Design and Development of an Accelerated Material Synthesis Platform for Automated Materials Research

by

Eunice I. Aissi

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2024

Authored by: Eunice I. Aissi
Department of Mechanical Engineering
August 23, 2024

Certified by: Tonio Buonassisi
Professor of Mechanical Engineering, Thesis Supervisor

Accepted by: Nicolas Hadjiconstantinou
Chairman, Department Committee on Graduate Theses

Design and Development of an Accelerated Material Synthesis Platform for Automated Materials Research

by

Eunice I. Aissi

Submitted to the Department of Mechanical Engineering
on August 23, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

ABSTRACT

Materials development is the foundation for innovation in many industries and fields, however, this process is traditionally slow and resource-intensive. Most often, new materials are developed and characterized on the time scale of years which can limit the pace of scientific and industry innovation. I address the material synthesis and characterization bottleneck by presenting a framework that I believe is suitable for smaller labs: Self-built, low-cost automation. The design philosophy is to de-risk the lab automation process by keeping costs low, failing fast, and leveraging common resources in electronic systems and additive manufacturing. I present an improved version of a low-cost but high-throughput inkjet material printer developed by Siemenn *et al.* and adapted to operation in the glovebox, hood, and benchtop environments. The tool is capable of depositing gradients of droplets with unique compositions at a rate of up to 1000 materials per minute, is self-built and cost around $500. I also present a computer-vision-enabled high-throughput material characterization algorithm for stability quantification through color degradation. The synthesis and characterization methods are validated on a methylammonium lead iodide ($MAPbI_3$) and formamidinium lead iodide ($FAPbI_3$) perovskite material system. X-ray diffraction (XRD), X-ray photoelectron spectroscopy (XPS), and hyperspectral imaging measurements show equivalence between high-throughput synthesis and more traditional spin-coating methods. Results obtained through the high-throughput stability characterization method are aligned with stability trends reported in literature and has an accuracy of 96.9% when compared to ground-truth degradation as measured by a domain expert.

Thesis supervisor: Tonio Buonassisi
Title: Professor of Mechanical Engineering

Acknowledgments

I would like to express my deepest appreciation to my Advisor Tonio Buonassisi for his professional guidance, feedback and support. I'm also extremely grateful to Alexander E. Siemenn and Basita Das for their mentorship and continued belief in my potential. Additionally, this endeavor would not have been possible without the generous support from First Solar, who partially funded the work presented in this thesis.

I could not have undertaken this journey without my undergraduate advisor Russell Scott Ray who mentored me and introduced me to the world of academia and my classmate and, at times teacher, Fang Sheng for her support and instruction.

Lastly, but definitely not least, words cannot express my gratitude to my family and partner who supported me and lifted me up when I doubted myself.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Automation in research is a growing field that has recently gained attention for its potential to accelerate and improve the quality of scientific outcomes.[1] However, acquiring the human and capital resources to establish and maintain these advanced tools can be daunting. For smaller labs investigating a wide range of energy materials, there is demand for low-cost, flexible, high-throughput (HT), and high-quality materials synthesis and characterization tools.

In this thesis, I develop low-cost high-throughput synthesis and characterization methods for automated materials exploration. I present an improved version of a low-cost but high-throughput ink-jet material printer developed by Siemenn *et al.* [2] and adapted to operation in the glovebox, hood, and bench-top environments. The printer takes two or more fluid precursors and prints a materials gradient within a set of droplets, line segments, or another user-specified shape. The tool is self-built, leveraging a low-cost 3D printer chassis, and is affordable at about \$500. Our build cycle takes one day and utilizes open-source platforms for its firmware. The printer can deposit up to 1,000 different material compositions in under one minute and is easy to use, requiring only three steps: creating a print pattern, loading the precursors, and printing the materials.

One area of interest for accelerated materials discovery is perovskite materials, compounds of structure ABX_3, where A and B are cations and X is a non metallic anion, that can be used as solar cell materials.[3] The perovskite crystal structure was first discovered in 1839[4] and since then a subset of these materials know as metal halide perovskites have shown great promise in optoelectronic applications such as LEDs, lasers, and solar cells. [5] Metal halide perovskite solar cells have been reported to reach power conversion efficiencies as high as 26.1% while being light weight and easy of synthesize. [6] However, they currently suffer from low stability and can degrade quickly when exposed to irradiation and heat. [7] This limits their applicability as solar cell materials and has slowed their adoption. Consequently, there has been great interest in optimizing and discovering perovskite solar cell ink formulas to both improve their stability and understand their modes of degradation. The material space is however vast for both composition and additives making it a suitable materials system for exploration through high-throughput combinatorial material synthesis.

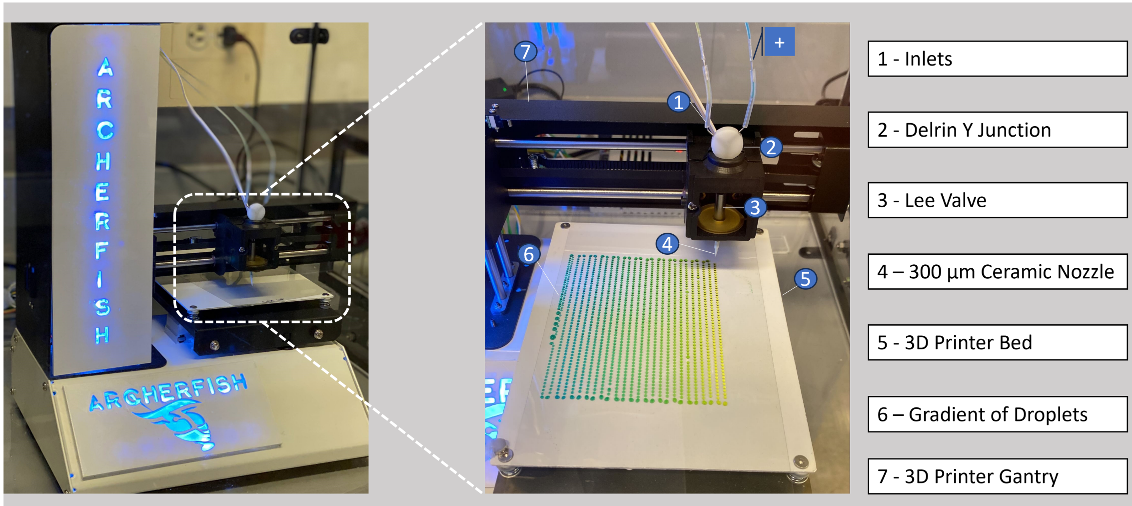


Figure 1.1: The Archerfish platform, is a retrofitted 3D printer that can deposit gradients of droplets at a rate of up to 1000 droplets a minute. It consists of two home-built 3 ml syringe pumps which dispense streams of fluid that join at a y-junction and are generated into droplets by a Lee valve forcing flow through a 300 μm diameter ceramic nozzle. Note that the fluid inlets from the syringe pumps are marked by a + sign.

I therefore validate the material-synthesis capacity of the methods I develop in this thesis on the methylammonium lead iodide ($MAPbI_3$) and formamidinium lead iodide ($FAPbI_3$) perovskite material system by showing equivalence to traditional spin-coated samples through X-ray diffraction, X-ray photoelectron spectroscopy (XPS), and hyperspectral imaging measurements. I also demonstrate high-throughput characterization of the perovskite materials through an algorithm that quantifies color changes in each material as a proxy for material degradation. In summary, I present and validate a framework of customized automation focused on improving R&D reproducibility and speed for smaller academic labs.

1.2 An overview of the Archerfish Material Printer

The Archerfish platform illustrated in Figure 1.1 was first invented by Alexander Siemenn and James Serdy in the Accelerated Materials Development for Sustainability Lab. [2]. Archerfish is a low-cost and high-throughput droplet deposition system that can create gradients of droplets at a rate of up to 1000 droplets per minute. In the first version of this platform, the plungers of two 3 ml syringes were replaced with a 4-40 lead screw and actuated with 28BYJ-48 stepper motors and ULN2003 drivers. These home-built pumps where controlled via an Arduino board in the ascending and descending stair-case flow pattern in Figure 1.2. The two streams from each pumps were then joined in a spherical y-joint machined from delrin and ink-jetted through a piezoelectric lee valve and out of a 300 μm ceramic nozzle. Lastly, the nozzle of a Monoprice MP select Mini Pro 3D printer was removed and replaced with the droplet generator to deposit the droplets at locations specified by the g-code uploaded to the printer. With this design, researchers at the Accelerated Materials development lab were able to create a high-throughput materials synthesis platform for use with simple solution

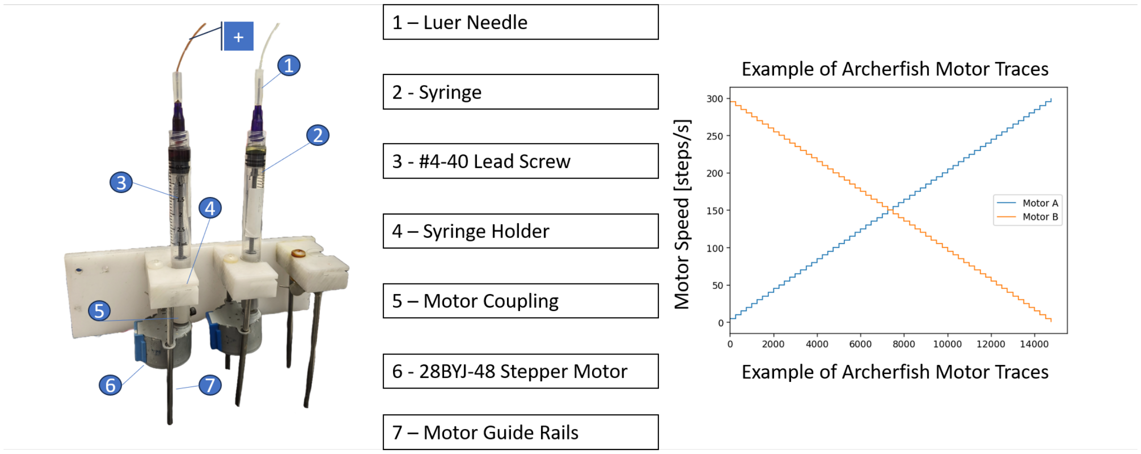


Figure 1.2: The Archerfish syringes are driven by $3 28BYJ- 48 stepper motors coupled to a 4-40 lead screw to force the plunger forward and dispense fluid. The motors create a fluid gradient from 100% of the fluid in syringe B controlled by motor B to 100% of the fluid in syringe A controlled by motor A using a descending and ascending speed profiles respectively. Note that the fluid outlets from the syringe pumps are marked by a + sign.

based chemistries. However, the system had several limitations included a lack of proper communications protocols that prevented it from being used in environmental enclosures such as a glove box, a near necessity for experiments with lead containing perovskites.

1.3 Thesis Overview

This master's thesis designs and develops improved high-throughput material synthesis and characterization platforms and algorithms to accelerate the discovery of novel energy materials.

Redesigning the Archerfish High-Throughput (HT) Platform: In order to allow for a wider range of material experiments, I implement several improvements to the communications and hardware components of the Archerfish platform. Chapter 2 describes these improvements to the Archerfish system's hardware and electronic components while motivating and explaining design choices.

Algorithms for HT Characterization: HT synthesis without characterization limits the pace of materials discovery. Chapter 3 describes an algorithm I developed to extract a key material property of perovskite materials, stability, printed using a high-throughput droplet deposition system like Archerfish.

Limitations of The Archerfish System: Despite its potential, the Archerfish platform is still imperfect. Chapter 4 describes its limitations and potential pathways to further improvements.

Chapter 2

Improving Archerfish: Advancements in Communication Protocols and Hardware

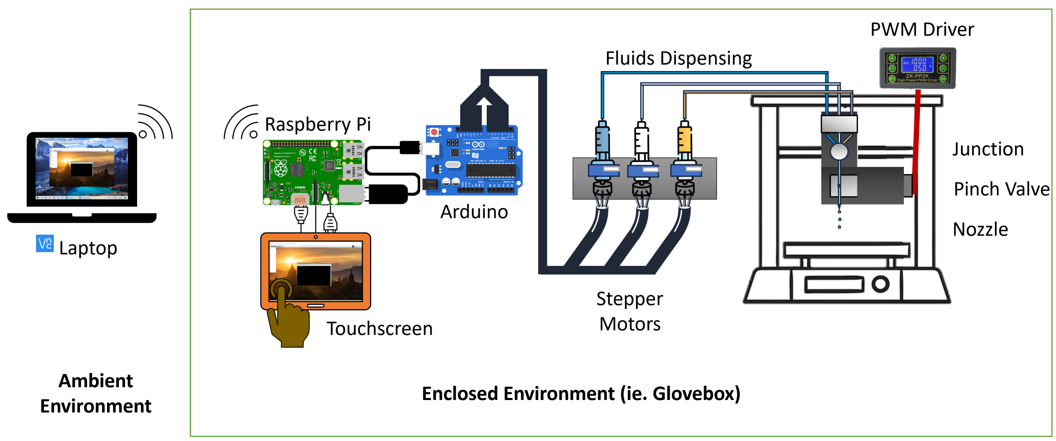


Figure 2.1: The second iteration of Archerfish was developed to address the chemical limitations of the initial proof of concept. Mainly, the system was fitted with wireless communication and a GUI for use with lead containing perovskites inside a glovebox.

2.1 Introduction and Overview

Section 1.2 describes the proof of concept version of Archerfish first designed by Siemenn *et al.* [2], however, it needed several updates in order to be utilized for chemical experiments. In this chapter I describe the updates I implemented for the communications, user interface, and aspects of the hardware in order to allow us to run chemical experiments in an inert environment. These updates to version one of Archerfish system are depicted in Figure 2.1.

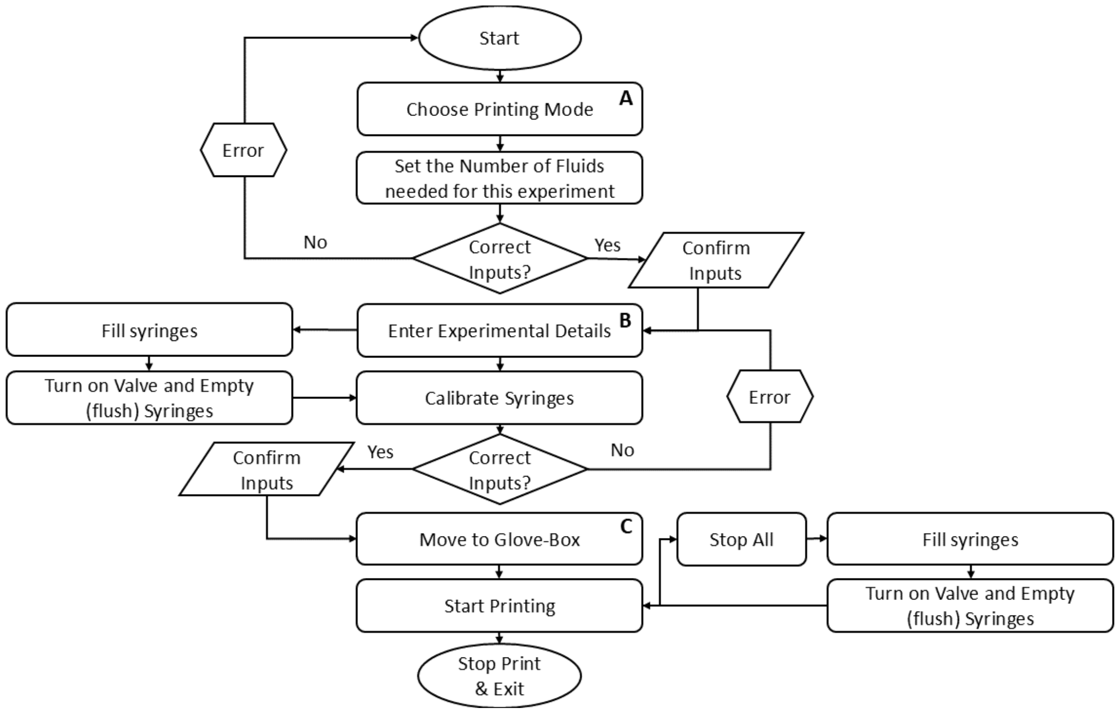


Figure 2.2: The flow diagram for the Archerfish system's Graphical User Interface (GUI).

