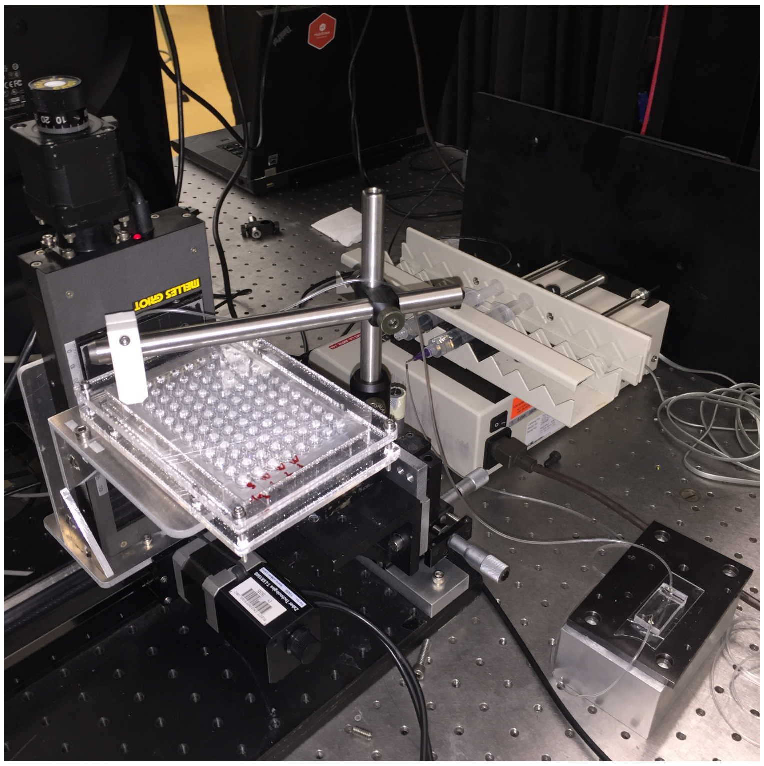
Automating Drug Injection Into Microfluidic Devices

Somesh Daga and Shamsuddin Rehmani

April 8, 2016

Project Sponsors:
Dr. Loic Laplatine
Dr. Karen Cheung

Project #30
ENPH 459
University of British Columbia



Contents

List of Figures

1 Executive Summary

This document serves as the Final Report for the project of 'Automating Drug Injection into Microfluidic Devices'. This project has been developed as part of the ENPH 459 Technical Project Course under the supervision of BioMEMS group members, Dr. Karen Cheung and Dr. Loic Laplatine. The BioMEMS group, located at the NanoSystems laboratory on the UBC Vancouver campus, develops microfluidic devices for applications in the health and environment industry.

The aim of this project was to design and automate a 3-axis linear stage system, to select and inject drugs contained on a 96-well microplate into microfluidic devices developed at the NanoSystems lab. The software requirements of the project were to create a template for defining user experiments and code for automating these experiments through the use of a Graphical User Interface (GUI). Additional objectives included the design and fabrication of mechanical components to secure the linear stages, a 96-well microplate and a selector needle.

A macro-enabled Excel workbook was created using Visual Basic, to provide users with a template to define experiments while providing features for guiding user input through color indicators, userforms and autocomplete routines. C# programs were developed for interfacing with the linear stages and a syringe pump to execute experiments and provide an estimate of the running time of experiments. A GUI was created using the .NET WinForms framework as the front-end of the drug injection system to run, stop, pause and resume experiments. Mechanical components, namely a needle holder, mounting brackets and a microplate enclosure, were fabricated through the use of water-jet cutting, laser cutting and 3D printing technologies.

The system has been tested through mock experiments and has met all of its initial objectives, allowing users to completely define experiments, load and execute them through the use of the GUI. Moreover, the GUI includes a progress bar to indicate the percentage completion of experiments. Due to workspace restrictions and time constraints, the system could not be tested under representative experiments on the scale of hours and days. As the system is integrated into practice at the NanoSystems laboratory, ongoing support will be provided by our group for any potential faults exposed through its use or feature requests required for improved performance.

In order to maintain a sterilized environment for the drugs contained, a PDMS membrane was laid across the top of the microplate. To evaluate the effect of cross-contamination of wells due to needle traversal, the currents across salt solutions and deionised (DI) water, resulting from AC 5VPP 10KHz excitations, were measured before and after running experiments. The results were inconclusive due to inadequate precision of available multimeters for concentrations below 1g/L of salt solutions. Moreover, an experiment to determine the error in estimated running time of experiments against actual running times was completed for experiments on the scale of 1 to 10 minutes. Errors for estimated times were within 1-2 seconds, however additional experiments on the scale of hours and days are yet to be performed as a test for scalability.

2 Introduction

Numerous experiments in the UBC NanoSystems Lab are being conducted where drug injection at various concentrations, durations and repetition rates on living cells or tissues are required. Such experiments typically last several days and are currently conducted manually, and therefore need to be automated. There exist commercially-sold devices to automate such experiments, however, they are limited to few channels (typically 4 to 16) and thus allow a restricted amount of drug samples to be used. To handle more drug samples for comprehensive tests, a titration plate of 96 wells can be used. Thus, the goal of this project is to create a modular system that can automate the selection of drugs from a 96-well microplate and inject them into any microfluidics setup. The proposed solution requires the use of a 3-axis linear stage system, a 96-well titration microplate, and a programmable syringe pump. The focus of this project is not on addressing the microfluidics aspects of experiments as the title may suggest, but rather on developing a platform for easy interaction with the linear stages and syringe pump such that the system can be adapted to any microfluidics setup by a knowledgeable user. The current scope of the project is to provide a platform for loading drug samples into microfluidic devices developed at the NanoSystems lab, with the potential of it being used by the BC Cancer Agency in the future.

The Project recommendation report starts with an overview of the experimental setup and outlines the system hardware used and notable features or problems developed through their use over the course of the project. Next the report provides a detailed description of the mechanical components that were designed and fabricated. The major portion of the report deals with the software implementation of the drug automation system. This includes the system specifications under which the software was developed, the features and code structure used to develop the experiment definition template (i.e a macro-enabled excel file to specify experiments), the automation and time calculation algorithms, and the GUI and its elements. The software section also talks about threads and processes required to implement the GUI and inter-process communication techniques and their limitations. Near the end of the report some quantitative results acquired from cross-contamination and time calculation experiments are provided to quantify the degree of cross-contamination and the accuracy of the time approximation code respectively. A conclusion to the project followed by the project deliverables, a financial summary of the project, further commitments to the project and recommendations on both the mechanical and software aspects are provided through the final sections of the report.

3 General Overview of The project

Figure 1 shows the complete Drug Automation Setup. The setup consists of three separate components: the 3-axis linear stage system, microfluidic device, and the programmable syringe pump. The microfluidic device will generally be placed under a microscope (not shown) for actual experiments. The aim of this setup is to allow the user to define experiments and automate drug infusion into biological samples housed in the microfluidic device.

