Final Report

Team RoboSquirt

May 5, 2014

Contents

Chapter 1

Executive Summary: Development of a Low Cost and Open Source Fluid Handler

Many biological, chemical, and tissue engineering studies depend heavily on the transport, organization, feeding, and analysis of cells. Whether for a specific goal like observing the link between phenotype and genotype in single cells, or for more routine lab work like transporting small organisms in solution or feeding cells, over 1000 laboratories worldwide share a common demand for efficient and reliable fluid transportation.

Current approaches to optimize fluid transportation in research settings are inadequate. The most laborious yet economical option, manual pipetting, yields low throughput experiments. Commercial liquid handlers automate the process of manual pipetting, leading to significant increases in throughput. However, the cost of commercial liquid handlers is exorbitantly high, often ranging from $10,000 to $40,000. There exists a demand for a low cost automated liquid handler that will help laboratories and clinics of any size access the same kind of large-scale assays typically reserved for well-funded major institutions.

In an effort to bring large-scale biology to the masses, five senior undergraduates from Rice University are developing RoboSquirt, a low cost, open source liquid handler capable of transporting low volumes. Specifically, the device will be able to accurately collect and dispense fluid into specific locations on a multi-well plate. RoboSquirt will cost approximately $250 to build and is designed to be highly customizable; it will be a versatile tool capable of fulfilling a large number of laboratories' liquid handling needs. In addition to complying with cost and accessibility constraints, Team RoboSquirt will address the following challenges and goals:

- Movement reliability: the robotic arm moves to the correct well on a multiwell plate
- Volumetric accuracy: the device can dispense with an accuracy of $10\mu L \pm 0.5\mu L$
- Single cell transport: the liquid handler is capable of dispensing a viable single cell into each well
- Bidirectional flow: the liquid handler can both collect and dispense solution

The RoboSquirt fluid handler has a reliable and simple physical design. The structural design is an X-Y frame which supports the motors that move the dispensing nozzle head from one place to another. The nozzle is able to collect or dispense fluid anywhere within the X-Y frame. The motors and pump are controlled by an Arduino microcontroller, which in turn is issued commands by software we have written. The syringe pump ensures easy-to-replicate volumetric and flow rate accuracy.

Team RoboSquirt has constructed the first working prototype of our fluid handler. Both the accuracy of movement and flow rate exceed the standards outlined in our design criteria, and the software component of the implementation is nearly complete. However, the transportation of single cells has not yet been tested with the fluid handler. Team RoboSquirt is in the process of performing further tests on the capabilities of RoboSquirt, as well as publishing the design and documentation of RoboSquirt on GitHub.

Chapter 2

Introduction: Optimizing Fluid Handling

Over 1000 biological laboratories world-wide can only can only afford to perform cellular experiments through tedious, error-prone manual pipetting, drastically slowing research into diseases such as Alzheimer's, Parkinson's, and diabetes. Commercial liquid handlers cost up to tens of thousands of dollars, and are limited by proprietary software. With this problem in mind, Team RoboSquirt has developed a low cost, open-source solution to bring automated liquid handling to every lab.

2.1 Basics of Liquid Handling

Automated liquid handling arose from the need to make the process of transporting liquid quicker and more accurate. Basic microliter syringes were the tools initially used to dispense low volumes of liquid into multi well plates. These syringes were replaced by the precise, spring-controlled micropipettes that are used in labs today. The latest advancements pertaining to liquid handling involve computer-run motorized systems [1], specifically fluid handler robots [2]. Fluid handlers have automated the tedious process of moving and dispensing substances for experiments, and they have greatly enhanced the throughput of DNA, RNA, and other assay-based experiments.

2.2 Currently Available Liquid Handlers

2.2.1 Pipetting

The simplest and cheapest solution to liquid handling is manual pipetting, whereby an individual transports measured volumes of liquid via media dispensers. The majority of pipettes function by creating a partial vacuum about the liquid-holding chamber, and as a lever is pressed down by the user, pressure is exerted on the fluid, pushing it out. There is a variety of pipettes available for a multitude of purposes and enabling differing levels of accuracy and precision. The action of manual pipetting, however, can be extremely laborious

and time-consuming. Many laboratories use well plates containing 384 wells, and as such the process of accurately dispensing liquid into such plates can take hours at a time.

2.2.2 Fluidigm C1

Figure 2.1: The Fluidigm C1

The Fludigm C1 (Figure 2.1) is an all-in-one unit that uses microfluidics to isolate cells and perform high throughput assays. An on-screen workflow allows users to easily select a variety of genomics related assays. The device is capable of performing DNA and RNA sequencing, gene expression analysis, PCR, and more. The powerful functionality comes at a great monetary cost and is inflexible with respect to the experiments that it can perform outside the scope of genomic experiments, such as electrophysiological measurements. The device is also incapable of performing simple fluid handling functions.

2.2.3 Splotbot

The Splotbot (Figure 2.2) is a design for an open source liquid handler capable of intelligent tracking for fluid handling experiments, created by Kliment Yanev and Juan M.P. Gutierrez. Using a PlayStation EyeToy, computer vision algorithms, and servo actuated syringes, the Splotbot can pick up and dispense liquid for artificial life handling experiments and perform general liquid handling procedures. The project was vital in providing insight as to how to produce a robust yet inexpensive framework based on a 3D printer. On the other hand, the syringe- based actuation mechanism is not conducive to interfacing with microfluidic devices or single cells [3].

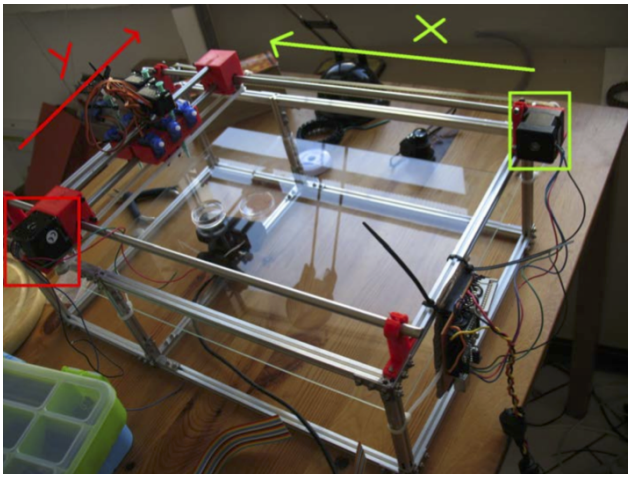


Figure 2.2: The Splotbot inspired the X-Y structural frame design of the RoboSquirt fluid handler.

2.3 Demand for an Alternative Fluid Handler

Due to the temporal inefficiency of modern micropipettes and both the high cost and inflexibility of automated liquid handlers, an open-source, automated liquid-handler would enable high-throughput fluid organization and transport in a broad spectrum of laboratory environments that are currently inhibited by the limitations of modern options. The ideal solution would be highly customizable so as to facilitate a broad spectrum of experiments. Specifically, the device would be able to interface with microfluidic devices, giving it the capacity to transport single cells and other small and precise fluid volumes. Hence, there ultimately exists a need for a versatile, inexpensive fluid handler capable of dispensing volumes on the same microscale as other, more expensive devices [4].

2.4 Potential Applications of Low-Cost, Automated Fluid Handler

An open source fluid handler could be used for a variety of lab needs, including the study of individual cell behavior, which requires the transport and dispensation of small volumes of liquid. The initial motivation for the development of such a device was a need to isolate the genes that are responsible for specific electrophysiological behavior in a given organism. The device is intended to be used in conjunction with microfluidic devices already developed in Robinson Lab. Experiments that could benefit from this application include the following:

- Magnetic properties of pigeon neurons: Pigeons can navigate during migration using the earths magnetic field, suggesting that they possess magnetic proteins expressed by their neurons. To test this, gene expression profiling (via qPCR) must be performed on individual cells in an attempt to relate this phenotypic expression to a particular genotype. The transportation of these isolated cells from the microfluidic device to a multi well plate requires a specialized fluid handler.

- Human olfactory cells: A smell map (analogous to a color wheel) could potentially be created by examining the chemical receptors present in human olfactory cells. As in the above pigeon experiment, individual cells must be transported from the microfluidic device to a multiwell plate by a specialized fluid handler.

- C. elegans worms: The nervous systems of C. elegans are not well understood by the scientific community because it is difficult to isolate individual worms for experiments. These worms are approximately 1 mm long, so the fluid handler can also be used to transport and manage individual worms, opening the opportunity to perform experiments on these specimen.

In addition to the aforementioned experiments, an open-source fluid handler would also assist in many flow-based applications. Tasks like feeding cells, moving and sorting small organisms in solution, and performing molecular assays could be conducted in high throughput via this device, thus saving both time and money.

2.4.1 Global Target Market

Almost all biological labs in the country have a need to dispense precise volumes of liquid, and hence this deice could conceivably behoove a full panoply of academic and industrial laboratories in the United States, and potentially many laboratories in other countries.

2.5 Design Considerations and Solution

To address the aforementioned needs, Team RoboSquirt has collaborated with Robinson Lab at Rice University to create and disseminate the design instructions (both hardware and software) for an open-source, low-cost liquid handler capable of dispensing single cells. The initial motivation for the development of RoboSquirt was a need to isolate the genes that are responsible for specific electrophysiological behavior in a given organism. The device is intended to be used in conjunction with microfluidic devices already developed in Robinson Lab.

2.5.1 RoboSquirt Requires Modular, Customizable Software

The limitations of currently available microfluidic devices indicate the need for a cost- effective, easy-to-assemble microfluidic device capable of isolating and dispensing individual cells. Our end goal is to create an open source fluid handler, complete with a powerful and versatile user interface. Very generally, this interface must be able to:

- Provide a virtual representation of the physical space over which the handler is dispensing.

- Accept commands sent by other programs running on the same computer, such as those programs controlling different parts of the same experiment.

- Interpret and relay these commands to the fluid handler.

- Accept raw commands inputted directly to the interface by the user.
- Provide a versatile command library to enable functionality for a large variety of experiments.

In short, the current fluid handler software is a large system containing many interconnecting parts, and it needs to be highly configurable in order to satisfy the wide range of end uses. The first iteration of this fluid handler implemented its software in Matlab. While MATLAB was a suitable language in which to perform initial testing, it is not an object-oriented language, and thus cannot easily manage a large system. In addition, while most end users would already have MATLAB on their lab computers, creating the interface as a stand- alone program would increase both its accessibility and usability. For these reasons, we decided to rebuild the system in Java, which would satisfy and allow for solutions to all these constraints. As an aside, by restarting and thus re-architecturing the system, we can more easily add additional functionality that the original system was not designed for.

2.5.2 RoboSquirt Utilizes 3D Printable Parts

The hardware of the microfluidic device should be just as accessible as the devices software. The design and assembly of the devices hardware consists of a combination of purchased and 3D printed components. Printing components allows complex shapes to be easily and inexpensively manufactured. In order to facilitate this, our project will include 3D models that the user can download and print.

2.5.3 RoboSquirt will be Disseminated Over the Internet

In order to distribute open source code, STL files, and assembly instructions, a website needs to be created that will enable worldwide access to these parts of the project. In addition to these downloadable files, the website must also include an instruction manual and troubleshooting guide, as well as a forum through which users can communicate, ask and answer questions, etc.

2.5.4 RoboSquirt has Specific Functional Requirements

Regarding functionality of the completed device, there are certain critical concerns that must be addressed. First, the device must be able to transport live cells without affecting their health or structure. Specifically, the device must be designed so that it does not damage a cell when subjecting it to high flow rates. The device needs to be able to handle volumes as small as $10\mu L$ (estimation based on 1/10th the volume of a 384 well plate) and it must possess bi-directional flow capabilities so that cells may be transferred from one container to another, such as from a microfluidic device to a multi well plate. In addition, the frame and arm of the fluid handler must be built in a way such that they are resistant to minor bumps, and do not need to be frequently calibrated.

2.5.5 Standards

Because this liquid handler is not directly used to diagnose, prevent, or treat disease and in no way comes into contact with patients, it does not fall under the category of medical device and hence there are no relevant FDA or AAMI standards that should affect its design and development. Similarly, although our prototype is technically an electronic and consequently should be governed by IEEE regulations, it does not fall into any of the specific categories that require such standards, such as wireless LANs, radiation detectors, or security systems.

2.5.6 Patents

Because we will not be making any profit off this design, we will not be breaching pre-existing patents.

2.6 Expectations for RoboSquirt

Fluid handler robots dispense substances into designated containers and automate processes in chemical and biological laboratories worldwide. However, current fluid handlers are expensive and do not interface smoothly with external microfluidic devices used for cell isolation. An open-source, low-cost, and versatile fluid handler would increase the accessibility of process automation, specifically for processes that involve transporting single cells from a microfluidic device to a multi well plate. To address this demand, Team RoboSquirt is creating a low-cost, fluid handler capable of transporting single cells. The finalized design will be disseminate on the internet to provide a multitude of laboratories with the information and code needed to create the device itself, enabling an increase in high-throughput fluid-based experiments worldwide.

Presented in the following pages of this binder is a detailed walkthrough of Team RoboSquirt's design procedure. Subsequent sections will provide further context regarding the demand and applications for the device, the specific criteria required of the design, a roadmap of the team's solutions, documentation of the team's advancements, and other specific considerations such as a safety plan, parts list, and budget.

Chapter 3

Design Strategy for Low-Cost Fluid Handler

3.1 Problem Statement

Fluid handler robots dispense substances into designated containers and automate processes in chemical and biological laboratories worldwide. However, current fluid handlers are expensive and do not interface smoothly with external microfluidic devices used for cell isolation. An open source, low cost, and versatile fluid handler would increase the accessibility of process automation, specifically for processes that involve transporting single cells from a microfluidic device to a multi well plate.

3.2 Design Criteria

Our team is ultimately designing an automated liquid handler capable of isolating, transporting, dispensing, and collecting single cells. To do so, the device must comply to certain requirements regarding repeatability, volume accuracy, and flow manipulability needed to ensure proper functionality and cell viability. The two primary limitations to be addressed in the design of this liquid handler are that it must be low-cost to make this device easily constructible within a university or industry laboratory, and versatile enough to enable various cell isolation and liquid handling applications. In order to do be versatile, the interface needs to be able to accept, interpret, and relay commands sent by other programs running on the same computer.

Table 3.1: Design Criteria for a Low Cost and Open Source Fluid Handler

Parameter or Characteristic	**Specified Value(s)**	**Testing Method**	**Point Value**	**Comments**
Movement Reliability	100 wells	Distribute fluid to 100 wells on a 384 well plate and visually assess (with the naked eye) that liquid accurately dispensed into the appropriate wells.	20	Key goal. Repeatable functionality is essential.
Volumetric Accuracy	10 μL	Dispense 10 μL into a well, then determine the change in mass of the plate. Convert to volume using the liquid's density. Repeat 100 times.	20	Key goal. 10 μL is based on 1/10th the volume of a single well in a 384 well plate. Will be challenging but is requested by customer.
Cell Isolation	1 cell/well	Use a microscope to visually assess the number of cells within a filled 384 well plate.	20	Key goal to enable matching phenotype and genotype of individual cells.
Low Cost	$< $250	Evaluate and sum costs of individual components.	15	Should be considerably less than current models on the market, (about $1000).
continued on next page				

continued from previous page				
Parameter or Characteristic	**Specified Value(s)**	**Testing Method**	**Point Value**	**Comments**
Bidirectional Flow	2 directions	Assess and compare amount of fluid both dispensed and collected by RoboSquirt.	15	Facilitated by a bidirectional pump. Aspiration may be a challenge.
Code Versatility	0 errors	Interface code with current Robinson lab projects. Determine number of errors outputted by code.	10	Stretch goal. May depend on the progress of Robinson Lab's other projects.