2.2 Improvements to the Communications Architecture

The primary limitation of the Archerfish system was its communications architecture. Originally it utilized prescribed printing flow rates for each syringe and a set deposition pattern that could be activated once the machine was turned on. To change precursors, the user had to remove the syringe pump motors and mount a smaller pump pre-programmed to move in reverse and draw in liquid. This meant that there was no way to easily change precursors, the composition of the gradient, how many droplets were printed, or where the droplets were deposited. Chemical experiments, however, require flexibility for composition and deposition as well as peripheral features like data logging. Moreover, chemical synthesis needed to occur in an enclosed environment such as a glove box where users had limited mobility and dexterity due to bulky chemical safety gloves. Features that required fine movements for reoccurring actions such as exchanging a motor to reload precursors were therefore intractable.

To allow for more flexibility, a Raspberry Pi was added as a control center to run a Graphical User Interface (GUI). The pi can be controlled wirelessly from outside the glove box via a bluetooth connection to a Virtual Network Computing (VNC) client. The VNC mirrored the raspberry pi screen on a laptop outside the glove box, allowing a user to set printing parameters through the GUI and control the hardware inside the glove box. Once the user connects to the raspberry pi, they are instructed to choose one of three printing modes: batch, two-fluid gradient, and constant+ two-fluid gradient. The pi then establishes a connection with the arduino and sends motor control instructions via serial communication to set the flow rates for the syringe pumps.

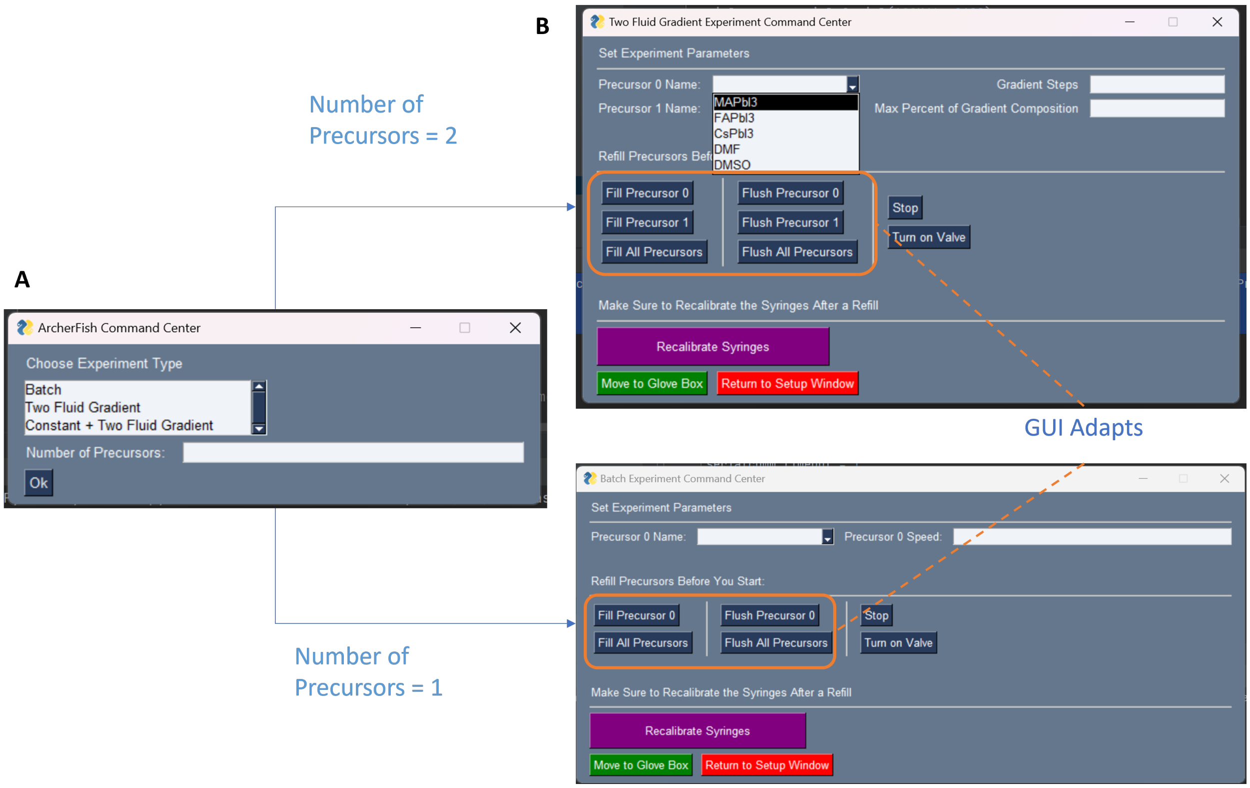
In batch mode, the relative flow rates of each syringe pump is set by the user in the GUI, the printed droplets are therefore multiple copies of the same composition. In two-fluid gradient mode, the flow rates of two syringe pumps change from a starting percentage of one precursor to a percentage of the other, creating a linear gradient between the two precursors loaded into the syringe pumps. This percentage value is chosen by the user and allows for finer resolution into sub-regions of a gradient of interest. For example, if a user notices interesting features in the regions between the droplets of composition $A_{0.25}B_{0.75}$ and the $A_{0.75}B_{0.25}$, they could specify these compositions as end points of a new gradient with all the droplets between this range, thereby zooming into this compositional space. In constant+ two-fluid gradient mode, a third syringe pump dispenses a precursor at a constant flow rate alongside the gradient adding a third component to the droplet composition, this is usually a solvent. After choosing a printing mode, the GUI asks for user inputs regarding experimental details such as the chemical names for the loaded precursors and saves these in a text file called Experimental log for later reference. Once all experimental details have been captured, the GUI goes to a simple run screen that features a prominent start and stop button. At this point, the user can enter the glove-box space using the bulky protective glove-box gloves and any further interactions with the Archerfish system takes place on the touch screen inside. The touchscreen is to facilitate use of the system and features two large start and stop buttons that can be pressed even with the bulky gloves that decrease the dexterity of the user.

It is important to note that the GUI also guides the user with confirmation and error messages to ensure that all necessary data is properly captured and the user follows the proper path through the GUI. The full GUI path is shown in Figure 2.2 and images of the three main windows along the path are shown in Figure 2.3. The GUI windows are dynamic and adaptable to each experimental set-up, they change their input fields depending on the type of print and the number of fluids the experiment requires.

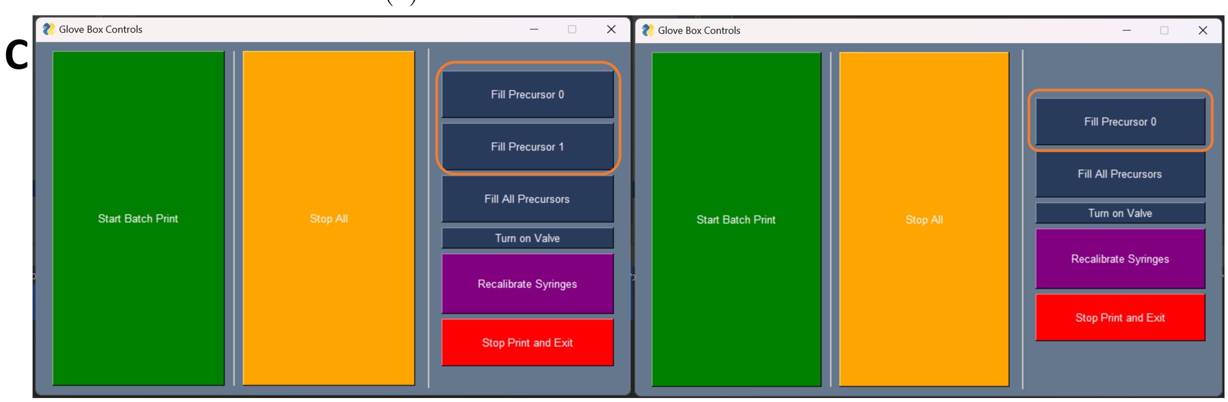
2.3 Hardware Updates: Enhancing Droplet Uniformity, and Crystallization

Beyond the flexibility of the system, two challenges limited our ability to print gradients of perovskite droplets: droplet uniformity, and uniform annealing. Test prints revealed that the unknown internal volume of the initial Archerfish Lee valve resulted in reservoirs of trapped fluid that required large volumes of transfer fluid to completely clear out. The unknown chemical composition of the wetted materials also created a potential for side reactions that could adversely impact the deposited materials purity. Consequently, the lee valve was replaced with a 24 V Precigenome solenoid pinch valve to provide breaks in the outlet fluid stream and generate droplets. Unlike the lee valve, the pinch valve's surfaces are not wetted, the actuator strikes a 1/32" ID x 3/32" OD silicone tube containing outlet fluid in a square wave pattern. A new 3D printed casing shown in Figure 2.4 was designed to hold the valve and the nozzle steady while printing as this was shown to minimize satellite droplets and improve droplet uniformity.

Annealing is a crucial step for perovskite crystallization and therefore requires a uniform



(a) First two windows in the GUI.



(b) Last window of the GUI.

Figure 2.3: The flow diagram for the Archerfish system's Graphical User Interface (GUI) with screenshots of the three main screens. The first screen is used to choose the type of experiment the user is running, next the set-up screen asks for experimental details including precursor names and saves this information for experimental logging. This screen also includes a hardware control interface to load the syringe pumps and changes its input fields based on the experiment type and the number of pumps being used. Lastly, a simple glovebox window with prominent start and stop buttons allow the user to control the printer while working in an enclosed environment with limited mobility. Note that the GUI adapts its configuration to the type of experiment and the number of syringe pumps being used for each experiment.

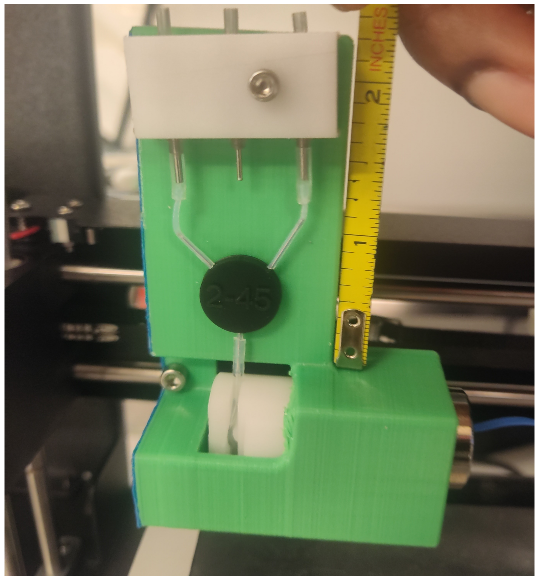


Figure 2.4: A New 3D printed casing was designed to hold an external pinch valve and the dispensing nozzle for improved droplet uniformity.

heating plate. However, infrared images of the commercial hot plate often used for chemical experiments revealed a non-uniform heating surface that resulted in noticeable degradation and poor crystallization of the perovskite droplets. Additionally, moving the samples from the print bed to the hot plate led to unpinning of the droplet edges further wetting of the glass substrate creating pools of materials instead of uniform droplets. In order to create viable perovskite samples, a new heating plate was designed and built for annealing perovskite droplets directly on the 3D printer bed and is shown in Figure 2.5. The new hot plate consisted of a flexible heating element sandwiched between a 1/8" thick glass bottom plate and an 1/8" thick aluminum top plate. Two foam pads where placed bellow the glass plate for structural stability. The heating element was controlled via an InkBird PID temperature controller with a k-type thermocouple for temperature feedback. Infrared images of the new heating plate revealed a near thermally uniform surface, further images where taken with a carbon sheet over the aluminum to decrease the impact of reflectance on the infrared image results. These images, though not shown here, also indicated improved thermal uniformity when compared to the commercial product.

2.4 Successes in Printing Organic Perovskite Materials

The $FAPbI_3$ - $MAPbI_3$ hybrid organic-inorganic perovskite material system was used to validate the improved Archerfish's ability to create gradients of materials in a high-throughput manner. This system was chosen because it is established with many data sets in literature that could be cross referenced for comparison with the Archerfish produced droplets. [9] Furthermore, each precursor, $FAPbI_3$ and $MAPbI_3$, could be mixed in solution form to produce $FA_{1-x}MA_xPbI_3$ droplets with $0 \geq x \geq 1$ that could then be annealed at 150°C for 15

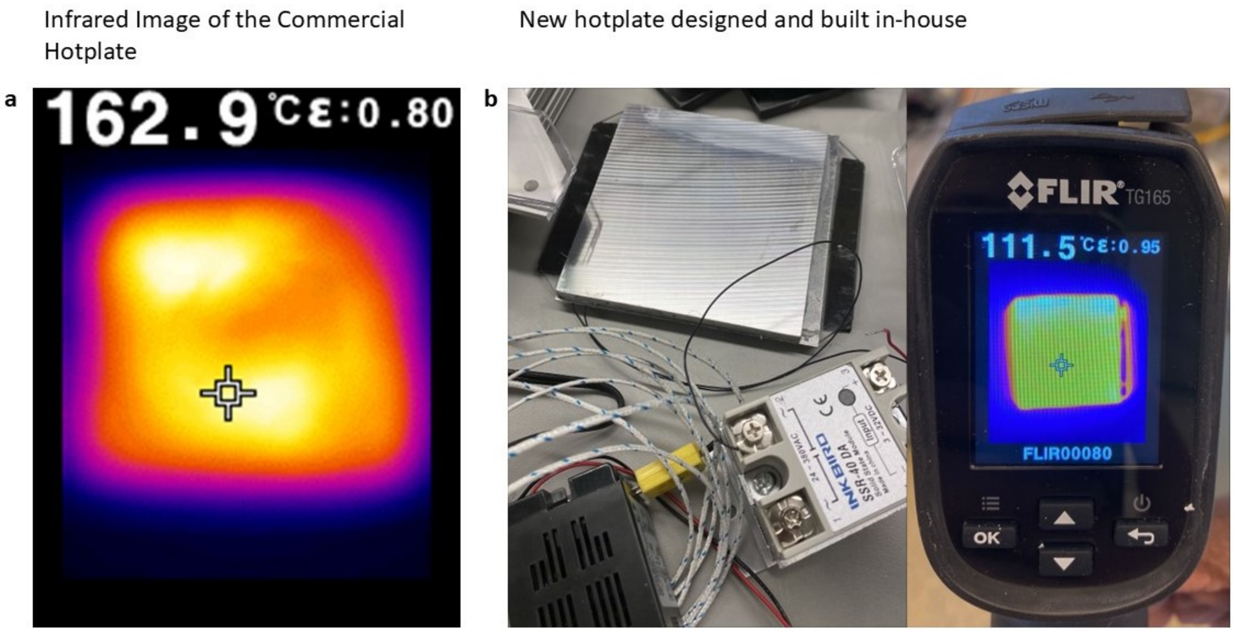


Figure 2.5: a) Thermal non-uniformity observed through infrared imaging of a commercial hotplate initially used to anneal Archerfish perovskite samples. b) The new hotplate built in lab shows better uniformity than its commercial counterpart.

minutes and crystallize to create the perovskite materials. Archerfish synthesized perovskite gradients are printed on a clean 2"×3" glass substrate in a serpentine pattern with the print head moving at a speed of 38 mm/s. The pinch valve actuates at 5Hz and 11% duty cycle depositing 70-80 droplets of unique compositions in roughly 16.5 seconds. Each print uses approximately 0.15ml of fluid, including fluid used to flush the lines before each print. Figure 2.6 show results of X-ray diffraction (XRD), X-ray photoelectron spectroscopy (XPS), and hyperspectral imaging that confirm the presence of a gradient across the Archerfish droplets ranging in composition from pure $MAPbI_3$ to $FAPbI_3$.