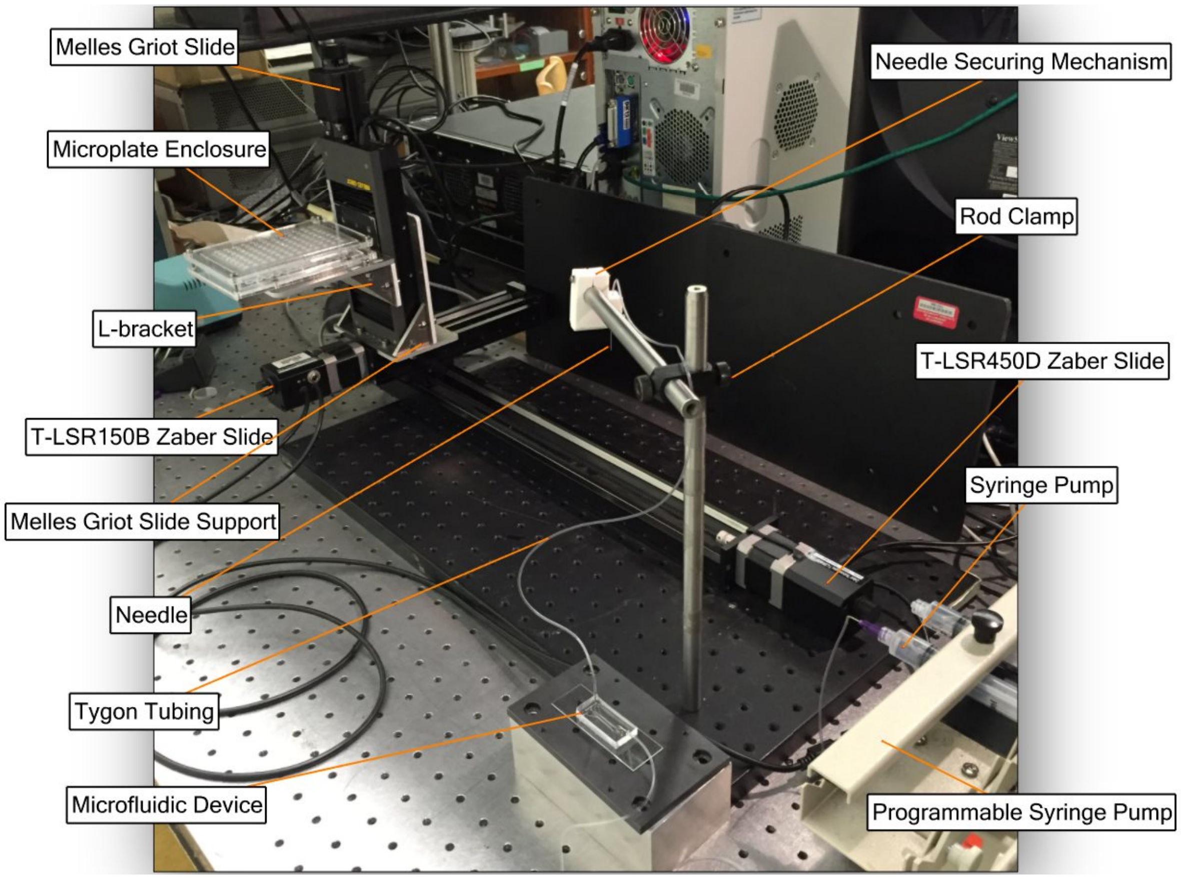
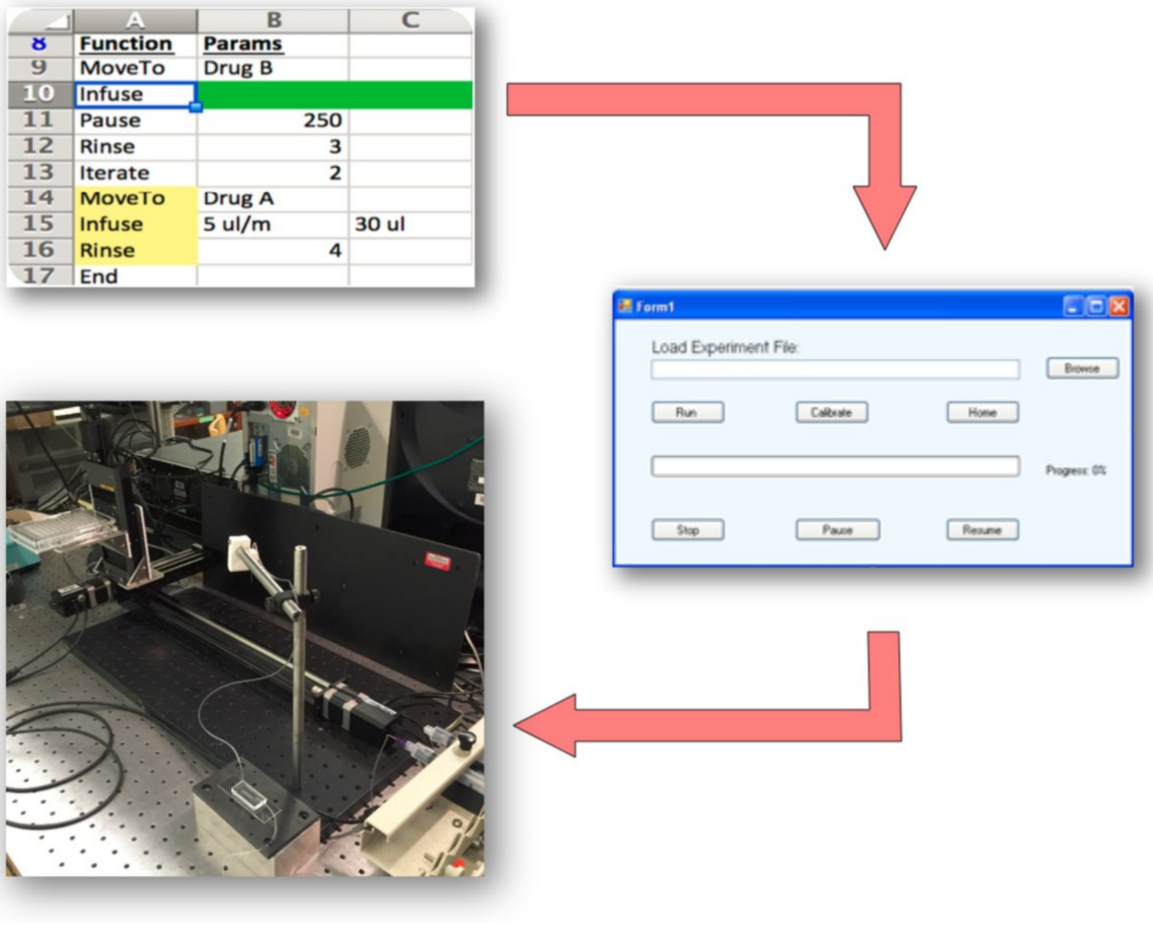


Figure 1: Labeled picture of complete Experimental Setup

Figure 2 below shows how the drug automation setup works as a whole. The user will input instructions into a programmed excel file which contains two sheets. In the first sheet, the user will sequentially input the experiment instructions using custom functions coded in excel and specifying their respective parameters. These functions are explained in detail in Table 1. While in the second sheet, the user would input all the drugs and their respective microplate well locations (see Appendix A for microplate dimensions and well location ordering). It is recommended that the user fill sheet 2 first, so that when the user is entering the experiment instructions in sheet 1, the built in functions can automatically identify the chemicals and their well locations from the other sheet. An example instruction set can be seen in the figure. Once the excel file has been defined, the user will save it and run it through the GUI executable. This GUI allows the user to browse for the excel file defining the experiment and after clicking the run button, will first run a time calculation script to get an estimate of the overall time of the experiment and then run the main drug automation code that will translate these instructions to physical movement of the linear stages and syringe pump. As this automation code is running, a progress bar will keep track of the experiment completion based on the time estimate received earlier.



	A	B	C
8	Function	Params	
9	MoveTo	Drug B	
10	Infuse		
11	Pause	250	
12	Rinse	3	
13	Iterate	2	
14	MoveTo	Drug A	
15	Infuse	5 ul/m	30 ul
16	Rinse	4	
17	End		

Figure 2: Drug automation operation overview

4 System Hardware

The entire setup was comprised of a:

- Zaber T-LSR450D Linear Stage
- Zaber T-LSR150B Linear Stage
- Melles Griot 17NST101 Linear Stage
- Melles Griot 17MST001 NanoStep Stepper Motor Controller Module
- Melles Griot 17MMR001 Main Rack
- KDS230 Infusion/Withdrawal Syringe Pump
- Host Computer with Windows XP Operating System and PCI-CAN card

Specifications for the linear stages and the syringe pump are included in Appendix A.

The subsections below describe properties of the system and/or complications in the hardware that arose over the course of the project, and how they were resolved.

4.1 Controller Area Network

The Melles Griot 17MMR001 Rack is designed to accommodate up to 7 different controller modules.

The rack uses a CAN Bus Topology to communicate with the host computer and supports daisy-chaining of multiple racks. The CAN Bus network requires termination at both ends of the chain through the use of a resistor combination across the ground and power leads of the network. A terminator for the rack was made through the use of a D-Sub adapter with the termination resistors soldered within the housing of the adapter.

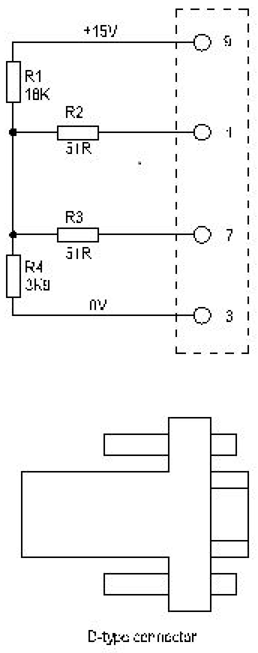


Figure 3: CAN Bus terminator schematics

4.2 KDS230 Syringe Pump Communication

The KDS230 Syringe Pump implements a RS-232 communication standard through the use of 4-pin RJ-11 (standard telephone) connectors. Initially, a RJ-11 to DB9 Serial port connector was obtained from the Engineering Physics Project lab. However, it was seen that it was unable to transmit reply data from the pump to the host due to a mismatch of pin connections. Due to a lack of RJ11 Serial communication standards, a cable for connecting the pump to the host computer could not be obtained through commercial means.

In order to overcome this problem, a RJ-11 cable was used with a RJ-11 socket, and communications established through the computer's serial port by soldering the leads of a D-Sub 9 connector to the required pins on the socket (according to the pinout diagram below).

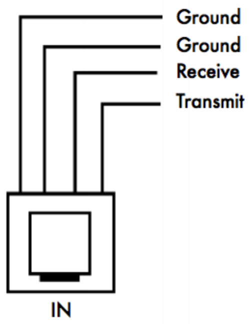


Figure 4: Syringe Pump RJ-11 Pinout

4.3 Damaged Home Sensor

The Zaber T-LSR450D and T-LSR150B utilize a home sensor, a form of a Hall effect sensor, to identify the minimum and maximum positions of the linear stages. Operation of the stages require the use of the in-built home function immediately after power-up to establish their travel limits.

Over the course of the project, the home sensor for the T-LSR450D was damaged, preventing the use of the homing capabilities of the slide. This resulted in the maximum position of the stage being stored as its final position before power down, which in turn resulted in an unpredictable minimum position and restricted travel distance for the stage.

Due to the time constraints on the project, the linear stage could not be sent back to the factory for a home sensor replacement. The proposed solution in order to continue working with the stage was to ensure that it was moved to the end of its range (i.e. the maximum position) before powering it down.

5 Mechanical Design

Although most of the focus of this project was on the software design, there was some mechanical design and fabrication involved. The mechanical design comprises of the following components:

- Waterjet-cut Aluminum supports for connecting Melles Griot and Zaber Slides
- Waterjet-cut L-bracket to secure the microplate enclosure to the Z-slide
- Laser-cut Microplate Enclosure
- 3D printed Needle Holder/Securing Mechanism

Figure 5 show a 3D model of the drug automation set-up. It does not, however, show the syringe pump and the microfludic device.