3.3 Choosing a Pump

Fluid handlers have automated the tedious process of moving and dispensing substances for experiments, and they have greatly enhanced the throughput of assay based experiments. Three general mechanisms drive fluid movement in liquid handlers: piston pumps, vacuum pumps, and gear based pumps. RoboSquirt must select a pump which is low-cost, reliable, and accurate.

3.3.1 Option 1: Piston Pump

Pistons provide a mechanically simple, reliable way of withdrawing and dispensing fluid. Movement of a sealed piston in a tube causes pressure changes that withdraw and dispense fluid. Depending on the system, the piston is either in direct contact with the fluid or separated by a constant volume of air. [5] Since a piston pump has a shared inlet and outlet, this system is not capable of continuous flow.

3.3.2 Option 2: Vacuum Pump

Utilizing the same principle of inducing pressure change to create force, vacuum pumps use air pumps to create suction or pressure in order to move fluid. Like piston pumps, vacuum pumps cannot create continuous flow. These pumps also require extra feedback sensors and circuitry to control their valves [5].

3.3.3 Option 3: Gear-based pump

Motors create cyclical or reciprocating motion to move fluid. Gear and peristaltic pumps can continuously move fluid by compressing parts of the channels that pass through, and are capable of bidirectional flow [5]. Of particular note is the peristaltic pump's ability to move

small quantities of liquid by pushing air, which is useful for pumping volumes smaller than the space between peristaltic pump rollers.

Chapter 4

Final Design and Device Implementation

Work for this device began in May 2013 by Rahul Roy and Ben Avants. During their summer project, they designed and constructed the device's physical frame and dispensing head as well as completed the Arduino and Matlab framework for controlling the device. By the end of the summer the robot was capable of tracking a given set of wells over five different well plate sizes.

4.1 Software

Thus far, we have already made significant software advancements. A working program has been written that mimics the functionality of the original Matlab code, and the new program possesses abstractions that will allow for a wider range of capabilities than those supported by the first iteration of software. The user can easily change the specifications of well plates being used, change the number of well plates, as well as change the size of the rectangular area over which the fluid handler move, all via the new graphic user interface. These wells can then be selected to queue up a set of commands that will subsequently be sent to the handler.

4.1.1 Overview

The main bulk of software is a Java program that allows the user to easily set up their experiment and communicate with the physical device with a certain level of abstraction. In general, the user interacts with the GUI (Graphical User Interface) to design their experiment, then executes this workflow. From there, the program runs automatically, communicating with the Arduino to complete all specified tasks.

To do this, our program implements a Model-View-Controller (MVC) architecture. An MVC architecture is useful when the designer wishes to completely insulate the View (what the user sees and interacts with) from the Model (the backend system that keeps track of state, performs any necessary computations, and communicates with external devices or programs), using a Controller (which contains Adapters) to mediate and limit the interaction

between the two. By separating the two, a designer can easily change the layout of the GUI with the safety of never changing how the underlying functionality works.

As such, this large program can be divided up meaningful sections that handle discrete tasks. Currently, we have the main view, serial communication model, plate model, task model, external communication model, and serialization model. Below is a diagram outlining the purpose of each section, how it connects to other sections, and how they connect to external modules.

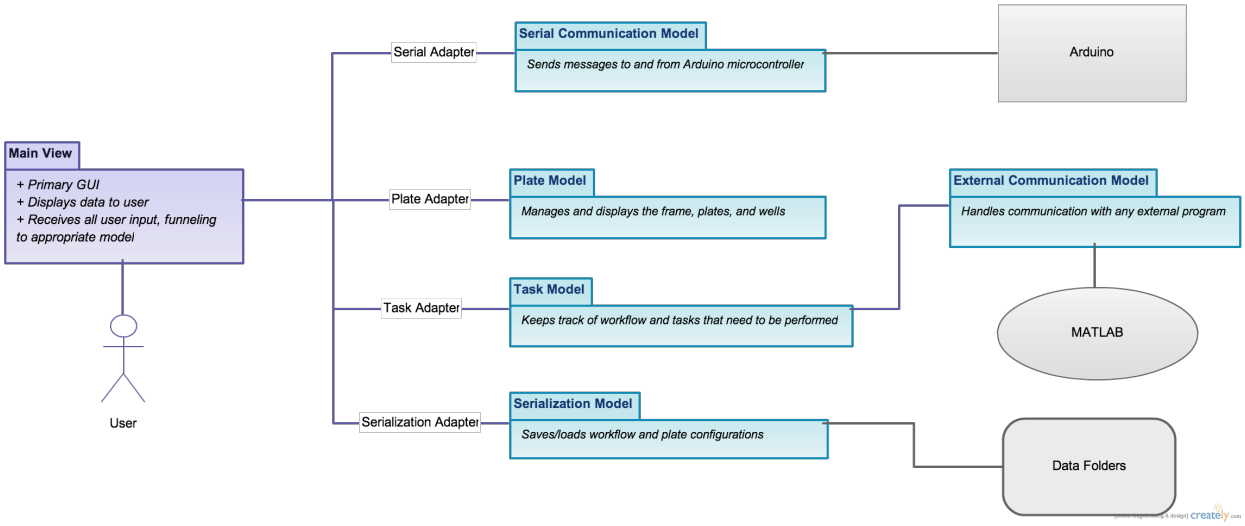


Figure 4.1: Use Case Diagram for RoboSquirt's MVC software implementation

The following sections detail how the user interacts with the program from the view, and the main concepts implemented within each model.

4.1.2 Main View

As mentioned in the overview, the main view handles all user input into the program. It consists of two main sections: on the left, an area that visualizes the physical frame bounds, plates laid beneath it, and tasks to be executed and, on the right, tabbed panes that contain all parameters, buttons, etc. that that the user can interact with. Figure X shows the entire GUI, with the tabbed pane on its first option, "Plate Setup". In this tab, the user inputs data directly from the multi-well plate's data sheet, where they want this well positioned in real space, and how they would like the wells to be ordered. These settings generate the 6-well plate seen, positioned with its top-left corner at (0, 0), with the well numbers ascending across their rows. Not shown in this screenshot, the user could choose to lay down multiple plates at once, with different specifications for each. The user can click the "Save Specs" button to save their input plate specifications to an external text file. Since this file persists across multiple program sessions, the user only ever has to input their specifications once. When the "Load Specs" button is clicked, the currently selected specification is loaded into the appropriate parameter inputs. This tab also allows the user to change the size of the area plates are laid on, as some users may make the end device slightly differently and the software needs to be flexible to such changes.

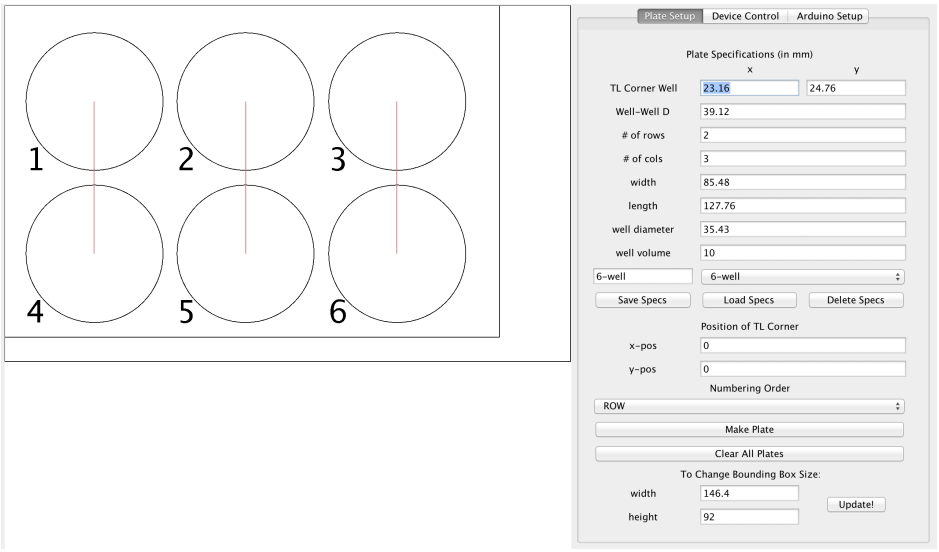


Figure 4.2: RoboSquirt's main GUI view

The next tab in the view, "Device Control", allows the user to manage the device workflow and which tasks it needs to perform. A screenshot of it is shown below in Figure 5.3. To allow for modular experiments, the user can make "stages" that can be executed independently or all at once (in order). Each stage can have any number of tasks, which are defined as any flow of liquid. In this section, parameters for tasks can be set, such as:

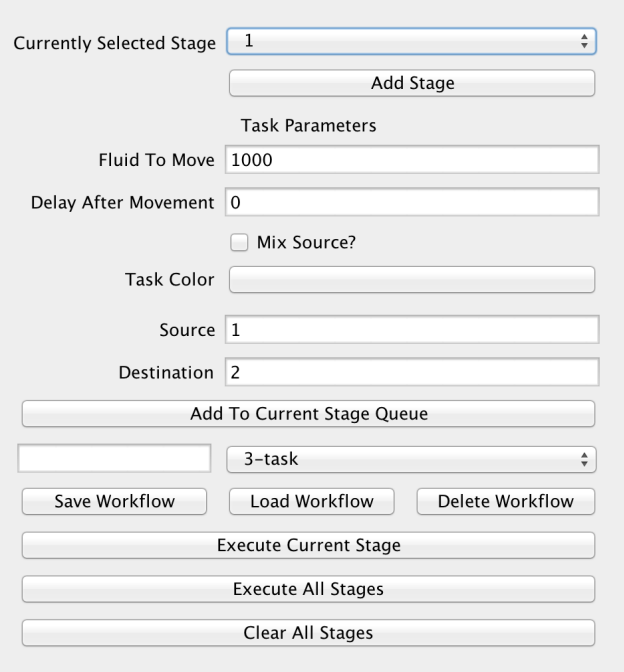
- the amount of fluid to move
- how long to delay after the task
- whether the device should mix wells before moving liquid
- what color to code the task

Once these parameters have been set, the user has a number of options for declaring the source and destination of the fluid. In particular, the user can designate well numbers for the source and destination fields to move fluid from one well to another, or designate "EXTERNAL" as the source in order to move fluid into a well from an external source. Alternatively, the user can also click and drag wells to easily queue tasks to and from these locations.

Also on this tab, the user can save and load a "workflow" in the same manner as plate configurations in the "Plate Setup" tab. A workflow is defined as the location and type of each plate laid out, as well as the tasks that have been assigned to each. By saving and loading workflows, the user can easily repeat the same experiment with minimal effort in configuration.

A new feature in the GUI is the ability to visualize list and visually organize tasks that have been created. Once tasks have been added to the task queue, the user sees these tasks as a tree in the device control panel. The tasks are visualized as a tree since high-level, more abstract tasks are composed with lower level tasks, down to the lowest level of tasks that

Figure 4.3: Device control GUI for RoboSquirt



actually communicate with the Arduino. With this mini-GUI, the user can see the order of tasks that are going to be executed, as well as drop and drop tasks to reorganize them.

The user can also now create tasks by choosing which task to create and then filling out the corresponding parameters that come up (instead of being limited to moving from wells to wells and external sources to wells). Fitting with this, the user can also save not only workflows, but any set of abstract tasks for future reference and execution. If the user wanted to, for instance, create a dilution task, they could create it out of more basic tasks, save this abstract task, then load it as a small part of a larger workflow later.

The last (and simplest) tab is one used to establish communication with an external microcontroller. When the Arduino is connected and the "Rescan" button is clicked, it becomes available in the drop down box. Clicking "Connect!" establishes a serial connection with it. The program automatically scans for serial connections on program startup as well.

4.1.3 Models

The previous section detailed how the user interacts with the program and the result of each interaction, but the view itself is only a set of buttons and inputs for communicating with the actual program logic. For example, when the "Make Plate" button in the "Plate Setup" tab is clicked, all the view does is ship the currently selected specifications to the plate model, which handles all plate creation and actually draws it to the screen. As of now, we have 5 distinct models implemented, each with a discrete function. Models may request data from other models, but must do so via adapters they are initialized with, in order to minimize coupling in the application. By doing this, the designer can rest assured that any changes made in one model will not have side effects in other models. The sections that

follow are much more technical in that they detail how the functionality outlined above is actually realized.

4.1.4 Serial Communication Model

This model handles all communication with the Arduino. Whenever a request is made by the view, it scans for all serial ports occupied on the computer, returning an ArrayList containing the name of each. When the request to connect to a certain serial port is made, a method is called that opens an input and output stream, using Java's BufferedReader and OutputStream object, respectively. When serial input is received, the received String is parsed and, if it is a signal that the Arduino is done with its current task, this model sends a request to the task model to execute the next task in the queue. Any other model wishing to communicate with the Arduino must receive the output stream directly from this model.

4.1.5 Plate Model

This model manages the virtual representation of all physical objects: the device dimensions, all plates being used, and the wells on those plates. When a request to create a plate is made, this model creates a Plate object that contains the location of the plate and its specifications. This is added to an ArrayList of all plates currently being used, and all wells on this plate are initialized to their own Well object. These well objects are added to a dispatcher in the model that utilizes the Observer-Observable design pattern, such that all wells can be quickly and effortlessly communicated with when desired. When a request to draw the screen is made from the view, this model calls draw methods on each plate and well object, which use their stored parameters to draw themselves to the view. When this happens, it also pushes a request to the task model to draw all tasks that have been designated by the user. This model also keeps track of any information regarding the device's state, encapsulated in an ArmState object, which can be used by other models to make reactive decisions on the device's operation.

4.1.6 Task Model

This model keeps track of all tasks to be performed by the device. Tasks themselves are organized in an abstraction hierarchy as shown in Figure 4.4.

Serial tasks are ones that communicate directly with the Arduino through the output stream acquired from the serial communication model. Drawing tasks (MoveWellToWellTask and MoveFromExternalTask) are ones that are composed of serial tasks but do not themselves communicate with the Arduino. By using such a hierarchy, tasks can be drawn to the screen when needed, but later deconstructed to their lowest level to be sent to and executed by the Arduino. MultiTasks are there strictly for organizational purposes, and can themselves be a combination of any other task. Using this, stages can be easily implemented by a top-level MultiTask (all tasks combined) containing many MultiTasks (one for each stage).