2.4.1 Methods

Materials

3"×2"×1mm glass slides (C&A Scientific) are cleaned using deionized water (DI, $< 1.0\mu$S/cm, VWR), Hellmanex III (VWR), and isopropyl alcohol (IPA, $\geq$ 99.5%, VWR) to be used as substrates. Lead iodide powder (PbI_2, 99.999% trace metal basis, Sigma-Aldrich), formamidinium iodide powder (FAI, >99.9%, Greatcell Solar Materials), methylammonium iodide (MAI, >99.9%, Greatcell Solar Materials), dimethylformamide (DMF, $\geq$ 99.8%, Sigma-Aldrich), and dimethylsulfoxide (DMSO, $\geq$ 99.9%, Sigma-Aldrich) are used to prepare the perovskites.

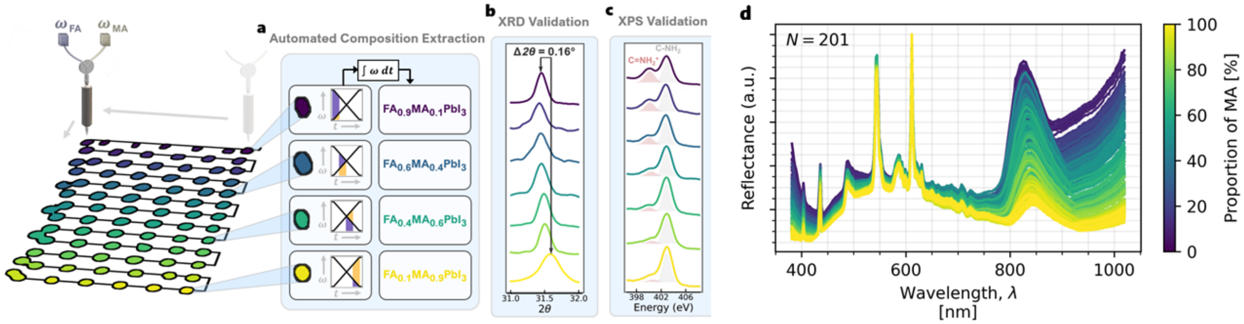


Figure 2.6: $FAPbI_3$ - $MAPbI_3$ hybrid organic-inorganic perovskite droplets printed using the Archerfish system where characterized using X-ray diffraction (XRD), X-ray photoelectron spectroscopy (XPS), and hyperspectral imaging. (b) A $\Delta 2\theta$ of 0.152 deg occurs in an XRD peak corresponding in a change of composition from $MAPbI_3$ to $FAPbI_3$. (c) Similarly, a change in the peak corresponding to the $C = NH_2$ double bond found in formamidinium indicates a gradient from $MAPbI_3$ to $FAPbI_3$. (d) Lastly, a gradual change in the reflectance spectra of the droplets obtained using hyperspectral imaging indicate a change in composition from pure $MAPbI_3$ to $FAPbI_3$. This figure and its measurements are reproduced with permission from Siemenn and Aissi *et al.* [8], the measurements where taken by Fang Sheng and Alexander Siemenn.

Substrate Preparation

Glass slide substrates are prepared for printing the perovskite samples using a three-step cleaning process: (1) ultrasonication for 5 minutes in DI water with 2%vol. Hellmanex III solution, (2) ultrasonication for 5 minutes in DI water only, and (3) ultrasonication for 5 minutes in IPA. Once cleaned, the substrates are transferred to an inert nitrogen environment glovebox with moisture levels < 10ppm.

Perovskite Preparation

$FAPbI_3$ (formamidinium lead iodide) and $MAPbI_3$ (methylammonium lead iodide) are prepared as 0.6M liquid-based precursors for high-throughput printing. For printing, 2mL of each precursor is prepared in an inert nitrogen environment glovebox with moisture levels < 10ppm. First, 3.2mL DMF is mixed with 0.8mL of DMSO to make 4mL of 4 : 1 DMF:DMSO solution. Then, 1.106g of PbI_2 powder is dissolved into the 4mL of 4 : 1 DMF:DMSO to make a PbI_2 stock. Next, the 4mL PbI_2 stock is split in half, pipetting 2mL of stock per vial. Lastly, 0.206g of FAI powder is dissolved into one of the 2mL PbI_2 stock vials and 0.191g of MAI powder is dissolved into the other making 0.6M $FAPbI_3$ and 0.6M $MAPbI_3$, respectively.

2.5 Chapter Summary

In this chapter, I present improvements to the Archerfish platform's communication protocols and hardware, then validate the capabilities of the new system by printing gradients of hybrid organic-inorganic perovskite materials and showing compositional equivalence to spin-coated samples with XRD, XPS, and hyperspectral imaging measurements. In the next chapter, I will develop an algorithm for high-throughput characterization of materials deposited with the new Archerfish platform.

Chapter 3

High - Throughput Characterization of Perovskite Materials

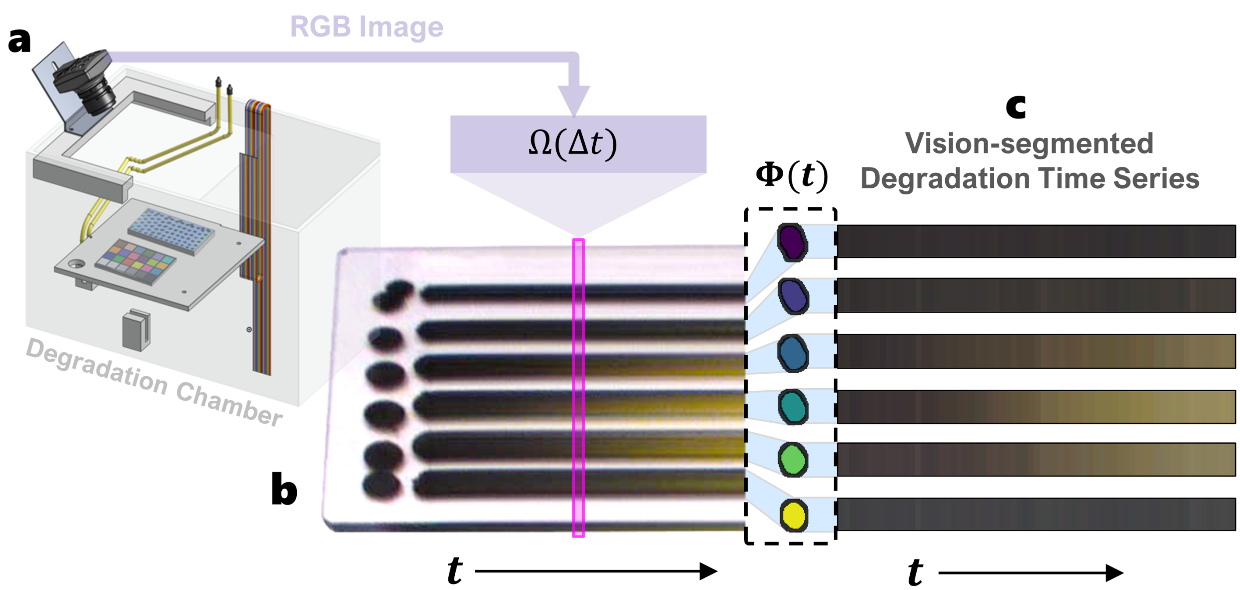


Figure 3.1: **a-c** Automatic degradation testing and measurement of computer vision-segmented perovskite deposits. **a**, The samples are placed in the degradation chamber with specified environmental conditions for a total of two hours. **b**, RGB images of the samples are taken every 30 seconds for two hours to resolve the time-dependent color change in material. **c**, Computer vision is used to segment each deposited sample over time, $\Phi(t)$, to compute the degradation intensity metric, I_c. This figure reproduced with permission from Siemenn and Aissi *et al.* [8]

3.1 Introduction and Overview

The Archerfish system presents a unique opportunity as a high-throughput synthesis method to explore the perovskite material space, however high-throughput material synthesis without characterization is insufficient to accelerate the rate of materials discovery. In this chapter, I present a high-throughput characterization algorithm to extract the stability of Archerfish produced perovskite samples.

As a lead halide perovskite degrades, it changes color from black to yellow, a result of a phase change and/or decomposition of the structure [10–12]. We leverage this RGB-detectable degradation mechanism [13] and use parallelized computer vision segmentation to automate the detection of degradation within perovskites, as shown in Figure 3.1c. Three sets of Archerfish produced $FAPbI_3$ to $MAPbI_3$ gradients, also called samples, totalling $N = 201$ droplets, were degraded for 2 hours in an environmental chamber as shown in Figure 3.1 under 0.5 suns illumination at 34.5°C. The perovskite gradients where synthesized as explained in section 2.4. RGB images of each gradient were taken every 30 seconds as the perovskite droplets degraded. We compute the degradation intensity, I_c, of each perovskite droplet by integrating its change in color, R, over time, t [10]:

$$I_c(\widehat{X}, \widehat{Y}) = \sum_{R=\{r,g,b\}} \int_0^T |R(t; \widehat{X}, \widehat{Y}) - R(0; \widehat{X}, \widehat{Y})| dt, \tag{3.1}$$

where T is the duration of the degradation and the three reflectance color channels are red, r, green, g, and blue, b, for each sample, $(\widehat{X}, \widehat{Y})_n \in N$. High I_c indicates high color change, corresponding to high degradation; I_c close to zero indicates low color change and low degradation.

3.2 Detecting Perovskite Degradation from RGB Time Series Data

3.2.1 Segmentation and Identification of Droplets

To start the analysis of the RGB images obtained at each time step of a samples degradation, the first image in the series is chosen and used to identify the location of the droplets within all the RGB images. Because the camera and the substrate do not move, the droplets only need to be identified in one image and these positions are used for all subsequent images in the series. First, the image is cropped to remove most of the background and include only the droplets on the glass substrate. This cropped image is then segmented using a series of morphological operations and the open-cv watershed algorithm.[8] The result is a gray-scale image with each droplet identified and labeled as shown in Figure 3.2. After segmentation the image is eroded and dilated to remove noise and maximum the number of pixels captured for each droplet.

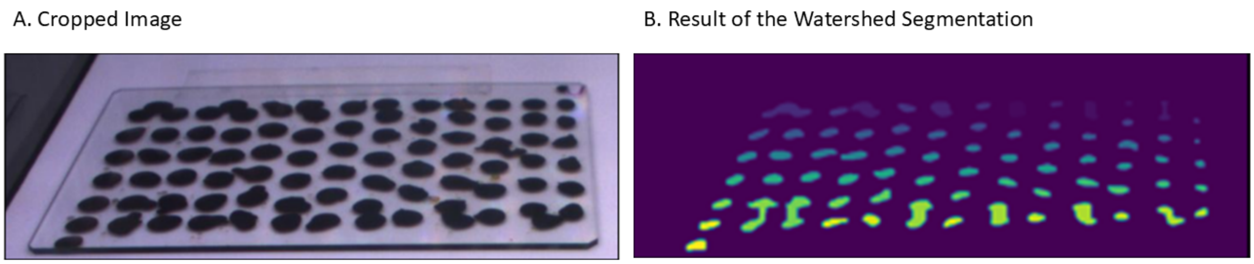


Figure 3.2: a) The droplets on the glass substrate cropped out of the first image in the series. b) Each droplet is identified and labeled as the output of the watershed image segmentation algorithm.

3.2.2 Color Calibration

To use color as a reproducible and repeatable quantitative proxy for degradation, color calibration needs to be applied because the illumination conditions in the environmental chamber may create distortions to the true sample color. At the beginning of the degradation study, an image of a reference color chart (X-Rite Colour Checker Passport; 28 reference color patches), I_R, is taken under the same illumination conditions as the perovskite semiconductor samples. Images at each time step, $\Omega(\Delta t)$, are transformed into CIELAB colorspace and subsequently into a stable reference color space, CIE 1931 color space with a 2-degree standard observer and standard illuminant D50, by applying a 3D-thin plate spline distortion matrix D [10, 14] defined by I_R and known colors of the reference color chart:

$$D = \begin{bmatrix} V \\ O_{4,3} \end{bmatrix} \begin{bmatrix} K & P \\ P^T & O_{4,4} \end{bmatrix}^{-1} \tag{3.2}$$

Here, $O(n,m)$ is an $n \times m$ zero matrix, V is a matrix of the color checker reference colors in the stable reference color space, P is a matrix of the color checker RGB colors obtained from I_R, and K is a distortion matrix between the color checker colors in the reference space and in I_R. The matrix V,K,P, and O are further described in Equation 3.3,

$$\begin{gathered} V = \begin{bmatrix} 1 & x_1' & y_1' & z_1' \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N' & y_N' & z_N' \end{bmatrix}, \\ P = \begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & y_N & z_N \end{bmatrix}, \\ K = \begin{bmatrix} 0 & \dots & U(r_{1N}) \\ U(r_{21}) & \dots & U(r_{2N}) \\ \vdots & \vdots & \vdots \\ U(r_{N1}) & \dots & 0 \end{bmatrix}, \end{gathered} \tag{3.3}$$

where $N = 24$ and is the number of colour patches on the X-Rite colour card, and $U(r_{ij})$ is given by:

$$\begin{aligned} U(r_{ij}) &= 2r_{ij}^2 \log(r_{ij} + 10^{-20}) \\ r_{ij} &= \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \end{aligned} \tag{3.4}$$

Here, r_{ij} is the Euclidean distance between the CIELAB values of the color patches under the current illumination given by x_i, y_i, and z_i and CIE 1931 color space values of the color patches given by x_j, y_j, and z_j. Vectorization techniques are employed for more efficient calculation of the distortion matrix D. The result of the color calibration algorithm is shown in Figure 3.3 and it observed that the purple hue produced by the illumination condition has been rectified.

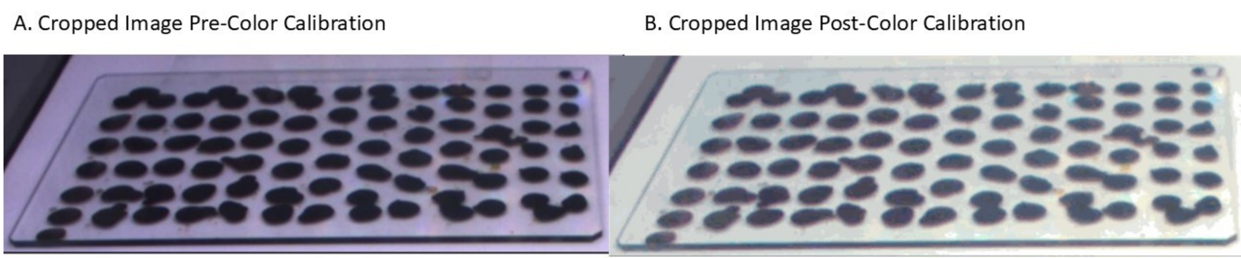


Figure 3.3: a) The cropped image of the perovskite droplets before it is processed by the color-calibration algorithm. b) The same image post calibration, note that the true color of droplets are now accessible and the purple hue created by the illumination conditions have been rectified.