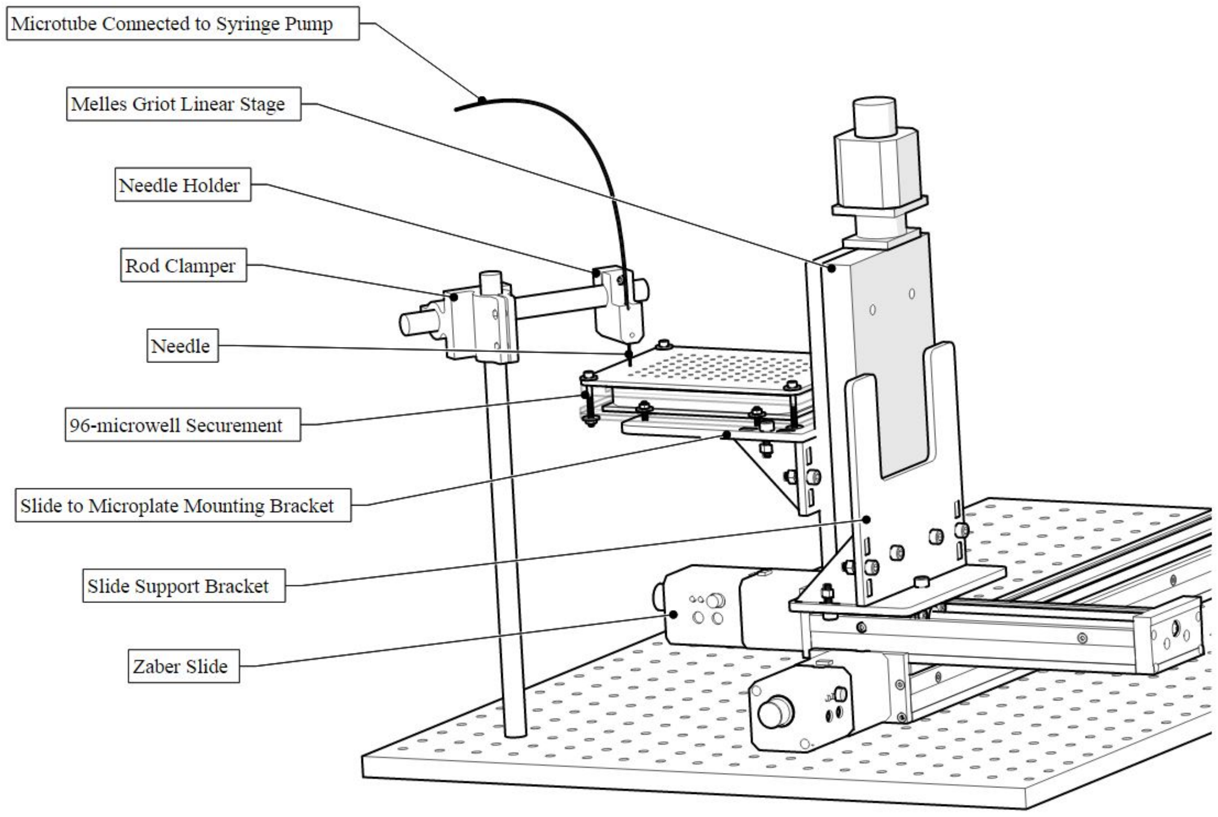

Microtube Connected to Syringe Pump
Melles Griot Linear Stage
Needle Holder
Rod Clamper
Needle
96-microwell Securement
Slide to Microplate Mounting Bracket
Slide Support Bracket
Zaber Slide

Figure 5: 3D model of drug automation setup

5.1 Melles Griot Supports

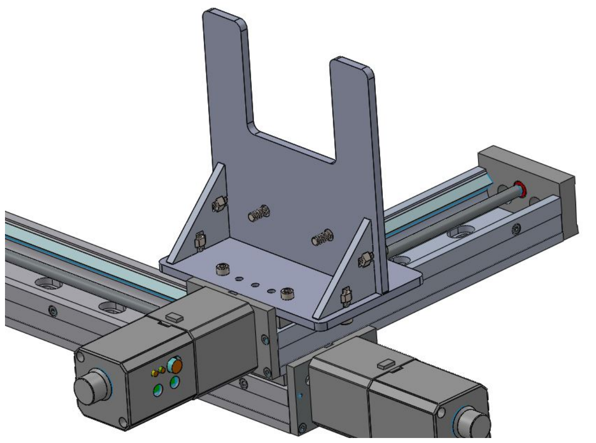


Figure 6: Z-slide supports secured on the zaber slide

Since we are using three linear stages from two different companies, one from Melles Griot and two from Zaber Technologies, there was a need to design supports to secure them to each other. The Zaber slides were used as the X-axis and Y-axis slides due to a larger range of motion as compared to the Melles Griot linear stage. The Melles Griot slide is attached to the to Y-axis stage using the designed bracket shown in Figures 6 and 8.

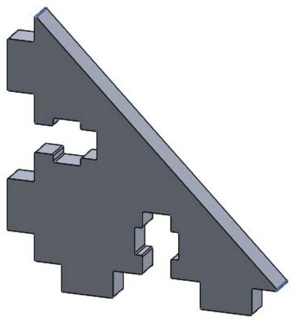


Figure 7: Triangular supports used to make the Z-slide Support bracket

This support design is made up of 4 waterjet-cut parts, each using 5 mm thick aluminum 6061 sheets. The slots in the triangular part (Figure 7 prevent the nut from rotating when the bolt is being screwed through them and allow for easy tightening of the bolts.

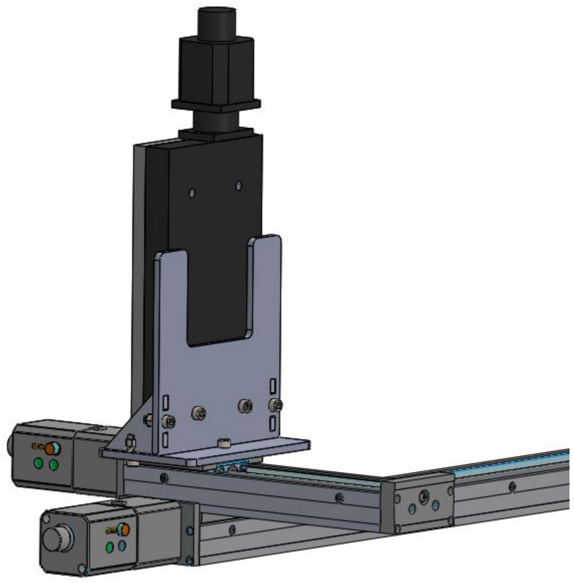


Figure 8: Z-slide supports securing the Melles Griot slide

5.2 L-bracket connecting Microplate to Z-slide

The L-bracket components were also waterjet-cut and utilize the same design as the supports for the Melles Griot stage as seen in figure 9. However, four additional holes (3 mm diameter) were cut to secure the microplate enclosure described in the following subsection. In addition, a deep slot cut was made to reduce material usage and limit the load on the linear stage.

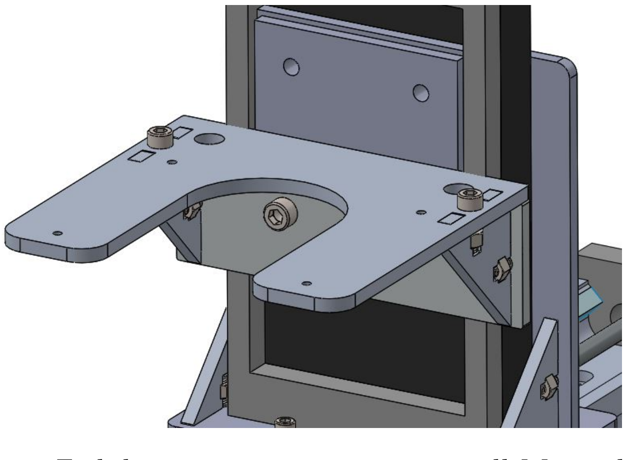


Figure 9: Z-slide supports to secure 96-well Microplate

5.3 Microplate Enclosure Components

In order to secure the microplate to the z-slide and secure a PDMS film on top of the microplate, the following enclosure was designed. Four acrylic pieces were laser-cut, two each for the top and bottom plates of the enclosure. These pairs were then joined together through the use of acrylic welding. M3 screw sizes are used to tighten the assembly as seen in Figure 10. The bottom plate includes additional four 3 mm holes that are used to secure the enclosure to the L-bracket.

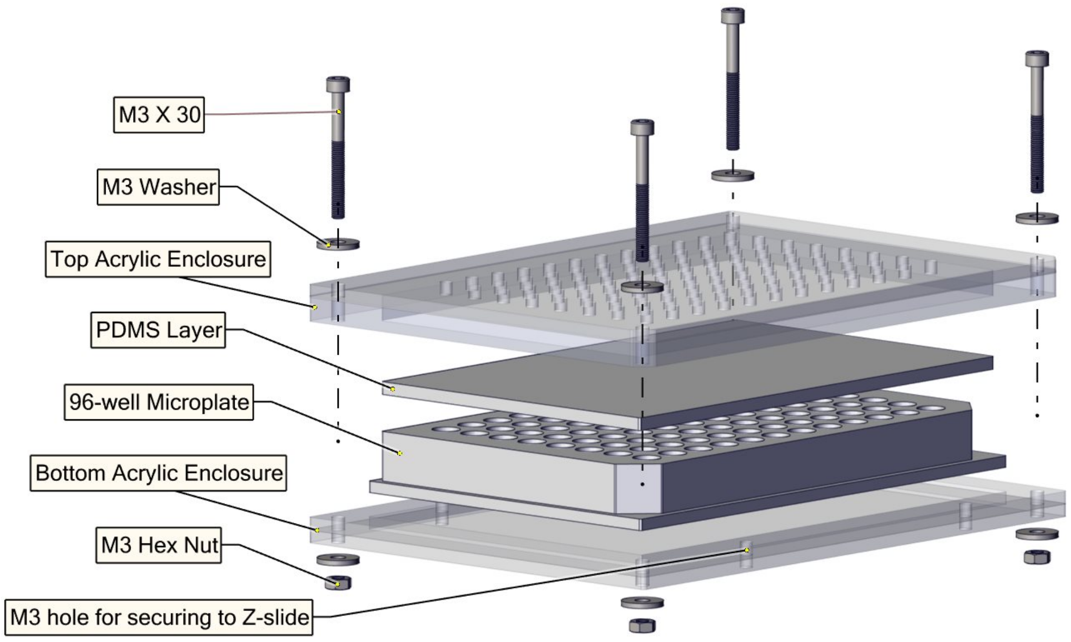


Figure 10: Needle Securing Setup

5.4 Needle Securing Mechanism

The last component of the mechanical design is the needle securing mechanism (or needle holder). The 3D model of this design is shown in the figure below. The needle securing mechanism was 3D printed and uses a nut and screw to clamp it to the support rod. This rod is then clamped to another rod oriented vertically using another 3D printed part labeled as rod clamp in the figure. However, this part although 3D printed was not used since the lab already had a similar clamp which allowed for easier adjustments and resulted in a sturdier design. This can be seen in the complete mechanical setup (Figure 1). To hold the needle in place, an M3 screw was tightened until the needle was firmly secured.

