DESIGN AND AUTOMATION OF AN INJECTABLE POLYBUBBLE DRUG DELIVERY PLATFORM

An Undergraduate Research Scholars Thesis

by

WHITNEY SOUERY

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Corey Bishop

May 2019

Major: Biomedical Engineering

TABLE OF CONTENTS

ABSTRACT

Design and Automation of an Injectable Polybubble Drug Delivery Platform

Whitney Souery
Department of Biomedical Engineering
Texas A&M University

Research Advisor: Dr. Corey Bishop
Department of Biomedical Engineering
Texas A&M University

The current state of controlled release therapeutics is limited by the challenges of: (1) developing a fabrication method that accounts for the unique sensitivities of amino acid-based cargo; and (2) designing a high-throughput drug-loading strategy that allows for both low-cost and high-resolution production of therapeutics. Addressing the above challenges will not only aid in expanding healthcare accessibility worldwide but also in optimizing the therapeutic potential of drugs. In this paper, we will report on a novel, high-throughput, automated polybubble fabrication system that seeks to address the limitations outlined above. The automated polybubble fabrication system enables polybubble production to be scaled up to meet patient demand on an industrial level, in addition to providing versatility in treatment scope. In this way, a single platform can be tailored to a variety of disease applications by allowing for loading of multiple drugs. Moreover, lyophilizing the polybubbles will allow for improved long-term stability during transport and storage, making the platform well suited for application in developing countries with limited access to medical infrastructure.

ACKNOWLEDGEMENTS

I would first like to thank my research advisor, Dr. Corey Bishop, for providing the idea for this project and allowing me the chance to be part of this incredible research opportunity. I am deeply grateful for all of Dr. Bishop's continuous support, encouragement, and motivation, and this project would not exist without him.

I would also like to recognize Shreedevi Arun Kumar, Mohamed Harhash, and Jacob Good from Dr. Bishop's lab, without whose effort this project would also not have been possible. In addition, I would like to acknowledge Sachin Shinde for his invaluable assistance with Python. And of course, a very big thank you to all of the other members of Dr. Bishop's Pharmacoengineering Laboratory for their input to the project and daily inspiration.

Thank you to Regan Manayan, without whose love and support this work would not have been possible. Thank you, Regan, for never doubting in me and reminding me of the upsides to every situation.

Lastly, a very large thank you to my family for teaching me the power of never, never, never giving up. Thank you, Mom, Dad, CC, and Rico, for your unconditional love and support for everything I do.

CHAPTER I

INTRODUCTION

The current state of controlled release therapeutics is limited by the challenges of: (1) scaling up production from the laboratory to industry to allow for high-throughput manufacturing of therapeutics; (2) improving overall treatment accessibility by addressing long-term stability and shelf-life of the therapeutic; and (3) lastly, tailoring such a treatment platform to account for synergistic, time-dependent drug dosing. My research question seeks to address whether an automated, injectable polybubble drug delivery system can resolve the above challenges. Automation of such a platform would enable microbubble production to be scaled up to meet patient demand on an industrial level, in addition to providing versatility in treatment scope by allowing for loading of multiple drugs. While such automated platforms have already been designed for this purpose, these platforms are associated with design limitations including cost, low resolution, and high margin of error. In order to expand the accessibility of disease treatment worldwide, an effective, low-cost automated drug-loading platform with a high degree of sensitivity must be developed. Addressing these challenges would not only expand healthcare accessibility but also optimize the therapeutic potential of drugs. The expected outcome of this project is to develop an platform capable of (1) automated polybubble fabrication for high-throughput applications and (2) lyophilization to facilitate better long-term storage outcomes.

Despite the advances and recent successes in the field of controlled release drug delivery, the current state of the field is limited by many challenges, ranging from encapsulating drug to ensuring predictable delivery of functional cargo. For example, many drug-loading strategies, such as single- and double-emulsion, rely on the use of organic solvents, which preferentially favors the delivery of hydrophobic cargo [1-3]. In the presence of organic solvents, protein- or antigen-based cargo may denature more rapidly and require temperature-sensitive transport, consequently posing as a challenge for delivery to remote locations.

The sensitivity of amino acid-based cargo to water also poses as an additional challenge. Many biodegradable polymers commonly used for therapeutic delivery, such as poly(lactic-co-glycolic acid) (PLGA) and poly(ε-caprolactone) (PCL), degrade hydrolytically [4]. In addition to causing early release of depot, unintended exposure to water either *ex vivo* or due to pre-existing humidity could also destabilize and consequently harm amino acid-based depots. As a result, there is a critical need for therapeutics that account for the sensitivity of amino acid-based cargos.

Developing a high-throughput drug-loading strategy is also key for developing successful and effective treatments that can easily be adapted for use in locations without accessible healthcare. While such automated platforms have already been designed for this purpose, these platforms are associated with design limitations including cost, low resolution, and high margin of error [5-7]. In order to expand the accessibility of disease treatment worldwide, an effective, low-cost automated drug-loading platform with a high degree of sensitivity must be developed. Addressing these challenges would not only expand healthcare accessibility but also optimize the therapeutic potential of drugs.

Objectives/Goals

In this project, I will seek to develop a novel automated, injectable polybubble drug delivery system that addresses the current lapses in the field of controlled drug delivery. The final platform is expected to: (1) allow for production of controlled-release therapeutics to be scaled up to meet patient demand on an industrial level and (2) incorporate a freeze-drying process that enables the shelf life of the treatment to be extended, making the platform well suited for application in developing countries with limited access to medical infrastructure.

This research would substantially contribute to the field of controlled-release therapeutics by increasing the feasibility of such drug delivery systems, which are currently hindered by their lack of compatibility in traditional healthcare systems.

CHAPTER II

METHODS

System Architecture

For control of the injection process, LabVIEW™ , and Python™ programming software was used in coordination with two Agilis-P Controller with Encoder Feedback Model CONVEX-AGL and two KDS Legato 200 Series Syringe Pumps. LabVIEW™ was used to control the Agilis-P micromotors. Python™ was used to execute both LabVIEW® sub-programs, as demonstrated in **Figure 1**.

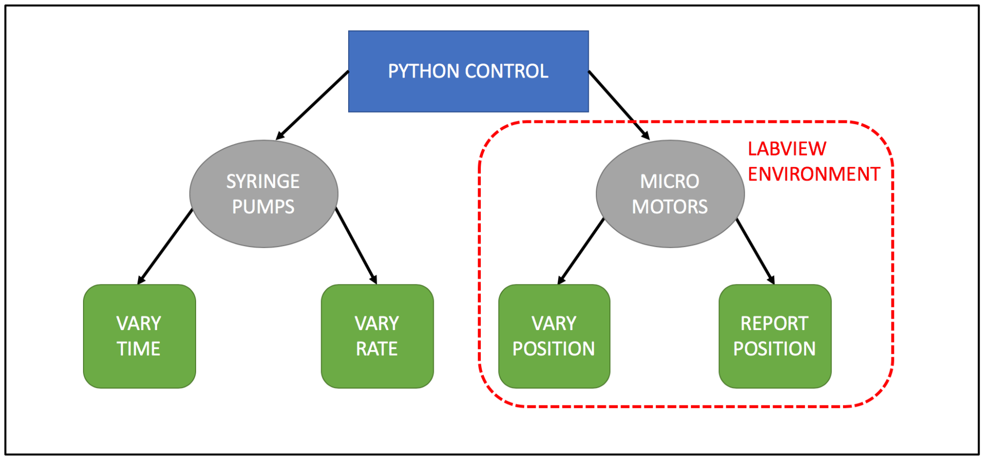


Figure 1. Overview of System Architecture

Automation of micromotors

Micromotor control is executed within the LabVIEW environment. Two separate programs within LabVIEW enable micromotion movement and motor localization.

Controlling Micromotor Motion

To control the relative motion of the micromotors, both the OR and PR_SET methods from the CONVEX-AGP command library were utilized within a flat sequence structure. The OR method is used to transition each micromotor from the “not referenced” state to “ready” state when suppled with the unique instrument key of each micromotor.

Following execution of the OR method, the PR_SET method is called. The PR_SET method controls the movement of each micromotor by accepting a unique step value, referred to as StepComX, where X represents the unique instrument key. The value of StepComX is set by the user during the initial input step via Python.

Localizing Micromotors

To localize the micromotors, the TP method from the CONVEX-AGP command library is utilized. The TP method reports the absolute position of the micromotors given the unique instrument key of the micromotor. The CurrentPosition variable is then displayed as output from the Python program.

Integration of Syringe Pump and Micromotor Function

Python was used to integrate both the syringe pump and micromotor processes in a single script. Python relays micromotor input to the LabVIEW sub-programs using the LabVIEW Automation labview_automation Python package licensed under the Massachusetts Institute of Technology [8].

Syringe Pump Functionality within Python

Communication with both syringe pumps in established using a serial port connection in Python, which allows the syringe pump to interface with the computer using pump chain commands.

The functions *runPumpX, stopPumpX*, and *PumpRunTimeX* allow for user-controlled syringe pump functionality, where X indicates the identity of the syringe pump (1 or 2). *runPumpX* simulates the “run” function of the syringe pump; similarly, *stopPumpX* simulates the "stop" function. The *PumpRunTime* function allows for a user to input a specific run time using both the "run" and "stop" functions, combined with the sleep function in Python's datetime library.

