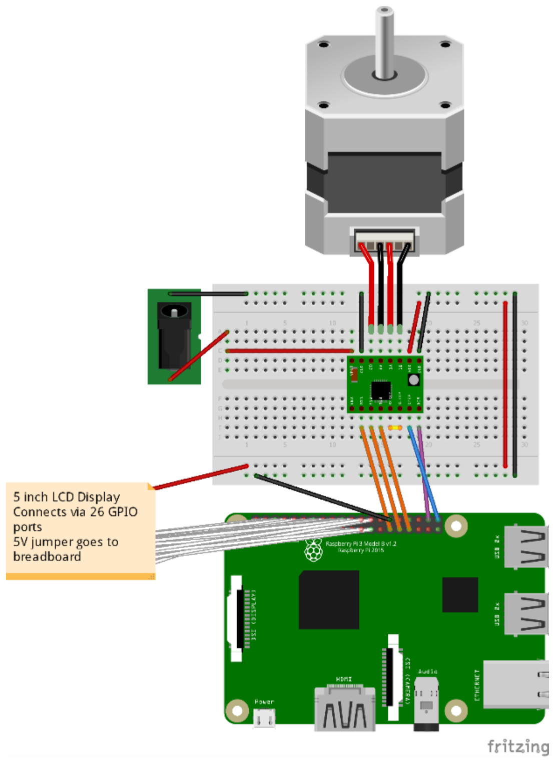
With everything put together I modified my electronics setup to incorporate the 2 limit switches. After I ran my testing suite to ensure the carriage moved properly, I reworked my infusion calculations outlined in Section 5c to compensate for the wider pitch of the new threaded rod. With the new lookup table, I was able to test a full infusion and continuous activation using the limit switches. Here I also tested my 60mL syringe which fit comfortably into the new holders and ensured my lookup table was in working order.

Now I had a more appropriate pump for my project, I began work on cleaning up the electronics.

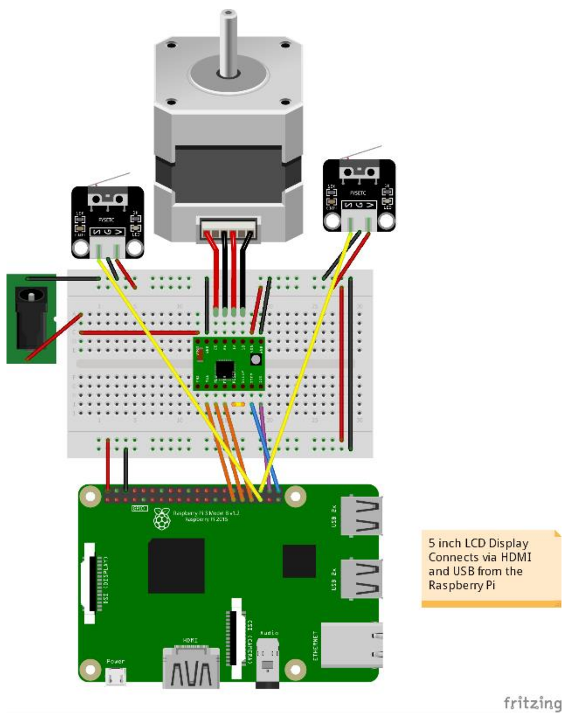
6d. Building the Electronics

Before building each version of the Syringe Pump, I designed the electronics systems using Fritzing – an open source tool for building electronics designs. Below are both designs for the electronics systems:

<u>ELECTRONICS DIAGRAM V1</u>

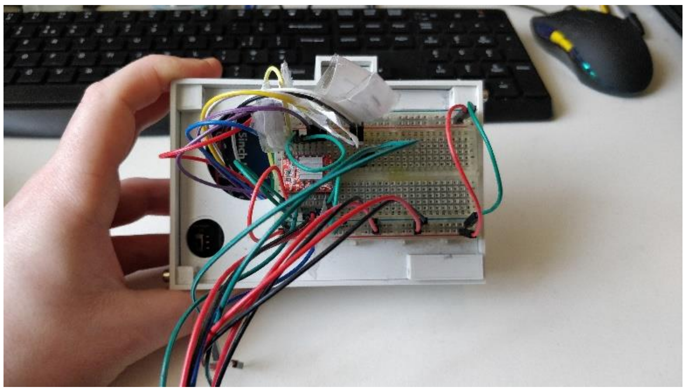


<u>ELECTRONICS DIAGRAM V2</u>

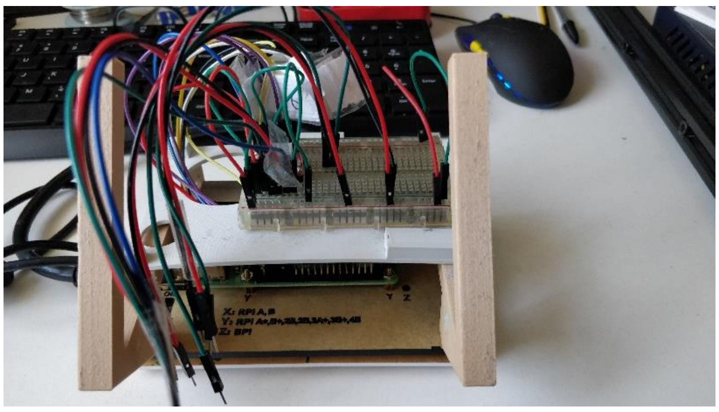
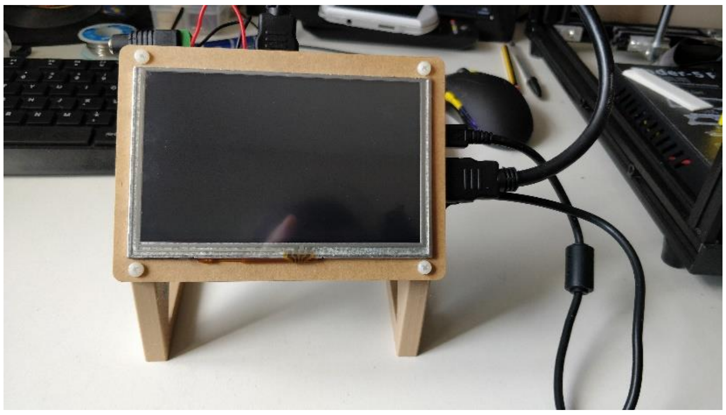


Using these as a guide, I wired up a breadboard and tested each version of the pump. For Syringe Pump v1 I made use of a 3D printed case for the Raspberry Pi and touchscreen, attaching the breadboard to the back. The files for this case were created by bredita and uploaded to Thingiverse [47]. For Syringe Pump v2, a CNC cut screen housing was included with the touchscreen and I 3D printed legs created by Grey3000 on Thingiverse [48] so the enclosure could stand. Images of these breadboards and cases are shown below:

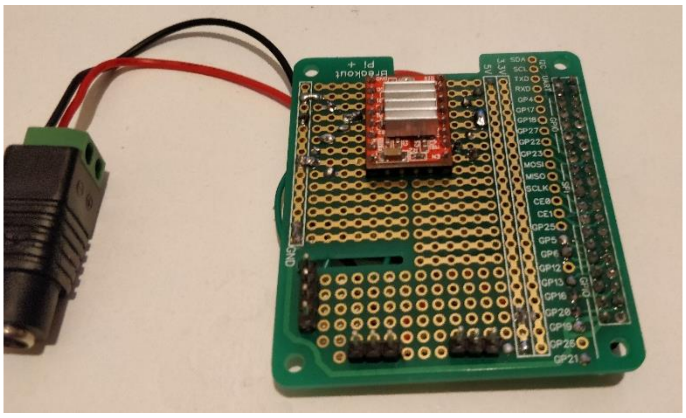
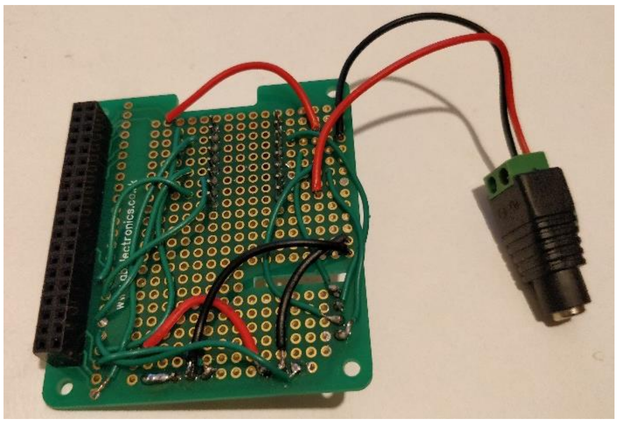
<u>BREADBOARD V1</u>