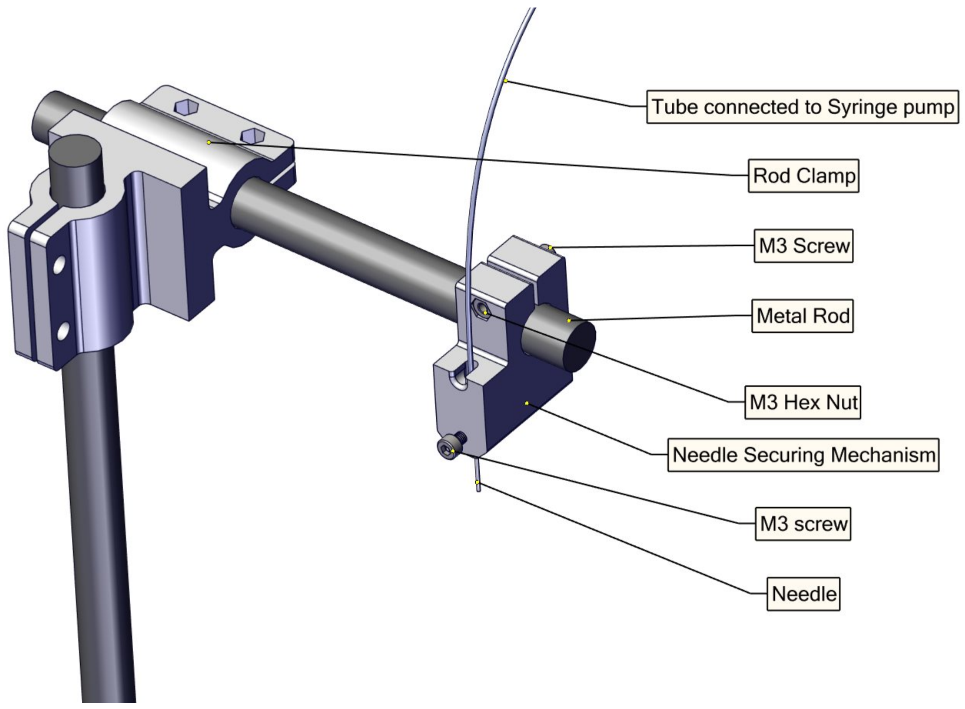


Figure 11: Microplate Enclosure

6 Software Design

This section provides an overview of the system/software requirements for the project, and details the purpose, structure and features provided by the software developed.

The software aims of the project were to:

1. Create a template for the user to create/define experiments
2. Develop code to automate the defined experiments through interfacing with the 3-axis linear stage system and syringe pump
3. Estimate time of completion for experiments based on experiment definitions and hardware parameters
4. Develop a Graphical User Interface (GUI) as the front-end to load and run the experiments

The majority of the code was developed using the C# language and the Visual Basic environment in Microsoft Excel.

6.1 System Requirements

The following are the specifications of the system under which the software was developed:

- Windows XP Operating System

- Microsoft Visual Studio 2010 Express Edition
- .NET Framework Version 4.0
- Microsoft Excel 2010

Moreover, a MelHost V5.0.1 software package obtained from ThorLabs' customer server was installed to interface with the 17MST101 NanoStep Stepper Motor Controller Module. The features provided by this software package are:

1. MG_17 Configuration Utility - Allows creation of required configuration files for Melles Griot controllers
2. ActiveX Drivers - Libraries for interfacing with controller through programming environments

The Windows XP Operating System only supports up to the .NET Framework version 4.0 with the last major release being version 4.5. Moreover, it only supports up to the Visual Studio 2010 edition. The .NET 4.5 Framework introduced asynchronous communication features which serve an important role in Graphical User Interface (GUI) development. Microsoft NuGet asynchronous feature releases for the .NET 4.0 Framework require the use of Visual Studio 2012 and upwards. The restrictions posed by the unavailability is detailed in the Graphical User Interface (GUI) section.

The .NET Framework largely supports backward-compatibility, allowing the software developed to work on systems with .NET Framework Version 4.0 and upwards.

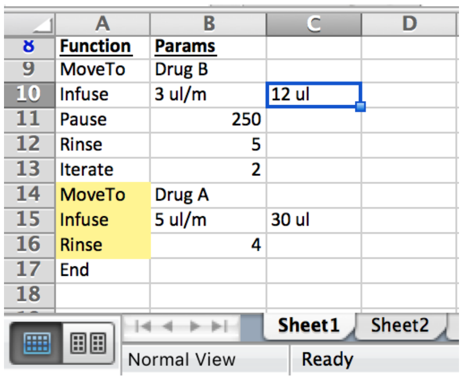
6.2 Experiment Definition Template

The Experiment Definition Template serves as a means to specify the instructions/functions to run an experiment and indicate the positions of drugs contained on the 96-well microplate.

The template is a macro-enabled Excel file that has been programmed through the developer-mode Visual Basic environment in Microsoft Excel 2010. This particular solution was chosen due to familiarity of lab personnel with spreadsheet software, specifically Microsoft Excel. The following sections outline the use of the template, structure of the code, functions available for experiment definition, the use of color indicators to guide user input and other features of interest.

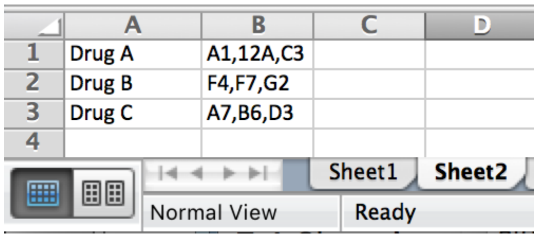
6.2.1 Template Structure

The experiment definition requires the use of Sheets 1 and 2 of the Excel workbook template. Sheet 1 requires the user to specify functions and parameters to be executed as part of the experiment. The functions available are detailed in section 6.2.3. Details of the positions of the drugs contained in the microplate are contained in Sheet 2. The figure below shows a sample experiment defined through the template:



	A	B	C	D
8	Function	Params		
9	MoveTo	Drug B		
10	Infuse	3 ul/m	12 ul	
11	Pause	250		
12	Rinse	5		
13	Iterate	2		
14	MoveTo	Drug A		
15	Infuse	5 ul/m	30 ul	
16	Rinse	4		
17	End			
18				

(a) Experiment Procedure



	A	B	C	D
1	Drug A	A1,12A,C3		
2	Drug B	F4,F7,G2		
3	Drug C	A7,B6,D3		
4				

(b) Drug Positions

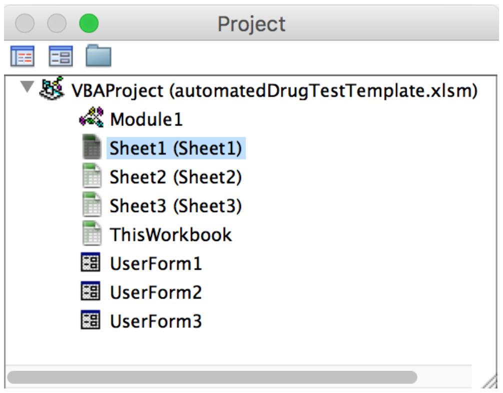
Figure 12: Sample Experiment Definition Template

As can be seen in Figure 12a, the experimental procedure is defined through use of functions (in column A) and parameters that are made available to the user. Sheet 2 (Figure 12b) contains the names of the drugs in column A and a comma-separated list of well positions in column B of the corresponding row.

6.2.2 Visual Basic Code Structure

The Visual Basic environment for Excel is based on event-driven programming. This means that routines are executed based on user interaction with Excel elements such as cells, rows, sheets, workbook etc. The event-driven methodology allows for guiding user input in real-time as detailed in section 6.2.4 and was a major consideration in our choice of solution.

Routines can be defined on a Workbook, WorkSheet and/or Module level as seen in figure 13 below.



VBAProject (automatedDrugTestTemplate.xlsm)
Module1
Sheet1 (Sheet1)
Sheet2 (Sheet2)
Sheet3 (Sheet3)
ThisWorkbook
UserForm1
UserForm2
UserForm3

Figure 13: Excel VBA Project Explorer

Routines for workbook initialization are defined on the Workbook level in 'ThisWorkbook'. Code for configuring the Sheet 1 environment to assist user input and experiment definition is on the WorkSheet level and included in 'Sheet1 (Sheet1)'. Module level code allows global access of variables across the elements seen in the project explorer. The use of the UserForm elements is covered in section 6.2.5.

6.2.3 Functions and Auto-Completion

The following table summarizes the functions and parameters available to the user to define the experiment on Sheet 1:

Function	**Number of Parameters**	**Parameters**
MoveTo	1	**Chemical** - Specify drug from the drop-down list. Note that the drop-down list is populated using the drugs contained in Sheet 2.
Infuse	2	**Flow Rate** - Specify the value and units of the flow rate to infuse at **Volume** - Specify the value and units of volume to infuse
Withdraw	2	**Flow Rate** - Specify the value and units of the flow rate to withdraw at **Volume** - Specify the value and units of volume to withdraw
Pause	1	**Time** - The amount of time in seconds to keep the experiment in a stationary state
Rinse	1	**Time** - The amount of time in seconds to spend in each of 3 wells for the Rinse cycle
Iterate	1	**No. of Loops** - The number of times to loop over the functions after the Iterate function until an 'End' statement is reached (Looped over functions are indicated through yellow-shaded cells)

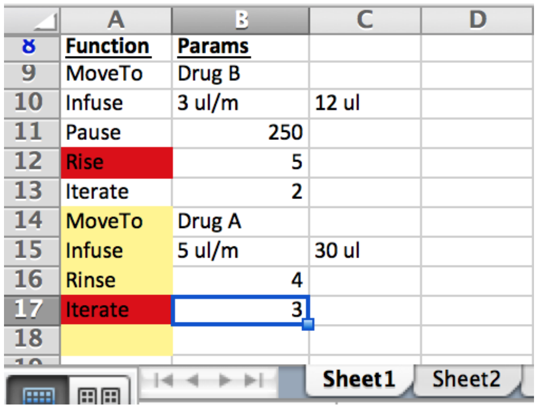
Table 1: Summary of Functions and Parameters

The parameters are set through input boxes or user forms that pop-up when the associated cells are selected (see section 6.2.4).