Micromotor Functionality within Python

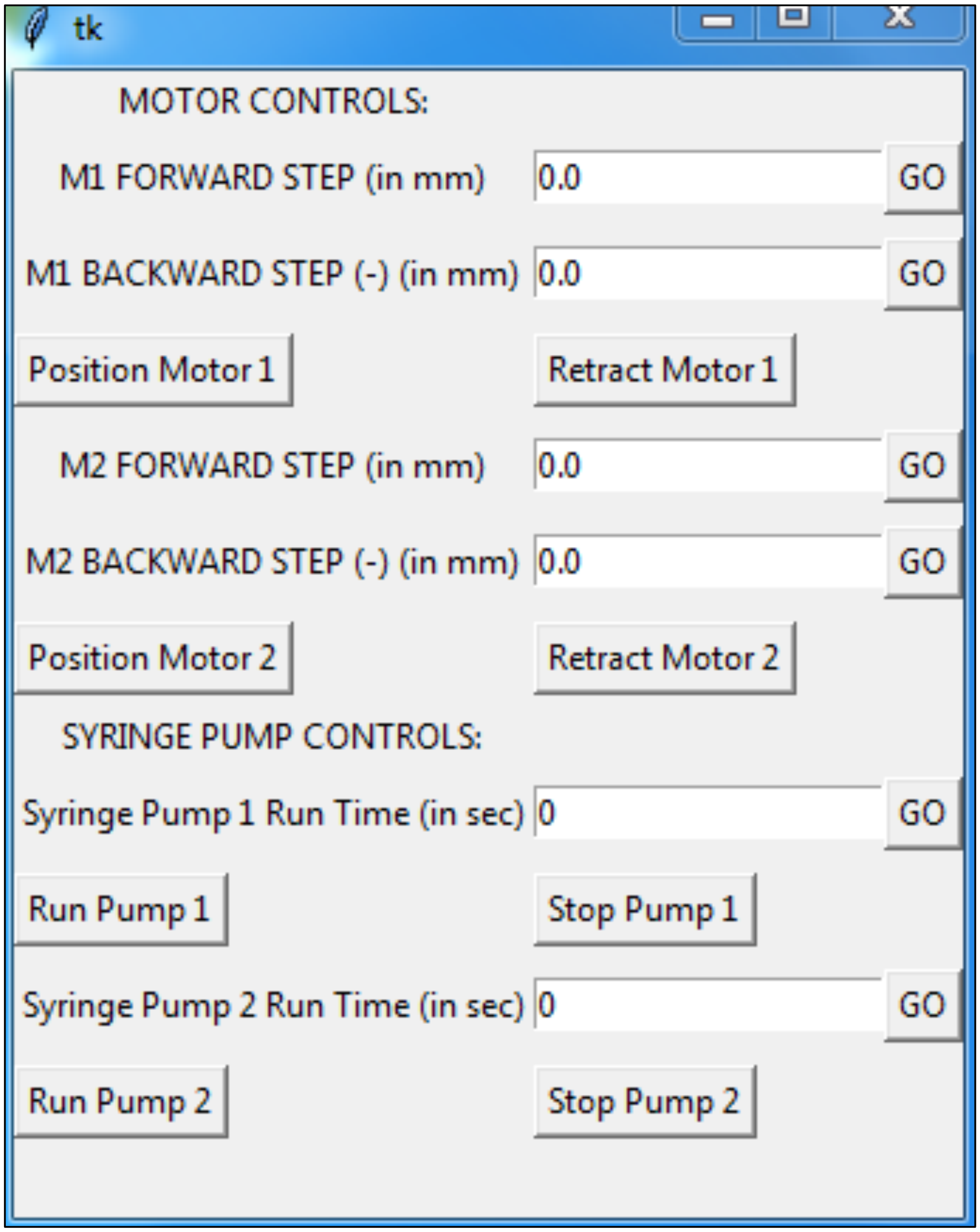
The functions *runMM1_forward*, *runMM2_forward*, *runMM1_backward*, and *runMM2_ backward* open and call the LabVIEW VI controlling micromotor motion. After initiating micromotor movement, the final absolute position of the micromotor's location is determined from the LabVIEW VI and displayed in the text file output, accompanied by a timestamp and the name of the function called.

The functions *retract_MM1* and *retract_MM2* allow the micromotor to be recalled to an absolute position of zero. Motion is enabled using the LabVIEW VIs controlling micromotor motion, with a pre-set step value of zero. After initiating micromotor movement, the final absolute position of the micromotor is determined from the LabVIEW VI and displayed in the text file output, accompanied by a timestamp and the name of the function called.

The functions *position_MM1* and *position_MM2* output the current position of the micromotors using the localization program. The outputted position value is also printed to the text file output, accompanied by a timestamp and the name of the function called.

Graphical User Interface (GUI)

All motor and syringe pump controls are displayed in the GUI shown in **Figure 2**. The GUI was created using the Tkinter toolkit within Python.



M1 FORWARD STEP (in mm) 0.0
M1 BACKWARD STEP (-) (in mm) 0.0
M2 FORWARD STEP (in mm) 0.0
M2 BACKWARD STEP (-) (in mm) 0.0
Syringe Pump 1 Run Time (in sec) 0
Syringe Pump 2 Run Time (in sec) 0

Figure 2. Graphical User Interface displayed by Python Script

Data Recording

A time log of all keystrokes made during script execution are recorded in an output text file using the open and write functions in Python. Each time the script is initiated, the text file is overwritten to contain data from the new session.

System Construction

To construct the automated polybubble fabrication system, a wooden platform was built to house the syringe pumps, micromotors, and injection site. The following sections will detail the construction and design specifications of the system.

Polymer and Drug Loading

Polymer and drug are injected using 316 stainless steel tubing purchased from McMaster-Carr. The polymer tubing has an outer diameter of 0.02 inches and an inner diameter of 0.016 inches. The tubing used to inject polymer is housed within a 316 stainless steel shaft with an outer diameter of 0.12 inches and an inner diameter of 0.09 inches. Similarly, the drug tubing has an outer diameter of 0.016 inches and an inner diameter of 0.008 inches. The drug tubing is housed within a shaft with the same specifications as used for the polymer tubing.

To help further stabilize positioning during the injection process, caps were fabricated for both the polymer and drug tubes using 30A silicone rubber purchased from Grainger to fit within the space between the shaft and tubing. Silicone was laser cut using a carbon dioxide Gravograph laser printer.

Caps were designed using AutoCad 2018 software. The polymer tubing cap has an outer diameter of 0.085 inches and an inner diameter of 0.025 inches. The drug tubing cap has an outer diameter of 0.085 inches and an inner diameter of 0.015 inches. Caps are fit onto both the drug and polymer tubing before polybubble fabrication.

Spatial Alignment of Polymer and Drug Tubing

The consistency of the polybubble fabrication system is highly dependent on the orientation of both the polymer and drug tubes. The micromotor controlling the position of the polymer tubing is oriented vertically, with the motor at the baseline position of zero.

In a similar manner, the drug-carrying micromotor is mounted on a goniometer fixed at an angle of 17.8 degrees on a wall with an incline of 79.39 degrees, as shown in **Figure 3** Panel A. Before drug injection begins, the drug-carrying motor starts at the baseline position of zero. After polymer is dispensed and the polymer tubing is moved, the drug tubing moves 5 mm in the direction of the injection vial site, as shown in **Figure 3**. The positions of the polymer tubing and path of the drug tubing are shown in **Figure 3** Panel B.

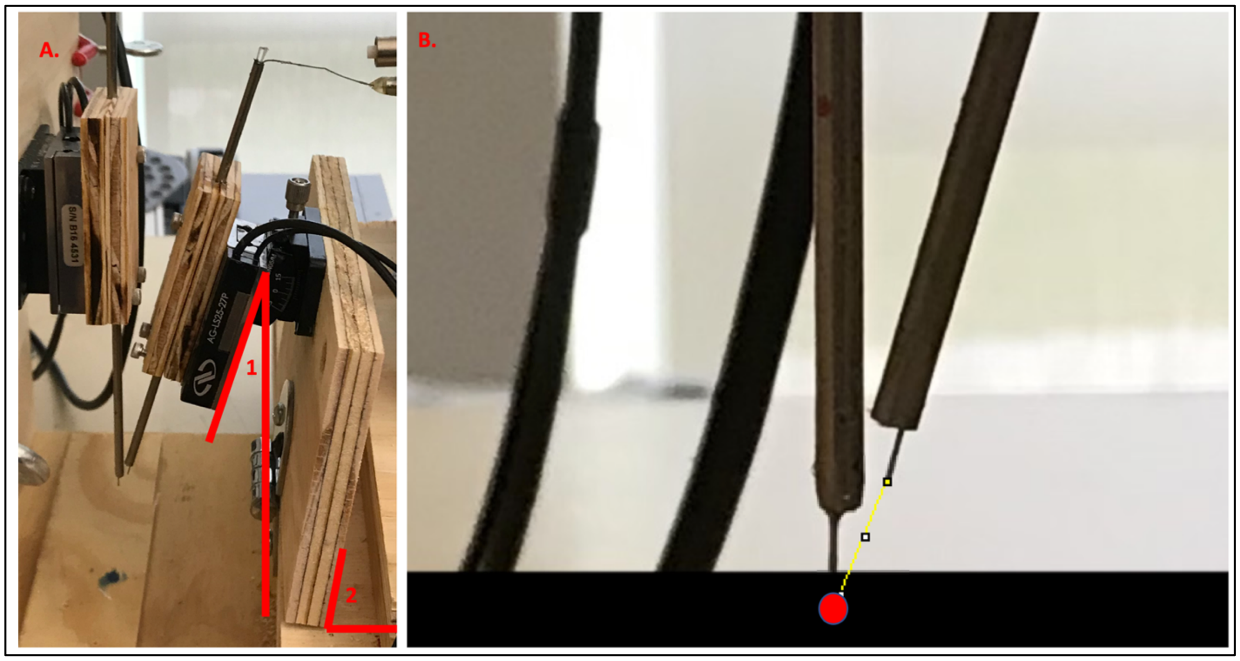


Figure 3. Alignment of polymer and drug tubing during the polybubble injection. Panel A indicates the angles of the goniometer (1) and wall (2). Angle 1 is approximately 17.8 degrees, and angle 2 is approximately 79.388 degrees. Panel B indicates the positioning of the polymer tubing upon injection, as well as the path (shown in yellow) of the drug tubing following dispensing of polymer. The yellow line is a distance of approximately 5 mm, with the red circle indicated the position of drug formation. Quantification of all angles and distances was performed using ImageJ software.