BREADBOARD V2

Once the breadboard systems were functional, I ordered a Raspberry Pi breakout board to replace the breadboard and soldered the components and headers onto it. This made setting up the pump a lot easier: insert the motor and limit switches into their headers, plug in the 12V and 5V power supplies and the system boots up. Below are images of the populated breakout board:

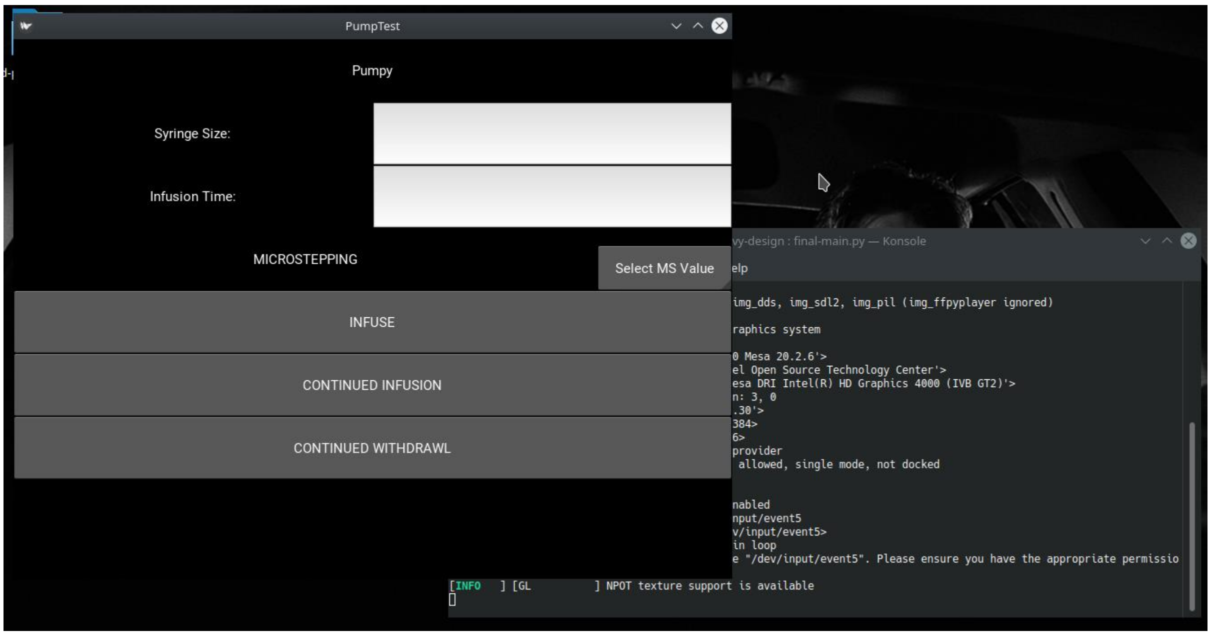
6e. Designing a User Interface

The next step was to work on a GUI to operate the device. As outlined in Section 5f I designed interfaces for 2 different libraries: Kivy and PyQt5. The features my GUI required were:

- Touchscreen friendly, no keyboard or mouse required
- Choice of syringe size and micro-stepping value
- Ability to continuously withdraw and infuse for setting up a syringe
- Stores and loads previous operations

As I worked with each library, I attempted to make the GUI comfortable to look at with a minimal set of visible features. This way new users wouldn't be overwhelmed by complicated features on startup.