3.2.3 Composition Extraction

After identifying the location of the droplets and performing color calibration, it is necessary to determine estimated values of the droplets composition in order to extract meaningful scientific information. However, due to the positioning of the camera, a perspective change is needed to access the true distance between the droplets and back-propagate their composition using the syringe pump speeds and the travel path of the printer head. [8] The algorithm for this change in perspective consists of two parts: corner detection and warping. Corner detection involves a series of morphological and mathematical operations to detect the four corners of the 2" x 3" glass substrate onto which the droplets are deposited. First the cropped RGB image of the droplets on their substrate is converted to gray-scale and thresholded. Canny edge detection is then performed on the thresholded image to extract its dominant contours. The contour corresponding to droplets are removed using the pixel locations from the segmentation algorithm in Section 3.2.1. A Hough Lines transform is then performed on the remaining contours to determine the dominant lines in the image. For each dominant line, the slope and y-intercept is calculated and used to form a linear equation representation to determine the angles between the lines and their intersection points. These angles and points are used to remove extraneous lines that most likely do not belong to the glass

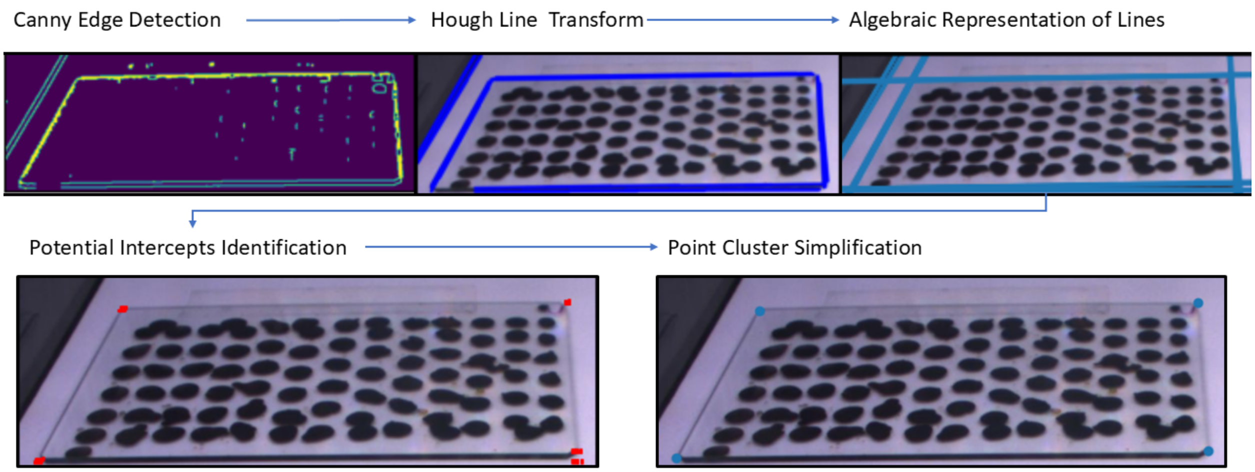


Figure 3.4: The first step in composition extraction is determining the corners of the glass substrate onto which the droplets are printed. This is done through a series of computer vision morphological operations and algebraic representations.

substrate. Lines that have less than 10 degrees between them, for example, are likely to be clusters of lines corresponding to the same edge in the image, they are therefore removed and only one of the lines is kept as a candidate for the edge of the glass substrate. Moreover, because the substrate is a rectangle whose edges intersect within the image, we remove lines that intersect outside the image as they are unlikely to corresponding the the edges of the substrate. Likewise, the intersection points that lie outside the image or along the edges of the image are removed. Finally, clusters of points that emerge near the corners of the substrate can be joined to elucidate the true location of the substrate corners within the image. This is achieved by taking the Euclidean distance between all remaining intersection points and condensing pairs of points into midpoints and repeating this process until all points are sufficiently far from one another. These steps result in the identification of the substrate edges and four corners as shown in Figure 3.4 which are then used to obtained a perspective transform and warp the image into a birds eye view of the droplets as shown in Figure 3.5.

3.2.4 Instability Calculation

Using the color-calibrated images, droplet composition, and droplet pixel locations given by Φ, a final array, $R(t; \widehat{X}, \widehat{Y})$ of the average color at time t for perovskite semiconductor of composition $FA_{1-x}MA_xPbI_3$ is created. The color of each droplet is measured to determine the stability metric I_c [10], calculated using Equation 3.1. A time series representation of the color of each droplet in a sample is given in Figure 3.6.

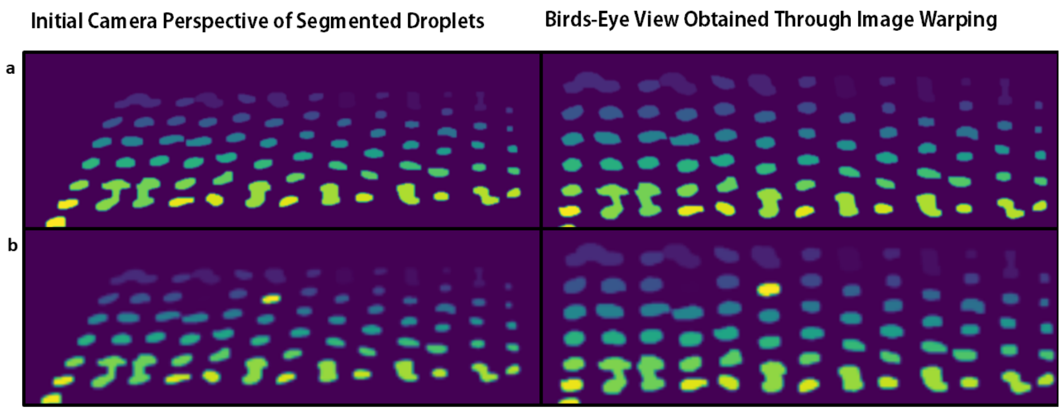


Figure 3.5: The second step in composition extraction is a perspective transform to obtain a bird's eye view of the droplets. a) This row provides the results of the perspective warp on all the Archerfish droplets. b) In this row, the same droplet is highlighted in yellow to indicating the retention of spatial information for each droplet, the yellow droplet appears in the same position relative to other droplets in both the initial and the warped perspectives.

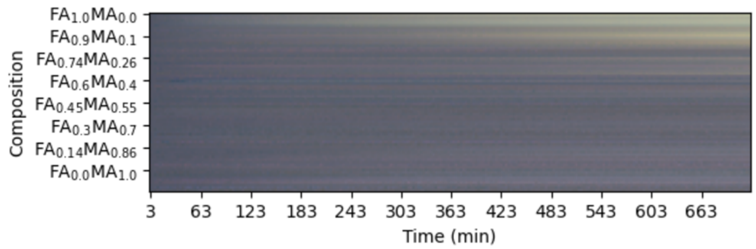


Figure 3.6: An example of a time series color change of every droplet in a sample. Each horizontal line represents the color of a droplet along the $FA_{1-x}MA_xPbI_3$ ($0 \geq x \geq 1$) compositional gradient over the degradation experiment.

3.3 Results

The performance of the degradation autocharacterization is demonstrated by comparing the output I_c to the ground truth degradation, obtained from the pre- and post-band gap deviation [10, 15]. Figure 3.7a illustrates the output of the autocharacterization where high computed I_c values strongly correspond to the occurrence of the ground truth degradation in the samples (yellow scatter points). The determination of ground truth degradation is conducted by a human domain expert. This classification performance of the autocharacterization algorithm achieves a maximal accuracy of 96.9%, relative to the ground truth. The yellowing pattern of the FA-rich samples is shown in Figure 3.7b as a result of the phase change from favorable cubic phase α-$FAPbI_3$ to the non-perovskite hexagonal phase δ-$FAPbI_3$ [11]. Furthermore, running a full degradation detection computation using autocharacterization takes only 20 minutes per 200 samples, given 48000 total degradation images over the 2-hour degradation experiment. This is a significant speedup from the stan-

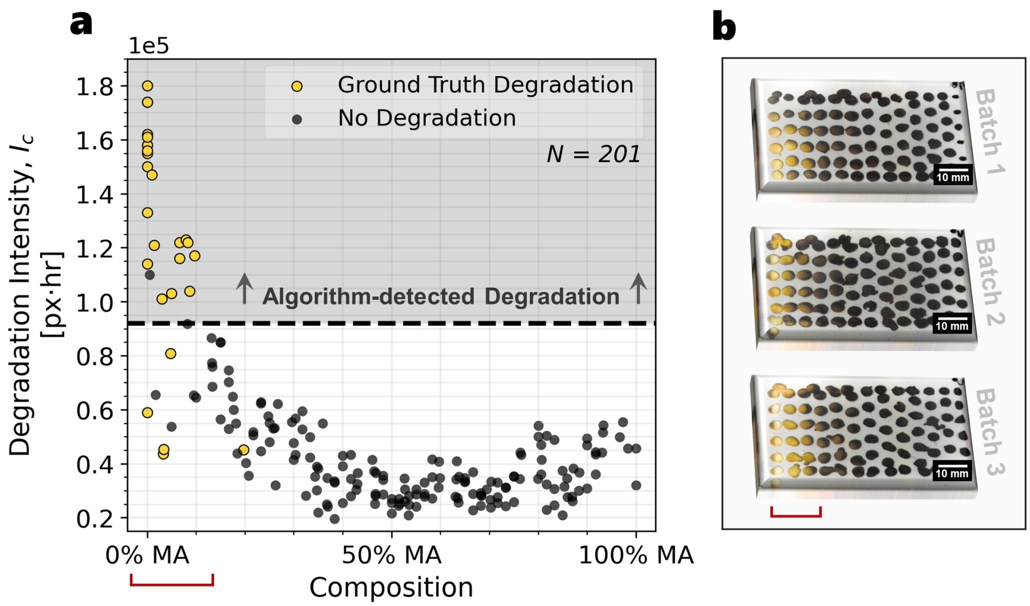


Figure 3.7: **a**, Performance of the autocharacterization of degradation intensity, I_c, relative to the ground truth degradation determined by a domain expert (yellow scatter points) on $N = 201$ unique perovskite samples across 3 independent trials. The black dashed line indicates the split between high and low I_c values, corresponding to high and low degrees of degradation, respectively. **b**, Images of the three batches of $FA_{1-x}MA_xPbI_3$ gradient samples after the 2-hour controlled degradation. The leftmost samples are FA-rich and the rightmost samples are MA-rich. The yellowed FA-rich compounds have undergone a phase transition from α-$FAPbI_3$ to $\delta-FAPbI_3$ and are considered as "ground truth" degradation samples if they exhibit a deviation of > 0.02eV in band gap from pre- to post-degradation, evaluated by a domain expert. This figure reproduced with permission from Siemenn and Aissi *et al.* [8]

dard microscopy or XRD methods of determining degradation, which can take hours or days to identify the degradation of an equivalent number of samples.

Using the fast and accurate stability autocharacterization tool developed in this thesis, we tractably generate an ultra-high resolution stability trend for the $FA_{1-x}MA_xPbI_3$ series, shown in Figure 3.7a and this trend has not been reported at such a high resolution yet in literature. Prior literature reports stability compositional resolutions from $0 \leq x \leq 1$ for 11 compositions [16], 9 compositions [17], and 7 compositions [18] using conventional characterization methods. Moreover, Charles *et al.* [16] reports the stability at $x \approx 0.1$ compositional increments from $0 \leq x \leq 1$ using 6 time steps, amounting to a total of 66 temporal data points. Comparatively, using automated characterization, we report the stability at $x \approx 0.008$ unique compositional increments from $0 \leq x \leq 1$ using 240 time steps, amounting to 28800 unique temporal data points (with 48000 total temporal data points). Thus, with autocharacterization, we achieve over a 10x increase in the compositional resolution and a 40x increase in the temporal resolution for a total of a 436x increase in the number of unique data points reported for the $FA_{1-x}MA_xPbI_3$ stability series, to our knowledge.

Furthermore, with this high-resolution stability trend, we note the same regions of high-degradation appear in Figure 3.7a as do in the literature for the α-$FAPbI_3 \rightarrow \delta$-$FAPbI_3$ degradation pathway at $0.0 \leq x \leq 0.15$, with the optimal low-degradation region occurring at $x \approx 0.40$ [16, 19].

Through the generation of ultra-high resolution trends, we may achieve a better understanding of complex semiconductor composition-property relationships to enable higher-performance design of materials in the future.

3.4 Chapter Summary

In this chapter, I provided a study that successfully combines high-throughput synthesis and characterization methods to speed up the rate of information gathering about a material system. Gradients of $FA_{1-x}MA_xPbI_3$ ($0 \geq x \geq 1$) perovskite solar cell materials with N=201 droplets where synthesized using the improved version of the Archerfish system described in Chapter 2 and characterized using computer vision image processing algorithms to extract their stability, a key material property for perovskites.

Chapter 4

Limitations of the Platform

4.1 Introduction and Overview

The architecture presented for high-throughput combinatorial printing with Archerfish is promising and has been used to print gradients of several different material systems [8]. However, with its current design, Archerfish can only print gradients, not specific compositions, and has many limitations that can result in poor reliability and reproducibility of samples. These limitations can be categorized into three main catagories: compositional control, droplet generation, and environmental and crystallization control. Each category has unique limitation considerations for the current tool but also invites opportunities for growth and design.

4.2 Lack of Compositional Control

Compositional control refers to the ability to set and determine the compositions of individual droplets. With Archerfish's current architecture, it is not possible to set a specific composition for each droplet or determine a droplet's exact composition without additional measurements, such as Energy-dispersive X-ray spectroscopy (EDS). This limits Archerfish to printing gradients with known endpoints that are pre-loaded as precursors in the syringes. Compositional control is limited by various factors but major contributors such as pressure build-ups and fluid reservoirs have been identified and are further discussed here.

Pressure build-ups in areas of compliance within the fluid lines create non-linearities in flow rates. For example, the syringes use a silicone rubber plunger and as a result, fluid is not always dispensed when the motors push the plunger, instead, the plunger compresses to accommodate the resulting pressure from the positive displacement as shown in Figure 4.3. Discontinuities in expected flow rates are also linked to the pressure build-up in the compliant plunger or fluid lines. Another main hindrance to compositional control is the presence of fluid reservoirs that result in hard-to-predict mixing behavior, which ultimately average out to produce a gradient, but do not seem to be linear at each time step. Reservoirs within the valve generate vortices and can result in residual contamination that complicates the transition between fluids, producing unknown compositions at the output droplet. To reduce cross-contamination between runs, reservoirs must also be purged. These purging

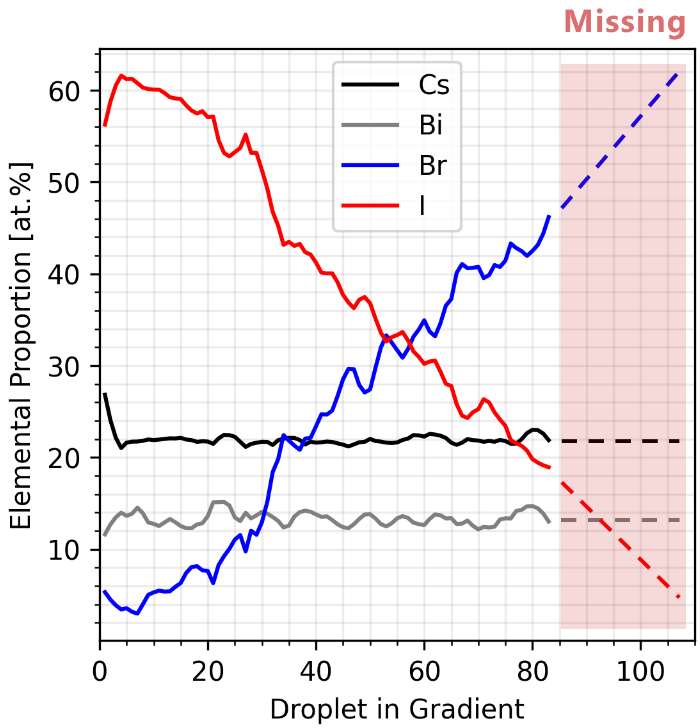


Figure 4.1: Energy-dispersive X-ray spectroscopy (EDS) elemental composition traces. These elemental traces are shown for a $Cs_3Bi_2I_9$-$Cs_3Bi_2Br_9$ (cesium bismuth iodide-cesium bismuth bromide) perovskite gradient printed using Archerfish where each droplet has its EDS spectrum measured. We note the abrupt stop in the compositional shift between iodine and bromine due to improper tuning of the Archerfish print settings. Approximately 80% of the entire gradient is shown to be successfully printed here, the missing portion of the gradient is visualized using dashed line projections. This figure reproduced with permission from Siemenn and Das *et al.* [20]

steps take much longer than the prints themselves, sometimes up to a minute, and waste precursors. Fluid inertia can also pose a problem as pulses in the syringe pump can lead to large volumes of liquid being dispensed periodically, adding more deviation to the expected flow rate. Due to the step-wise rotation of the stepper motors, the syringe pumps create oscillations in the fluid flow, as shown in Figure 4.2 that do not affect the overall gradient composition but could impact the composition of individual droplets. Over-pressurization and back-flow in the fluid lines further limit compositional control and create leaks in the system. Differences in flow rates and thus fluid pressure can at times lead to back-flow and cross contamination between fluid lines, with one fluid forcing its way into another syringe instead of exiting through the nozzle. This behavior is aided by compliant materials in the fluid path that expand to allow build-ups of fluid. This problem could be resolved by the use of check-valves in the flow path thereby forcing the fluid to exit through the nozzle. Lastly, important questions regarding the impacts of pressure, fluid viscosity, temperature, and other parameters on fluid composition and deposition require further analysis and study.

These questions have not yet been explored, but are critical to precise compositional control.