The code implements auto-complete routines, such that when enough characters are typed in to match a single function, moving to another cell completes the function name.

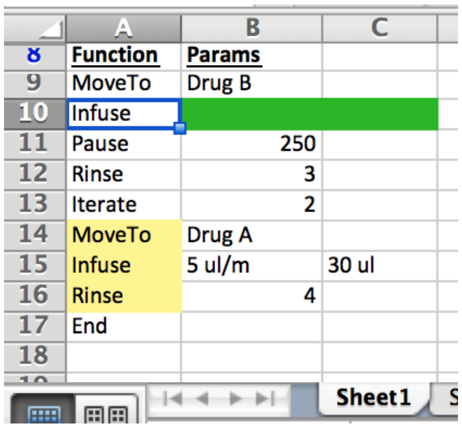
6.2.4 Color Coding

The template uses color indicators on Sheet 1 for parameter prompting, warning against erroneous functions/structure and looped statements.



	A	B	C	D
8	Function	Params		
9	MoveTo	Drug B		
10	Infuse	3 ul/m	12 ul	
11	Pause	250		
12	Rise	5		
13	Iterate	2		
14	MoveTo	Drug A		
15	Infuse	5 ul/m	30 ul	
16	Rinse	4		
17	Iterate	3		
18				

(a) Incorrect function/structure



	A	B	C
8	Function	Params	
9	MoveTo	Drug B	
10	Infuse	🟩	🟩
11	Pause	250	
12	Rinse	3	
13	Iterate	2	
14	MoveTo	Drug A	
15	Infuse	5 ul/m	30 ul
16	Rinse	4	
17	End		
18			

(b) Parameter Prompting

Figure 14: Color Indicators in Excel Template

3 different colors are used as indicators in the template:

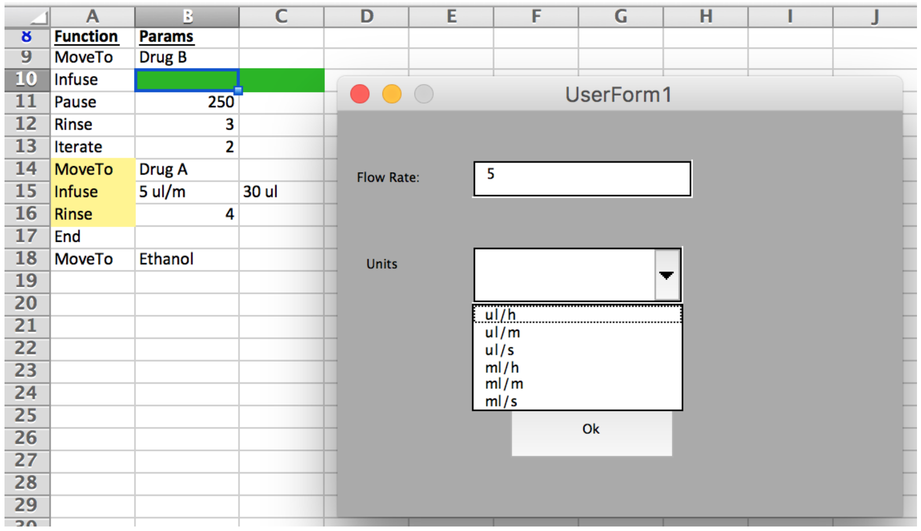
- Red - The red indicators are seen in 2 situations which are observed in Figure 14a. The first is when the user inserts an incorrect function name as can be seen in Row 12. The second happens when the user uses an incorrect structure. For example, the template only allows for the implementation of single looping i.e. no nested loops. So using an 'Iterate' function below another 'Iterate' before encountering an 'End' statement results in an error.

- Green - Green boxes appear to the right when a valid function name in entered in Column A. These indicate the no. of parameters required by the function. Selecting a green filled cell opens an input box or user forms to allow for parameter assignment.

- Yellow - Statements enclosed between 'Iterate' and 'End' are filled in yellow to indicate the scope of the loops to the user

6.2.5 User Forms

User Forms are Excel elements that allow for addition of multiple controls (e.g. drop-down list, text box, labels) in a single window. These are useful for obtaining user input for multiple related fields, such as that for flow rates (values and units) as shown below.



	A	B	C
8	Function	Params	
9	MoveTo	Drug B	
10	Infuse		
11	Pause	250	
12	Rinse	3	
13	Iterate	2	
14	MoveTo	Drug A	
15	Infuse	5 ul/m	30 ul
16	Rinse	4	
17	End		
18	MoveTo	Ethanol	

UserForm1

Flow Rate: 5

Units

Figure 15: User Form for Flow Rate Input

6.3 Automation Code

The automation code is a C# program that interfaces with the linear stages and syringe pump to automate the experiment defined through the Excel Definition Template.

The code requires the use of three 3rd party libraries:

1. Zaber.Serial.Core - A C# library providing objects and methods to interface with the Zaber linear stages

2. Interop.MG_17Drivers - Library for loading Melles Griot Configurations and interfacing with the NanoStep Stepper Motor Controller module

3. EPPlus - .NET library for managing Excel spreadsheets. Available through the NuGet package manager on Microsoft Development Platforms.

The syringe pump does not utilize any 3rd party libraries. RS-232 commands and replies are established through writing and reading strings over the Serial Port. The status of the pump is obtained through parsing reply data using regular expressions. Pump is polled until an infusion/withdrawal process is complete to continue the experiment. The pseudocode below (Algorithm 1) outlines the implementation of the automation code.

Algorithm 1 Drug Injection Algorithm

```
 1: procedure MAIN(gui_procId, excel_file)
 2:     $sheet1 = excel\_file.Open().Sheet1 \leftarrow$ Open Excel file using EPPlus
 3:     $log\_file = File.Create(excel\_file.path + "\_log.txt") \leftarrow$ Create log file
 4:     $connect\_pipe(gui\_procId) \leftarrow$ Connect to GUI using passed in Process ID
 5:     $Initialize() \leftarrow$ Initialize serial ports and configurations for linear stages/pump
 6:     $pipe.Write("Start") \leftarrow$ Signal GUI to start progress bar
 7:     for (row=startRow, row <= LastRow, row++) do
 8:         $sheet1.readRow()$
 9:         $list.store(method)$
10:         $list.store(parameters)$
11:         if method = Iterate then
12:             $loops = parameters$
13:             $iterateStartRow = row$
14:         else if method = End then
15:             $loops = loops - 1$
16:             if loops > 1 then
17:                 $row = iterateStartRow$
18:         else
19:             $InvokeMethod(list[0], list[1:]) \leftarrow$ Call method e.g. Withdraw, Infuse, MoveTo
20:             $log\_file.Write("Executing:$ list[0] on Row row at DateTime.Now")
21:             $list.Clear()$
```

The automation code is the back-end of the Drug Injection system and is launched when an experiment in run from the Graphical User Interface.

6.4 Experiment Time Approximation

The Experiment Time Approximation code is a C# program to calculate the approximate running time of an experiment created using the Experiment Definition Template.

The code works by processing instructions as in the Automated Drug Injection code, but rather than physically executing the instructions, it calculates the time for each step. It utilizes a switch-case structure to identify the function and then uses its parameters to calculate the time for the associated step. Mechanical speeds of linear stages were obtained through querying stored settings on the devices.

Sources of error in estimating total experiment time include,

- Lag times associated with command execution by linear stages/syringe pump
- Time taken for code computations/execution
- Status polling of syringe pump. For example, if the syringe is currently infusing/withdrawing, it is polled every 5 seconds for its status in order to determine when it has stopped and the program should continue. So the error associated with each infusion/withdrawal step can be up to 5 seconds. A lot of infusion/withdrawal steps can cause this error to accumulate to a significant value.

The first 2 of these error sources are not significant, especially as only a rough approximation of the total time is required (about +/- 5%). The status polling times are not likely to have a significant impact as the length of the infusion/withdrawal steps will constitute a much greater percentage of the total experiment time as opposed to the errors due to polling. However, small error percentages in long experiments can result in large differences between the estimated and actual duration.

In order to determine trends between error in time approximation and experiment length, an experiment was performed as detailed in section 7.2.

6.5 Graphical User Interface (GUI)

The Graphical User Interface (GUI) serves as the front-end of the Drug Injection system. Its primary purpose it to allow users of the Drug Injection system to run experiments defined through the Experiment Definition Template. The following sections will walk the readers through the GUI controls/elements used, the use of threads and processes to manage UI events and run code in the background, and the use of inter-process communication techniques to pass data between applications.

The GUI was created using the WinForms framework in C# (or Windows Forms) available through Visual Studio. Through initial research, it was determined that the majority of windows interfaces for .NET applications are developed through the WPF (Windows Presentation Foundation) and WinForms frameworks. The WPF framework is a newer technology, built from scratch, and does not use the standard controls defined by the Windows API. While WPF offers more capabilities for richer user interfaces, it has a much steeper learning curve than WinForms and it was unlikely that these richer capabilities would provide significant benefit to an application of a small scale. Moreover, the Visual Studio software provides a better design interface for WinForms applications. Keeping in mind that a goal of this project was to develop a solution that could easily be built upon in the future by other developers, it was decided that the WinForms framework would best meet this goal.