6ei. Kivy



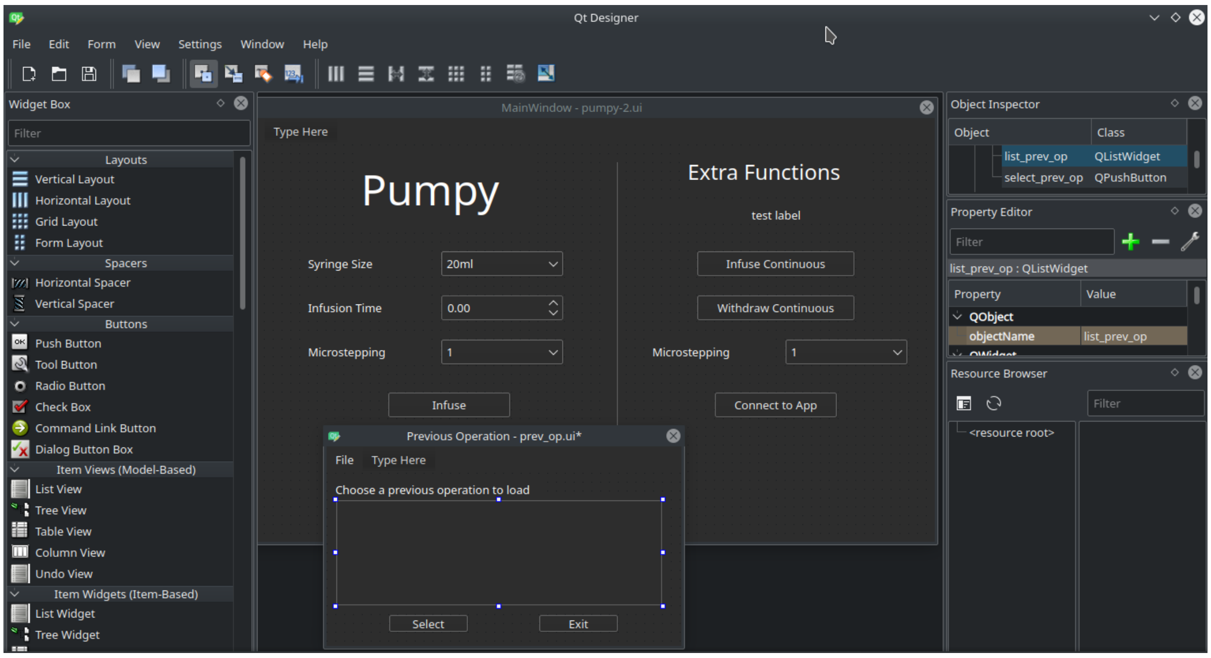
```
img_dds, img_sdl2, img_pil (img_ffpyplayer ignored)

raphics system

0 Mesa 20.2.6'>
el Open Source Technology Center'>
esa DRI Intel(R) HD Graphics 4000 (IVB GT2)'>
n: 3, 0
.30'>
384>
6>
provider
 allowed, single mode, not docked


nabled
nput/event5
v/input/event5>
in loop
e "/dev/input/event5". Please ensure you have the appropriate permissio
[INFO   ] [GL          ] NPOT texture support is available
```

Working with Kivys layout system made it difficult to resize elements on the screen. This meant labels, buttons and input boxes would fill entire grids and appeared ugly when displayed. There is also no CSS support to create a theme, choose fonts and change colours – instead this is handled on an element by element basis which can be tedious.

For numerical inputs I had planned to implement a virtual keyboard that would pop up when the user clicked on the 'Syringe Size' or 'Infusion Time' input boxes. This feature would work on a lone application, but when testing it with my GUI I couldn't get it to display consistently. When it did show up the keyboard suffered from input latency and didn't feel comfortable to use.

For these reasons I chose to switch libraries to PyQt5.