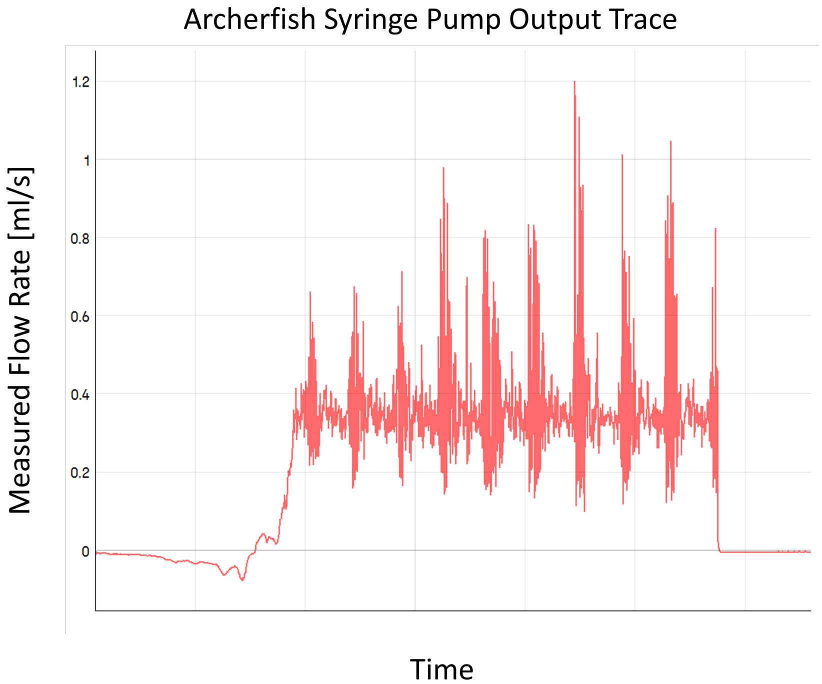


Figure 4.2: The measured flow rate from the current \$10 Archerfish syringes with a constant motor speed. Large and small oscillatory spikes with magnitudes of up to 333% of the set flow rate occur periodically due to the rotation of the stepper motor. The x-axis is in 20 second intervals.

The lack of compositional control reduces the reliability of Archerfish and makes it more difficult for researchers to use the system. For example, it took three researchers two days to find the right parameters to create an end-to-end gradient for the $FAPbI_3$-$MAPbI_3$ perovskite series due to the complex relationship between the droplet wetting properties, reservoirs, compliant regions, and standing waves within the fluid lines. Figure 4.1 illustrates the EDS-measured elemental traces of an 80-droplet $Cs_3Bi_2I_9$-$Cs_3Bi_2Br_9$ (cesium bismuth iodide-cesium bismuth bromide) perovskite series printed with Archerfish that did not have its printing parameters properly tuned. This improper tuning resulted in approximately only 80% of the entire gradient being printed, stopping before reaching the $Cs_3Bi_2Br_9$ end point.

4.3 Droplet Generation

Archerfish can deposit gradients of uniformly thick droplets on most prints but this uniformity can vary based on the precursor molarity and properties of the fluid or substrate. Substrates with higher wettability often distort the droplets, limit droplet packing, and add variability to their shape. It has been observed that wetting behavior can even change between prints of the same materials on the same substrate. Moreover, the relationship between droplet shape and the PWM driver has not been characterized in detail, making it

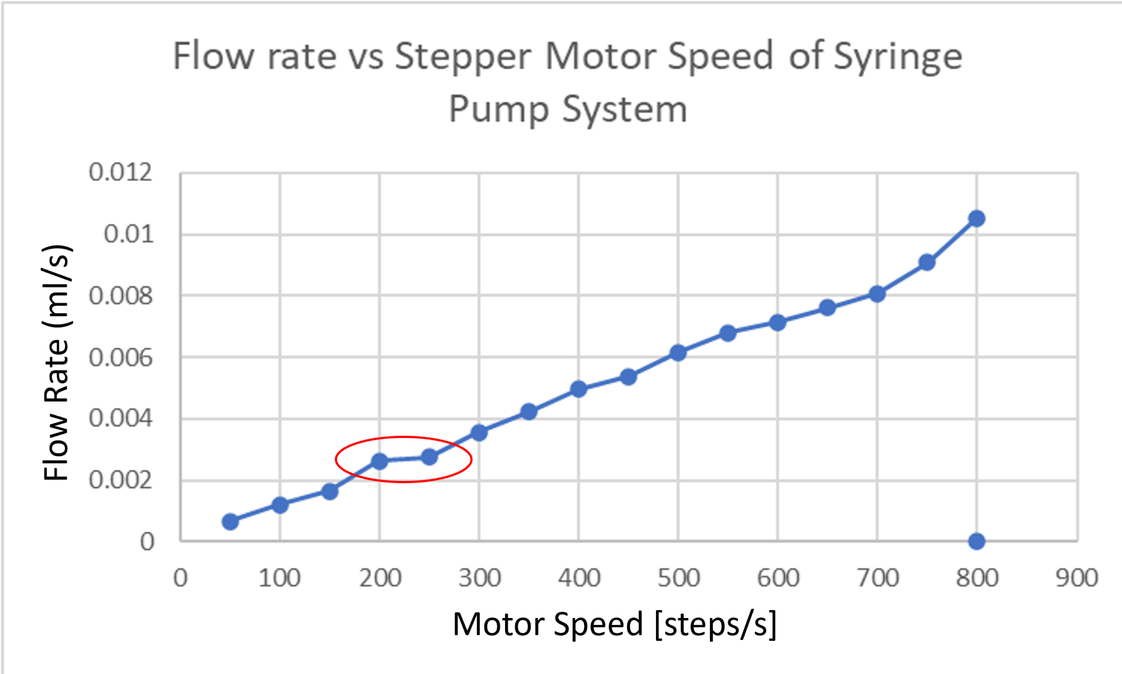


Figure 4.3: Results of a flow rate measurement test indicates non-linearities in the expected flow rate due to a pressure build-up effect. The syringe was refilled between the data points circled in red, despite an increase in the motor speed, the flow rate after a refill remains nearly the same. This effect is a result the rubber plunger in the syringe relaxing when drawing in fluid and therefore needing to re-pressurize before any fluid is dispensed.

difficult to change the droplet generation parameters along with the gradient parameters in accordance with the substrate wettability.

Archerfish droplets with low molarity are subject to the coffee ring effect, in which higher evaporation rates at the droplet's edge cause radial migration of the species in the fluid. This phenomenon produces a droplet with a thicker outer ring of high species concentration and a thinner inner area of low species concentration [21]. Figure 4.4 illustrates these differences in species migration rates for a low molarity Archerfish droplet. For most material systems, the downstream characterization effects of this coffee ring can be avoided through post-processing techniques and focusing characterization on only the uniform inner area of the droplet. Satellite droplets can also form during the droplet deposition process, which do not hinder characterization but are, nonetheless, undesirable.

4.4 Environmental and Crystallization Control

Environmental and crystallization control refers to the ability to control and create uniform crystallization conditions across all droplets on an Archerfish print. Crystallization control is not necessary for all Archerfish samples, however, some material systems cannot be studied without it. Since Archerfish is designed to be a general experimental tool, no aspect of the device is devoted to post-processing any one specific material system. For example, 0.5

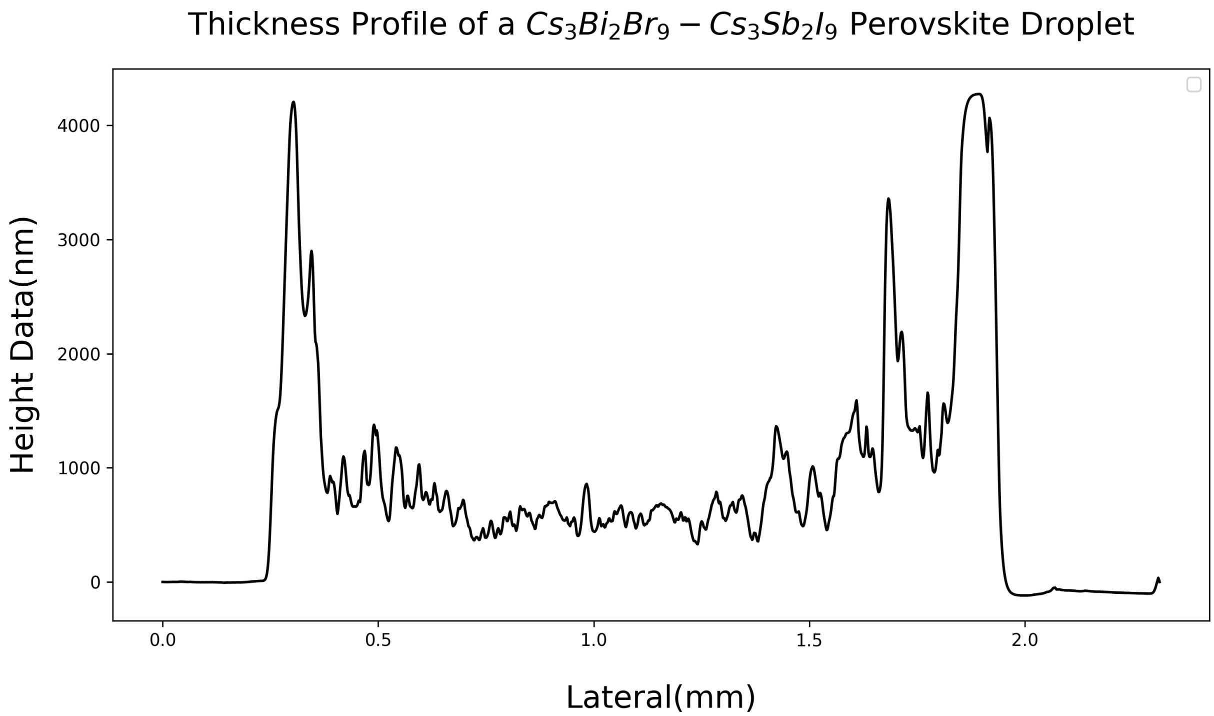


Figure 4.4: The profile of a droplet along the $Cs_3Bi_2Br_9$ to $Cs_3Sb_2I_9$ perovskite gradient obtained though stylus profilometry. A coffee ring affect can be seen, with high ridges along the edge of the droplet and a near-flat film in the middle.

Archerfish does not maintain a constant temperature, pressure, or humidity around samples while or after they are printed, as these conditions would vary across different material systems.

This lack of environmental control presented a challenge for the synthesis of perovskite gradients, limiting our ability to study crystallization-dependent properties like stability. Inconsistent degradation patterns across Archerfish prints arose due to the non-uniform crystallization of droplets. Figure 4.5 shows the controlled degradation of two perovskite samples of the same $FAPbI_3$-$MAPbI_3$ compositional gradient. Varying degradation patterns were observed due to non-uniformities in droplet geometry and hot plate surfaces during annealing. While not in the original scope of the project, future versions of Archerfish should include tunable environmental control to expand the material systems that can be synthesized and studied.

4.5 Unknown Mixing Mechanisms

Due to the small pipe diameters and low flow rates, the Archerfish fluid streams operate in the laminar flow regime making is nearly impossible for passive mixing to occur at the length scales of the fluid lines after the junction. The flow regime can be identified via the value of the Reynolds number which is defined as:

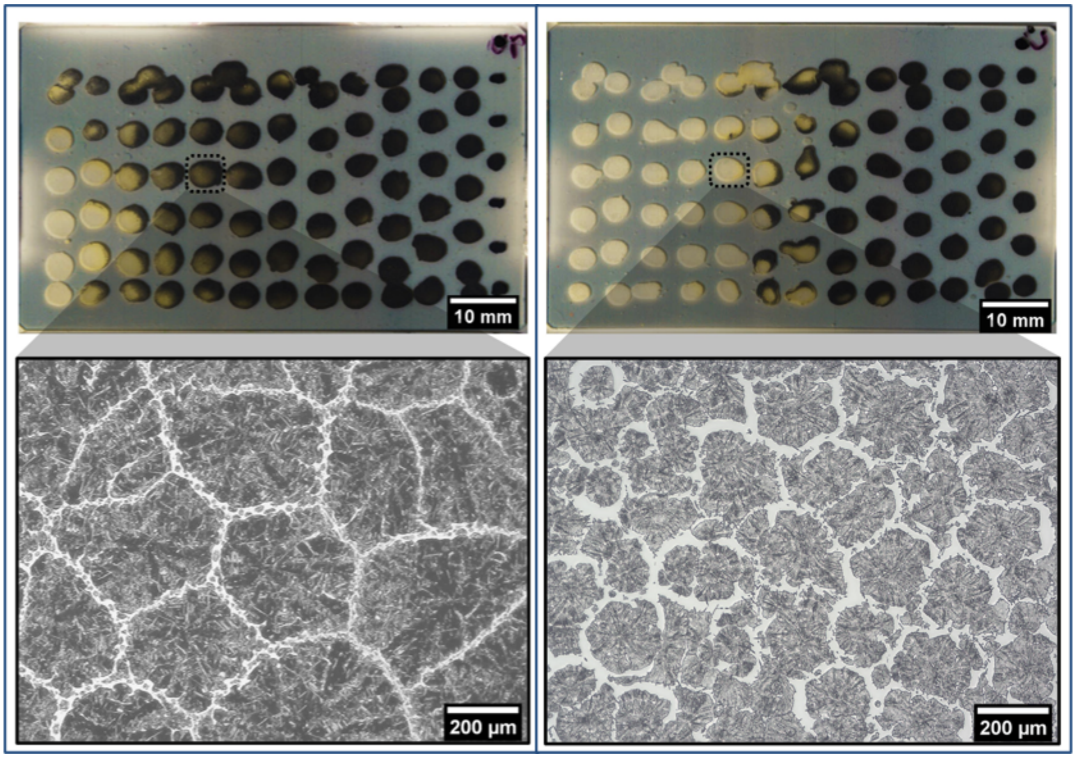


Figure 4.5: Two different Archerfish prints of the same $FAPbI_3$-$MAPbI_3$ compositional gradient after annealing and controlled degradation. Yellow samples are degraded while black samples are not degraded. These two perovskite gradients exhibit different degradation patterns despite being of the same composition and degrading under the same conditions. The differences in crystallization, as shown by the SEM images, transpired from spatial non-uniformities in the annealing and deposition processes. This figure reproduced with permission from Siemenn and Das *et al.* [20]

$$Re = \frac{\rho u L}{\mu} \tag{4.1}$$

where ρ is the fluid density, u is the fluid velocity, L is the pipe diameter, and μ is the dynamic viscosity of the fluid. Passive mixing can be expected in turbulent flow regimes characterized by Reynolds numbers above $2,000$. For Archerfish operating with near 1 centipoise fluids, the Reynolds number ranges from 0.3 to 30. Figure 4.6 provide optical microscope images of a 4-way junction machined from acrylic for visual inspection of the Archerfish junction. Laminar flow with no mixing is observed, as expected due to the low Reynolds numbers of this system. However, optical microscopy images from before and after the old Archerfish system's Lee valve indicate that mixing could potentially be achieved by agitation from a valve as shown in Figure 4.7. Precursor flows with higher viscosity or lower Reynolds numbers could require more folds or obstacles within the plumbing lines to promote more uniform mixing.