Polybubble Fabrication

Poly(ε-caprolactone) (PCL) and PCL tri-acrylate (PCL-TA) were synthesized using the method described previously by Cai, et al. [9] PCL and PCL-TA were blended in a 1:3 ratio. The PCL and PCL-TA blend was subsequently mixed with 10% 1 mg/mL acriflavine in dimethyl sulfoxide for visualization. Trypan Blue Solution 0.4% purchased from Thermo Fischer Scientific was mixed with 5% carboxymethyl cellulose (CMC) to serve as the model drug for prototyping purposes. Polybubbles were injected into a 10% CMC solution.

All polybubble fabrication was executed through the Python GUI. To fabricate the polybubbles, the drug tubing was first positioned 5 mm away from the designated injection site. Polymer (PCL and PCL-TA blend) was first dispensed from the syringe pump at a pre-set rate of 3 microliters per second for 8 seconds. Next, the polymer tubing was moved 10 mm in the direction towards the micromotor.

To dispense drug, the drug tubing was moved 5 mm in the direction of the polybubble. Drug was dispensed from the tubing for 2 seconds, after which the drug tubing was moved 10 mm away from the polybubble.

Following drug injection, polybubbles were cured under ultraviolet light for 20 minutes and then lyophilized overnight.

CHAPTER III

RESULTS AND DISCUSSION

System Architecture

Automation of Micromotors

Controlling Micromotor Motion

Panel A of **Figure 4** displays the LabVIEW subprograms that are used to control the motion of the drug- and polymer-carrying micromotors. Inputting the forward or backward step value into the GUI text boxes labeled “M1 Forward Step”, “M1 Backward Step”, “M2 Forward Step”, and “M2 Backward Step” moves the motors by the desired amount.

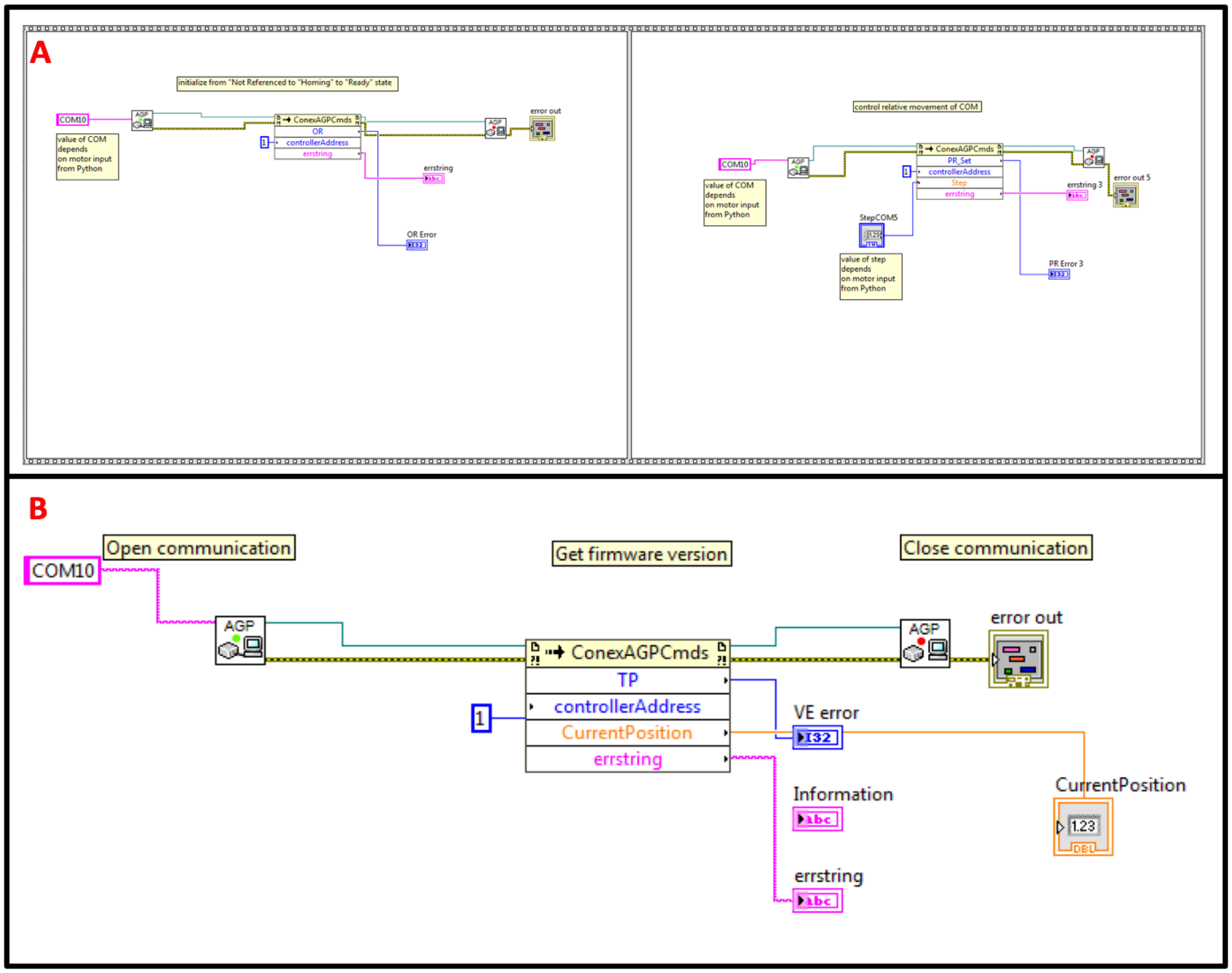


Figure 4. Screenshots of LabVIEW subprograms used in Python, where COM10 represents the instrument key of one micromotor. Panel A demonstrates control of micromotor motion, and Panel B demonstrates how localization of each micromotor was carried out.

Negative values move the motors in the direction toward the injection vial, while positive values move the motor away from the injection vial, as demonstrated by the coordinate system shown in **Figure 5**.

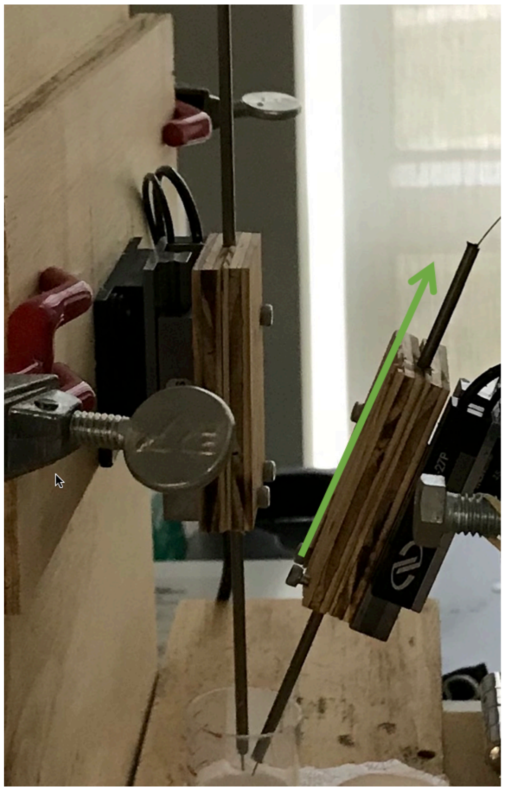


Figure 5. Demonstration of coordinate system used to control micromotor motion, with the green arrowhead indicating the direction of motion from positive values inputted for the "M1 Forward Step", "M1 Backward Step", "M2 Forward Step", and "M2 Backward Step."

Localizing Micromotors

Panel B of **Figure 4** demonstrates the LabVIEW program used to localize the absolute position of the micromotors. Selecting the "Position Motor 1" or "Position Motor 2" command on the GUI provides the user with the absolute position of the micromotor.