This hierarchy is parsed, deconstructed, and acted upon via the Visitor design pattern. When a request to draw the tasks is made, a DrawVisitor is applied to the MultiTask corresponding to the current stage, which recursively applies the visitor to its children.

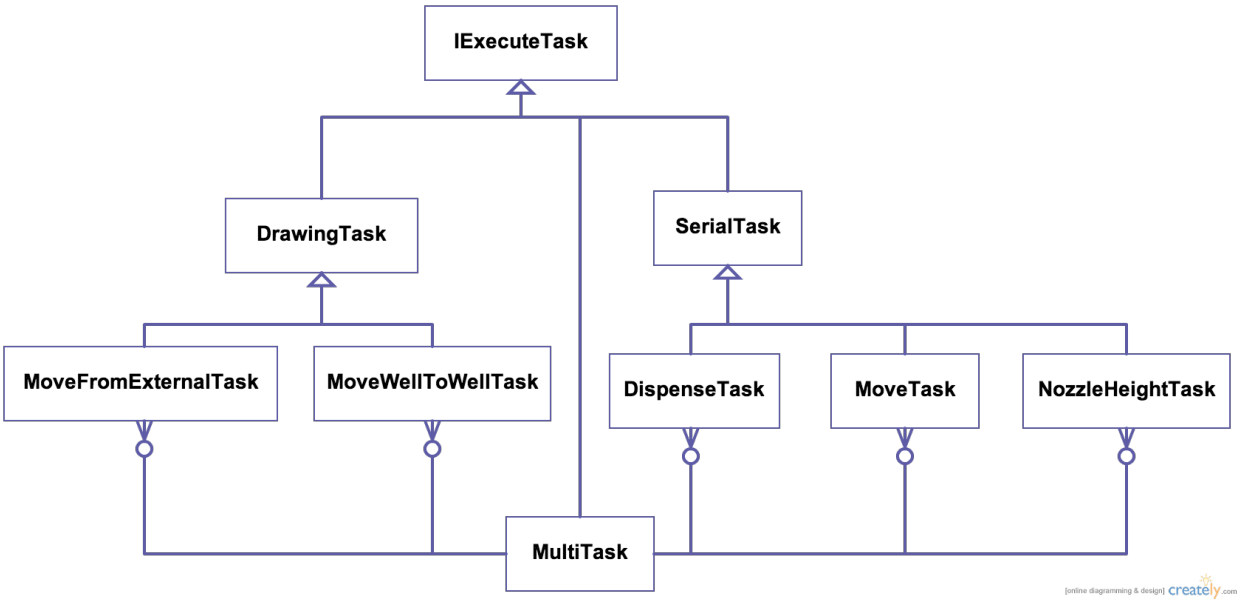


Figure 4.4: Task model hierarchy

When the DrawVisitor is applied to a DrawingTask, it forces the task to draw itself to the screen, and when it is applied to a SerialTask, nothing happens. In this way, abstract tasks consisting of multiple steps can be drawn, while raw tasks can be ignored. Conversely, when a DeconstructVisitor is applied to the current stage, the execution tree is parsed, adding serial tasks to an execution queue while doing nothing with DrawingTasks that are encountered (we do not want to give the Arduino any drawing information).

For instance, the view may send information that the task model recognizes as a "MoveWellToWellTask". This task is recognized to consist of the following subtasks:

1. move to well 1
2. lower the appropriate distance based on the well 1's dimensions
3. remove given amount of fluid
4. raise same distance
5. move to well 2
6. lower the appropriate distance based on well 2's dimensions
7. dispense given amount of fluid
8. raise same distance

By using an abstract task like this, though, the user only has to create one task to accomplish all 8 of these tasks. This task is populated with its sub-tasks and immediately put in this stage's task list. When a request to draw tasks is made, this abstract MoveWellToWellTask is drawn on the screen, resulting in a line between the wells where fluid will be moved.

When the device is run and must actually move the fluid, this task is broken down into its 8 sub-tasks and sent to the Arduino.

4.1.7 External Communication Model

This model handles all communication with external programs using server sockets. When the application is started, a socket is opened and can be populated by any external program that knows the port number. MATLAB is one such program, but any language that handles socket communication could connect and send data objects. The communication protocol is JSON, simply a regularized way to format data as strings. JSON is the most popular way to send data between programs, and so it has parsers in nearly every language.

When a task is received by this model, it gives it directly to the task model, where it is added to the execution queue and acted upon just like any other task. In this way, users can dynamically alter the program while it's running through their own experiment's program, adding and changing tasks the device is completing as data is collected on their end.

4.1.8 Serialization Model

This last model handles all saving to and loading from data files. As of now, it is able to save plate specifications and workflows, though it is organized in such a way that it can be easily extended to save additional types. When a request is made to save one of these types, this model serializes the objects and writes them to the appropriate directory of the data folder using a file name specified in the view. These saved types can be accessed via a request to read the saved files, given the same file name. Whenever a change to the data folders is made, the view requests a complete update of files currently in the appropriate folder, populating its drop-down lists with the results. In this way, whenever a file is saved or deleted, the change is immediately seen by the user. When a request to save workflow is received, this model communicates with the task and plate models to obtain a copy of their current task queue and plate ArrayList, respectively. Conversely, when a request to load a workflow is received, this model deserializes the one with the appropriate name, forces the task and plate models to load the deserialized task queue and plate ArrayList, and forces an update so that this change is immediately reflected in what the user sees.

4.2 Hardware

4.2.1 Overview

In order to accomplish the task of low volume fluid handling the hardware is composed of a frame on which a carriage moves in the X-Y plane, much like the tip of a laser cutter, a syringe pump, and a set of valves, all controlled by an Arduino.

Each component was selected to be easily manufactured or purchased without the need for a machine shop, expensive shipping, or quote requests for obscure parts. In addition, the distribution of build instructions and modifiable models allows the device to be tailored to specific needs. It is important to note that although the device does not receive G-code like

most Computer Numerical Controlled (CNC) machines, the code is freely distributed and documented for any purpose, including modification for use with G-code if desired.

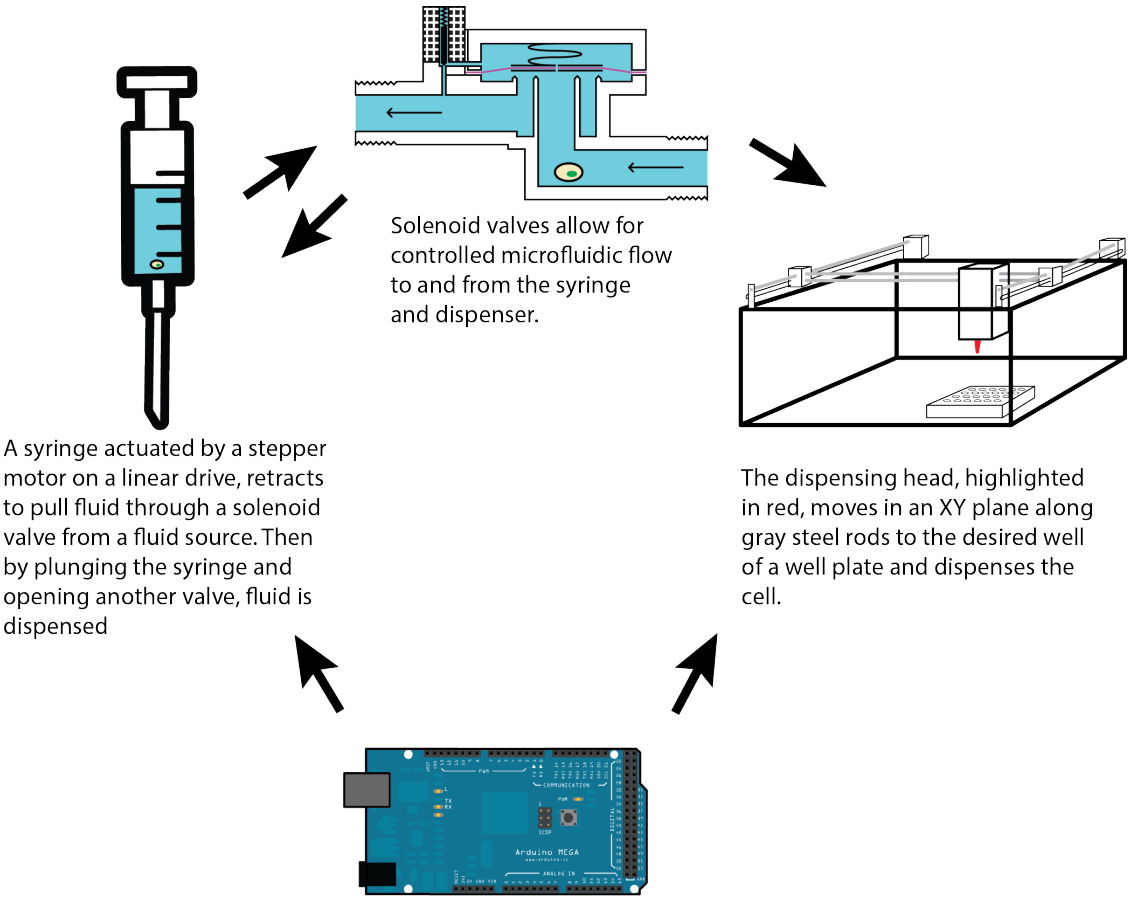


Figure 4.5: Flowchart depicting the interaction of RoboSquirt's different components

Above is a flowchart depicting how the syringe pump, valves, dispenser and Arduino interact.

4.2.2 Syringe Pump

In our initial assessment and research on microfluidic pumps, we believed that an in-line peristaltic pump would provide the greatest level of accuracy while minimizing harm to cells with shear or puncture from moving parts. Unfortunately, even with the smallest easily accessible pump on the market, we experienced fluid accumulation and wildly inconsistent flow rates and volumes. Variations stemmed from basic sources of error such as the height and position of the pump, fluid source and valves, demanding a reassessment of the pumping mechanism.

As a well proven fluid handling approach, syringe pumps promised an inexpensive method of transporting fluid that might be free of the kinds of errors experienced with the peristaltic pump. Our design is based on one found on Thingiverse for a gel extruder designed for use with syringes and tubing of the same diameter as that found on our device (Figure 4.6.

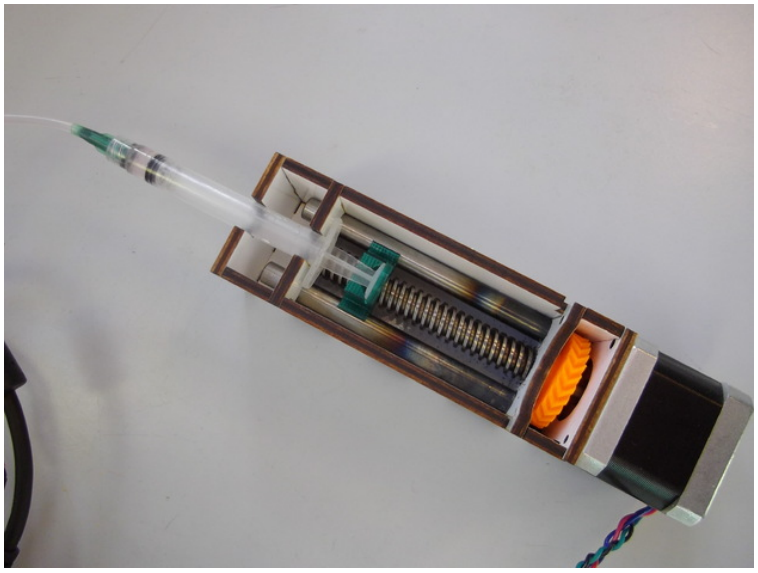


Figure 4.6: Syringe pump prototype from Thingiverse on which RoboSquirt will base pump design

The device is constructed using laser cut plywood and two 3D printed parts. We modified the original design to work with the A4988 stepper driver and NEMA14 motor used in the dispenser of the device. It was impossible to find a low-volume supplier for the originally specified 12mm drive requires a minor change to the 3D printed syringe actuator. The original build instructions also request lathe work on the lead screw which is not necessarily accessible to all users, so the gear that interfaces with the motor and screw must be modified to accommodate simpler 3mm set screws. We believe that these modifications will not inhibit the functionality of the device and make it easier to assemble. The accuracy of stepper motor drives should also allow us to achieve the volumes we wish to dispense.

The only drawbacks for the design are the flex observed when force is applied to the plunger of the syringe that can delay or prevent actuation of small volumes, and continuous flow is impossible as the syringe must be retracted and depressed repeatedly.

4.2.3 Dispenser

The design for the dispenser stems from the Splotbot, which uses a fairly standard Cartesian system for determining the position of the dispensing head; however, the area over which the dispenser can move has been reduced to fit on a microscope table to fulfill our sponsor's needs. The carriage also differs greatly by moving the fluid actuating mechanism off of the device to accommodate the larger, more accurate linear stepper drive. The nozzle still lowers towards the plate in order to reduce splash and cell trauma.

Complex shapes such as the motor mounts, bearing holders, and servo arm that would be difficult to machine are 3D printed easily and inexpensively and, as previously mentioned, and can be freely changed. The support structure is composed of Makerbeam extrusions that are small, open-source components that, although somewhat expensive, are readily available and cut to size. The motors and carriers are the same as ones found on open-source robots and CNC machines; they are reliable, easy to use, and well-documented. The GT2 belts, bearings, and rods are found on many open-source 3D printers on the market. All of these readily available components do not sacrifice on reliability or accuracy as we have repeatedly placed head of the nozzle anywhere in its range within a fraction of a millimeter.

4.3 Electronic Circuitry

The device's four motors are controlled by four stepper motor driver carriers, which in turn receive signals from an Arduino Mega 2560.

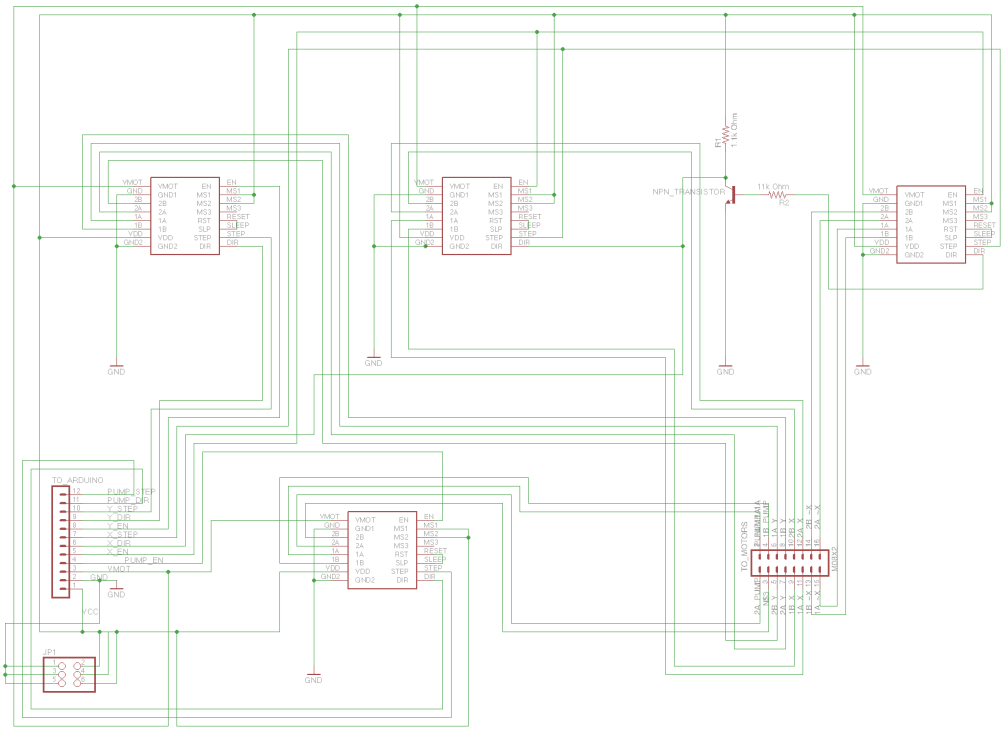


Figure 4.7: Printed circuit board schematic of Robosquirt motor driver circuitry

We use Pololu A4988 stepper motor driver breakout boards in our implementation to control the step (precision) and direction of the stepper motors. The motor drivers for the two X-direction motors receive the same signal from the Arduino, but since the motors move in the opposite direction, we use a PNP transistor to invert the direction of the second X-direction motor.