6.5.1 GUI Elements

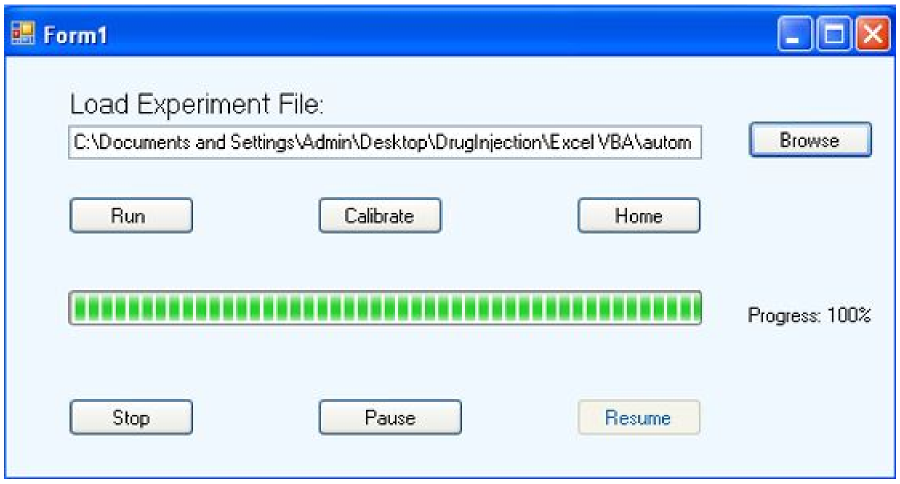


Figure 16: GUI for Automatic Drug Injection

This GUI application offers users with the following features:

- **Run** - Run Experiments by loading the experiment Excel file
- **Pause/Resume** - Pause and Resume running experiments
- **Stop** - Stop running experiments
- **Calibrate** - Move linear stages into position for calibration
- **Home** - Move linear stages back to their Home positions

6.5.2 Threading and Processes

The GUI uses threading to remain responsive to UI events during execution of experiments. It also makes use of multiprocessing features to run the Time Calculation and Automation code in the background.

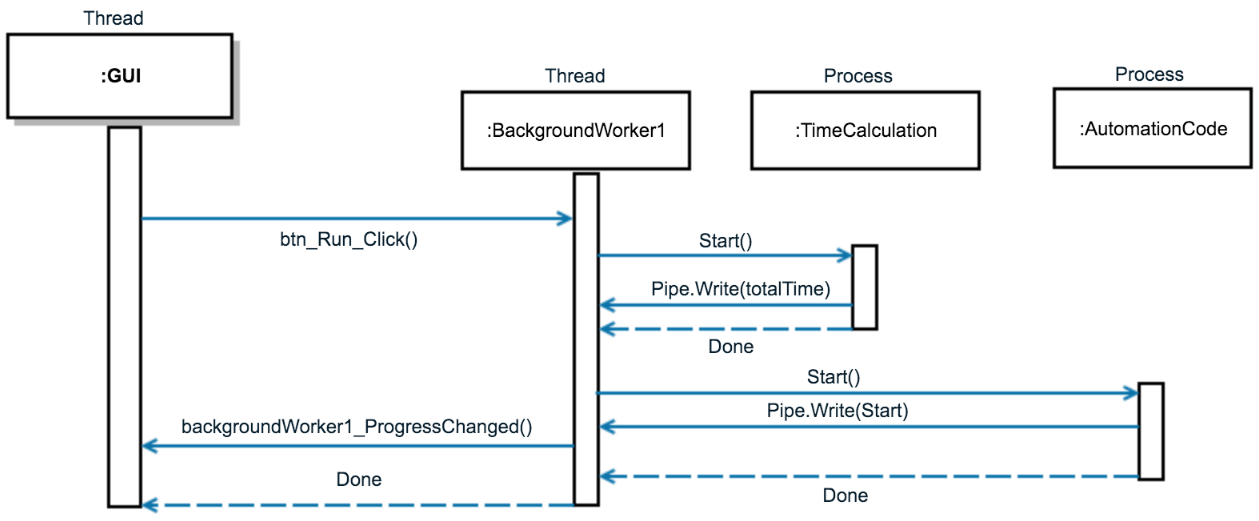


Figure 17: UML Sequence Diagram for Drug Injection Software

As seen in figure 17, the GUI (main) thread is initialized when the GUI application is started. After the user loads a file and clicks the run button, the GUI instantiates the backgroundWorker1 object to handle the running of the experiment. The backgroundWorker1 object is an instance of the BackgroundWorker class in C# WinForms, which creates a new thread with helper methods for communicating with the main thread. The use of this second thread was required in order to free the main thread to detect and act upon UI events, such as the user clicking the stop, pause and resume buttons. It is due to the features provided by the helper methods that the BackgroundWorker Class was used as opposed to the Thread class (in the System.Threading namespace) for traditional multi-threaded applications.

The run click event starts the execution of the new thread through the DoWork() member of the BackgroundWorker class. The DoWork() method spawns two processes over its lifetime, one for running the Time Calculation code and another for running the Automation code, in that order. Upon the spawning of each process, the BackgroundWorker object creates a pipe between the GUI application and the spawned process. The pipe is a unidirectional communication channel that relays information from the created process to the backgroundWorker1 thread. The pipe is used to send the total time calculated through the Time Calculation code to be used for reporting progress through the progress bar element of the GUI.

The progress bar starts when it receives a 'start' string message from the automation program through the pipe. Generally, there would not be a requirement for a start message as the progress bar can be started immediately after the creation of the automation code process. However, the system in use tends to occasionally exhibit a significant lag (~1-2 mins) between the start of the automation program and actual code execution. Once the progress bar begins, a StopWatch object is used to monitor the time elapsed since the 'start' message event and used to calculate the percentage progress of the experiment. The calculated percentage is reported to the UI thread using the BackgroundWorker class' ProgressChanged() method, which avoids the use of cross-thread calls and compromised thread safety that might arise from manual implementation of these helper methods.

6.5.3 Inter-Process Communication (IPC)

Inter-process communication refers to the use of pipes for communicating between 2 or more processes. As mentioned in the previous section, a unidirectional pipe was created between the spawned processes for the Time Calculation and Automation code to the GUI application. This was done through the use of AnonymousPipeServerStream and AnonymousPipeClientStream objects available through the System.IO.Pipes namespace. The implementation of the communication channels was largely trivial, however it did result in some interesting findings about the limitations of the .NET 4.0 framework.

As mentioned in section 6.1, the .NET 4.5 framework introduced libraries for asynchronous communication. The original plan for the GUI was to include an additional progress bar for reporting the progress of the currently executing step. This would have required communication of the start time of each step between the automation code and the GUI. However, due to an absence of asynchronous features in .NET 4.0, this could not be implemented within the given time constraints and workaround implementations would have increased complexity in the code. The primary issue in communicating data throughout the lifetime of the automation code arises from the pipe having to be continuously polled for incoming data by the GUI. The read methods provided by stream objects in C# are blocking, meaning that they block thread execution until data is received. This in turn would significantly delay the updating of the progress bars which were being updated from the same thread. A workaround would likely involve creating dedicated threads for the progress bars. Through discussions with the project sponsor, it was decided that the addition of a progress bar to show step execution was superfluous and not worth an increased complexity in the code.

7 Quantitative Results

7.1 Quantifying Cross-Contamination

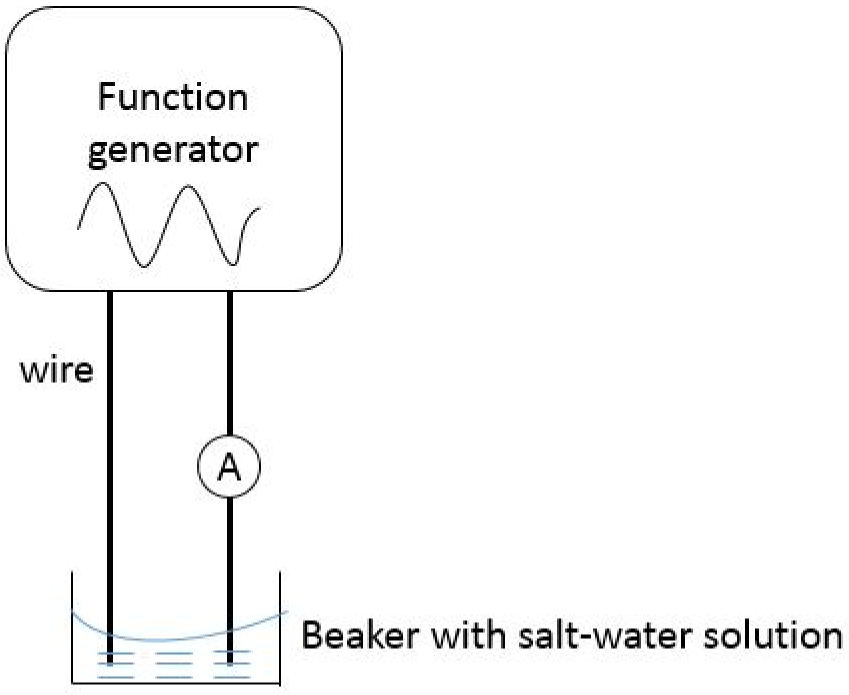


Figure 18: Experimental set-up for quantifying cross-contamination

In order to quantify any cross-contamination that may occur when the needle traverses from one drug to another, the setup shown in Figure 18 was used. The following experimental procedure was used to obtain the results:

- A function generator was set at an AC voltage of 10KHz with 5V (Peak to Peak).
- Two thin wires were connected to the function generator and ammeter, and acted as electrodes when immersed in salt water solutions. These wires were tightly taped together making sure that the distance between them (in this case the diameter of the wire) was kept constant throughout the experiment. Uneven distances between the electrodes would lead to differences in path lengths (and resistances) seen by the circuit.
- The wire pair was then dipped into different salt-water concentrations and the respective current readings were noted down. As the concentration was increased there was an increase in current since higher concentration solutions would conduct more easily due to presence of a greater concentration of charge carriers (ions).
- These measurements were then used to plot the concentration vs current plot shown in Figure 23.