6eii. PyQt5



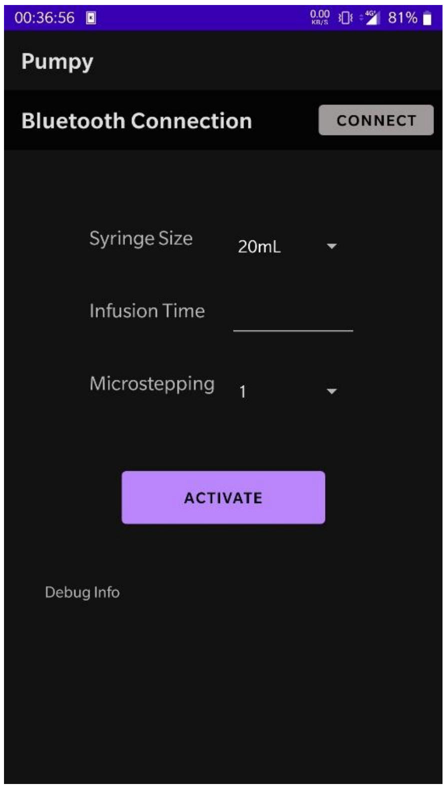
Upon first installing PyQt 5, it also installed the Qt Designer – an application for building GUIs graphically rather than programmatically. The GUI designed is exported as a '.ui' file that can be imported into Python to add functionality. This software gave me more control over each element's location on the screen, its size, and its theme. Each element uses CSS to style itself, allowing users to design each element however they want.

Once I decided I was happy using PyQt 5 as my GUI library I built the Pumpy interface along with a window to store, display, and load previous operations. I also decided not to use a virtual keyboard for user input, instead using a QSpinBox widget to let users select a time without need for any keyboard input. In the future I plan to add a time unit option, allowing users to have the software detect their time value as seconds, minutes, or hours.

Now that the GUIs were completed, I exported them as '.ui' files and imported them into a Python class. From here all my elements were available as class variables for me to bind functions to. I could then include my pump operation class known as Pumpy.py and test the GUI operating the Syringe Pump.

From here I was ready to begin work on the remote operation feature.

6f. Building an Android Application



Syringe Size 20mL

Infusion Time

Microstepping 1

For the final part of my project, I wanted users to be able to access the Pumpy system remotely from an Android device. This would allow users to pair their phone or tablet with the Raspberry Pi, send operations for the Pump to perform and receive feedback once the operation has completed.

I began by reading the Android documentation on the BluetoothAdapter and BluetoothDevice classes [49]. To initiate a connection, I would have to perform the following steps:

- Create a custom element to display any discoverable Bluetooth devices
- Create an activity that lets the user connect to a Bluetooth device from the above list, pair with the device and return to the main application
- When the main application detects it's paired with a device, it starts up a thread to get an active connection – this way data can be received and transmitted
- If an active connection is made, a connected thread starts up to handle the data being sent and received
- Our main application will continuously wait for messages to be sent to it from the connected thread, and allows for us to write data over the same thread

These activities, data structures and threads were implemented and tested by uploading the application to a OnePlus 5 Android phone. Once the application was able to connect to the Raspberry Pi and I knew that the connected thread was running, I had to implement a Bluetooth handler in Python so that my GUI could deal with data from the Android application.

To do this I used the pybluez library to pick up when the Android application was paired with the Raspberry Pi. From here the Python code could accept a connection from the application and begin handling data. Encapsulating this behavior into a class, I had to turn it into a thread so that the GUI could continue to be used while the Bluetooth handler ran in the background and queued up operations.

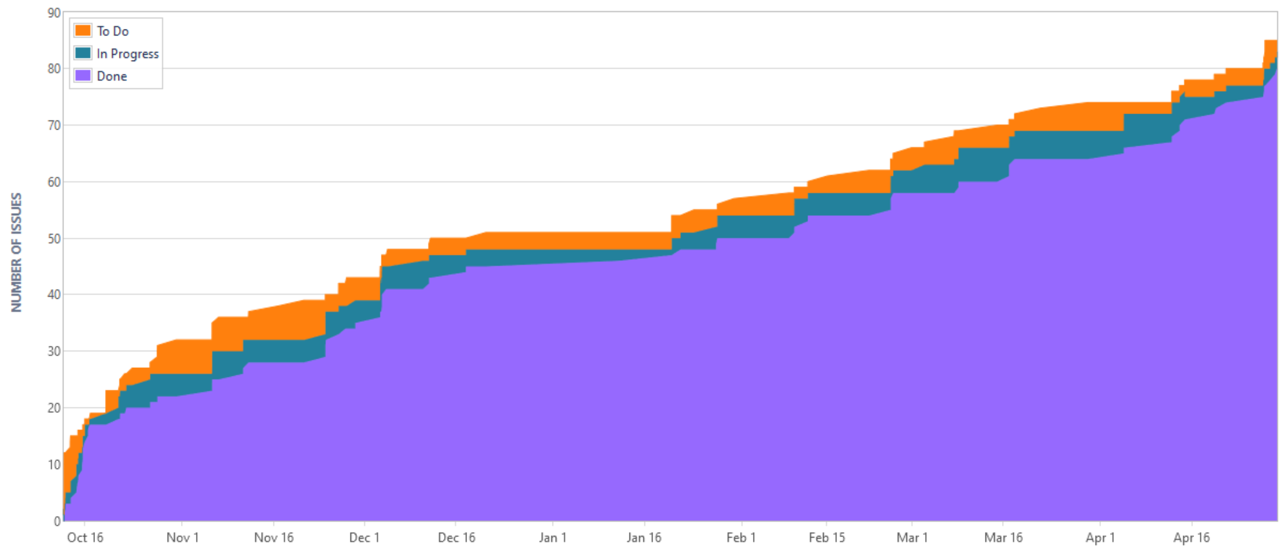
After some bug fixing and further testing, I had a functional Android application that could connect to my Syringe Pump over Bluetooth and send operation information to activate the pump for remote infusion.