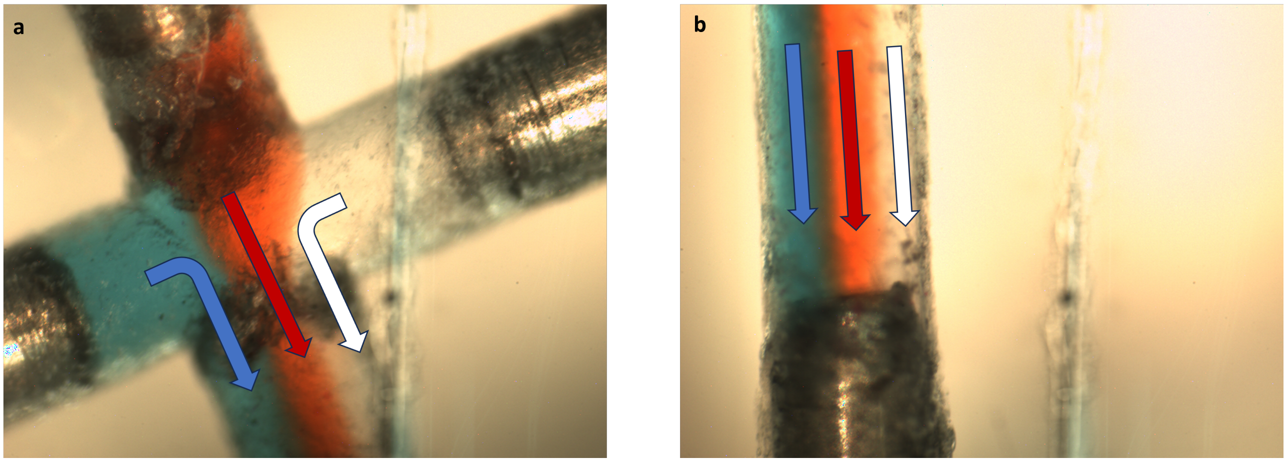


Figure 4.6: Optical microscope images through a machined acrylic 4-way junction with 1/32" inner diameters fluid paths of a) the point of fluid stream contact with clear separation and no mixing and b) the inlet to the droplet generator indicating, as expected, that no mixing occurs when the fluid streams meet. The arrows indicate the direction of flow.

4.6 Other Limitations

Beyond the concerns presented by a lack of compositional control, droplet generation, and environmental and crystallization control, several minor but impactful limitations warrant a mention.

4.6.1 Material Compatibility

The current wetted materials in the Archerfish plumbing lines are PTFE, silicone, and polypropylene. While PTFE is resistant to a wide range of chemicals, silicone and polypropylene are not, which limits the materials that can be studied with the current Archerfish design. Future iterations of the designs should therefore include a carefully chosen material path with highly chemically resistant tubing and actuators.

4.6.2 Human Introduced Variation

Full integration of each Archerfish subsystem is lacking. The retrofitted 3D printer and the actuators (pinch valve and microcontroller) are independently controlled. The pi data center controls the syringe pumps and valve, however the 3D printer gantry is not yet integrated into the central data center. There is no connection between the deposition location and the droplet generation leading to variation between prints, even with the same operator, as the 3D printer and droplet generators may not always be activated simultaneously. Human to human variation is also introduced between prints as the two sub-systems cannot be activated simultaneously and in the same order by every operator. One operator may choose to start the pinch valve before the 3D printer, and another may do the opposite.

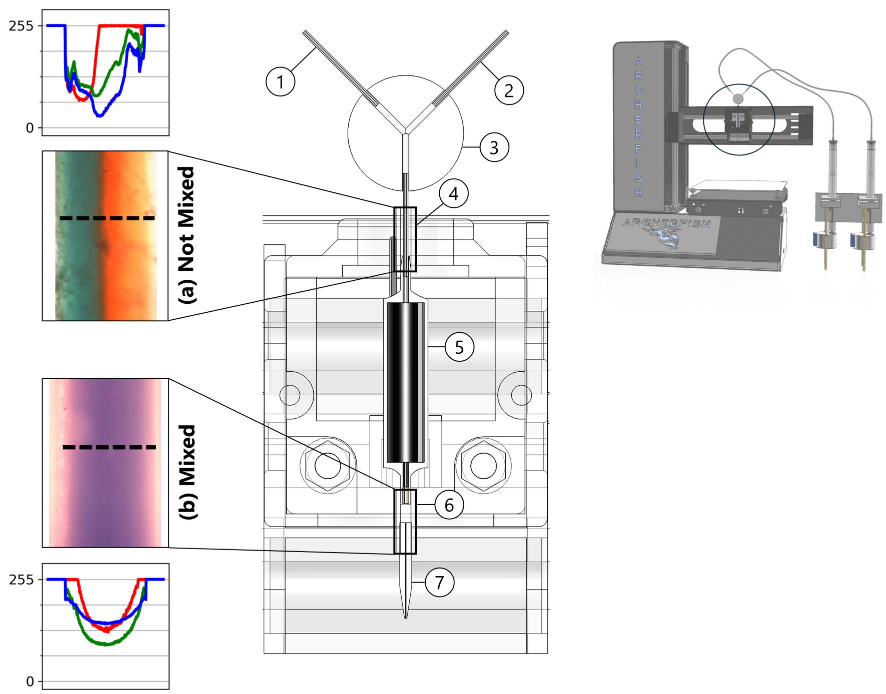


Figure 4.7: Optical microscope images through a machined acrylic three-way junction with inlets at 90 degrees. RGB color analysis show mixing at the outlet of the old Archerfish Lee valve, suggested that agitation via the valve could be a viable path to achieve known mixing for Archerfish droplets. Figure reproduced with permission from Siemenn and Das *et al.*. [20]

4.6.3 Lack of System Feedback

Human to human variations can be further exacerbated by the lack of real-time data acquisition during the printing procedure. Currently, there are no monitoring sensors in the fluid path or on the actuators, there is therefore no way to measure what the system is outputting. The syringe pumps for example, have shown discontinuities in flow rates but these discontinuities currently are not being measured. For example, an encoder on the stepper motors could provide insight on the true flow rate output of the syringe pumps, and time logs indicating when each subsystem is activated could help predict and explain variations between prints.

4.7 Chapter Summary

In this chapter, I present a thorough discussion of limitations and failure modes of the Archerfish system. These include: lack of compositional control, poor environmental control, droplet morphological instabilities, and unknown mixing behavior amongst others. In the next chapter I will provide a conclusion, with a summary and future work for the methods developed throughout this thesis.

Chapter 5

Conclusion

5.1 Summary

In this thesis I presented a high-throughput workflow for synthesizing and characterizing perovskite materials. I introduced the Archerfish system as developed by Siemenn *et al.* [2], explained its limitations, and described several improvements that I implemented to allow it to be used with perovskite materials. I presented XRD, XPS, and hyper-spectral imaging results that validated the improved systems ability to synthesis perovskite crystalline materials. I then demonstrated a high-throughput characterization algorithm to extract the stability of 201 perovskite droplets generated through the Archerfish platform. Lastly, I discussed the limitations of the platform and opportunities for future improvements.

5.2 Future Work

The work discussed in this thesis provides an example of low-cost but high-throughput automated research tools that can decrease synthesis time and increase productivity for small labs. The Archerfish platform as presented has a lot of potential, however as discussed in Chapter 4.1, there are several limitations that still require further development before it can be launched as tool for research. Future work includes further improvements to the Archerfish system specifically focused on mixing, more robust fluid lines and connections, and better communication protocols to join all components of the system under one central command center.

Appendix A

Archerfish Components List

Summary Table				
Description	Cost	Supplier	Part	Qty
Solenoid pinch valve; 2-way NO, 1/32" ID x 3/32" OD, 12/24V	\$108.00	Precigenome		1
Monoprice MP Select Mini 3D Printer (discontinued)	\$175.99	Mono Price		1
300 um Diameter Flow-focusing micro-nozzle outlet	\$50.00	Small Precision Tools	1551-120-437P 200 (10-11D-20)	1
Arduino Mega	\$48.90	Amazon	B0046AMGW0	1
ZK-PP2K 24V High Power PWM Driver	\$10.57	Amazon	B0C7H1QFR8	1
24V Variable Power Supply	\$45.99	Amazon	B08GFSVHLS	2
Raspberry PI 3 Model B+	\$44.41	Amazon	B0BNJPL4MW	1
Raspberry Pi Touch Screen Monitor	\$32.99	Amazon	B07RZYYNMZ	1
Bluetooth KeyBoard and Mouse	\$19.79	Amazon	B07WV5WN7B	1
28BYJ-48 DC 5V Stepper Motors + ULN2003	\$7.99	Amazon	B0CLYCM1CP	1
Fittings, Tubing, and Pumps	\$23.39	Home-Built + McMasterCarr		
Total	\$568.02			

Table A.1: General component list for the improved Archerfish system. Detailed bill of materials and building instructions for version one of the Archerfish Platform are provided by Siemenn *et al.* [20]

Appendix B

Archerfish Raspberry Pi Code

```
# import needed libraries


import PySimpleGUI as sg
import serial
import time
import numpy as np
from datetime import datetime

serialcomm = serial.Serial('COM4', 9600)
serialcomm.timeout = 1


# Save Experiment Setup Data in text file


def save_data(precursors, relative_speeds, mode):
    # saves precursor and relative speed data
    # may need to change to accommodate gradient type experiments
    now = datetime.now()
    f = open('Experimental_log.txt', 'a')
    # Input Date and Time then table of precursors and speeds for this specific experiment
    # P1 P2 ....
    # S1 S2 ....

    t_string = now.strftime("%H:%M:%S")
    d_string = now.strftime("%B %d, %Y")
    dt_string = d_string + ' ' + t_string + '\n'
    f.write(" \n")
    f.write(" ---------------------------------- \n")
    f.write(f"New {mode[0]} Experiment \n")
    f.write(dt_string)
```

```
    f.write(f"Precursors are:      {precursors} \n")
    if mode == ['Batch']:
        f.write(f'Relative Speeds are: {relative_speeds} \n')
    else:
        f.write(f'Steps, Percent of 3rd Precursor, and Gradient Range are: {relative_speeds} \n')
    f.close()


# Set Up Window Layout


def setup_window(def_mot, max_precursor_number):
    choose = ['Batch', 'Two Fluid Gradient', 'Constant + Two Fluid Gradient']
    layout = [[sg.Text("Choose Experiment Type", justification="c")],
              [sg.Listbox(choose, size=(30, len(choose)), key='-EXP-')],
              [sg.T("Number of Precursors: ", justification="r"), sg.I(key="-IN-")],
              [sg.Button("Ok")]]

    window_title = "ArcherFish Command Center"
    window = sg.Window(window_title, layout)

    while True:
        event, values = window.read()
        # cont = ''
        if event == sg.WINDOW_CLOSED:
            break
        if event == "Ok":
            # Output the set-up choices
            # Put in error messages in case they forget to put a number
            # of motors or don't choose an experiment
            if not values['-EXP-']:
                sg.popup_error("You must choose a mode", modal=True)
            if values['-EXP-']:
                choice = values['-EXP-']
                mot = values['-IN-']
                if choice == ['Batch']:
                    if len(mot) == 0:
                        mot = str(def_mot)
                elif choice == ['Two Fluid Gradient']:
                    mot = str(2)
                    cont1 = sg.popup_ok_cancel("For Two Fluid Gradient mode you must use two precursors.",
```

```
                            modal=True)
                    if not cont1 == 'OK':
                        window.close()
                        return choice, int(mot), cont1, int(max_precursor_number)
                elif choice == ['Constant + Two Fluid Gradient']:
                    mot = str(3)
                    cont2 = sg.popup_ok_cancel("For Constant + Two Fluid Gradient mode you must use three precursors.", modal=True)
                    if not cont2 == 'OK':
                        window.close()
                        return choice, int(mot), cont2, int(max_precursor_number)
                if int(mot) > int(max_precursor_number):
                    sg.popup_error("Maximum number of precursors is 3, enter a number lower than or equal to 3.")
                else:
                    cont = sg.popup_ok_cancel(
                        f"Opening command window for {choice[0]} experiment with {mot[0]} precursors.", modal=True)
                    if cont == 'OK':
                        window.close()
                        return choice, int(mot), cont, int(max_precursor_number)
                    else:
                        window.close()
                        return choice, int(mot), cont, int(max_precursor_number)
    window.close()


def batch_window(mots, def_mots, max_precur_num, max_speed, mode_input):
    print(mode_input)
    mode_str = str(mode_input)
    mode_str = mode_str[:-1]
    mode_str = mode_str[:-1]
    mode_str = mode_str[1:]
    mode_str = mode_str[1:]
    mode = [mode_str]

    print(mode)
    mots = int(mots)
    max_precur_num = int(max_precur_num)
    max_speed = int(max_speed)
```

```
    refills = []
    stopping = []
    chemicals = ['MAPbI3', 'FAPbI3', 'CsPbI3', 'DMF', 'DMSO']
    info = ['Gradient Steps', 'Max Percent of Gradient Composition', 'Percent of 3rd Precursor']
    layout = [[sg.T('Set Experiment Parameters')],
              [sg.HSeparator()]]
    if mode == ['Batch']:
        mode_num = 1
        for i in range(mots):
            istr = str(i)
            layout = layout + [[sg.T(f"Precursor {istr[0]} Name:"),
                                sg.Combo(chemicals, size=(20, 1), key=f"-P{istr[0]}-"),
                                sg.T(f"Precursor {istr[0]} Speed:"),
                                sg.I(key=f"-S{istr[0]}-")]]
        layout = layout + [[sg.T('')],
                           [sg.T("Refill Precursors Before You Start:")],
                           [sg.HSeparator()]]
    elif mode == ['Two Fluid Gradient'] or mode == ['Constant + Two Fluid Gradient']:
        mode_num = 2
        for i in range(mots):
            istr = str(i)
            if i == 2:
                layout = layout + [[sg.T(f"Constant Precursor Name:"),
                                    sg.Combo(chemicals, size=(20, 1), key=f"-P{istr[0]}-"), sg.Push(),
                                    sg.T(info[i]),
                                    sg.I(key=f"-S{istr[0]}-", size=(20, 1))]]
            else:
                layout = layout + [[sg.T(f"Precursor {istr[0]} Name:"),
                                    sg.Combo(chemicals, size=(20, 1), key=f"-P{istr[0]}-"), sg.Push(),
                                    sg.T(info[i]),
                                    sg.I(key=f"-S{istr[0]}-", size=(20, 1))]]

        layout = layout + [[sg.T('')],
                           [sg.T("Refill Precursors Before You Start:")],
```

```
                            [sg.HSeparator()]]
    # print(layout)
    for j in range(mots):
        jstr = str(j)
        if j == 0:
            layout_left = [[sg.Button(f"Fill Precursor {jstr[0]}")]]
            flush = [f"Flush Precursor {jstr[0]}"]
            layout_right = [[sg.Button(f"Flush Precursor {jstr[0]}")]]
            refills = refills + [f"Fill Precursor {jstr[0]}"]
            stopping = stopping + ['0000']
        else:
            layout_left = layout_left + [[sg.Button(f"Fill Precursor {jstr[0]}")]]
            layout_right = layout_right + [[sg.Button(f"Flush Precursor {jstr[0]}")]]
            refills = refills + [f"Fill Precursor {jstr[0]}"]
            flush = flush + [f"Flush Precursor {jstr[0]}"]
            stopping = stopping + ['0000']
    for f in range (int(max_precur_num) - int(mots)):
        stopping = stopping + ['0000']
    layout_left = layout_left + [[sg.Button("Fill All Precursors")]]
    layout_right = layout_right + [[sg.Button("Flush All Precursors")]]
    refills = refills + ["Fill All Precursors"]
    flush = flush + ["Flush All Precursors"]
    layout_stop = [[sg.Button("Stop")], [sg.Button("Turn on Valve")]]
    layout_columns = [[sg.Column(layout_left),sg.VSeparator(), sg.Column(layout_right), sg.VSeparator(), sg.Column(layout_stop)]]

    layout = layout + layout_columns + [[sg.T('')],
                        [sg.T('Make Sure to Recalibrate the Syringes After a Refill')],
                        [sg.HSeparator()],
                        [sg.Button("Recalibrate Syringes", button_color='purple', size=(30, 2))],
                        [sg.Button('Move to Glove Box', button_color='green'), sg.Button('Return to Setup Window',