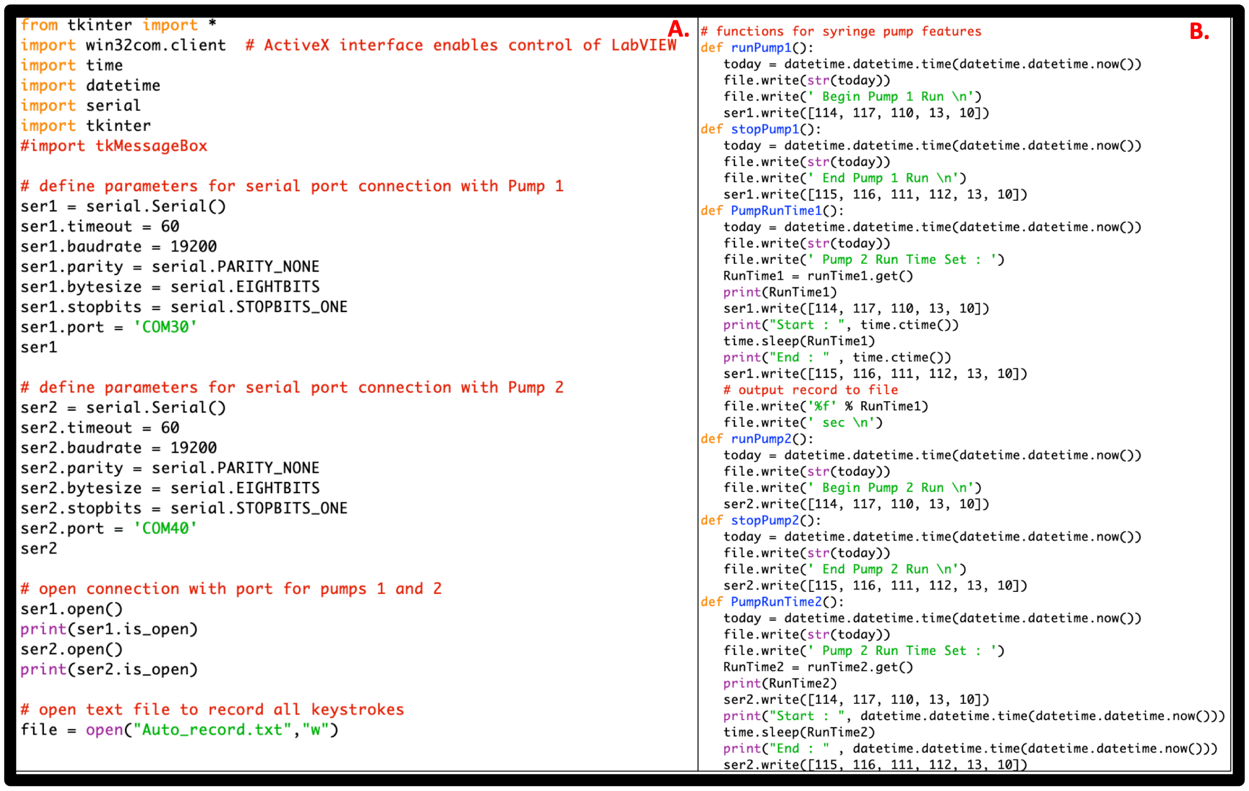
Integration of Syringe Pump and Micromotor Function

Syringe Pump Functionality within Python

Figure 6 displays the commands used to toggle the syringe pump on and off. Execution of the syringe pump is associated with a timestamp indicating the time the syringe pump was turned on and off, which is subsequently displayed to the user in the terminal window.

Micromotor Functionality within Python

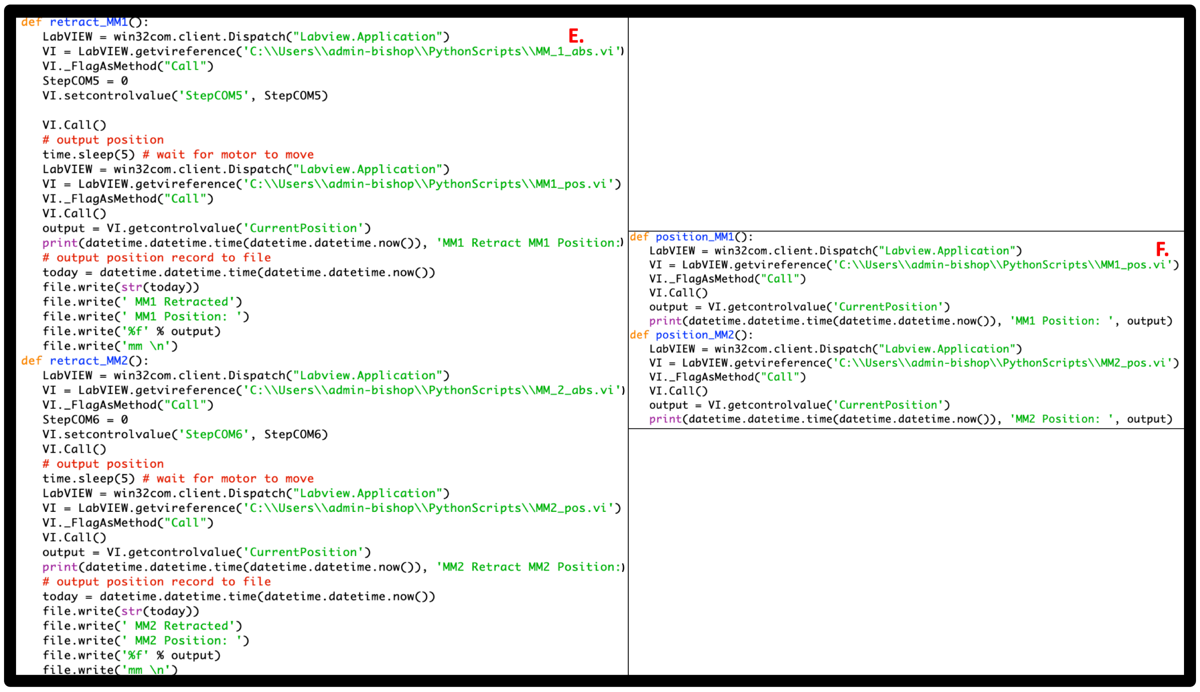
Figure 6 displays how control over the LabVIEW environment was integrated within Python. All micromotor control is carried out using the Python program.



```
from tkinter import *
import win32com.client  # ActiveX interface enables control of LabVIEW
import time
import datetime
import serial
import tkinter
#import tkMessageBox

# define parameters for serial port connection with Pump 1
ser1 = serial.Serial()
ser1.timeout = 60
ser1.baudrate = 19200
ser1.parity = serial.PARITY_NONE
ser1.bytesize = serial.EIGHTBITS
ser1.stopbits = serial.STOPBITS_ONE
ser1.port = 'COM30'
ser1

# define parameters for serial port connection with Pump 2
ser2 = serial.Serial()
ser2.timeout = 60
ser2.baudrate = 19200
ser2.parity = serial.PARITY_NONE
ser2.bytesize = serial.EIGHTBITS
ser2.stopbits = serial.STOPBITS_ONE
ser2.port = 'COM40'
ser2

# open connection with port for pumps 1 and 2
ser1.open()
print(ser1.is_open)
ser2.open()
print(ser2.is_open)

# open text file to record all keystrokes
file = open("Auto_record.txt","w")
```

```
# functions for syringe pump features
def runPump1():
   today = datetime.datetime.time(datetime.datetime.now())
   file.write(str(today))
   file.write(' Begin Pump 1 Run \n')
   ser1.write([114, 117, 110, 13, 10])
def stopPump1():
   today = datetime.datetime.time(datetime.datetime.now())
   file.write(str(today))
   file.write(' End Pump 1 Run \n')
   ser1.write([115, 116, 111, 112, 13, 10])
def PumpRunTime1():
   today = datetime.datetime.time(datetime.datetime.now())
   file.write(str(today))
   file.write(' Pump 2 Run Time Set : ')
   RunTime1 = runTime1.get()
   print(RunTime1)
   ser1.write([114, 117, 110, 13, 10])
   print("Start : ", time.ctime())
   time.sleep(RunTime1)
   print("End : " , time.ctime())
   ser1.write([115, 116, 111, 112, 13, 10])
   # output record to file
   file.write('%f' % RunTime1)
   file.write(' sec \n')
def runPump2():
   today = datetime.datetime.time(datetime.datetime.now())
   file.write(str(today))
   file.write(' Begin Pump 2 Run \n')
   ser2.write([114, 117, 110, 13, 10])
def stopPump2():
   today = datetime.datetime.time(datetime.datetime.now())
   file.write(str(today))
   file.write(' End Pump 2 Run \n')
   ser2.write([115, 116, 111, 112, 13, 10])
def PumpRunTime2():
   today = datetime.datetime.time(datetime.datetime.now())
   file.write(str(today))
   file.write(' Pump 2 Run Time Set : ')
   RunTime2 = runTime2.get()
   print(RunTime2)
   ser2.write([114, 117, 110, 13, 10])
   print("Start : ", datetime.datetime.time(datetime.datetime.now()))
   time.sleep(RunTime2)
   print("End : " , datetime.datetime.time(datetime.datetime.now()))
   ser2.write([115, 116, 111, 112, 13, 10])
```

Figure 6. Python program used to execute syringe pump and micromotor functionality. Panel A displays the parameters defined after connecting each syringe pump using the serial port. Panel B displays the functions used to define the syringe pump controls. Panels C-F display the functions used to define the micromotor controls. Panels G-H display the code used to create the GUI.