Chapter 5

Scaled Representations of RoboSquirt

Refer to Appendix A to view the CAD Drawings for RoboSquirt's Design.

Chapter 6

Manufacturing Cost Estimate

The cost to manufacture RoboSquirt is around $350, no matter the volume of production. A more detailed breakdown of manufacturing cost estimate can be found in Appendix B.

Figure 6.1: Estimated high and low cost ranges for low, medium, and high volumes of manufacturing of RoboSquirt.

Total Manufacturing Cost	Low Volume(1-5 devices per batch)	Medium Volume (5-100 devices per batch)	High Volume (100+ devices per batch)
Low Cost	363.51	341.38	332.37
High Cost	381.74	356.50	337.15

Chapter 7

Testing Strategies and Results

7.1 Volume Flow Rate Test: Peristaltic Pump

7.1.1 Purpose

In order to meet the target flow rate of 10 μL/s and minimum volume dispensable of 10 μL, the output of the prototype must be measured.

7.1.2 Procedure Overview

The pump was set up with a valve controller at its input. The pump's output tube fed the water into a container on a scale that is accurate to 0.0005 g at minimum. Commands were issued to the pump in milliseconds; by filling the tubing before the valve controller and issuing dispense commands, the rate at which the pump moves fluid was measured. In addition, by decreasing the number of milliseconds the pump is running, the minimum volume that the pump can consistently handle was measured. There were 21 trials.

7.1.3 Results

Peristaltic Pump

For each flow time, 7 trials were run. The minimum flowrate we achieved was 35μL/s at a command of 1000ms as opposed to the shorter 750ms (Fig. 7.1) . Data varied greatly due to inconsistencies in pump flow due primarily to accumulation within the pump itself. The pump behavior varied with tube length from the source and to the valve controller, with the height of the water source, and with pump orientation. Due to the dependence of the flow on the location of the peristaltic pump relative to the dispensing head, the peristaltic pump is not suitable for our needs.

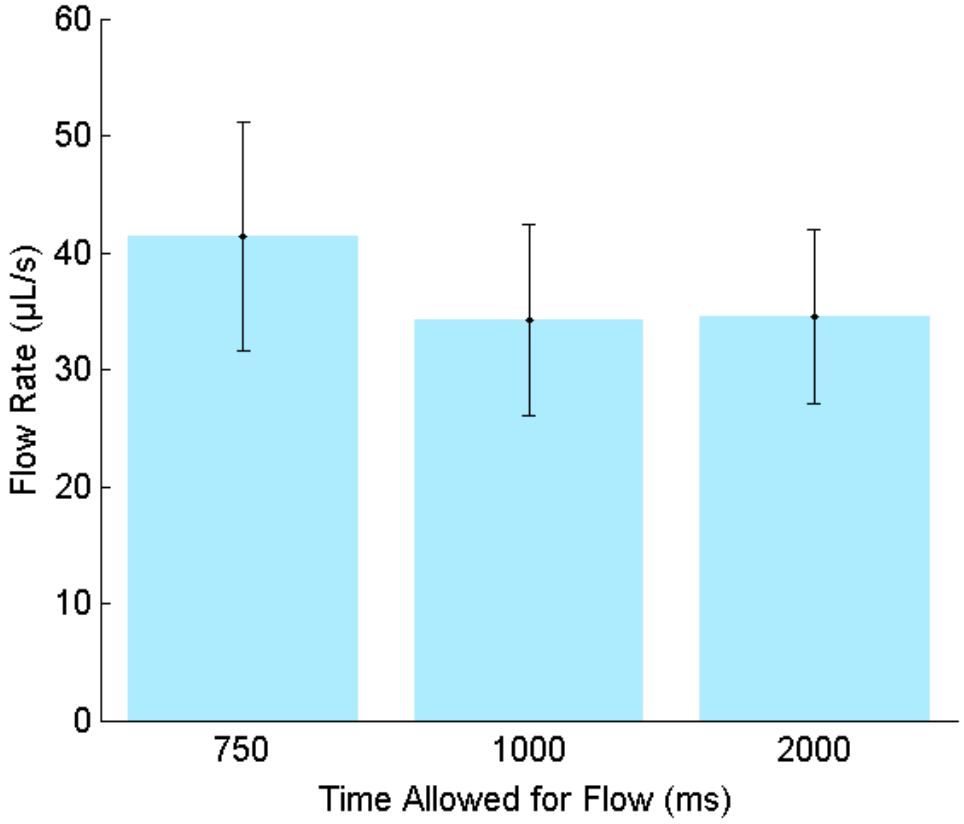


Figure 7.1: The flow rate of the peristaltic pump is inconsistent

7.2 Volume Flow Rate Test: Syringe Pump

7.2.1 Purpose

See "Volume Flow Rate Test: Peristaltic Pump".

7.2.2 Procedure Overview

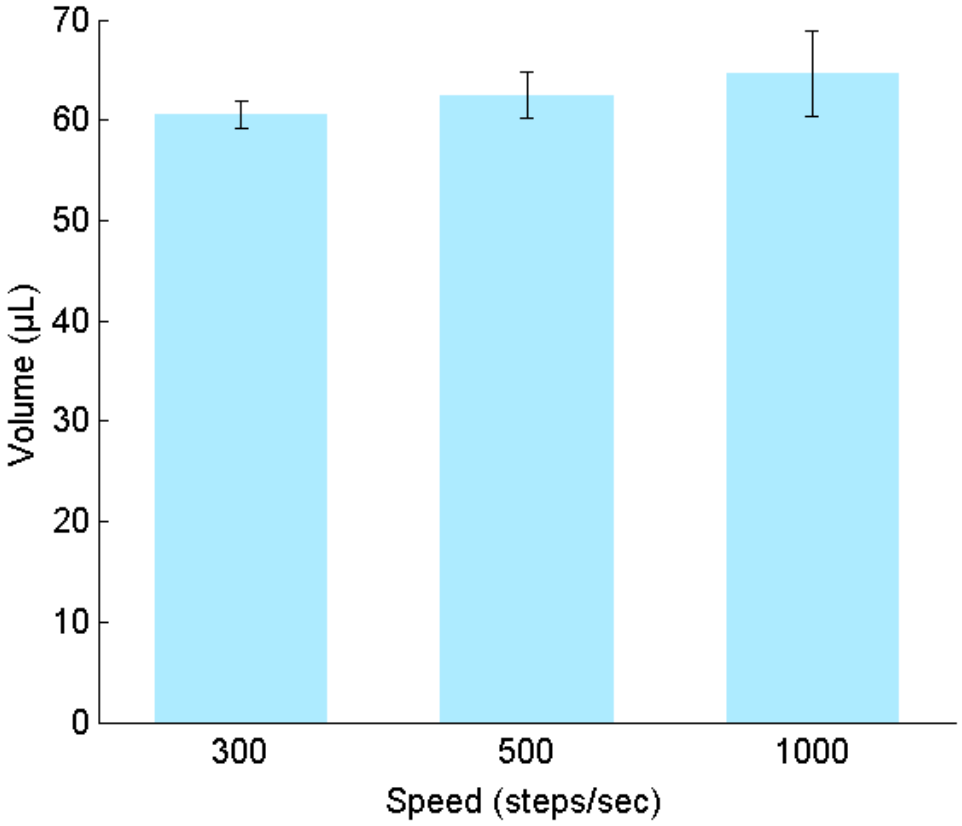
The volume of water dispensed by the syringe pump was tested as a function of speed (steps/sec), acceleration (steps/sec^2), and rotation of stepper motor (steps). Steps are arbitrary units that represent the rotation of the stepper motor used in the syringe pump, in which a greater number of steps corresponds to a greater angle of rotation.
The arduino code controlled speed, acceleration and steps. The volume of water was calculated by measuring the mass of the fluid that left the pump's output using a scale that is accurate for masses as low as 0.0005 g.

When acceleration was 500 steps/sec^2, and the rotation of the motor was 1000 steps, as speed increased, the mean volume of the fluid dispensed and the standard deviation increased (Fig 7.2).

When speed was 500 steps/sec and rotation was 100 steps, the mean volume of the fluid remained relatively constant, but the standard deviation increased (Fig. 7.4). When speed was 300 steps/sec and acceleration was 500 steps/sec^2, as the rotation increased, the volume dispensed increased. The standard deviation remained relatively stable. Five trials were conducted. When acceleration and speed were held constant, there was a high correlation between rotation and volume dispensed. However, for the device to be used in a laboratory setting, there needs to be a higher correlation. When the speed and acceleration were decreased, the device was more precise at dispensing volumes. The device is able to dispense volumes as low as 4 μ with a standard deviation of 0.1 μL.

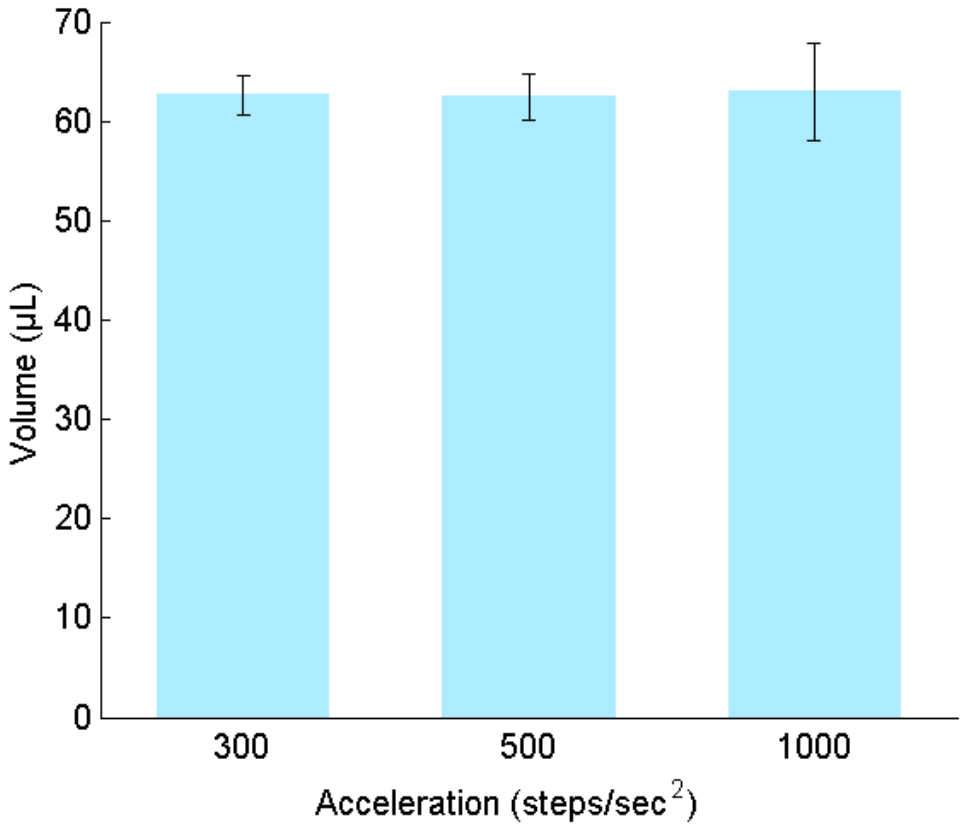
From these tests, it was apparent that increased speed and increased acceleration made the effects of variables not accounted for (e.g. pump placement, tube length, and the time it takes for fluid to stop moving) more significant.

In order to account for these effects, we are standardizing the placement of the syringe pump with respect to the nozzle and the length of tubing. With a calibration routine that is run when the tube is changed or pump is moved, we can mitigate error from these factors.



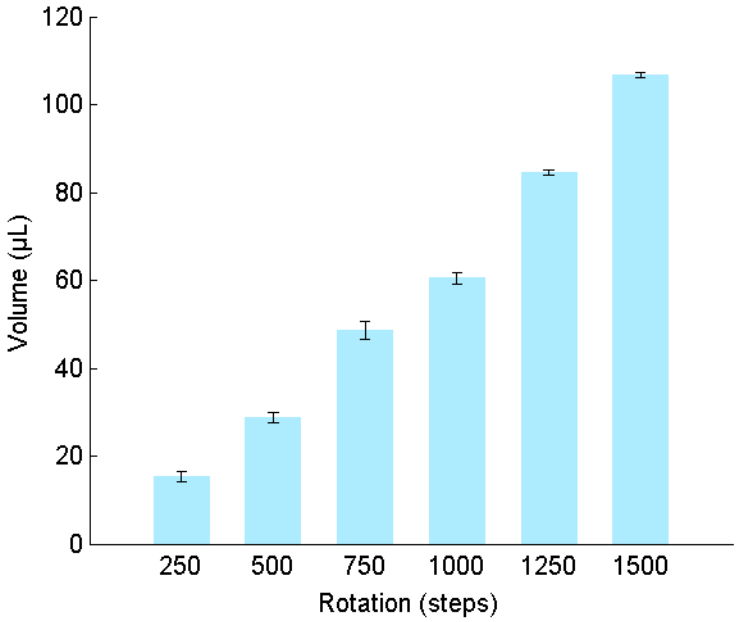
	Speed (steps/sec)		
	300	500	1000
Mean Volume (μL)	60.6	62.4	64.4
Standard Deviation (μL)	1.3	2.3	4.3

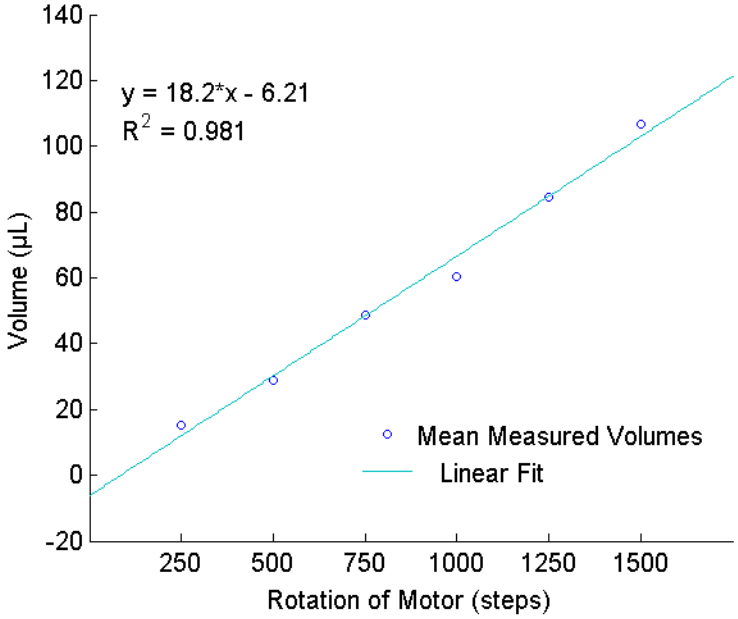
Figure 7.2: Greater speeds reduced precision in the syringe pump. Acceleration and rotation were held constant, and speed was varied. For each speed, 11 trials were run.