7. Results and Conclusion

7a. Project Planning

This project was developed using an Agile methodology, allowing me the flexibility to decide what area of the project I worked on and routinely adjust my goals. Being able to have weekly meetings with my supervisor gave me the chance to take a step back and analyze the work I had done. From there I could gain a better perspective on how long certain features would take and reinforce long term goals to myself, ensuring I didn't fall into a programming rabbit hole.

To track my progress, I made use of Jira: a workflow tool for Agile development. It provided me with the tools to break down the bigger aspects of my project into smaller achievable goals. With this comes a sense of scale, showing you the backlog of tasks completed and the problems you've ignored for some time. Having not used a tool like Jira before, I found it invaluable for managing my time when lots of things were happening at once.



The above diagram is a cumulative flow diagram generated by Jira, showing the life cycle of issues as they pass through the development process. From this we can see that during the Christmas Holidays (December 16th – January 16th) and around the end of March when the Easter Holidays began, issues were not passing through the development cycle. This was due to taking time off the project over Christmas and a backlog of other assignments during March.

The graph also shows that issues didn't pile up during the project, with between 4 and 6 tasks actively in development at any one time.

7b. Conclusions and Future Work

In the introduction I outlined 5 goals I hoped to achieve with this project, with the end goal being to design and build a Syringe Pump system that proved the viability of a cheap and open source solution to expensive medical equipment.

I was able to take the Open-Source Syringe Pump Libraries [12] initial proof of concept and create a working prototype. From here I decided on modifications that would improve on the original design, adding features that could help turn this into a viable product such as easy syringe withdrawal, improved stability, and position detection with limit switches. Then I put together the electronics to operate my modified Syringe Pump and built a GUI that provides an easy to use touchscreen interface for basic operation. Finally, I created an Android application to connect to the system via Bluetooth for remote operation.

Exploring the viability of a cheap and open source medical Syringe Pump is a difficult goal to achieve, since there is no tangible outcome unlike the rest of this project. I have been able to show that the system is capable of the same functions as commercial syringe pumps, albeit unrecognized by any medical authority. To bring my project to this goal, I would need to present the idea to a charity such as Glia who are providing open source medical equipment at a low cost in developing countries [35].

There is a need for this kind of product in the medical industry. The open source development website Just One Giant Lab (JOGL) has a project for an Open Source Low-Cost Syringe Pump adapted to Hospital Uses [50]. I contacted one of the developers of the project, Adam Krovina back in November 2020 to discuss the current goals of the project. They had a couple of Electronic Engineering PhD students working on a custom embedded software system and previously had people working on the certification and regulation of such a device. As it stands today, work on the project has been slow – but the need for such a device hasn't changed.

My project has taken the ideas of this JOGL project in a consumer electronics direction, focusing on usability and reproducibility. There are still many improvements to be made to the Syringe Pump system: support for multiple syringe brands, extensibility of remote operation to handle multiple pumps at one time, status updates about the current progress of the infusion.

My personal goal with this project was to see what I could produce with the tools provided by the open source community and to try and give something back.