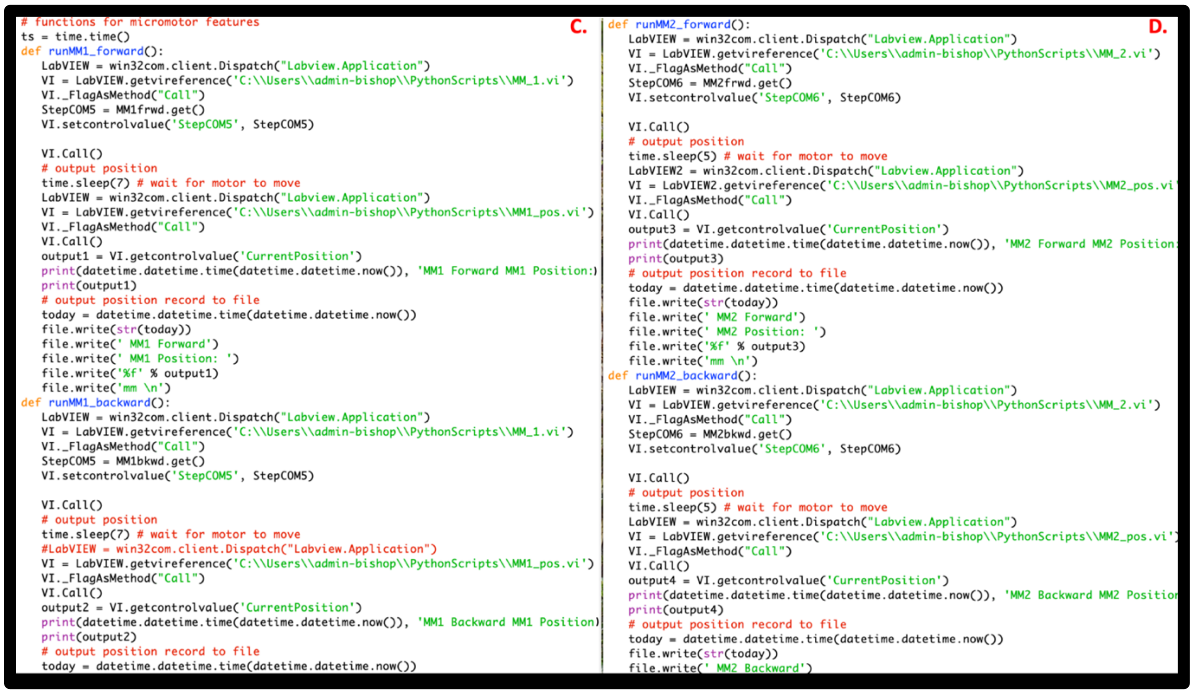


E.

```
def retract_MM1():
    LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM_1_abs.vi')
    VI._FlagAsMethod("Call")
    StepCOM5 = 0
    VI.setcontrolvalue('StepCOM5', StepCOM5)

    VI.Call()
    # output position
    time.sleep(5) # wait for motor to move
    LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM1_pos.vi')
    VI._FlagAsMethod("Call")
    VI.Call()
    output = VI.getcontrolvalue('CurrentPosition')
    print(datetime.datetime.time(datetime.datetime.now()), 'MM1 Retract MM1 Position:
    # output position record to file
    today = datetime.datetime.time(datetime.datetime.now())
    file.write(str(today))
    file.write(' MM1 Retracted')
    file.write(' MM1 Position: ')
    file.write('%f' % output)
    file.write('mm \n')
def retract_MM2():
    LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM_2_abs.vi')
    VI._FlagAsMethod("Call")
    StepCOM6 = 0
    VI.setcontrolvalue('StepCOM6', StepCOM6)
    VI.Call()
    # output position
    time.sleep(5) # wait for motor to move
    LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM2_pos.vi')
    VI._FlagAsMethod("Call")
    VI.Call()
    output = VI.getcontrolvalue('CurrentPosition')
    print(datetime.datetime.time(datetime.datetime.now()), 'MM2 Retract MM2 Position:
    # output position record to file
    today = datetime.datetime.time(datetime.datetime.now())
    file.write(str(today))
    file.write(' MM2 Retracted')
    file.write(' MM2 Position: ')
    file.write('%f' % output)
    file.write('mm \n')
```

F.

```
def position_MM1():
    LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM1_pos.vi')
    VI._FlagAsMethod("Call")
    VI.Call()
    output = VI.getcontrolvalue('CurrentPosition')
    print(datetime.datetime.time(datetime.datetime.now()), 'MM1 Position: ', output)
def position_MM2():
    LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM2_pos.vi')
    VI._FlagAsMethod("Call")
    VI.Call()
    output = VI.getcontrolvalue('CurrentPosition')
    print(datetime.datetime.time(datetime.datetime.now()), 'MM2 Position: ', output)
```



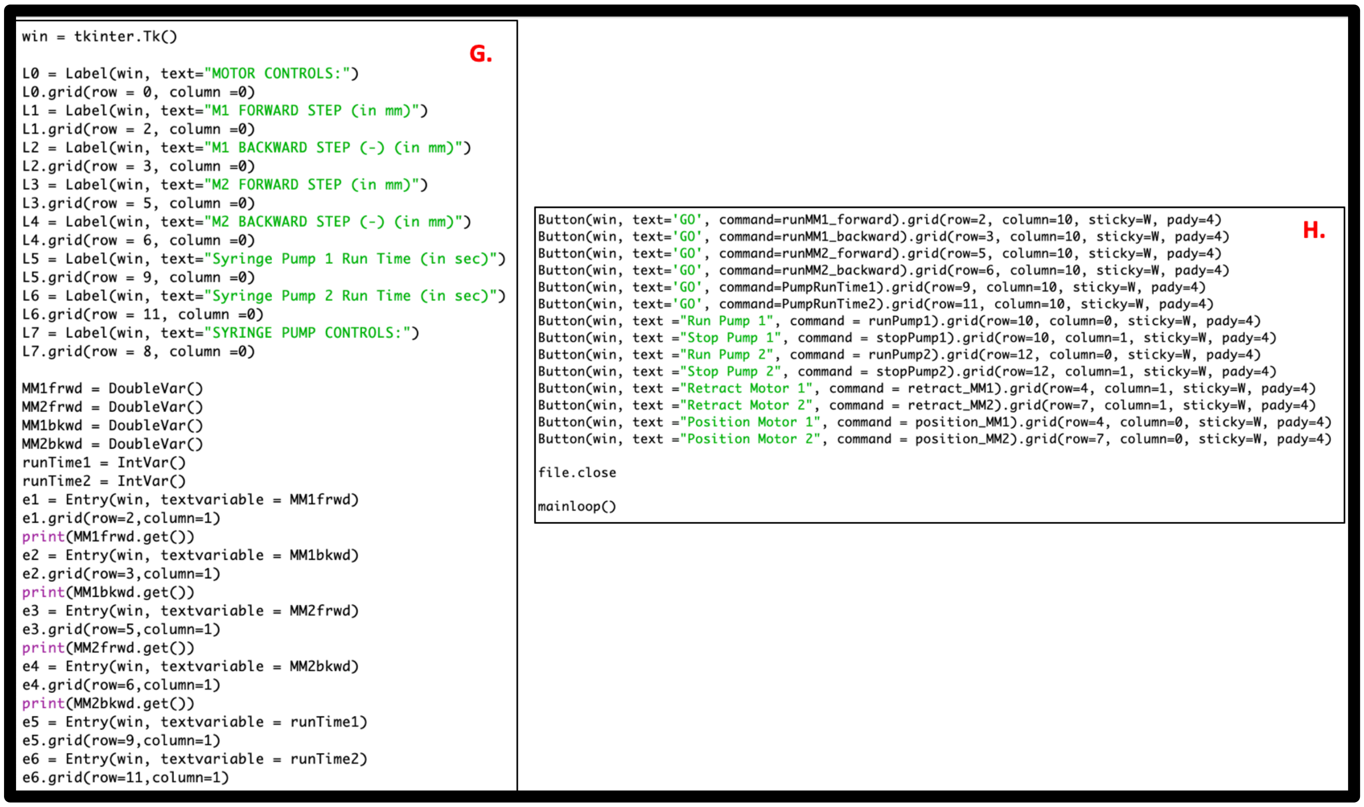
C.

```
# functions for micromotor features
ts = time.time()
def runMM1_forward():
    LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM_1.vi')
    VI._FlagAsMethod("Call")
    StepCOM5 = MM1frwd.get()
    VI.setcontrolvalue('StepCOM5', StepCOM5)

    VI.Call()
    # output position
    time.sleep(7) # wait for motor to move
    LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM1_pos.vi')
    VI._FlagAsMethod("Call")
    VI.Call()
    output1 = VI.getcontrolvalue('CurrentPosition')
    print(datetime.datetime.time(datetime.datetime.now()), 'MM1 Forward MM1 Position:
    print(output1)
    # output position record to file
    today = datetime.datetime.time(datetime.datetime.now())
    file.write(str(today))
    file.write(' MM1 Forward')
    file.write(' MM1 Position: ')
    file.write('%f' % output1)
    file.write('mm \n')
def runMM1_backward():
    LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM_1.vi')
    VI._FlagAsMethod("Call")
    StepCOM5 = MM1bkwd.get()
    VI.setcontrolvalue('StepCOM5', StepCOM5)

    VI.Call()
    # output position
    time.sleep(7) # wait for motor to move
    #LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM1_pos.vi')
    VI._FlagAsMethod("Call")
    VI.Call()
    output2 = VI.getcontrolvalue('CurrentPosition')
    print(datetime.datetime.time(datetime.datetime.now()), 'MM1 Backward MM1 Position
    print(output2)
    # output position record to file
    today = datetime.datetime.time(datetime.datetime.now())
```

D.

```
def runMM2_forward():
    LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM_2.vi')
    VI._FlagAsMethod("Call")
    StepCOM6 = MM2frwd.get()
    VI.setcontrolvalue('StepCOM6', StepCOM6)

    VI.Call()
    # output position
    time.sleep(5) # wait for motor to move
    LabVIEW2 = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW2.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM2_pos.vi
    VI._FlagAsMethod("Call")
    VI.Call()
    output3 = VI.getcontrolvalue('CurrentPosition')
    print(datetime.datetime.time(datetime.datetime.now()), 'MM2 Forward MM2 Position
    print(output3)
    # output position record to file
    today = datetime.datetime.time(datetime.datetime.now())
    file.write(str(today))
    file.write(' MM2 Forward')
    file.write(' MM2 Position: ')
    file.write('%f' % output3)
    file.write('mm \n')
def runMM2_backward():
    LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM_2.vi')
    VI._FlagAsMethod("Call")
    StepCOM6 = MM2bkwd.get()
    VI.setcontrolvalue('StepCOM6', StepCOM6)

    VI.Call()
    # output position
    time.sleep(5) # wait for motor to move
    LabVIEW = win32com.client.Dispatch("Labview.Application")
    VI = LabVIEW.getvireference('C:\\Users\\admin-bishop\\PythonScripts\\MM2_pos.vi'
    VI._FlagAsMethod("Call")
    VI.Call()
    output4 = VI.getcontrolvalue('CurrentPosition')
    print(datetime.datetime.time(datetime.datetime.now()), 'MM2 Backward MM2 Positio
    print(output4)
    # output position record to file
    today = datetime.datetime.time(datetime.datetime.now())
    file.write(str(today))
    file.write(' MM2 Backward')
```

Figure 6. Continued.