	Acceleration (steps/sec^2)		
	250	500	1000
Mean Volume (μL)	62.6	62.4	63.0
Standard Deviation (μL)	2.0	2.3	4.8

Figure 7.3: Greater acceleration reduced precision. Speed and rotation were held constant, and acceleration was varied. For each acceleration, 11 trials were run.





	Rotation (steps)					
	250	500	750	1000	1250	1500
Mean Volume (μL)	15.3	28.8	48.7	60.6	84.6	106.8
Standard Deviation (μL)	1.2	1.2	2.1	1.3	0.7	0.55

Figure 7.4: Greater acceleration reduced precision of the syringe pump. Speed and rotation were held constant, and acceleration was varied. For each acceleration, 11 trials were run.

7.3 Dispensing Location Test

7.3.1 Purpose

The device must accurately place the head of the arm over whatever receptacle or liquid source a user selects. The device is considered accurate if the error in each direction (x and y) measured from the center of the well is less than the distance equal to ± 5 % of the diameter of the well.

7.3.2 Procedure Overview

The arm was sent 50 mm in the the x- and y- axes, 10 times in each direction. Then, we measured the distance traveled using calipers. Then, to assess consistency over the range in motion, we sent the arm 10, 20, 30, and 40 mm in each direction, a total of 8 commands. Each time the arm reached the distance specified via the user interface, the arm returned to the origin and went to the next distance specified. For all tests, percent error and the average of the results was calculated. If the average discrepancy between the specified distances and the actual distances in the x- and y- directions was less than 5%, movement will be considered accurate.

7.3.3 Results

Table 7.1: Movement Testing Results

	Y-Distance (mm)	X-distance (mm)
	49.98	50.25
	49.96	50.18
	49.93	50.21
	49.90	50.21
	49.96	50.19
	49.92	50.38
	49.99	50.22
Average	49.95	50.23
Standard Deviation	0.03	0.07

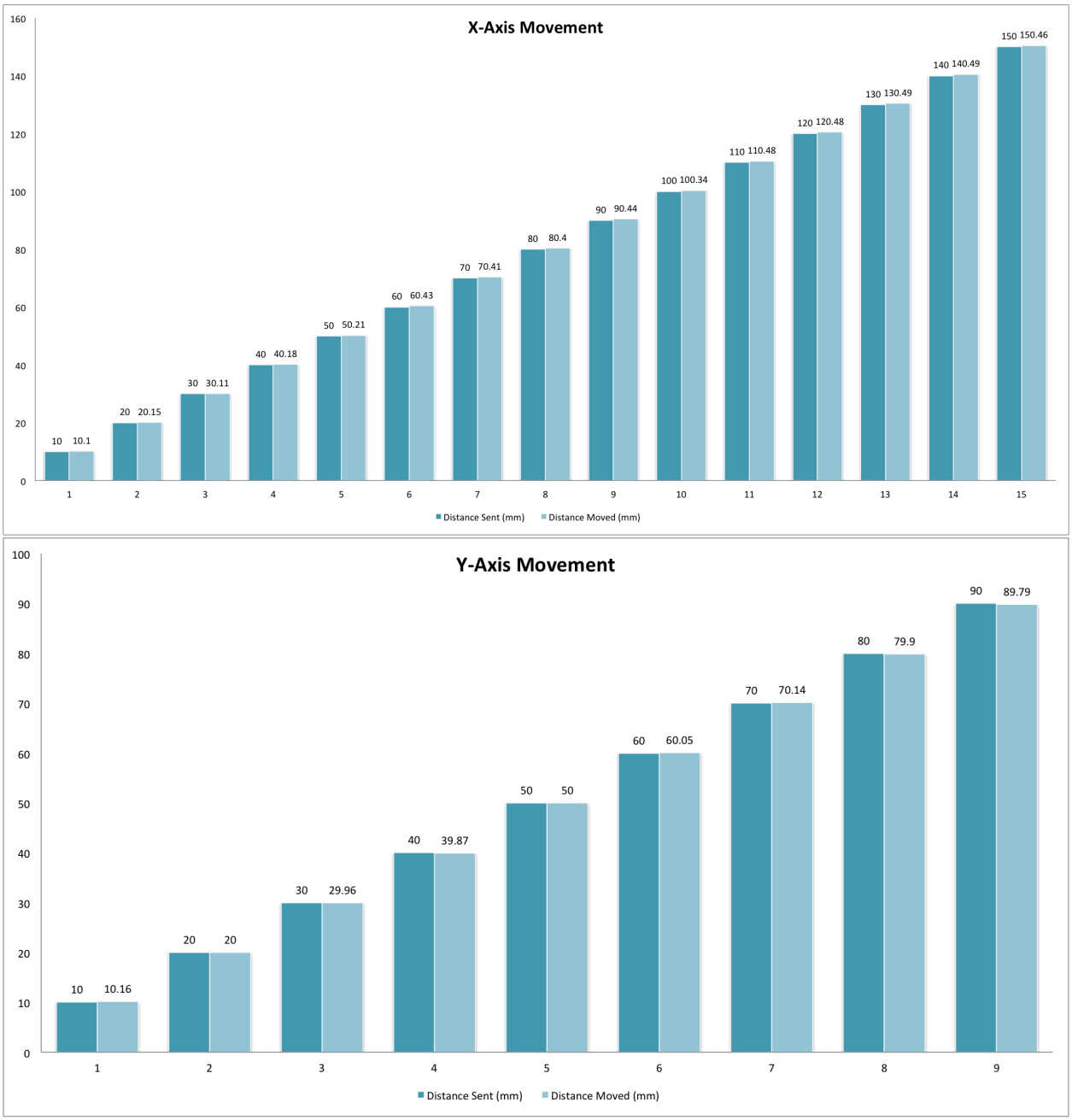


Figure 7.5: The distance between the specified value and the actual value was no more than 0.4 mm

7.4 3D Printing Test

7.4.1 Purpose

We must make our project completely accessible to all who wish to build it. We need to make sure both professional and desktop 3D printers are capable of constructing the parts generated in OpenSCAD.

7.4.2 Procedure Overview

We will print the models that are not the XY frames structural components because those had been printed on the original Splotbot design by a desktop printer. The desktop printer that we will use is a Solidoodle 3. We will qualitatively determine that functional components and fastener holes are reproduced accurately by replacing the parts made by a professional printer with the parts made by a desktop printer and checking that the parts fit.

7.4.3 Expected Data

3D printed components that fit RoboSquirt as well as the professional printed components.

7.4.4 Results

Other than the nozzle, the core components of the device, including the servo holder and arm printed accurately. The components printed by the desktop printer fit the device. The nozzle will require a redesign to be desktop printer friendly as the support structure proved impossible to remove.

7.5 Future Testing

7.5.1 Quantification of Cells

Light Microscopy

Since the concentration of cells being transported by RoboSquirt will be low, each well in a multi-well plate should not have more cells than can be counted by hand. The entire well can be scanned by a microscope, and the amount of cells can be recorded. Advantages of light microscopy include cost and time. This method does not require special reagents, and there are no waiting periods. Furthermore, it does not require special expertise. The biggest disadvantage of light microscopy in relation to RoboSquirt is inaccuracy. The cells will most likely not have had enough time to adhere to the surface after being deposited in a well. This means that the cells can float outside of the field of view of the microscope as the plate is being shifted. If the cells were given enough enough time to adhere to a plate, the cells may have have enough time to divide, thereby obscuring the results. Furthermore, other particles such as dust may be mistaken for cells if the cells are not deposited in a sterile environment. The results would depend on how much fluid is in the well, and how much fluid is dispensed

by the device. For each given volume, the average cell concentration would be computed as well as the standard deviation.

7.5.2 Determining Cell Viability

Trypan Blue Assay

The Trypan Blue assay distinguishes between living and dead cells by assuming that living cells will not take up Trypan blue. Cells are counted using a hemocytometer. The data would be in the form of a table showing the mean amount of cells as well as the standard deviation. This method is prone to human bias. However, because our device is intended to dispense low volumes of cells, the accuracy of this test will suffice[6].

Chapter 8

Summary and Recommendations

In its current iteration, RoboSquirt has a strong basis for a low-cost solution for automated liquid handling. As stated by our design criteria and objectives, Our device must:

- Distribute fluid to a specified location on a well plate.
- Dispense 10 μL into wells.
- Dispense a single cell into a well.
- Cost less than $250.
- Aspirate and dispense fluid.
- Communicate with programs in other languages.

To address these needs, our design uses:

- Highly accurate stepper motors on an XY frame to achieve movement to .03mm precision.
- A syringe pump to achieve as low as 4.0 μL volumes with a precision of 0.1 μL.
- Currently, the apparatus to isolate cells has not been fully developed such that this could be tested.
- The syringe pump motor affixed to a lead screw and 3D printed nut to pull and depress the syringe plunger for bidirectional flow.
- Java, an open source and widely used programming language.

However, from the beginning, RoboSquirt was designed to be modified, and even now is not in the most optimal form. All improvements to the design should bear in mind the following:

- Simplicity: RoboSquirt was designed to be built without specialized knowledge of a machine shop and using increasingly available tools in the maker revolution such as a 3D printer and laser cutter.

- Openness: All non-printed or laser cut components are meant to be accessible without the need to place quote requests, for association with a firm or institution, to hunt for an obscure source, or to place bulk orders. In addition, we strongly encourage that improvements to the main design or changes to include new capabilities be shared online.

- Functionality: Although it can be built in a garage, RoboSquirt is designed to compete with counterparts at far higher price points.

For those hoping to undertake this project or something similar, searching for solutions that are developed during the course of the project can provide ideas for improvements to the design. For example, the syringe pump is based on a Bowden syringe extruder found on Thingiverse made by user JelleAtProtospace; however, ideas from other syringe extruders, such as the Cellstruder, may also work. In order to better understand the nature of open source projects, good suppliers from which to find parts, and places from which ideas can be drawn, becoming familiar with the RepRap Project, Thingiverse, and other sites. Certain features of RoboSquirt already require changing for improvement:

- The nozzle model should be modified to be more desktop printer friendly.

- The nozzle can be redesigned for modularity with larger gauge tube.

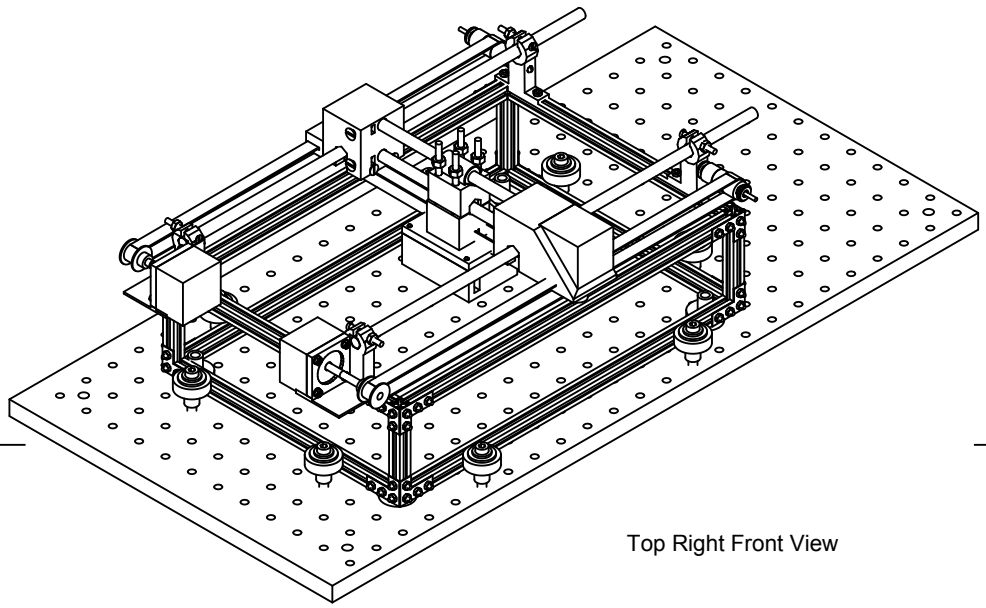
- The motor carriage along the X-axis should be designed to more easily incorporate the timing belts without use of epoxy.

- The Y-axis needs a more permanent solution on the idler than a screw through a GT2 pulley.