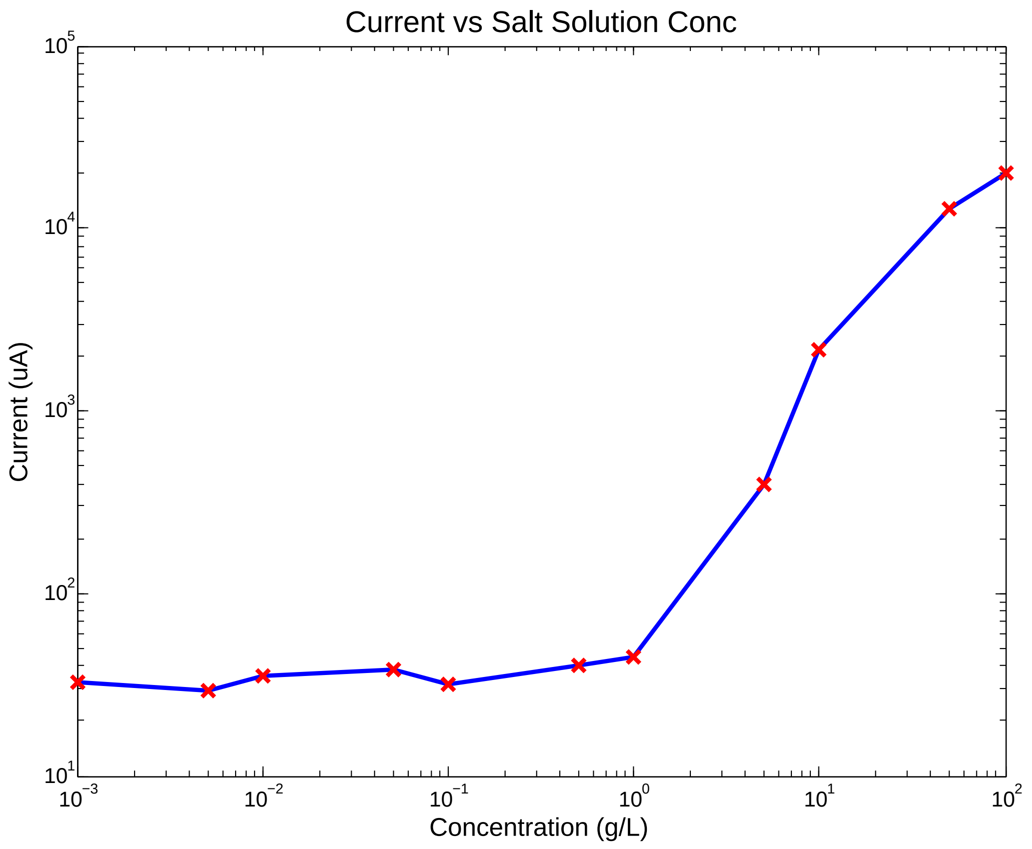


Figure 19: Concentration(g/L) vs. Current(mA) plot

Once these results were established, the following steps were performed:

- Three wells in the microplate were filled with DI (deionised) water and one was filled with 100g/L salt concentration solution (the maximum concentration available).
- A drug automation experiment was performed, where the needle started out in the salt solution and then traversed each of the wells containing the deionised water, spending 10 seconds in each. The wells were labelled (1, 2 and 3) with 1 being the first well of DI water traversed, and 2 and 3 being the following wells traversed in that order.
- After the experiment, the wire pair was dipped in each well and the current measurements were noted down and compared against the graph in Figure 23

From this experiment, it was seen that the wells containing the now 'contaminated' DI water allowed currents of 20 uA for 5V peak-to-peak excitation as before. However, these values are lower than those measured to establish the current vs concentration graph. Moreover, the ammeter was not sensitive enough to register noticeable change for low concentration readings (as is seen for all concentrations below 1g/L in figure 23). Therefore, the results of the experiment were inconclusive. Perhaps, a better alternative to this experiment would be to use fluorescence techniques, replacing the salt water solutions with dyed water.

7.2 Accuracy of Experiment Time Approximation Code

Towards the end stages of the project, we considered evaluating the accuracy of the time estimation code through running experiments of various lengths, ranging from very short to very long ones. However, due to higher priority tasks involving software bugs, feature improvements and project hand-off, we were not able to complete such experiments.

It was proposed that the trends in estimation errors versus experiment running times be obtained and incorporated as compensation measures in the code. We did manage to get in five tests ranging from a minute to 10 minutes. In each of these tests, the estimation was within 1-2 seconds of the actual running time. However, more representative experiments on the scale of multiple hours to days will provide a better bearing on the accuracy of estimation. This has been included as a recommendation in section 10.2

8 Conclusion

The Experiment Definition template has been thoroughly tested through numerous mock experiments and found to be bug-free and working as expected for tested inputs/actions. It is able to successfully utilize custom-defined functions to completely specify experiments, and provide features for auto-completion of function names, using color indicators to guide user input and obtain input for function parameters through the use of user form elements. The drug automation code works successfully for all defined functions to execute the experiments as required. Estimation on the running durations of experiments were found to be accurate to within 1-2 seconds for experiments on the time scale of one to ten minutes. However, experiments were not conducted with more representative experiment time scales (on the order of hours) due to time constraints. All GUI features have been tested and are running as intended, allowing users to load experiment files and run, stop, pause and resume such experiments.

In addition, experiments were conducted to evaluate the degree of cross-contamination of wells by measuring currents established (by AC voltage excitation) across deionised water 'contaminated' by salt water solutions through needle traversal. However, results were inconclusive due to inadequate sensitivities of the measuring equipment. Possible alternatives include the use of fluorescence techniques used in conjunction with dyed substances. The current mechanical design for the supports, microplate enclosure, and needle holder, although working as required, can certainly be improved upon as discussed in section 10.1

9 Project Deliverables

This section details the state of the material contributions to the project made over the last 4 months to be handed over to the NanoSystem lab, an approximate financial summary of the entire project and outstanding commitments that will continue on past the report submission stage.

9.1 List of Deliverables

The table below summarizes the deliverables and their state at the time of report submission.

	Deliverables
Mechanical Design	**Melles Griot Support:** Made up of four parts waterjet cut out of 5mm thick 6061 aluminum sheet. M3 bolts and nuts were used to assemble these parts as shown in Figure 6. The support is used to secure the Melles Griot Linear stage to the T-LSR150B Zaber Slide. **L-bracket to connect Microplate to Melles Griot Stage:** Also made up of four waterjet-cut parts using aluminum 6061 sheets and assembled using M3 bolts and nuts as shown in Figure 9. One side of the bracket connects to the Melles Griot stage while the other secures the microplate enclosure. **Microplate Enclosure:** Made up of 4 laser cut acrylic plates two of each were acrylic welded to make the top and bottom enclosure(refer to Figure 10). 30 mm long M3 bolts were used to secure microplate enclosure and its contents. **Needle Holder:** This part was 3D printed at the UBC Engineering Physics project lab (refer Figure 11). It uses an M3 bolt to clamp the holder to the rod and another screw to hold the needle in place.

Software Design	**Experiment Definition Template:** A macro-enabled Excel file to define the positions of drugs on the 96-well microplate, and predefined functions to specify the experiment. It uses color indicators and user forms to guide user input. The template is in a fully functional state with features being tested thoroughly. However, given the various types of user interactions with the worksheet, it may be possible that bugs exist for some actions. In any case, such bugs are unlikely to hinder the user from using the template successfully. **Drug Automation Code:** Reads the excel workbook with instructions and runs the experiment through automation of the 3-axis linear stage system and syringe pump. It also generates a log file which lists executed steps with time stamps. This is also working as expected and can be built upon by future developers as required. **Time Calculation Code:** The code provides an estimated time of running a particular experiment defined through the Experiment Definition Template. It is used by the GUI to display the running time of the experiment and update the progress bar. While time constraints prevented any significant testing of the accuracy of estimation, especially on the scale of the expected length of experiments (i.e. days), it was seen that it maintained an accuracy of estimation of within 1-2 seconds for experiments on the order of 10 minutes. The scalability of this estimation remains to be determined. **Drug Automation GUI:** The GUI allows users to load experiments defined through the definition template. It also provides utilities for linear stage calibrations, homing the setup, running, pausing/resuming and stopping the experiment. Over the course of numerous experiments, it has not exhibited any problems of note and is currently considered fully functional. However, experiments on the scale of multiple hours/days may reveal any potential faults.

Table 2: Summary of Project Deliverables

9.2 Financial Summary

This project has largely utilized pre-existing equipment available at the NanoSystems laboratory. Hence, financial records of the apparatus used do not exist. Nonetheless, an approximation of the total cost of the project is compiled below based on current market prices for each piece of equipment.

Part	**Total Cost ($)**
Zaber T-LSR450D Linear Slide	2932
Zaber T-LSR150B Linear Slide	2019
Melles Griot 17NST101 Linear Stage	617
Melles Griot 17MMR001 Main Rack	847
Melles Griot 17MST001 Stepper Motor Module	40
KDS 230 Infusion\Withdrawal Syringe Pump	2200
PDMS Film	-
Melles Griot Software\Drivers	Free
Microsoft Visual Studio 2010 Express	Free
96-well Microplate	4
CAN Bus Terminator	6
Aluminum 6061 5mm ($\sim$100 in^2)	17
Estimated Total Cost	**8754**

Table 3: Project Parts List

Some of the equipment used by this project was overkill, especially the use of linear stages with the range provided by the Zaber T-LSR450D slide (450 mm). However, they were used since these were readily available at the laboratory and significant costs would be involved in acquiring additional equipment to replace them.

9.3 Ongoing Commitments

The following list summarizes the ongoing commitments and support to be provided for the project past the ENPH 459 end date:

1. Fabrication of simple acrylic components through the laser cutter to provide an easier securing mechanism for the microplate. This is expected to be completed by the end of the week after the report submission.

2. Fabrication of a larger version of the triangular supports shown in figure 7. This is required to decrease the moment seen at the end of the L-bracket which is currently resulting in a fair amount of tilt. This is expected to be completed by the end of April.

3. Bug fixes and support for minor feature requests will be provided as the system is integrated into practice at the NanoSystem laboratory. This will depend on the results of its usage and potential faults that may be exposed through its use.

4. Addition of user documentation on the github project maintained at .rehmanis/Drug-Injection

10 Recommendations

10.1 Mechanical

The current mechanical design can be improved as follows:

1. **Melles Griot Support**: When the bolts are screwed in the holes that secure the bottom plate of Melles support to the Zaber slide, it touches the Melles Griot Slide and therefore can be hard to screw or unscrew. To remedy this, increase the distance from the hole center to the Melles slide.

2. **L-bracket connecting microplate to Melles Slide**: In the current design the L-bracket bends under the load of the microplate and aluminum piece. Thus the needle is not able to fully immerse in the well at the end of microplate. To minimize bending, the size of the triangular brackets (Figure 7) can be increased which will provide a sturdier support to the aluminum plate on which the microplate rests.