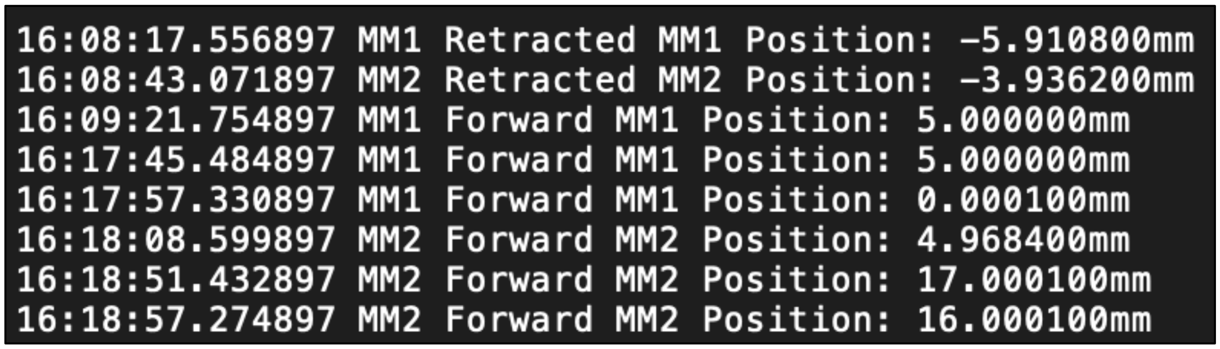
```
win = tkinter.Tk()

L0 = Label(win, text="MOTOR CONTROLS:")
L0.grid(row = 0, column =0)
L1 = Label(win, text="M1 FORWARD STEP (in mm)")
L1.grid(row = 2, column =0)
L2 = Label(win, text="M1 BACKWARD STEP (-) (in mm)")
L2.grid(row = 3, column =0)
L3 = Label(win, text="M2 FORWARD STEP (in mm)")
L3.grid(row = 5, column =0)
L4 = Label(win, text="M2 BACKWARD STEP (-) (in mm)")
L4.grid(row = 6, column =0)
L5 = Label(win, text="Syringe Pump 1 Run Time (in sec)")
L5.grid(row = 9, column =0)
L6 = Label(win, text="Syringe Pump 2 Run Time (in sec)")
L6.grid(row = 11, column =0)
L7 = Label(win, text="SYRINGE PUMP CONTROLS:")
L7.grid(row = 8, column =0)

MM1frwd = DoubleVar()
MM2frwd = DoubleVar()
MM1bkwd = DoubleVar()
MM2bkwd = DoubleVar()
runTime1 = IntVar()
runTime2 = IntVar()
e1 = Entry(win, textvariable = MM1frwd)
e1.grid(row=2,column=1)
print(MM1frwd.get())
e2 = Entry(win, textvariable = MM1bkwd)
e2.grid(row=3,column=1)
print(MM1bkwd.get())
e3 = Entry(win, textvariable = MM2frwd)
e3.grid(row=5,column=1)
print(MM2frwd.get())
e4 = Entry(win, textvariable = MM2bkwd)
e4.grid(row=6,column=1)
print(MM2bkwd.get())
e5 = Entry(win, textvariable = runTime1)
e5.grid(row=9,column=1)
e6 = Entry(win, textvariable = runTime2)
e6.grid(row=11,column=1)
```

```
Button(win, text='GO', command=runMM1_forward).grid(row=2, column=10, sticky=W, pady=4)
Button(win, text='GO', command=runMM1_backward).grid(row=3, column=10, sticky=W, pady=4)
Button(win, text='GO', command=runMM2_forward).grid(row=5, column=10, sticky=W, pady=4)
Button(win, text='GO', command=runMM2_backward).grid(row=6, column=10, sticky=W, pady=4)
Button(win, text='GO', command=PumpRunTime1).grid(row=9, column=10, sticky=W, pady=4)
Button(win, text='GO', command=PumpRunTime2).grid(row=11, column=10, sticky=W, pady=4)
Button(win, text ="Run Pump 1", command = runPump1).grid(row=10, column=0, sticky=W, pady=4)
Button(win, text ="Stop Pump 1", command = stopPump1).grid(row=10, column=1, sticky=W, pady=4)
Button(win, text ="Run Pump 2", command = runPump2).grid(row=12, column=0, sticky=W, pady=4)
Button(win, text ="Stop Pump 2", command = stopPump2).grid(row=12, column=1, sticky=W, pady=4)
Button(win, text ="Retract Motor 1", command = retract_MM1).grid(row=4, column=1, sticky=W, pady=4)
Button(win, text ="Retract Motor 2", command = retract_MM2).grid(row=7, column=1, sticky=W, pady=4)
Button(win, text ="Position Motor 1", command = position_MM1).grid(row=4, column=0, sticky=W, pady=4)
Button(win, text ="Position Motor 2", command = position_MM2).grid(row=7, column=0, sticky=W, pady=4)

file.close

mainloop()
```

Figure 6. Continued.

Data Recording

To maintain a record of how each polybubble was fabricated, a text file is continually updated with the details of each session, including the timestamps associated with execution of each control. **Figure 7** indicates a sample text file associated with the fabrication of a sample polybubble.



```
16:08:17.556897 MM1 Retracted MM1 Position: -5.910800mm
16:08:43.071897 MM2 Retracted MM2 Position: -3.936200mm
16:09:21.754897 MM1 Forward MM1 Position: 5.000000mm
16:17:45.484897 MM1 Forward MM1 Position: 5.000000mm
16:17:57.330897 MM1 Forward MM1 Position: 0.000100mm
16:18:08.599897 MM2 Forward MM2 Position: 4.968400mm
16:18:51.432897 MM2 Forward MM2 Position: 17.000100mm
16:18:57.274897 MM2 Forward MM2 Position: 16.000100mm
```

Figure 7. Example of text file output recording the fabrication protocol, as executed via user control within Python.

System Construction

Figure 8 displays the platform used to house the syringe pumps, micromotors, and injection site. The platform is constructed entirely of wood. A goniometer is affixed to the drug-loading micromotor to control the angle at which drug is injected within the polybubble.

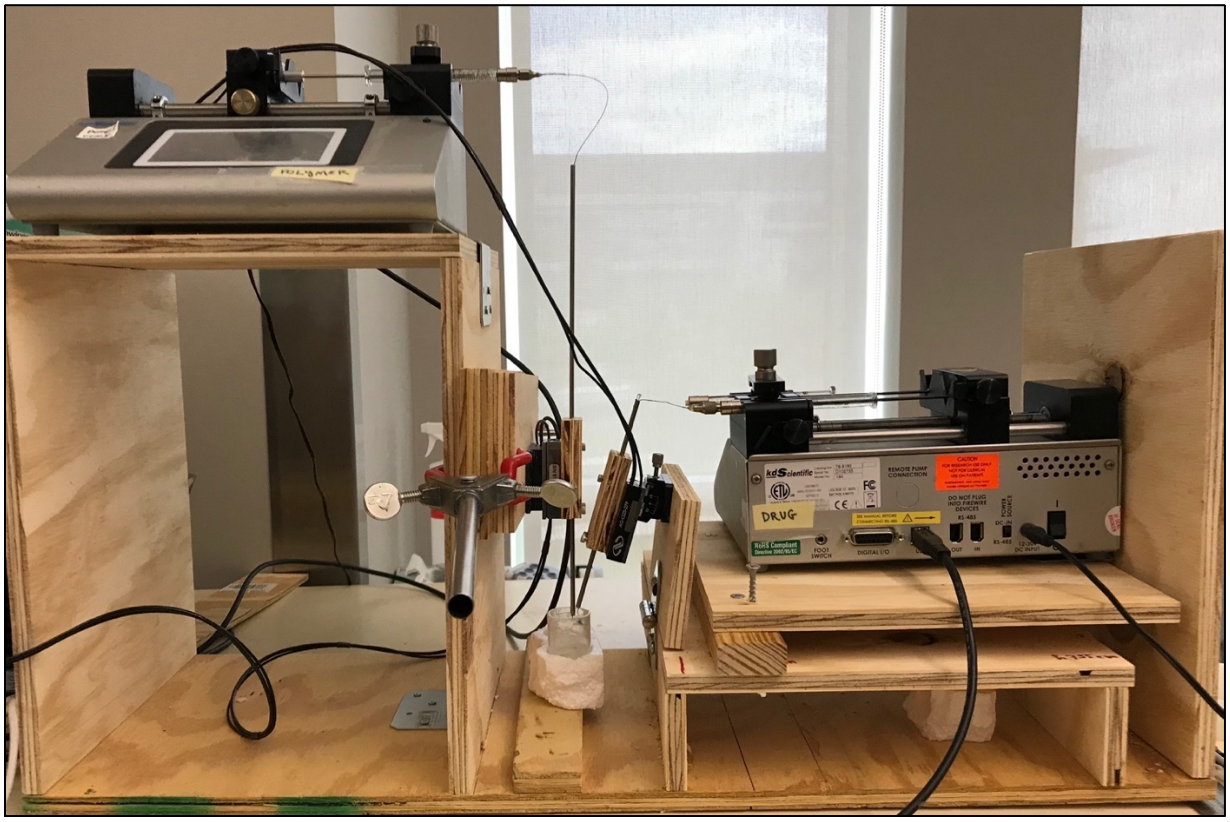


Figure 8. Display of the automated polybubble injection platform.