Chapter 9

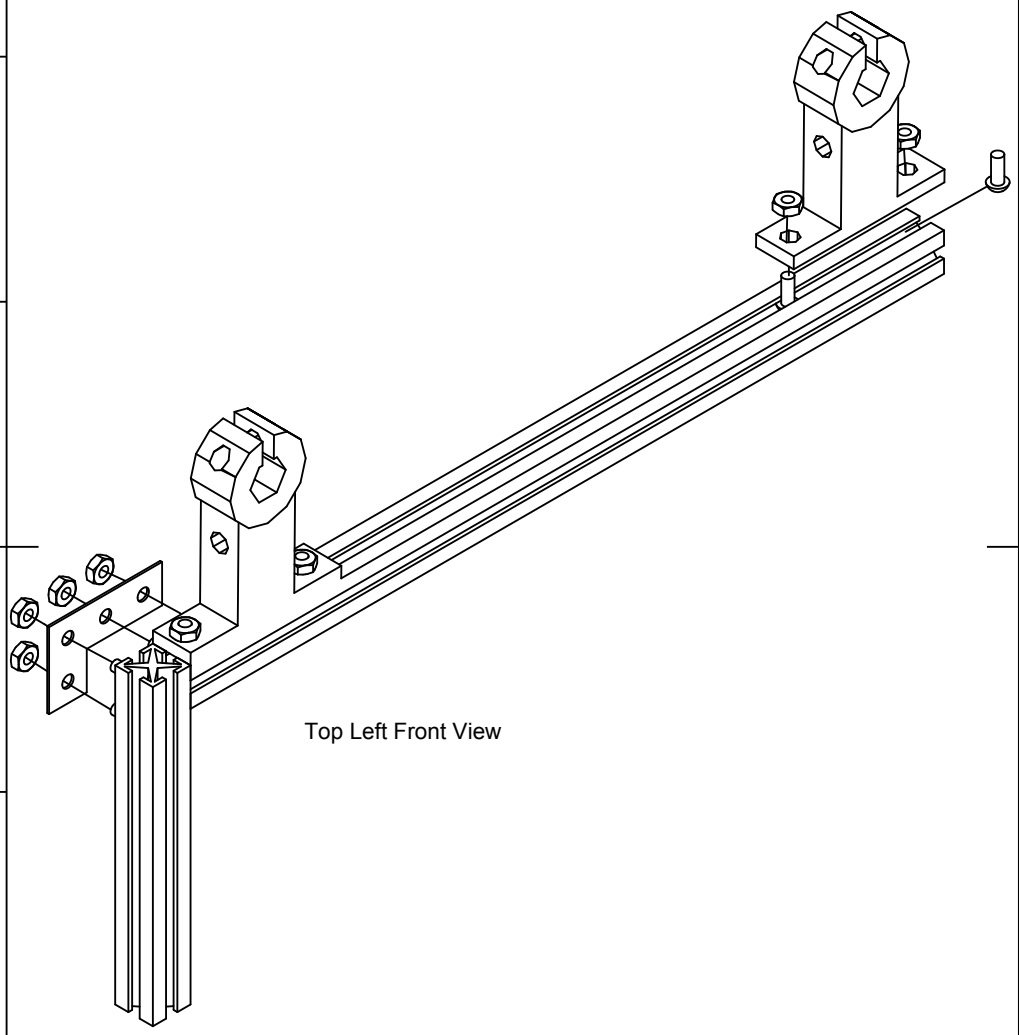
Appendix

9.1 Appendix A: CAD Drawings



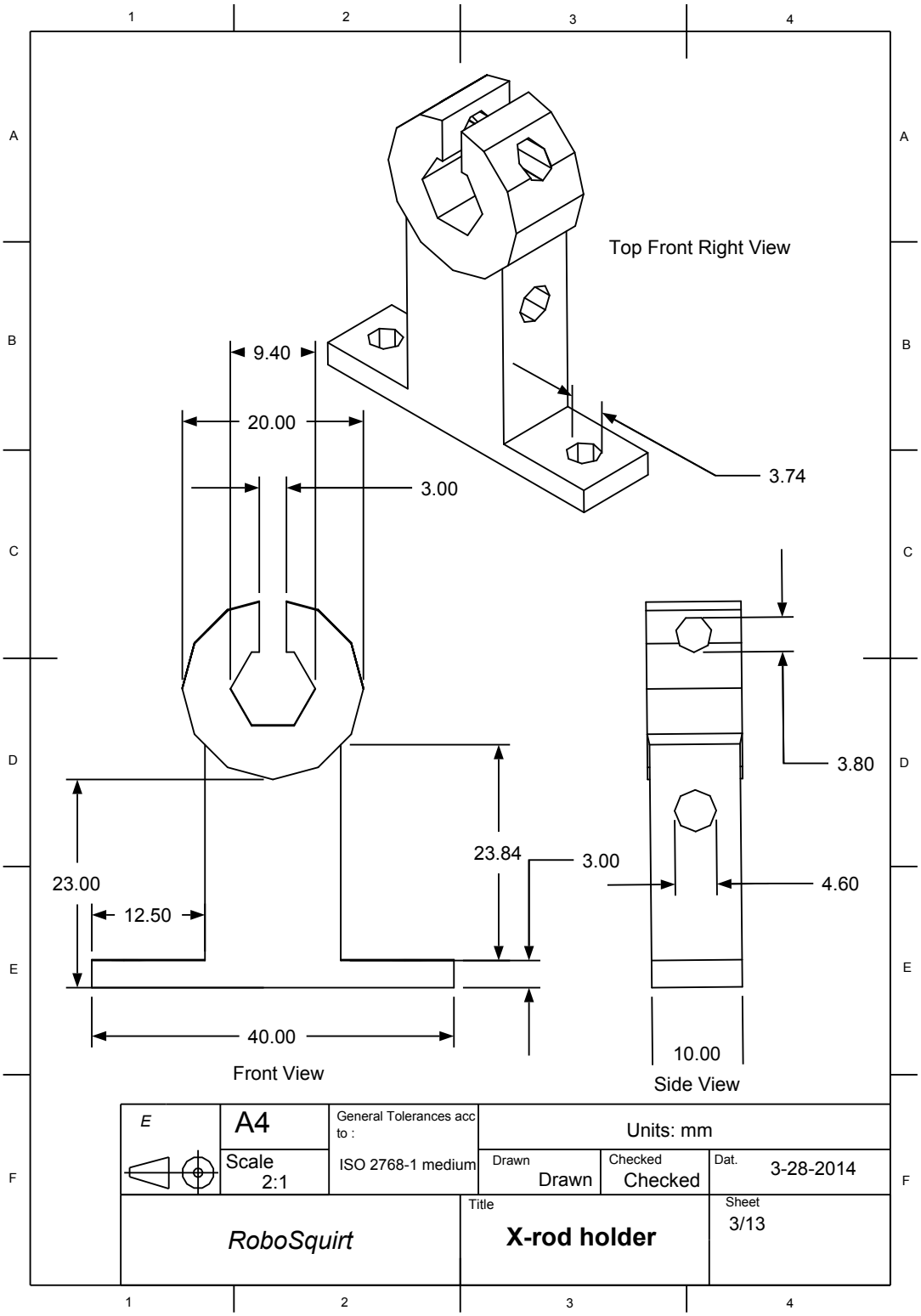
Top Right Front View

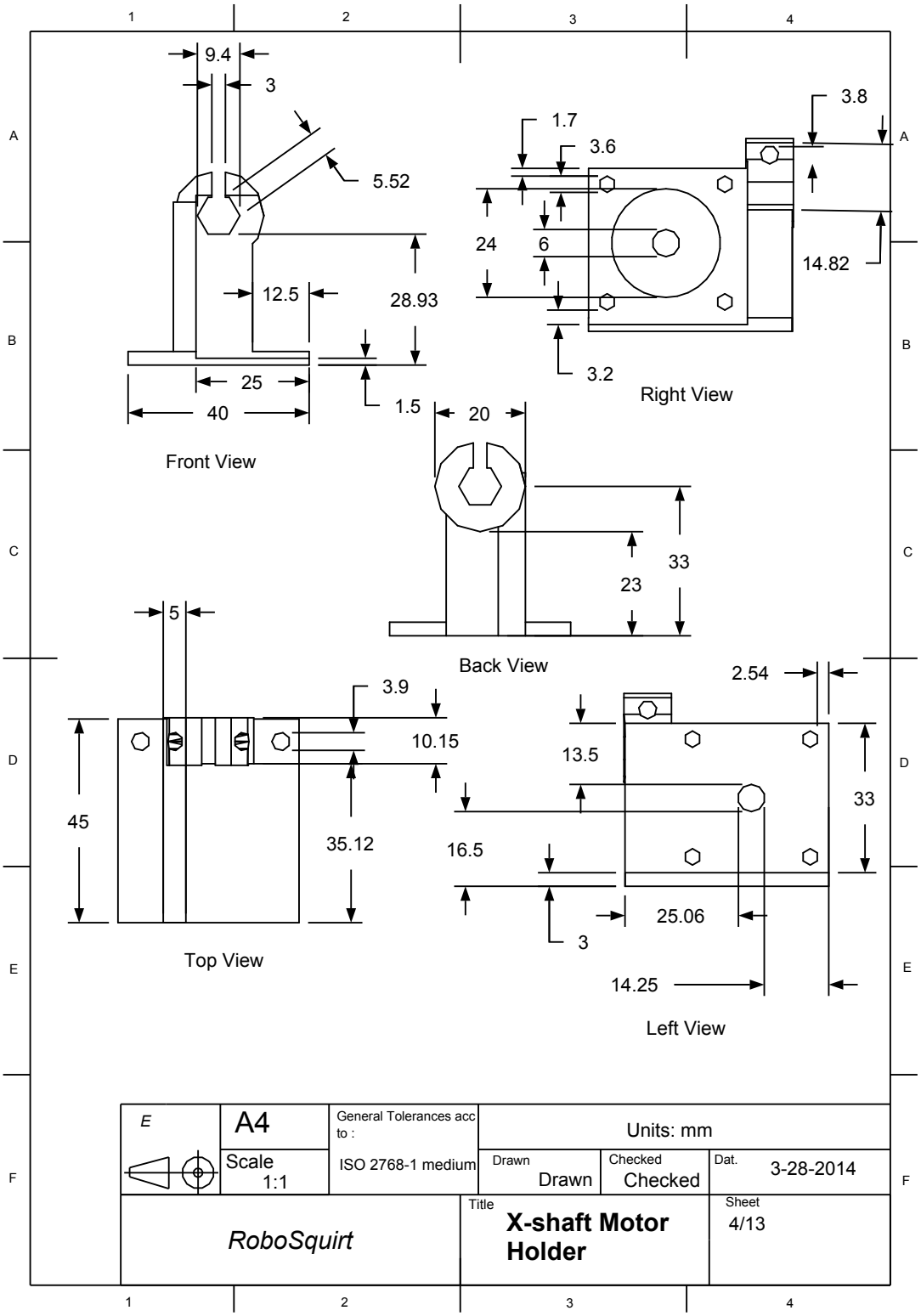
E	A4	General Tolerances acc to :	Units: mm			
	Scale 1:3.5	ISO 2768-1 medium	Drawn Drawn	Checked Checked	Dat. 3-28-2014	
RoboSquirt		Title **RoboSquirt Fluid-Handler**			Sheet 1/13	

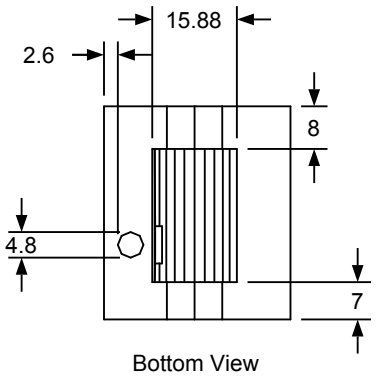


Top Left Front View

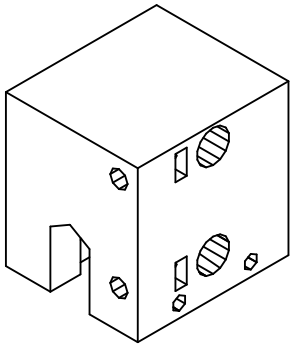
E	A4	General Tolerances acc to :	Units: mm			
	Scale 1:1	ISO 2768-1 medium	Drawn Drawn	Checked Checked	Dat. 3-28-2014	
RoboSquirt		Title **RoboSquirt Fluid-Handler**			Sheet 2/13	


Top Front Right View
9.40
20.00
3.00
3.74
3.80
23.84
3.00
4.60
23.00
12.50
40.00
10.00
Front View
Side View
E
A4
Scale 2:1
General Tolerances acc to :
ISO 2768-1 medium
Units: mm
Drawn
Drawn
Checked
Checked
Dat.
3-28-2014
RoboSquirt
Title
X-rod holder
Sheet
3/13

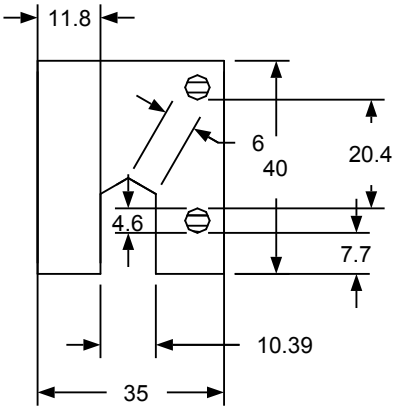

9.4
3
5.52
12.5
28.93
25
40
1.5
Front View
1.7
3.6
3.8
24
6
14.82
3.2
Right View
20
33
23
Back View
5
3.9
10.15
45
35.12
Top View
2.54
13.5
33
16.5
25.06
3
14.25
Left View
E
A4
Scale 1:1
General Tolerances acc to :
ISO 2768-1 medium
Units: mm
Drawn
Drawn
Checked
Checked
Dat.
3-28-2014
RoboSquirt
Title
X-shaft Motor Holder
Sheet
4/13