```

```python

    window_title = f"{mode[0]} Experiment Command Center"
    window = sg.Window(window_title, layout)
    op = bool
    speeds = []
    precur = []
    while True:
        event, values = window.read()
        if event == sg.WINDOW_CLOSED:
            break
        if event == "Turn on Valve":
            i = str(1) + str(stopping) + str(1)
            while True:
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break
        if event == 'Stop':
            stopping_str = str(stopping)
            i = str(1) + stopping_str + '0'
            while True:
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break
        if event == "Recalibrate Syringes":
            i = []
            for k in range(mots):
                i = i + ['-050']
            if mots < max_precur_num:
                for l in range(max_precur_num - mots):
                    i = i + ['0000']
            i = str(1) + str(i) + str(1)
            while True:
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break
            time.sleep(20)
            i = str(1) + str(stopping) + str(0)
            while True:
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break

        if event in refills:
            # print(event)
            i = []
            if event == "Fill All Precursors":
```

```
                for k in range(mots):
                    i = i + ['0600']
            else:
                wvent = str(event)
                for k in range(mots):
                    if k == int(wvent[15]):
                        # building the serial message as ['####', '####', '####']0
                        # if you are at the location within the message for the motor you want to communicate with
                        # make the speed 500, else make the speed 0
                        i = i + ['0600']
                        # print(k)
                    else:
                        i = i + ['0000']
            for kl in range(max_precur_num - mots):
                i = i + ['0000']
            i = str(i)
            i = str(1) + i + str(0)
            # print(i)
            # print(wvent[15])
            while True:
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break
        if event in flush:
            # print(event)
            i = []
            if event == "Flush All Precursors":
                for k in range(mots):
                    i = i + ['-200']
            else:
                wvent = str(event)
                for k in range(mots):
                    if k == int(wvent[16]):
                        # building the serial message as ['####', '####', '####']0
                        # if you are at the location within the message for the motor you want to communicate with
                        # make the speed 500, else make the speed 0
                        i = i + ['-600']
                        # print(k)
                    else:
                        i = i + ['0000']
            for f in range(max_precur_num - mots):
```

```
                i = i + ['0000']
            i = str(i)
            i = str(1) + i + str(1)
            # print(i)
            # print(wvent[15])
            while True:
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break

        if event == 'Return to Setup Window':
            op = False
            speeds = []
            precur = []
            break
        if event == 'Move to Glove Box':
            max_speed = int(max_speed)
            if mode == ['Batch']:
                op = True
                # create speeds list which just grabs the exact values inputted for precursor speeds
                # Speed values must be between 0 and 1, must sum to greater than 0.9 but not more than 1 ideally
                # create list of precursors used
                for k in range(mots):
                    kstr = str(k)
                    speeds = speeds + [values[f"-S{kstr[0]}-"]]
                    precur = precur + [values[f"-P{kstr[0]}-"]]
                # print(speeds)
                # Send Confirmation Message with Precursor and Speed Values
                ok_cancel = sg.popup_ok_cancel("You set up an experiment for " + str(precur) +
                                                    " at the respective speeds of " + str(speeds))
                # If user inputs less than 3 precursors, change speeds list to include
                if int(mots) < int(max_precur_num):
                    dif = int(max_precur_num) - int(mots)
                    for g in range(dif):
                        speeds = speeds + ['0000']
                if ok_cancel == 'OK':
                    # if ok_cancel == ok just continue on with the code
                    # save Speeds and Precursors to Data File
                    save_data(precur, speeds,mode)
```

```
# Check user inputted speed values for all precursors
if '' in speeds:
    sg.popup_error("One or more Motor Values were not entered")
    op = False
    speeds = []
    precur = []
# now to change our decimal values to motor values
# max_speed = 700
mot_speeds = []
val_ints = 0
speed_ints = 0
# print(len(speeds))
for p in range(len(speeds)):
    # print(speeds[p])
    # ie 0.3333*700 = 233.31
    val = round(float(speeds[p]) * max_speed)
    # = 233
    val_ints = val_ints + val
    speed_ints = speed_ints + float(speeds[p])
    # print(p)
    val = str(val)
    # = '233'
    g = len(val)
    # g = 3
    d = 4 - g
    # d = 1
    # generate zero string to add in front
    # print(d)
    for t in range(d):
        # print(t)
        if t + 1 == d:
            val = '-' + val
        else:
            val = '0' + val
    # 0233
    mot_speeds = mot_speeds + [val]
# print(mot_speeds)
if val_ints > max_speed * 1.14:
    # print(val_ints)
    sg.popup_error('Input speeds exceed the maximum motor speed sum')
    op = False
elif speed_ints < 0.9:
    # print(speed_ints)
```

```
                        sg.popup_error('Input concentrations do not sum to the minimum of 0.9')
                        op = False
                    else:
                        op = True
                        speeds = []
                        break
                    speeds = []
                    precur = []
                else:
                    speeds = []
                    precur = []
            if mode == ['Two Fluid Gradient'] or mode == ['Constant + Two Fluid Gradient']:
                op = True
                # create speeds list which just grabs the exact values inputted for precursor speeds
                # create list of precursors used
                for k in range(mots):
                    kstr = str(k)
                    precur = precur + [values[f"-P{kstr[0]}-"]]
                steps = values['-S0-']
                max_percent = values['-S1-']
                inputs = [steps, max_percent]
                percent = 0
                # Confirmation Message of Experiment Settings
                if mode == ['Constant + Two Fluid Gradient']:
                    percent = values['-S2-']
                    inputs = [steps, percent, max_percent]
                    ok_cancel = sg.popup_ok_cancel("You set up an experiment for " + str(precur) +
                                                   " with " + str(steps) + " steps, " + str(max_percent) + " Percentage Range, and " +
                                                   str(percent) + " Percent of 3rd Precursor. Note that the constant precursor " +
                                                   "MUST BE LOADED IN MOTOR C. Also, in the gradient MOTOR
```

```
                                                            B STARTS AT THE HIGHER PERCENTAGE.")
                else:
                    ok_cancel = sg.popup_ok_cancel("You set up an experiment for " + str(precur) +
                                                   " with " + str(steps) + " steps, " + str(max_percent) + " Percentage Range")


                # If user inputs less than 3 precursors, change stopping list to have three values
                # if int(mots) < int(max_precur_num):
                    # dif = int(max_precur_num) - int(mots)
                   # for g in range(dif):
                      #  stopping = stopping + ['0000']
                # in mode 2 #['s###', 's###', 's###']# becomes
                #       // mode ['steps','percent of third precursor','max percent of the gradient composition'] valve
                #       // for the max gradient percent and example is going from 25%-75% to 75-25 instead of 0-100 to 100-0
                #       // in this case the max percent will be 75 or 100
                #       // for the step numbers the max value is 999
                #       // for third precursor percentage the max value is 100
                #       // for max gradient percentage the max value is 100
                #       // #['0###', '0###', '0###']# where ### is an absolute percent ie 100 = 100%, 050 = 50%

                if ok_cancel == 'OK':
                    # if ok_cancel == ok just continue on with the code
                    # Make the speeds vector
                    s = len(steps)
                    m = len(max_percent)
                    p = len(str(percent))
                    speeds_steps = str(steps)
                    speeds_max_percent = str(max_percent)
                    speeds_percent = str(percent)
                    for i in range(4-s):
```

```
            speeds_steps = str(0) + str(speeds_steps)
        # print(speeds_steps)
        for i in range(4-m):
            speeds_max_percent = str(0) + str(speeds_max_percent)
        for i in range(4-p):
            speeds_percent = str(0) + str(speeds_percent)
        speeds = [speeds_steps, speeds_percent, speeds_max_percent]
        # print(speeds)

        # save Speeds and Precursors to Data File
        save_data(precur, speeds, mode)
        # Checking values are in the right range
        # Check user inputted speed values for all precursors
        if '' in inputs:
            sg.popup_error("One or more Experiment Parameters were not entered")
            op = False
            speeds = []
            precur = []
        # Max step is 999
        elif int(steps) > 999:
            sg.popup_error("Steps must be less than or equal to 999")
            op = False
        # Max 3rd percentage is 100
        elif int(percent) > 25:
            sg.popup_error("Maximum 3rd precursor Percentage is 25")
            op = False
        # Max Percentage Range is 100
        elif int(max_percent) > 100 or int(max_percent) < 60:
            sg.popup_error("Maximum percentage range is 60 to 100")
            op = False
        else:
            op = True
            break
        speeds = []
        precur = []

    else:
        speeds = []
        precur = []
```

```python

    if op:
        window.close()
        if mode == ['Batch']:
            mot_speeds = str(mot_speeds)
            glove_box('batch_window', mots, def_mots, mot_speeds, stopping, max_precur_num, max_speed, mode)
        else:
            speeds = str(speeds)
            print(speeds)
            glove_box('batch_window', mots, def_mots, speeds, stopping, max_precur_num, max_speed, mode)
        # print(mot_speeds)

    else:
        window.close()
        return op
    window.close()

def glove_box(experiment, number, def_mots, speeds, stopping, max_precur_num, max_sp, mode):
    max_speed = int(max_sp)
    # print(speeds)
    q = 3
    refills = []
    layoutrefill = []
    for j in range(number):
        jstr = str(j)
        layoutrefill = layoutrefill + [[sg.Button(f"Fill Precursor {jstr[0]}", size=(25, q))]]
        refills = refills + [f"Fill Precursor {jstr[0]}"]
    refills = refills + ["Fill All Precursors"]
    layoutrefill = layoutrefill + [[sg.Button("Fill All Precursors", size=(25, q))],
                                   [sg.Button("Turn on Valve", size=(25, 1))],
                                   [sg.Button("Recalibrate Syringes", size=(25, q + 1), button_color='purple')],
                                   [sg.Button("Stop Print and Exit", size=(25, q), button_color='red')]]
    # print(layoutrefill)
    layoutss = [[sg.Button("Start Batch Print", s=(25, 25), button_color='green'),
                 sg.VSeparator(),
```

```
                    sg.Button("Stop All", s=(25, 25), button_color='
                        orange')]]

    layout = [[sg.Column(layoutss), sg.VSeparator(), sg.Column(
        layoutrefill)]]

    window_title = 'Glove Box Controls'
    window = sg.Window(window_title, layout)
    # number is the number of motors or precursors
    # change speeds to a list that the arduino can read
    # stopspeeds = "['0000', '0000', '0000']0"
    stopspeeds = str(1) + str(stopping) + str(0)
    # print(stopspeeds)
    if mode == ['Batch']:
        startspeeds = str(1) + speeds + str(1)
    else:
        startspeeds = str(2) + speeds + str(1)
    # print(startspeeds)
    # build vector with equal speeds for all motors
    rat = 1 / number
    w = []
    w_speeds = []
    # w for warm-up
    for j in range(number):
        w = w + [rat]
    for p in range(number):
        # print(speeds[p])
        # ie 0.3333*700 = 233.31
        val = round(float(w[p]) * max_speed)
        # = 233
        # print(p)
        val = str(val)
        # = '233'
        g = len(val)
        # g = 3
        d = 4 - g
        # d = 1
        # generate zero string to add in front
        # print(d)
        for t in range(d):
            # print(t)
            if t + 1 == d:
                val = '-' + val
            else:
                val = '0' + val
        # 0233
        w_speeds = w_speeds + [val]
```

```
    for k in range(max_precur_num-number):
        w_speeds = (w_speeds + ['0000'])
    w_speeds = str(1) + str(w_speeds) + str(1)

    while True:
        event, values = window.read()
        if event == sg.WINDOW_CLOSED:
            break
        if event == "Turn on Valve":
            i = str(1) + str(stopping) + str(1)
            while True:
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break
        if event == "Recalibrate Syringes":
            i = []
            for k in range(number):
                i = i + ['-050']
            if number < max_precur_num:
                for l in range(max_precur_num - number):
                    i = i + ['0000']
            i = str(1) + str(i) + str(1)
            print(i)
            while True:
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break
            time.sleep(20)
            i = str(1) + str(stopping) + str(0)
            while True:
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break

        if event == 'Stop All':
            i = stopping
            i = str(1) + str(i) + '0'
            while True:
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break

        if event in refills:
            # print(event)
            i = []
            if event == "Fill All Precursors":
                for k in range(number):
```

```
                    i = i + ['0600']
            else:
                wvent = str(event)
                for k in range(number):
                    if k == int(wvent[15]):
                        # building the serial message as ['####', '####', '####']0
                        # if you are at the location within the message for the motor you want to communicate with
                        # make the speed 500, else make the speed 0
                        i = i + ['0600']
                        # print(k)
                    else:
                        i = i + ['0000']
            for f in range(max_precur_num - number):
                i = i + ['0000']
            i = str(i)
            i = str(1) + i + '0'
            # print(i)
            # print(wvent[15])
            while True:
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break
        if event == 'Stop Print and Exit':
            while True:
                i = stopspeeds
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break
                # print
            number = str(number)
            def_mots = str(def_mots)
            window.close()
            # print(number)
            eval(experiment + f'(f"{number}", f"{def_mots}", f"{str(max_precur_num)}", f"{max_speed}", f"{mode}")')
            break
        if event == 'Start Batch Print':
            # first run all the motors at same speed for 25 seconds
            # then run the given speed/settings values
            while True:
                i = w_speeds
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break
```

```
            time.sleep(7)
            while True:
                i = startspeeds
                print(i)
                serialcomm.write(i.encode())
                time.sleep(0.5)
                break
                # print(serialcomm.readline().decode('ascii'))
    window.close()


if __name__ == "__main__":
    default_motors = 3
    max_precursors = 3
    max_speed = 400
    # Changed from 700 to 400 because of over presurization issues
    # Gets set-up choices and checks that the user wants to continue with them
    con = True
    while True:
        if con:
            choices, moto, con, max_precur_num = setup_window(default_motors, max_precursors)
            if con == 'Cancel':
                # print("Con was non")
                con = True
            else:
                if not len(choices) == 0:
                    op = batch_window(moto, default_motors, max_precur_num, max_speed, choices)
                    if not op:
                        con = True
                else:
                    sg.popup_error("Not Yet Defined")
```

Appendix C

Archerfish Arduino Code

```
// Version 1 Python to Arduino Serial communication motor control
   Code
// Author Eunice Aissi on 9/9/2022
int trans = 22;
// Variables for running Serial Communication
char mode;
String incomingByte;
// Mode 2 Gradient + Constant solvent mode
char step1;
char step2;
char step3;
char prec_per1;
char prec_per2;
char prec_per3;
char grad_max1;
char grad_max2;
char grad_max3;
int step100;
int step10;
int stepone;
int prec_per100;
int prec_per10;
int prec_perone;
int grad_max100;
int grad_max10;
int grad_maxone;
int steps;
int prec_per;
int grad_max;
int maxStepSpeed = 500; // max stepSpeed = 800 before failure/
   unreliable
int precursor_range = 50;  // previously 300
int gradient_range = maxStepSpeed - precursor_range;
// Mode 1 Batch Mode
```

```
char speedA1;
char speedA2;
char speedA3;
char speedB1;
char speedB2;
char speedB3;
char speedC1;
char speedC2;
char speedC3;
char Sign1;
char Sign2;
char Sign3;
char valve;
int speedA100;
int speedA10;
int speedAone;
int speedB100;
int speedB10;
int speedBone;
int speedC100;
int speedC10;
int speedCone;
int speedA;
int speedB;
int speedC;

//Variables for runing motors copied from Aleks' 2_motor_Step_V4_AS code


// Stepper A pins
int a1 = 10;
int a2 = 11;
int a3 = 12;
int a4 = 13;
// Stepper B pins
int b1 = 2;
int b2 = 3;
int b3 = 4;
int b4 = 5;
//Stepper C pins - added by Eunice
int c1 = 6;
int c2 = 7;
int c3 = 8;
int c4 = 9;