Polymer and Drug Loading

As consistency is the ultimate goal of the polybubble fabrication process, additional measures are taken to maintain consistent positioning of the polymer and drug tubing between each fabrication process. The silicone caps are designed to fit between the inner and outer tubing within the drug and polymer shafts and remain stable throughout the injection process, helping to ensure that the position of the polymer and drug tubing remains consistent across all polybubble fabrication procedures.

Figure 9 shows the silicone caps used to stabilize the drug and polymer tubes during polybubble fabrication.

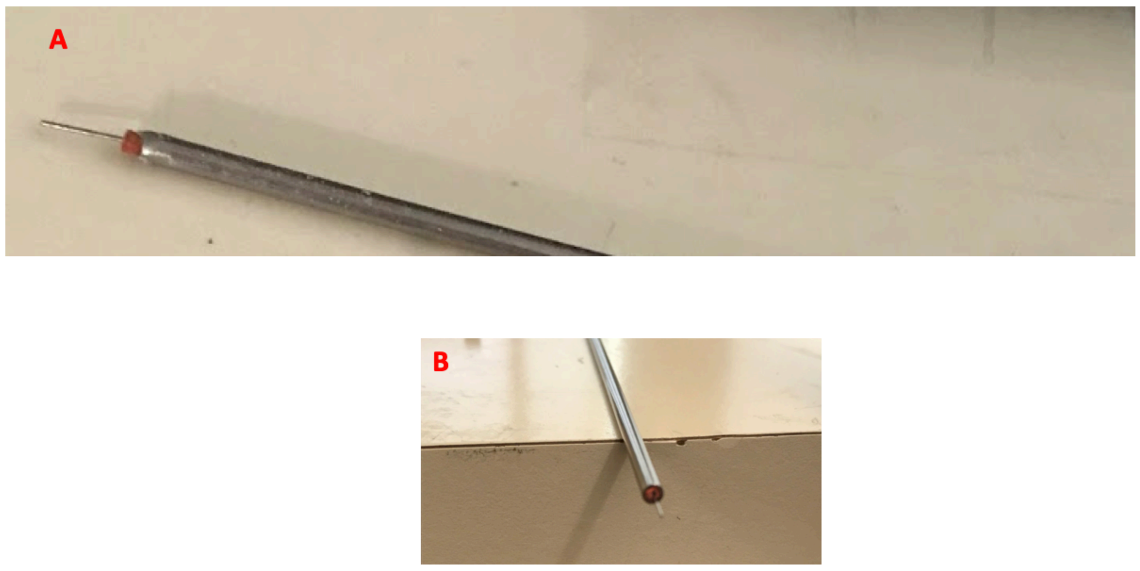


Figure 9. Silicone capping mechanism used for the A) the polymer tubing and B) the drug tubing.

Spatial Alignment of Polymer and Drug Tubing

The positioning of the polymer and drug tubes not only restricts the size of the polybubble but also controls the location of cargo injection. The approach described in this paper is catered to dispensing cargo to the center of the polybubble. It is important to note, however, that the prototype described here can be adapted to inject drug at any desired location within the polybubble, merely by controlling the angle of the drug injection and the positioning of the drug-carrying micromotor (**Figure 10**). Moreover, the size of the polybubble can also be controlled by varying the positioning of the polymer-carrying micromotor, as well as the timing of the polymer-dispensing syringe pump.

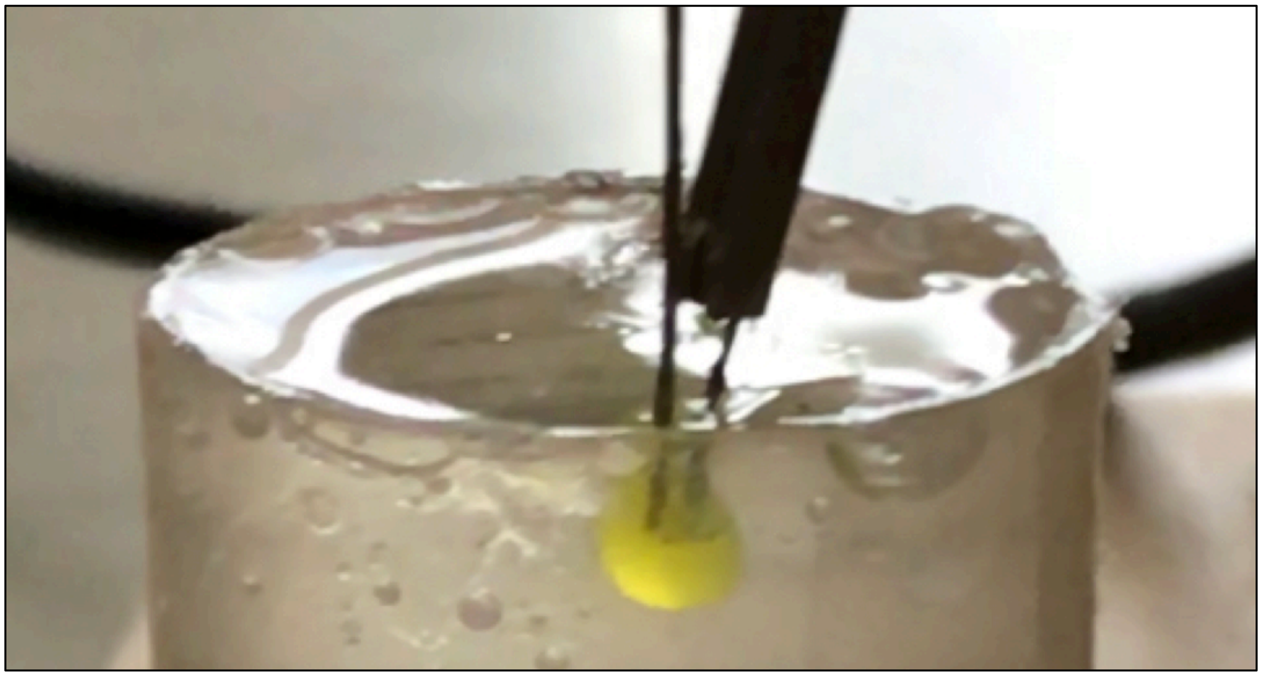


Figure 10. Demonstration of the drug injection process.

Polybubble Fabrication

The drug and polymer dispensing steps of the polybubble fabrication procedure take place in less than 2 minutes and require no user intervention, aside from placement and removal of the injection vial containing 10% CMC.

Polybubbles of approximately 3 mm were formed using the previously outlined fabrication protocol, as shown in **Figure 11**. With the exception of the placement and removal of the injection vial, the polybubble fabrication process is automated beginning from polymer loading and ending with drug injection.

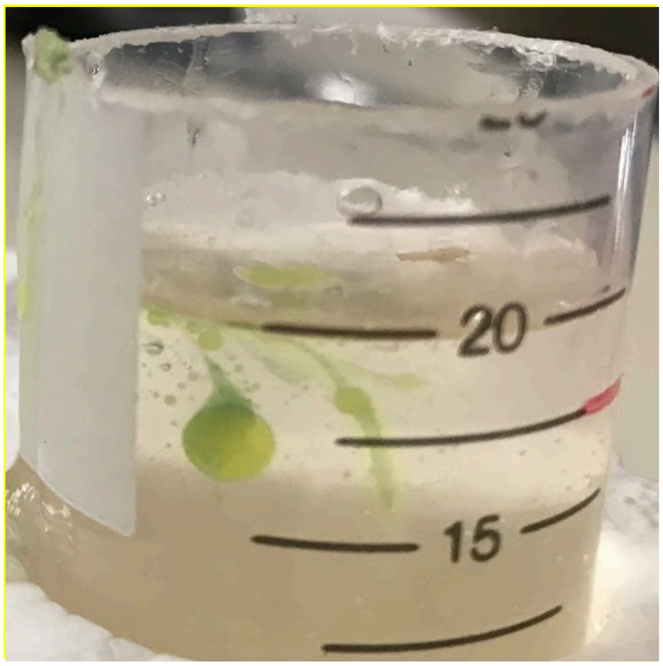


Figure 11. Example of a polybubble fabricated using the Python GUI.