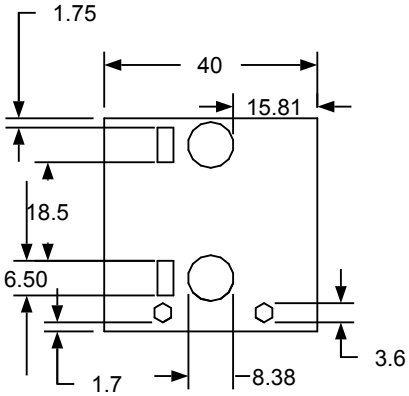
Bottom View



Top Right Front View

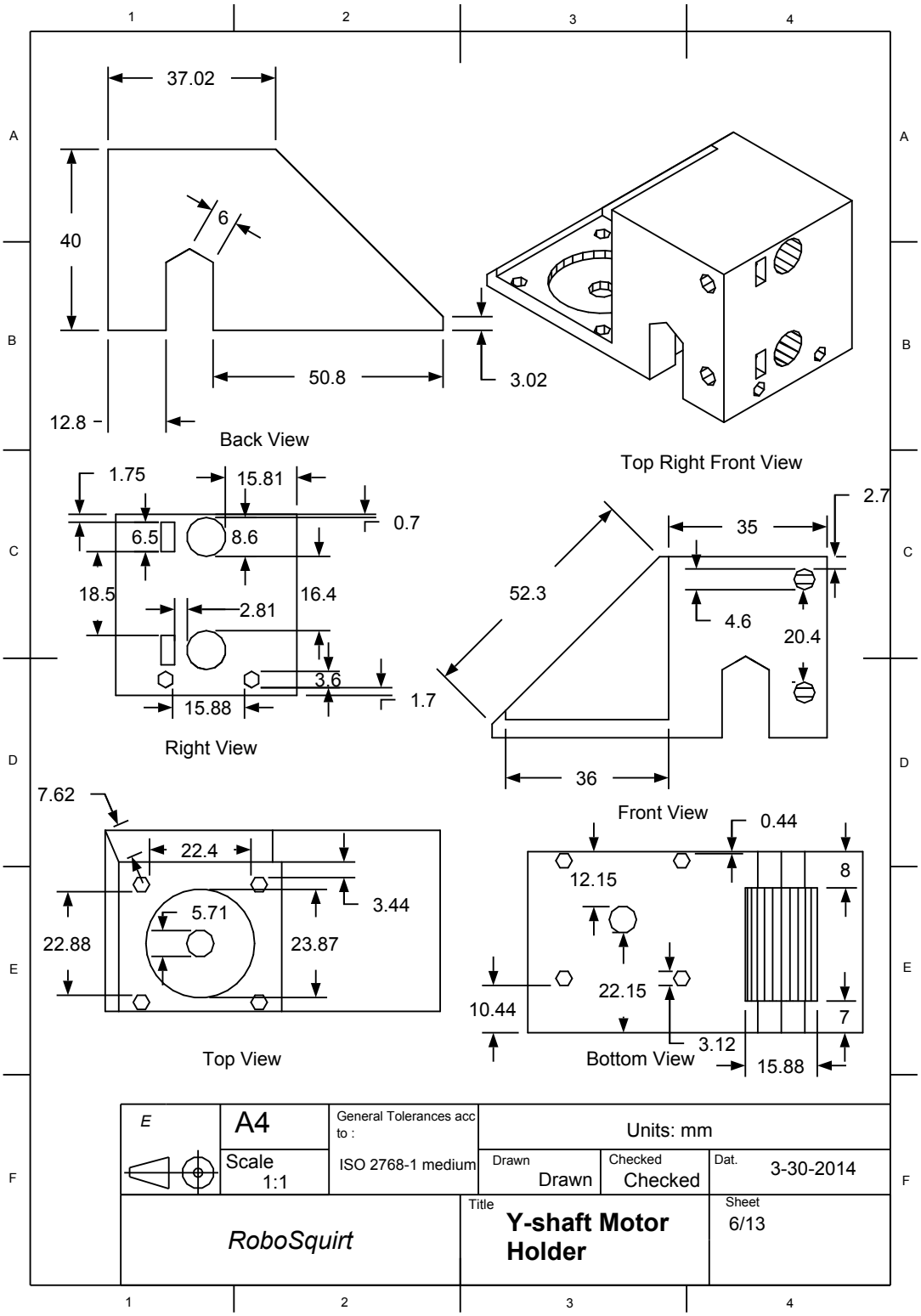


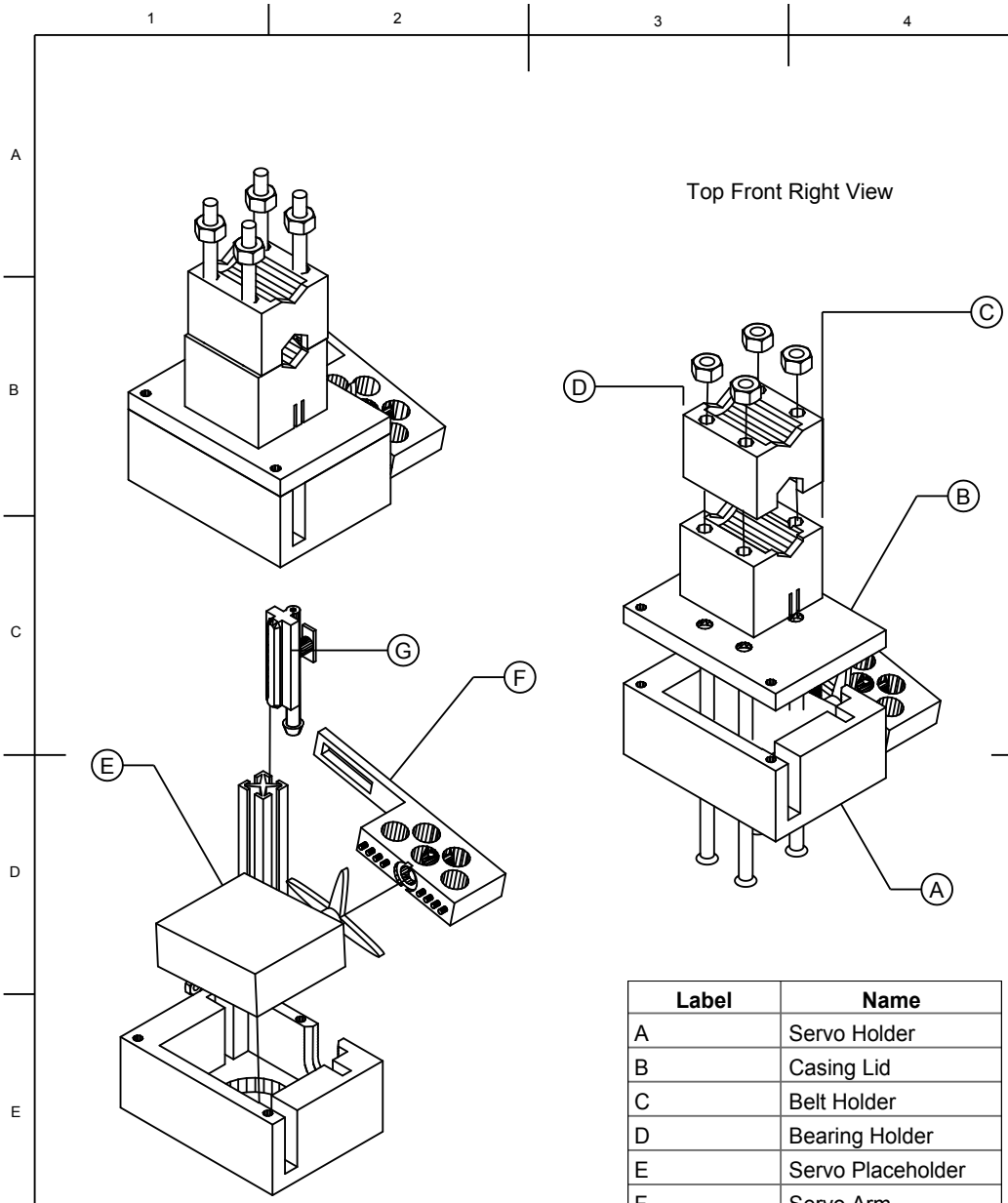
Front View



Right View

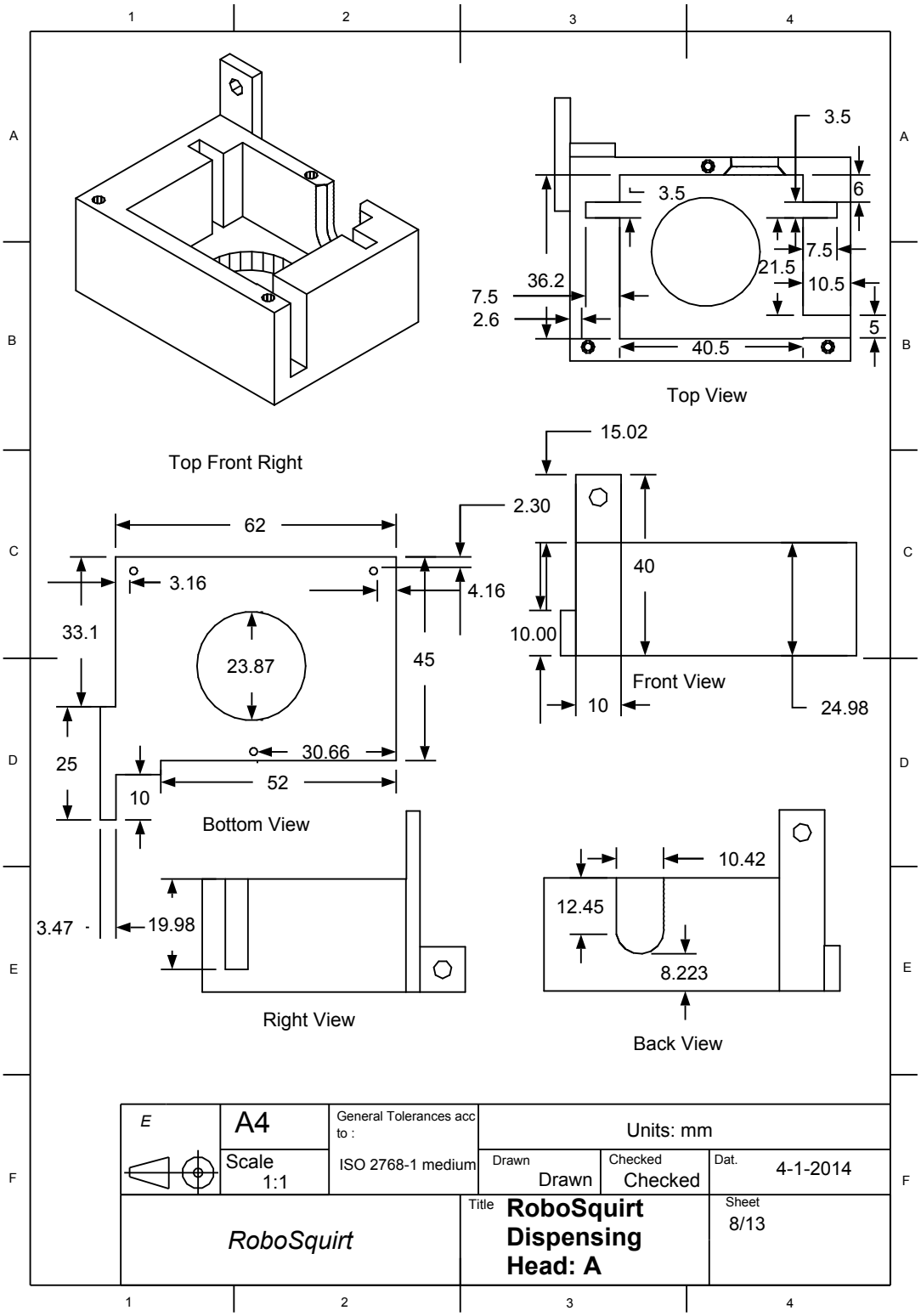
E	A4	General Tolerances acc to :	Units: mm		
	Scale 1:1	ISO 2768-1 medium	Drawn: Drawn	Checked: Checked	Dat.: 3-28-2014
RoboSquirt	Title: **Y-axis Tether Holder**				Sheet: 5/13

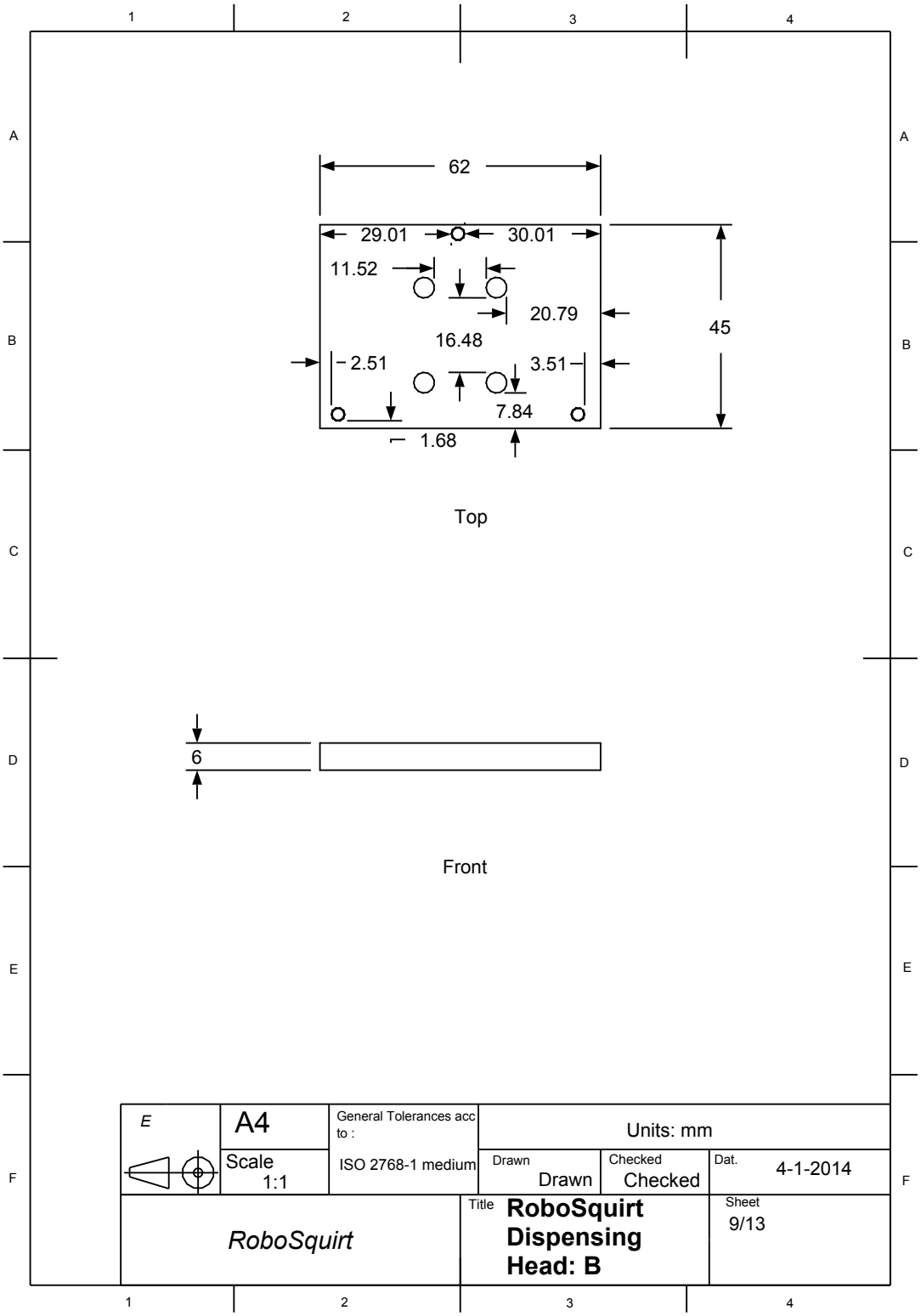

37.02
40
6
50.8
3.02
12.8
Back View
Top Right Front View
1.75
15.81
0.7
6.5
8.6
18.5
2.81
16.4
3.6
15.88
1.7
Right View
2.7
35
52.3
4.6
20.4
36
Front View
7.62
22.4
3.44
5.71
22.88
23.87
Top View
0.44
12.15
8
22.15
10.44
7
3.12
15.88
Bottom View
E
A4
Scale 1:1
General Tolerances acc to :
ISO 2768-1 medium
Units: mm
Drawn
Drawn
Checked
Checked
Dat.
3-30-2014
RoboSquirt
Title
Y-shaft Motor Holder
Sheet
6/13

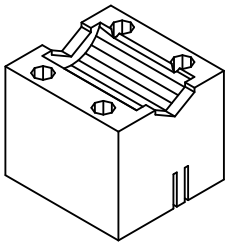


Label	Name
A	Servo Holder
B	Casing Lid
C	Belt Holder
D	Bearing Holder
E	Servo Placeholder
F	Servo Arm
G	Tube Holder

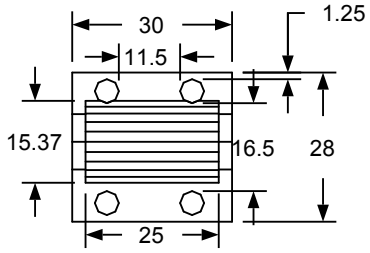
E	A4	General Tolerances acc to :	Units: N/A		
	Scale 1:1.5	ISO 2768-1 medium	Drawn: Drawn	Checked: Checked	Dat.: 3-30-2014
RoboSquirt	Title: **RoboSquirt Dispensing Head**				Sheet: 7/13