```

```
// always choose a maxStepSpeet value that is divisible by 10!!!!!
int stepAccel = 100; // don't really need to change this acceleration value
int stepTime =10; // seconds. don't exceed 30 seconds or it will run forever
                   // motors run for stepTime*(number of motors)
int stepDirection = -1; // -1 for CW (plunger goes down) and 1 for CCW (plunger goes up) [motor pin facing down]
char firstMotor = 'A'; // 'A' => first, motor A runs for stepTime then turns off and motor B runs for stepTime
                       // 'B' => first, motor B runs for stepTime then turns off and motor A runs for stepTime

float increment ;
// for Gradient + constant mode
unsigned long wait_time;
unsigned long purge_time;
unsigned long begin_time2;

#include <AccelStepper.h> // Tools > Manage Libraries ... > search for and install AccelStepper
#define HALFSTEP 8 // number of full steps for 28BYJ-48 stepper motors
AccelStepper stepperA(HALFSTEP,a1,a3,a2,a4); // define stepper 1 pins
AccelStepper stepperB(HALFSTEP,b1,b3,b2,b4); // define stepper 2 pins
AccelStepper stepperC(HALFSTEP,c1,c3,c2,c4); // define stepper 3 pins - added by Eunice
// motor controller pin map out (FULLSTEP, N1, N3, N2, N4) Npins are the motor controller pins
int buffer_time = 22000;

void setup() {
  // put your setup code here, to run once:
  // Serial communication set up
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(trans, OUTPUT);

  // Motor Control Setup
  // Stepper 1 settings
  stepperA.setMaxSpeed(maxStepSpeed);
  stepperA.setAcceleration(stepAccel);
  // Stepper 2 settings
  stepperB.setMaxSpeed(maxStepSpeed);
  stepperB.setAcceleration(stepAccel);
  // Stepper 3 settings
```

```
  stepperC.setMaxSpeed(maxStepSpeed);
  stepperC.setAcceleration(stepAccel);
}
void loop() {
  // put your main code here, to run repeatedly:
  while (Serial.available()>0) {
    incomingByte = Serial.readStringUntil(’/n’); // read till end of line
    incomingByte.trim();
    //Serial.print(incomingByte);
    // Save input values as speed values
    // input speed values #[’s###’, ’s###’, ’s###’]#
    // input mode[speed motor A, speed motor B, speed motor C]valve on/off
    // extract individual charcaters note zero indexing
    //Serial.print(incomingByte);
    valve   = incomingByte.charAt(25);
    mode    = incomingByte.charAt(0);
    if (valve == ’1’) {
     digitalWrite(trans, HIGH);
    }
    if (valve == ’0’) {
      digitalWrite(trans, LOW);
    }
    // mode one batch mode input speed values and run those speeds
    if (mode == ’1’){
      speedA1 = incomingByte.charAt(4);
      speedA2 = incomingByte.charAt(5);
      speedA3 = incomingByte.charAt(6);
      speedB1 = incomingByte.charAt(12);
      speedB2 = incomingByte.charAt(13);
      speedB3 = incomingByte.charAt(14);
      speedC1 = incomingByte.charAt(20);
      speedC2 = incomingByte.charAt(21);
      speedC3 = incomingByte.charAt(22);
      Sign1   = incomingByte.charAt(3);
      Sign2   = incomingByte.charAt(11);
      Sign3   = incomingByte.charAt(19);

      // turn them into integers and multiply by their place value
      String speedA1s = String(speedA1);
      String speedA2s = String(speedA2);
      String speedA3s = String(speedA3);
      String speedB1s = String(speedB1);
      String speedB2s = String(speedB2);
      String speedB3s = String(speedB3);
      String speedC1s = String(speedC1);
```

```
    String speedC2s = String(speedC2);
    String speedC3s = String(speedC3);

    speedA100 = 100* speedA1s.toInt() ;
    speedA10  = 10*  speedA2s.toInt() ;
    speedAone = 1*   speedA3s.toInt() ;
    speedB100 = 100* speedB1s.toInt() ;
    speedB10  = 10*  speedB2s.toInt() ;
    speedBone = 1*   speedB3s.toInt() ;
    speedC100 = 100* speedC1s.toInt() ;
    speedC10  = 10*  speedC2s.toInt() ;
    speedCone = 1*   speedC3s.toInt() ;
    //add them to make the final speed
    speedA = speedA100 + speedA10 + speedAone;
    speedB = speedB100 + speedB10 + speedBone;
    speedC = speedC100 + speedC10 + speedCone;
    //Serial.println(speedA);
    if (Sign1 == '-'){
      speedA = speedA * -1;
    }
    if (Sign2 == '-'){
      speedB = speedB * -1;
    }
    if (Sign3 == '-'){
      speedC = speedC * -1;
    }

    //input check to make sure the sum of speeds doesn't exceed 100%
    if (abs(speedA) + abs(speedB) + abs(speedC) > maxStepSpeed){// need to change this to absolute value
      Serial.print("Invalid Input");
    }
    stepperA.setSpeed(speedA);
    stepperB.setSpeed(speedB);
    stepperC.setSpeed(speedC);
  } // End of Mode 1

  if (mode == '2'){
    // mode 2, two motor making a gradient, third motor is constant
    // need increment for gradient and motor 3 speed
    // in mode 2 #['s###', 's###', 's###']# becomes
    // mode ['steps','percent of third precursor','max percent of the gradient composition'] valve
    // for the max gradient percent and example is going from 25%-75% to 75-25 instead of 0-100 to 100-0
    // in this case the max percent will be 75 or 100
```

```
      // for the step numbers the max value is 999
      // for third precursor percentage the max value is 100
      // for max gradient percentage the max value is 100
      // #['0###', '0###', '0###']# where ### is an absolute percent ie 100 = 100%, 050 = 50%
     // Extract the info you need, note that chafcaters are 0 indexed, spaces are characters
      step1 = incomingByte.charAt(4);
      step2 = incomingByte.charAt(5);
      step3 = incomingByte.charAt(6);
      prec_per1 = incomingByte.charAt(12);
      prec_per2 = incomingByte.charAt(13);
      prec_per3 = incomingByte.charAt(14);
      grad_max1 = incomingByte.charAt(20);
      grad_max2 = incomingByte.charAt(21);
      grad_max3 = incomingByte.charAt(22);
      // turn them into integers and multiply by their place value
      String step1s = String(step1);
      String step2s = String(step2);
      String step3s = String(step3);
      String prec_per1s = String(prec_per1);
      String prec_per2s = String(prec_per2);
      String prec_per3s = String(prec_per3);
      String grad_max1s = String(grad_max1);
      String grad_max2s = String(grad_max2);
      String grad_max3s = String(grad_max3);

      step100 = 100* step1s.toInt() ;
      step10  = 10*  step2s.toInt() ;
      stepone = 1*   step3s.toInt() ;
      prec_per100 = 100* prec_per1s.toInt() ;
      prec_per10  = 10*  prec_per2s.toInt() ;
      prec_perone = 1*   prec_per3s.toInt() ;
      grad_max100 = 100* grad_max1s.toInt() ;
      grad_max10  = 10*  grad_max2s.toInt() ;
      grad_maxone = 1*   grad_max3s.toInt() ;

      steps =step100 + step10 + stepone;
      prec_per = prec_per100 + prec_per10 + prec_perone;
      grad_max = grad_max100 + grad_max10 + grad_maxone;
      wait_time = 250;
      purge_time = 20000;
      mode2(steps,prec_per,grad_max,wait_time, begin_time2, maxStepSpeed, buffer_time, stepDirection, purge_time);
    } // end of Mode 2

  } // while ends
```

```

  stepperA.runSpeed();
  stepperB.runSpeed();
  stepperC.runSpeed();
} // End of Void Loop

// START of Special Functions


void mode2 (int steps, int prec_per, int grad_max, unsigned long wait_time, unsigned long begin_time2,int maxStepSpeed,
int buffer_time, int stepDirection, unsigned long purge_time) {
  // first step is determining the speed bounds based on the
  // max percent of the gradient composition and precursor percentage
  float prec_per_float = prec_per; // prec_per = % solvent
  float C = (prec_per_float/100);
  // because percentage of solvent = Solvent speed/ ( solvent speed + precursor total speed)
  // solve that solvent speed for a given percentage of solvent composition is S = CP/(1-C) where C is desired percent composition, P is precursor total speed, and S is solvent speed
  // this makes for a maximum of 25% is solvent composition.
  float stepperC_speed = ( C * precursor_range)/(1-C);   // precursor range refers to MA/FA
  // 2['0100', '0050', '0100']0
  float max_gradient_speed = precursor_range;
  float grad_max_float = grad_max;
  float max_gradient_comp_speed = max_gradient_speed * (grad_max_float/100); // max speed based on the gradient composition
  // ie. the maximum speed of the starting motor for the 75-25 composition start point
  float min_gradient_comp_speed = max_gradient_speed - max_gradient_comp_speed;
  // now find the increment
  float step_float = steps;
  float increment = (max_gradient_comp_speed - min_gradient_comp_speed)/step_float;
  float speedq = 0;
  stepperA.setSpeed(stepDirection * min_gradient_comp_speed);
  stepperB.setSpeed(stepDirection * max_gradient_comp_speed);
  stepperC.setSpeed(stepDirection * stepperC_speed); // replaced stepperC_speed with 800 worked
  begin_time2 = millis();
  while (millis()-begin_time2 < buffer_time ){
    // run motor speeds
    stepperA.runSpeed();
```

```
    stepperB.runSpeed();
    stepperC.runSpeed();
  }
  for (int i=0; i<steps+1; i+=1){
    if (i<steps){
      speedq = speedq + increment ;
      stepperA.setSpeed(stepDirection * speedq);
      stepperB.setSpeed(stepDirection * (max_gradient_speed - speedq)
         );
      unsigned long begin_time = millis();
      while (millis()-begin_time < wait_time ){
        // run motor speeds
        stepperA.runSpeed();
        stepperB.runSpeed();
        stepperC.runSpeed();
      }
    }
    else if (i>=steps){
      unsigned long begin_time = millis();
      while (millis()-begin_time < purge_time ){
        // run motor speeds
        stepperA.runSpeed();
        stepperB.runSpeed();
        stepperC.runSpeed();
      }
    }
  }
  stepperA.setSpeed(0);
  stepperB.setSpeed(0);
  stepperC.setSpeed(0);
  digitalWrite(trans, LOW);
}
```

References

[1] J.-P. Correa-Baena, K. Hippalgaonkar, J. van Duren, S. Jaffer, V. R. Chandrasekhar, V. Stevanovic, C. Wadia, S. Guha, and T. Buonassisi. "Accelerating Materials Development via Automation, Machine Learning, and High-Performance Computing". In: *Joule* 2 (8 Aug. 2018), pp. 1410–1420. ISSN: 2542-4785. DOI: 10.1016/j.joule.2018.05.009.

[2] A. E. Siemenn. *A System for High-Throughput Materials Exploration Driven by Machine Learning.* 2021.

[3] R. M. Hazen. "Perovskites". In: *Scientific American* 258.6 (1988), pp. 74–81. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/24989124 (visited on 08/23/2024).

[4] L. Ortega-San-Martin. "Introduction to Perovskites: A Historical Perspective". In: *Revolution of Perovskite: Synthesis, Properties and Applications*. Ed. by N. S. Arul and V. D. Nithya. Singapore: Springer Singapore, 2020, pp. 1–41. ISBN: 978-981-15-1267-4. DOI: 10.1007/978-981-15-1267-4_1. URL: https://doi.org/10.1007/978-981-15-1267-4_1.

[5] L. Zhang et al. "Advances in the Application of Perovskite Materials". In: *Nano-Micro Letters* 15 (1 Aug. 2023), p. 177. ISSN: 2150-5551. DOI: 10.1007/s40820-023-01140-3.

[6] X. Fan. "Advanced progress in metal halide perovskite solar cells: A review". In: *Materials Today Sustainability* 24 (2023), pp. 100–603. ISSN: 2589-2347. DOI: https://doi.org/10.1016/j.mtsust.2023.100603. URL: https://www.sciencedirect.com/science/article/pii/S2589234723002907.

[7] Y. Rong, Y. Hu, A. Mei, H. Tan, M. I. Saidaminov, S. I. Seok, M. D. McGehee, E. H. Sargent, and H. Han. "Challenges for commercializing perovskite solar cells". In: *Science* 361.6408 (2018), eaat8235. DOI: 10.1126/science.aat8235. eprint: https://www.science.org/doi/pdf/10.1126/science.aat8235. URL: https://www.science.org/doi/abs/10.1126/science.aat8235.

[8] A. E. Siemenn, E. Aissi, F. Sheng, A. Tiihonen, H. Kavak, B. Das, and T. Buonassisi. "Using scalable computer vision to automate high-throughput semiconductor characterization". In: *Nature Communications* 15.1 (2024), p. 4654.

[9] F. F. Targhi, Y. S. Jalili, and F. Kanjouri. "MAPbI3 and FAPbI3 perovskites as solar cells: Case study on structural, electrical and optical properties". In: *Results in Physics* 10 (2018), pp. 616–627. ISSN: 2211-3797. DOI: https://doi.org/10.1016/j.rinp.2018.07.007. URL: https://www.sciencedirect.com/science/article/pii/S2211379718311811.

[10] S. Sun et al. "A data fusion approach to optimize compositional stability of halide perovskites". In: *Matter* 4.4 (2021), pp. 1305–1322. DOI: 10.1016/j.matt.2021.01.008.

[11] Z. A. Nan et al. "Revealing phase evolution mechanism for stabilizing formamidinium-based lead halide perovskites by a key intermediate phase". In: *Chem* 7 (9 Sept. 2021), pp. 2513–2526. ISSN: 2451-9294. DOI: 10.1016/J.CHEMPR.2021.07.011.

[12] J. Wu, J. Chen, and H. Wang. "Phase Transition Kinetics of MAPbI3 for Tetragonal-to-Orthorhombic Evolution". In: *JACS Au* 3 (4 Apr. 2023), pp. 1205–1212. ISSN: 26913704. DOI: 10.1021/JACSAU.3C00060/SUPPL_FILE/AU3C00060_SI_002.ZIP.

[13] R. Keesey et al. "An open-source environmental chamber for materials-stability testing using an optical proxy". In: *Digit. Discov.* (2023). ISSN: 2635-098X. DOI: 10.1039/D2DD00089J.

[14] P. Menesatti, C. Angelini, F. Pallottino, F. Antonucci, J. Aguzzi, and C. Costa. "RGB Color Calibration for Quantitative Image Analysis: The "3D Thin-Plate Spline" Warping Approach". In: *Sensors* 12.6 (2012), pp. 7063–7079. ISSN: 1424-8220. DOI: 10.3390/s120607063.

[15] C. C. Stoumpos, L. Mao, C. D. Malliakas, and M. G. Kanatzidis. "Structure-Band Gap Relationships in Hexagonal Polytypes and Low-Dimensional Structures of Hybrid Tin Iodide Perovskites". In: *Inorg. Chem.* 56 (1 Jan. 2017), pp. 56–73. ISSN: 1520510X. DOI: 10.1021/ACS.INORGCHEM.6B02764/SUPPL_FILE/IC6B02764_SI_013.CIF.

[16] B. Charles, J. Dillon, O. J. Weber, M. S. Islam, and M. T. Weller. "Understanding the stability of mixed A-cation lead iodide perovskites". In: *J. Mater. Chem. A* 5 (43 Nov. 2017), pp. 22495–22499. ISSN: 20507496. DOI: 10.1039/C7TA08617B.

[17] O. J. Weber, B. Charles, and M. T. Weller. "Phase behaviour and composition in the formamidinium–methylammonium hybrid lead iodide perovskite solid solution". In: *J. Mater. Chem. A* 4 (40 Oct. 2016), pp. 15375–15382. ISSN: 2050-7496. DOI: 10.1039/C6TA06607K.

[18] A. Pisanu, C. Ferrara, P. Quadrelli, G. Guizzetti, M. Patrini, C. Milanese, C. Tealdi, and L. Malavasi. "The FA1–xMAxPbI3 System: Correlations among Stoichiometry Control, Crystal Structure, Optical Properties, and Phase Stability". In: *J. Phys. Chem. C* 121 (16 Apr. 2017), pp. 8746–8751. ISSN: 1520-5215. DOI: 10.1021/acs.jpcc.7b01250.

[19] A. Binek, F. C. Hanusch, P. Docampo, and T. Bein. "Stabilization of the Trigonal High-Temperature Phase of Formamidinium Lead Iodide". In: *J. Phys. Chem. Lett.* 6 (7 Apr. 2015), pp. 1249–1253. ISSN: 1948-7185. DOI: 10.1021/acs.jpclett.5b00380.

[20] A. Siemenn, B. Das, E. Aissi, F. Sheng, L. Elliot, B. Hudspeth, M. Meyers, J. Serdy, and T. Buonassisi. "A Retrofitted 3D Printer for High-throughput Combinatorial Experimentation via Continuous Printing." In: *ChemRxiv* (2024). DOI: 10.26434/chemrxiv-2024-fz16p.

[21] R. Sliz, J. Czajkowski, and T. Fabritius. "Taming the Coffee Ring Effect: Enhanced Thermal Control as a Method for Thin-Film Nanopatterning". In: *Langmuir* 36.32 (2020), pp. 9562–9570.