3. **Microplate Enclosure** An improvement on the current design would be to increase the hole size of the top acrylic enclosure (10 or make the holes tapered. Increasing the hole size would be easier to implement in terms of manufacturability and provide more room for error in calibration

4. **Needle Holder**. Since the needle holder was printed on low accuracy 3D printer, the slot that houses the thin needle was printed at an angle. Thus, when the needle is secured in this slot, it tilts. As a result, it may crash into the walls of the top acrylic enclosure and bend. To minimize this tilt, a better 3D printer can be utilized to print the needle holder.

10.2 Software

Software components that could be improved in the future include:

1. Improving accuracy of experiment time approximation code by deducing trends in the error with actual running times, and using these trends to compensate for the errors. This will require significant testing of the estimation code with a range of experiments, from very short to very long ones.

2. Addition of hardware and software for automated calibration of the needle with the wells of the microplate. A potential solution might involve the use of a camera and image processing technologies to determine the alignment and compensations required. This will likely be a highly technical and involved task, fitting of another project on its own.

3. Addition of error/exception handling procedures to propagate exceptions occurring in the automation/time calculation code to the GUI. This will require some research into capturing exceptions in a process and passing relevant information to the GUI process/application. This might be considerably more difficult than conventional exception handling. An alternative solution might be to implement the exception handling

locally and writing the exception trace to the log file for the experiment (this will be a much easier solution to implement).

A Appendix

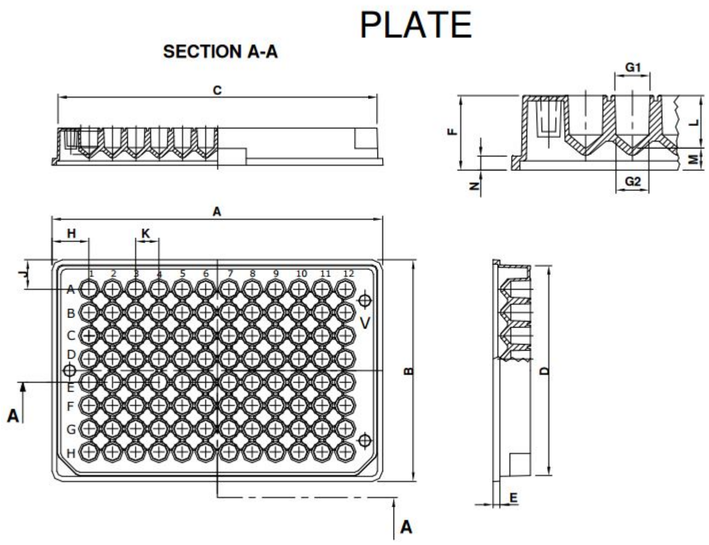
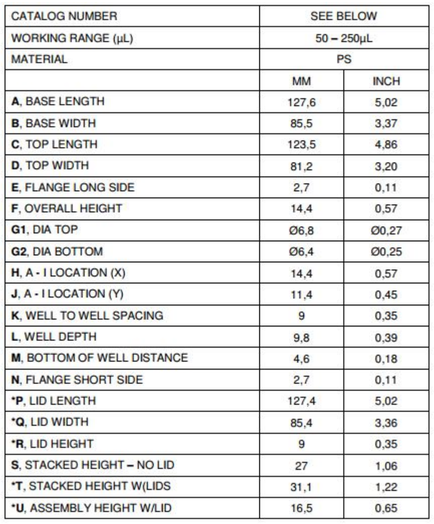
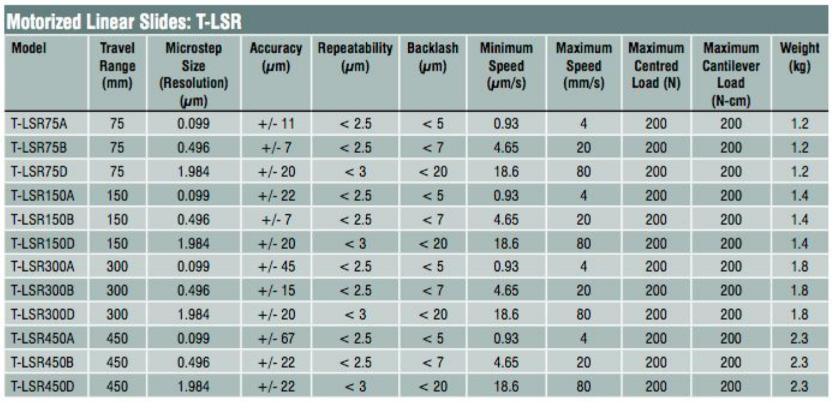


Figure 20: Microplate well drawings



CATALOG NUMBER	SEE BELOW	
WORKING RANGE (µL)	50 – 250µL	
MATERIAL	PS	
	MM	INCH
A, BASE LENGTH	127,6	5,02
B, BASE WIDTH	85,5	3,37
C, TOP LENGTH	123,5	4,86
D, TOP WIDTH	81,2	3,20
E, FLANGE LONG SIDE	2,7	0,11
F, OVERALL HEIGHT	14,4	0,57
G1, DIA TOP	Ø6,8	Ø0,27
G2, DIA BOTTOM	Ø6,4	Ø0,25
H, A - I LOCATION (X)	14,4	0,57
J, A - I LOCATION (Y)	11,4	0,45
K, WELL TO WELL SPACING	9	0,35
L, WELL DEPTH	9,8	0,39
M, BOTTOM OF WELL DISTANCE	4,6	0,18
N, FLANGE SHORT SIDE	2,7	0,11
***P**, LID LENGTH	127,4	5,02
***Q**, LID WIDTH	85,4	3,36
***R**, LID HEIGHT	9	0,35
S, STACKED HEIGHT – NO LID	27	1,06
***T**, STACKED HEIGHT W(LIDS	31,1	1,22
***U**, ASSEMBLY HEIGHT W/LID	16,5	0,65

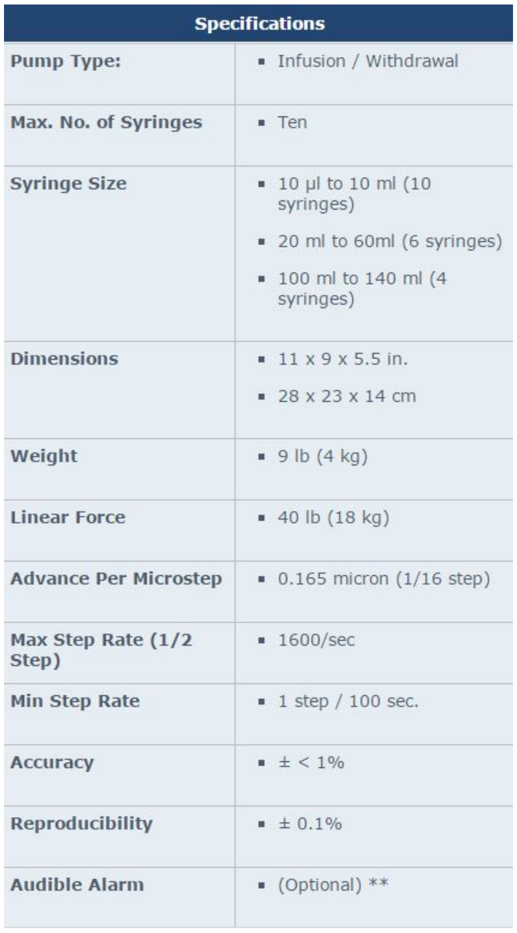
Figure 21: Microplate well dimensions for drawing



Motorized Linear Slides: T-LSR

Model	Travel Range (mm)	Microstep Size (Resolution) (μm)	Accuracy (μm)	Repeatability (μm)	Backlash (μm)	Minimum Speed (μm/s)	Maximum Speed (mm/s)	Maximum Centred Load (N)	Maximum Cantilever Load (N-cm)	Weight (kg)
T-LSR75A	75	0.099	+/- 11	< 2.5	< 5	0.93	4	200	200	1.2
T-LSR75B	75	0.496	+/- 7	< 2.5	< 7	4.65	20	200	200	1.2
T-LSR75D	75	1.984	+/- 20	< 3	< 20	18.6	80	200	200	1.2
T-LSR150A	150	0.099	+/- 22	< 2.5	< 5	0.93	4	200	200	1.4
T-LSR150B	150	0.496	+/- 7	< 2.5	< 7	4.65	20	200	200	1.4
T-LSR150D	150	1.984	+/- 20	< 3	< 20	18.6	80	200	200	1.4
T-LSR300A	300	0.099	+/- 45	< 2.5	< 5	0.93	4	200	200	1.8
T-LSR300B	300	0.496	+/- 15	< 2.5	< 7	4.65	20	200	200	1.8
T-LSR300D	300	1.984	+/- 20	< 3	< 20	18.6	80	200	200	1.8
T-LSR450A	450	0.099	+/- 67	< 2.5	< 5	0.93	4	200	200	2.3
T-LSR450B	450	0.496	+/- 22	< 2.5	< 7	4.65	20	200	200	2.3
T-LSR450D	450	1.984	+/- 22	< 3	< 20	18.6	80	200	200	2.3

Figure 22: Zaber Linear Slide Specifications



Specifications	
Pump Type:	▪ Infusion / Withdrawal
Max. No. of Syringes	▪ Ten
Syringe Size	▪ 10 µl to 10 ml (10 syringes) ▪ 20 ml to 60ml (6 syringes) ▪ 100 ml to 140 ml (4 syringes)
Dimensions	▪ 11 x 9 x 5.5 in. ▪ 28 x 23 x 14 cm
Weight	▪ 9 lb (4 kg)
Linear Force	▪ 40 lb (18 kg)
Advance Per Microstep	▪ 0.165 micron (1/16 step)
Max Step Rate (1/2 Step)	▪ 1600/sec
Min Step Rate	▪ 1 step / 100 sec.
Accuracy	▪ ± < 1%
Reproducibility	▪ ± 0.1%
Audible Alarm	▪ (Optional) **

Figure 23: Syringe Pump KDS230 Specifications