CHAPTER IV

CONCLUSIONS AND FUTURE DIRECTIONS

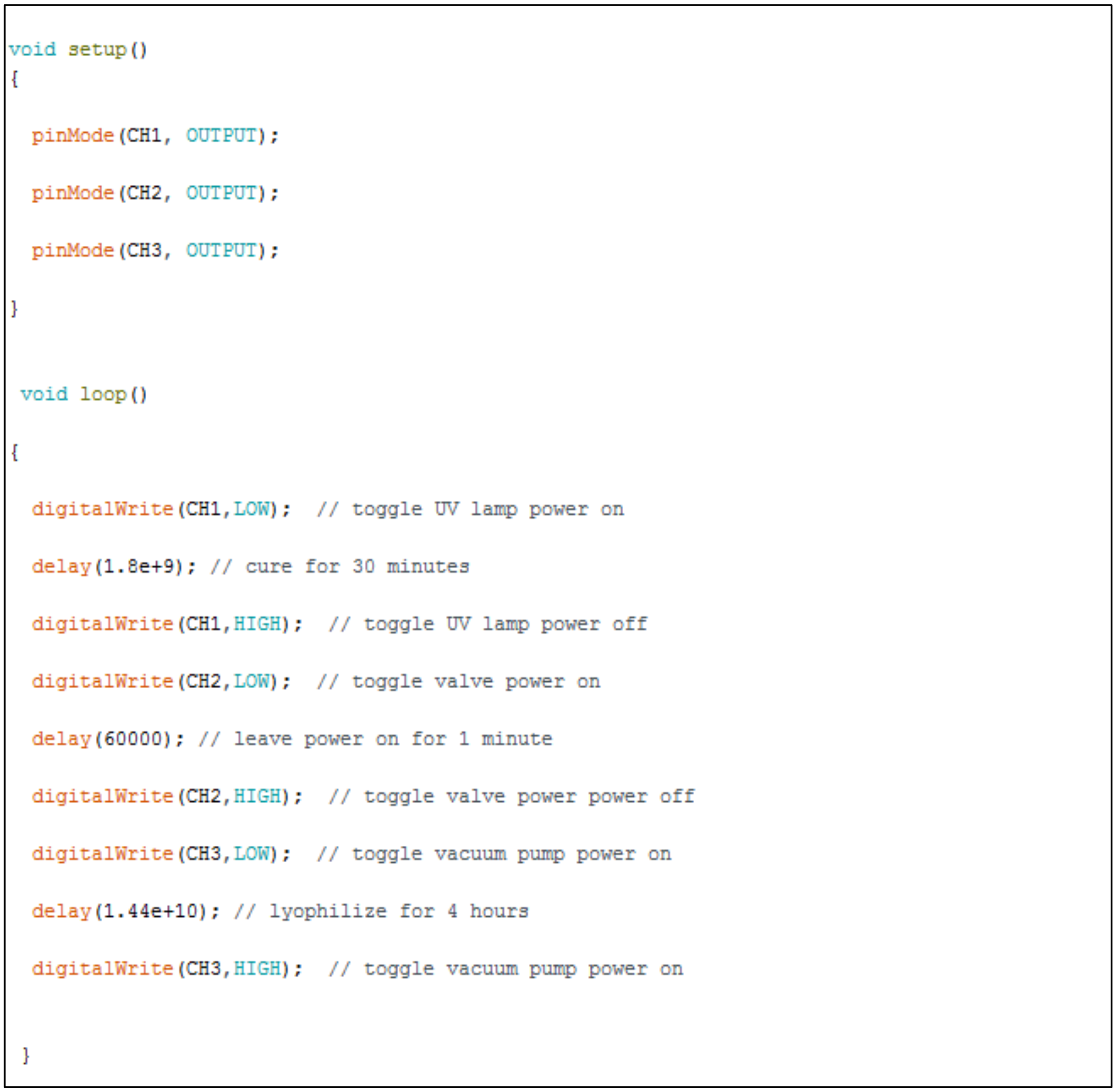
While the automated polybubble fabrication system in its current state can successfully form and inject polybubbles ranging from approximately 1-3 mm in diameter, current research efforts are focused on developing a method that allows for drug to be consistently injected into the center of polybubbles 1 mm in diameter. Consistently injecting the cargo within the center of the polybubble represents a critical step on the path to ultimate translation of the platform. Specifically, injecting drug into the center of the polybubbles allows for the most predictable patterns of drug release; as the PCL degrades via bulk degradation within the body, drug will be released over time. Work is currently being done to ensure that the system consistently produces and injects polybubbles of the desired size.

To further investigate the potential of the automated polybubble fabrication platform as a translatable technology, current studies are being undertaken to characterize polybubbles produced using the automated process. In these studies, polybubble morphology will be characterized using scanning electron microscopy; moreover, confocal scanning laser microscopy will be used to compare the location of cargo injection across multiple polybubble samples. Further studies will also compare the change in polybubble size before and after lyophilization.

In addition to the ongoing studies described above, future studies will analyze the release kinetics of the fabricated polybubbles. Drug release will be quantified upon altering the input parameters of the system; namely, run time of both the polymer and drug syringe pumps, as well as the location of the cargo within the polybubble. In this way, we aim for our polybubble fabrication system to provide not only full user-control over the size of the polybubbles but also the release kinetics, allowing each polybubble to be catered to the precise needs of both the therapeutic application and patient.

However, before the automated polybubble fabrication system can perform in high-throughput applications, several modifications are needed. First, to better accommodate high volumes of polybubble production, we plan to modify the site of polybubble injection such that polybubbles can be dispensed in a rotating fashion. In theory, this rotating platform could house multiple rows of injection vials and would be programmed to rotate by a specified angle following each polybubble injection. Such a modification would significantly impact the speed of polybubble production and cater to commercialization of the system and its eventual distribution.

Moreover, to further adapt the platform for such high-throughput applications, the polybubble fabrication system can be linked to an automated UV curing and lyophilization process. After forming polybubbles, vials can be transported via an automated conveyor belt to a UV curing and subsequent vacuum chamber. These modifications will allow the polybubble fabrication process to be entirely automated up to the point of use. Steps to integrate the UV curing and lyophilization process with the fabrication process are currently being taken, although further integration is needed. **Figure 12** demonstrates a program sample created using Arduino that can be further adapted for controlling the timing of the vacuum chamber, UV lamp, and cryogenic valve that regulates the release of nitrogen gas. A four-channel relay is used in combination with an Arduino Uno to interface between the electrical components and computer.



```
void setup()
{

  pinMode(CH1, OUTPUT);

  pinMode(CH2, OUTPUT);

  pinMode(CH3, OUTPUT);

}


 void loop()

{

  digitalWrite(CH1,LOW);  // toggle UV lamp power on

  delay(1.8e+9); // cure for 30 minutes

  digitalWrite(CH1,HIGH);  // toggle UV lamp power off

  digitalWrite(CH2,LOW);  // toggle valve power on

  delay(60000); // leave power on for 1 minute

  digitalWrite(CH2,HIGH);  // toggle valve power power off

  digitalWrite(CH3,LOW);  // toggle vacuum pump power on

  delay(1.44e+10); // lyophilize for 4 hours

  digitalWrite(CH3,HIGH);  // toggle vacuum pump power on


 }
```

Figure 12. Demonstration of an Arduino program that can be used to control the timing of lyophilization and UV-curing procedures, in combination with a four-channel relay. CH1, CH2, and CH3 each represent different relay channels that can be used for controlling the UV lamp, cryogenic valve, and vacuum pump, respectively.

It is our goal that, by further modifying the automated polybubble fabrication system for use in high-throughput settings, such a system will not only broaden the accessibility of healthcare but also optimize the therapeutic potential of drugs. The versatility of this novel approach to manufacturing polybubbles allows for each treatment to be catered to a unique patient by varying the type, dose, and, potentially, release kinetics of any controlled-release therapeutic. In the developing world, this translates into a wide range of treatments that can be manufactured using a single, transportable system, all by the push of a single button. By aiming to eventually deploy our polybubble fabrication system across the world, we hope to not only revolutionize medical care in the developing world, but also save the lives of the millions of adults and children that die each year due to lack of access to disease treatment.

REFERENCES

1. D. Dutta, C. Fauer, K. Hickey, M. Salifu, S.E. Stabenfeldt, Tunable delayed controlled release profile from layered polymeric microparticles, J Mater Chem B 5(23) (2017) 4487-4498.

2. L. Zhao, G. Shen, G. Ma, X. Yan, Engineering and delivery of nanocolloids of hydrophobic drugs, Adv Colloid Interface Sci 249 (2017) 308-320.

3. C. Wischke, S.P. Schwendeman, Principles of encapsulating hydrophobic drugs in PLA/PLGA microparticles, Int J Pharm 364(2) (2008) 298-327.

4. W.B. Liechty, D.R. Kryscio, B.V. Slaughter, N.A. Peppas, Polymers for drug delivery systems, Annu Rev Chem Biomol Eng 1 (2010) 149-73.

5. K.J. McHugh, T.D. Nguyen, A.R. Linehan, D. Yang, A.M. Behrens, S. Rose, Z.L. Tochka, S.Y. Tzeng, J.J. Norman, A.C. Anselmo, X. Xu, S. Tomasic, M.A. Taylor, J. Lu, R. Guarecuco, R. Langer, A. Jaklenec, Fabrication of fillable microparticles and other complex 3D microstructures, Science 357(6356) (2017) 1138-1142.

6. S.E. Gratton, P.D. Pohlhaus, J. Lee, J. Guo, M.J. Cho, J.M. Desimone, Nanofabricated particles for engineered drug therapies: a preliminary biodistribution study of PRINT nanoparticles, J Control Release 121(1-2) (2007) 10-8.

7. J.E.L. Jaeyun Kim, Soo Hyeon Lee, Soo Hyeon Lee, Jung Ho Yu, Jung Hee Lee, Tae Gwan Park,*and Taeghwan Hyeon, Designed Fabrication of a Multifunctional Polymer Nanomedical Platform for Simultaneous Cancer-Targeted Imaging and Magnetically Guided Drug Delivery, Advanced Materials 20 (2008) 478-483.

8. Massachusetts Institute of Technology, python_labview_automation. Accessed 2016.

9. S.W. Lei Cai, Poly(ε-caprolactone) acrylates synthesized using a facile method for fabricating networks to achieve controllable physicochemical properties and tunable cell responses, Polymer 51(1) (2010) 164-177.