Top Front Right
3.5
3.5
6
7.5
21.5
10.5
36.2
7.5
2.6
5
40.5
Top View
15.02
2.30
62
3.16
4.16
33.1
23.87
45
40
10.00
Front View
10
24.98
25
30.66
52
10
Bottom View
3.47
19.98
Right View
10.42
12.45
8.223
Back View
E
A4
Scale 1:1
General Tolerances acc to :
ISO 2768-1 medium
Units: mm
Drawn
Drawn
Checked
Checked
Dat.
4-1-2014
RoboSquirt
Title
RoboSquirt Dispensing Head: A
Sheet
8/13

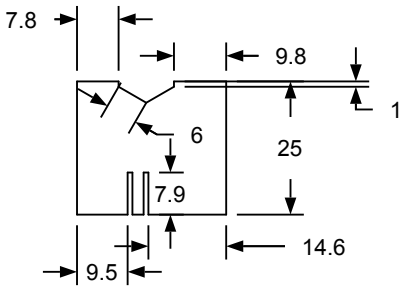

62
29.01
30.01
11.52
20.79
16.48
45
2.51
3.51
7.84
1.68
Top
6
Front
E
A4
Scale 1:1
General Tolerances acc to :
ISO 2768-1 medium
Units: mm
Drawn
Drawn
Checked
Checked
Dat.
4-1-2014
RoboSquirt
Title
RoboSquirt Dispensing Head: B
Sheet
9/13



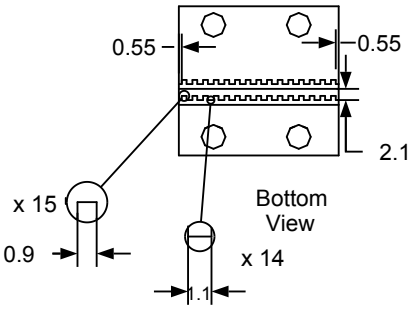
Top Right Front View



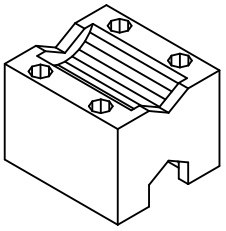
Top View



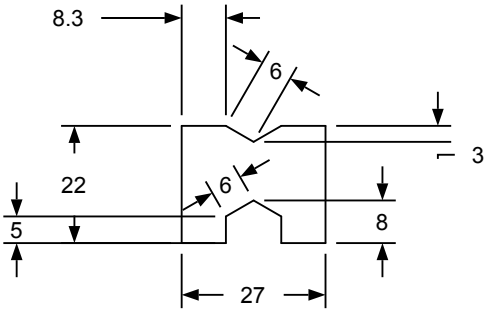
Right View



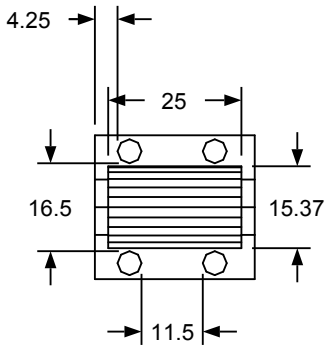
E	A4	General Tolerances acc to :	Units: mm		
	Scale 1:1	ISO 2768-1 medium	Drawn Drawn	Checked Checked	Dat. 4-1-2014
RoboSquirt	Title **RoboSquirt Dispensing Head: C**				Sheet 10/13



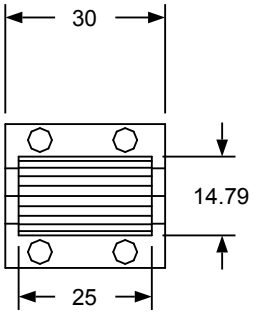
Top Front Right View



Right View

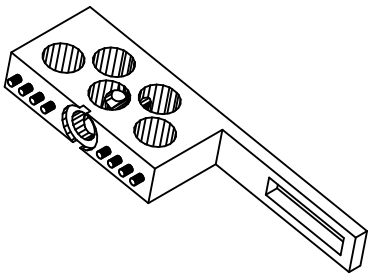


Bottom View

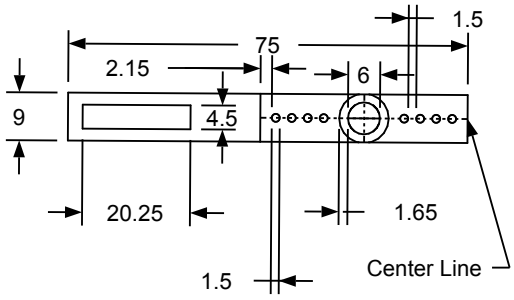


Top View

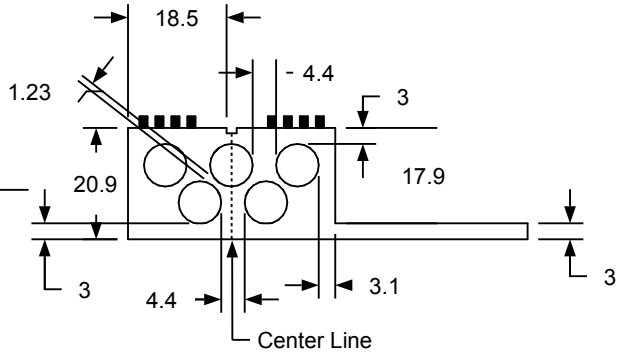
E	A4	General Tolerances acc to :	Units: mm		
	Scale 1:1	ISO 2768-1 medium	Drawn Drawn	Checked Checked	Dat. 4-1-2014
RoboSquirt	Title **RoboSquirt Dispensing Head: D**				Sheet 11/13



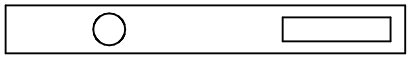
Top Front Right



Front View

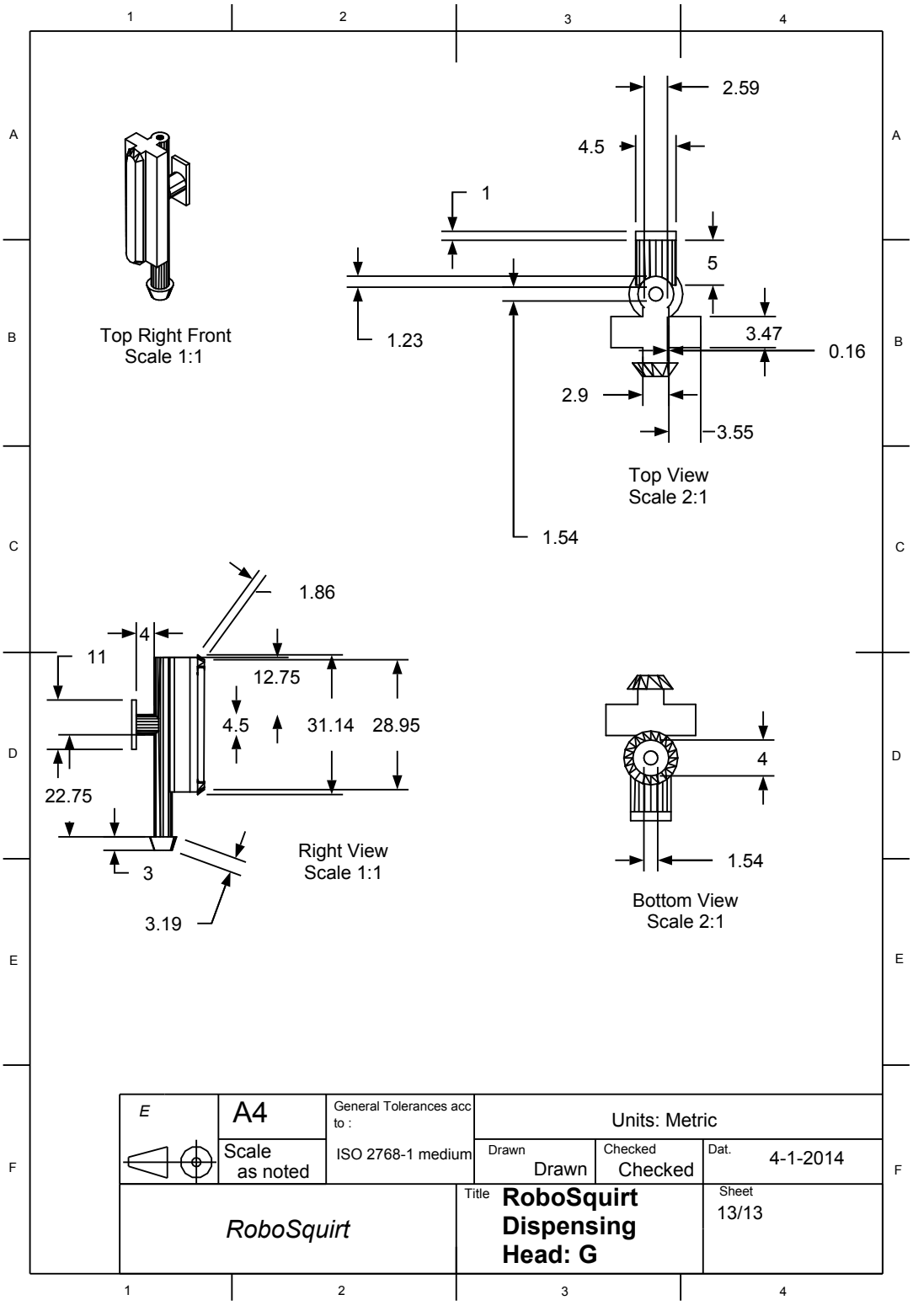


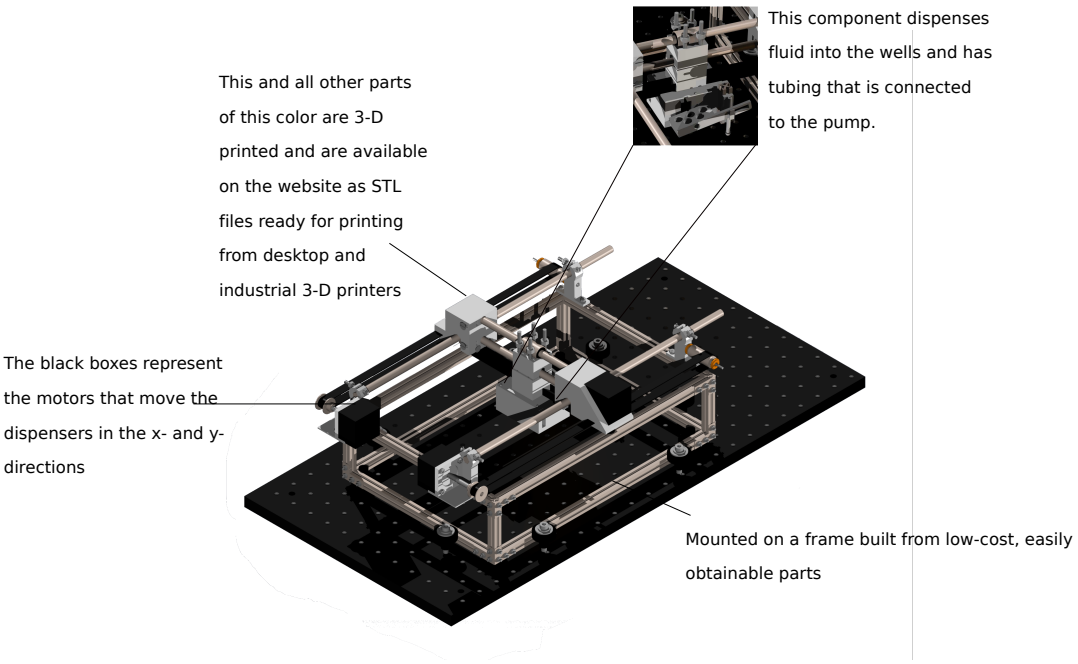
Top View



Back View

E	A4	General Tolerances acc to :	Units: mm		
	Scale 1:1	ISO 2768-1 medium	Drawn Drawn	Checked Checked	Dat. 4-1-2014
RoboSquirt	Title **RoboSquirt Dispensing Head: F**		Sheet 12/13		


Top Right Front
Scale 1:1
2.59
4.5
1
5
1.23
3.47
0.16
2.9
3.55
Top View
Scale 2:1
1.54
1.86
4
11
12.75
4.5
31.14
28.95
22.75
3
3.19
Right View
Scale 1:1
4
1.54
Bottom View
Scale 2:1
E
A4
Scale
as noted
General Tolerances acc
to :
ISO 2768-1 medium
Units: Metric
Drawn
Drawn
Checked
Checked
Dat.
4-1-2014
RoboSquirt
Title
RoboSquirt
Dispensing
Head: G
Sheet
13/13


This component dispenses fluid into the wells and has tubing that is connected to the pump.
This and all other parts of this color are 3-D printed and are available on the website as STL files ready for printing from desktop and industrial 3-D printers
The black boxes represent the motors that move the dispensers in the x- and y-directions
Mounted on a frame built from low-cost, easily obtainable parts

9.2 Appendix B: Projected Manufacturing Costs

Item	Low Volume(1-5 devices per batch)	Medium Volume (5-100 devices per batch)	High Volume (100+ devices per batch)
Makerbeam Kit	$118.18	$118.18	$118.18
3x 8mm Steel Rod	$14.43/rod	$14.43/rod	$14.43/rod
4x NEMA 14 Stepper motors	(1-2devices) $15.95/motor	$14.36/motor	$14.36/motor
	(3-5) $14.36/motor		
Servo	$12.95	(6-9) $12.95	(100+) $10.36
		(10-99) $11.66	
GT2 Belt and 2 pulleys	$15.8	$15.8	$15.8
LM8UU Bearings	$13.52/12 pack	$13.52/12 pack	$13.52/12 pack
608ZZ Bearings	(1-3 devices) $10.37/8 pack	(6-24) $19.49/16 pack	(100-199) $53.95/100 pack
	(4-5) $19.49/16 pack	(25-99) $53.95/100 pack	(200+) $345/800 pack
Arduino (Least expensive shipping)	$17.99	(6-9) $17.99	$13.85
		(10-49) $16.19	
		(50-99) $15.29	
4x A4988 Drivers	(1 device) 11.95/chip	(6) $9.95/chip	(100-124) $7.95/chip
	(2-5) $9.95/chip	(7-25) $8.95/chip	(125+) $6.95/chip
		(26-100) $7.95/chip	
10 ft IDxOD .023"x.039" tubing (Sci Com Inc.)	$14	$7.99/10ft	$7.50/10ft
12"rod	$16.38	$16.38	$16.38
Hardware cost	$2	$1	$0
Printed parts	Strongly Variable	Strongly Variable	Strongly Variable
Total			
High Cost	381.74	356.496666666667	337.151875
Low Cost	363.506666666667	341.381875	332.373984962406

Figure 9.1: The above table details the breakdown of costs for low, medium, and high volume manufacturing.

Bibliography

[1] The history of automated liquid handling LabAutopedia (2009).

[2] Hudson History of liquid handling (2013).

[3] Gutierrez, J. M. P. Automatic liquid handling for artificial life research Master's thesis University of Southern Denmark (2013).

[4] Liquid handling LabX October 2013.

[5] Gurevitch, D. M. Liquid handling: Theory and practice LabAutopedia (2010).

[6] Frei, M. Cell viability and proliferation Technical report Sigma Aldrich (2011).