8. References

[1] https://www.jnj.com/johnson-johnson-announces-completion-of-acquisition-of-auris-health-inc

[2] https://apps.who.int/iris/bitstream/handle/10665/44407/9789241564045_eng.pdf?sequence=1

[3] https://www.3dcrowd.org.uk/about-us/

[4] https://3dprint.nih.gov/collections/covid-19-response

[5] Chagas et al. (2020) – Leveraging open hardware to alleviate the burden of COVID-19 on global health systems

[6] Thomas et al. (2020) – Ten Days to Implementation of 3D-Printed Masks for a Level-I Orthopedic Trauma Practice During the COVID-19 Pandemic

[7] Franco-Bedoya et al. (2017) - Open Source Software Ecosystems: A Systematic Mapping

[8] https://www.fda.gov/media/78369/download

[9] https://www.mariecurie.org.uk/professionals/palliative-care-knowledge-zone/symptom-control/syringe-drivers

[10] Garcia et al. (2018) – Low-cost touchscreen driven programmable dual syringe pump for life science applications

[11] Booeshaghi et al. (2019) – Principles of open source bioinstrumentation applied to the Poseidon syringe pump system

[12] Wijnen et al. (2014) – Open-Source Syringe Pump Library

[13] https://www.appropedia.org/Open-source_syringe_pump

[14] https://www.youmagine.com/designs/syringe-pump

[15] https://www.appropedia.org/Lynch_open_source_syringe_pump_modifications

[16] 8ARM: Open Source Precision Pump (2015) - https://www.diva-portal.org/smash/get/diva2:915985/FULLTEXT01.pdf

[17] https://web.archive.org/web/20070611152544/https://opensource.org/docs/osd

[18] https://www.arduino.cc/en/Guide/Introduction

[19] https://source.android.com/

[20] https://httpd.apache.org/

[21] Dryden et al. (2017) – Upon the Shoulders of Giants: Open-Source Hardware and Software in Analytical Chemistry

[22] https://ohwr.org/welcome

[23] https://www.journals.elsevier.com/hardwarex

[24] https://openhardware.metajnl.com/

[25] Woern er al. - (2018) - RepRapable RecycleBot: Open Source 3D printable extruder for converting plastic into 3D printing filament

[26] Pearce J – Nature Vol. 491. 519-521- Make nanotechnology research open-source

[27] Jones et al. (2009) – RepRap: The Replicating Rapid Prototyper

[28] https://patents.google.com/patent/US5121329

[29] https://www.grandviewresearch.com/industry-analysis/3d-printing-industry-analysis

[30] https://commonslibrary.parliament.uk/research-briefings/cbp-8515/

[31] https://science.howstuffworks.com/science-vs-myth/everyday-myths/how-long-does-it-take-for-plastics-to-biodegrade.htm

[32] Graham et al. (2005) – The Syringe Driver and the subcutaneous route in palliative care: the inventor, the history and the implications

[33] https://www.sps.nhs.uk/wp-content/uploads/2018/02/NRLS-1287-Safer-ambulatorrs-RRR-2010.12.13-v1.pdf

[34]https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/757999/dhsc-review-of-syringe-driver-safety-actions.pdf

[35] https://glia.org/

[36] Pavlovsky et al. (2018) – Validation of an effective, low cost, free/open access 3D printed stethoscope

[37] https://www.creality3dofficial.com/products/ender-5-pro-3d-printer?variant=39278008401993

[38] https://hackaday.com/2016/08/29/how-accurate-is-microstepping-really/

[39] https://www.thingiverse.com/thing:4194094

[40] https://www.raspberrypi.org/documentation/usage/gpio/

[41] https://processing.org/

[42] https://pypl.github.io/PYPL.html

[43] https://coderslegacy.com/python/python-gui/

[44] https://kivy.org/doc/stable/tutorials/pong.html

[45] https://gs.statcounter.com/os-market-share/mobile/worldwide

[46] https://developer.apple.com/programs/how-it-works/

[47] https://www.thingiverse.com/thing:2287941

[48] https://www.thingiverse.com/thing:2311344

[49] https://developer.android.com/guide/topics/connectivity/bluetooth

[50] https://app.jogl.io/project/185/OpenSyringePump