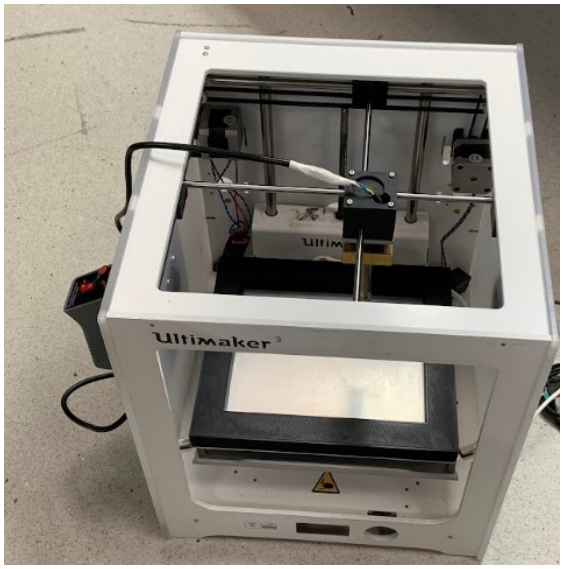
TU Delft Delft University of Technology

DELFT UNIVERSITY OF TECHNOLOGY

MASTER THESIS

FLEXIFABRICATE: PLASTIC WELDING TECHNIQUE USED TO JOIN PLASTIC LAYERS FOR ORIGAMI ASSISTIVE GLOVE FABRICATION



Hamed Abolhassan Beigi

DELFT UNIVERSITY OF TECHNOLOGY

MASTER THESIS

FlexiFabricate: Plastic welding technique used to join plastic layers for origami assistive glove fabrication

Author:
Hamed Abolhassan Beigi (5048478)

Graduation committee:
Dr. ir. Arno Stienen, Chair & Supervisor
Prof.dr. Frans van der Helm, Committee member

August 22, 2024

Mechanical Engineering, department of BioMechanical Engineering

CONTENTS

ACKNOWLEDGMENT

The authors would like to thank Dr. ir. Arno Stienen, for his valuable guidance and support throughout this research. Arno's knowledge and guidance were key in shaping my research, and his consistent support made a big difference in bringing this work to completion.

I also acknowledge Delft University of Technology for providing the necessary resources.

Hamed Abolhassan Beigi.
Delft, August 2024

***Abstract*—This paper aims to explore innovative fabrication methods for assistive gloves using origami design principles. The study addresses the limitations of current assistive gloves, such as bulkiness and lack of customisation, by developing a novel fabrication method that improves ergonomic design and manufacturing flexibility. The proposed method integrates 3D printing and welding techniques to create lightweight, customisable gloves that enhance rehabilitation for individuals with hand impairments, particularly stroke survivors. The paper details the design process, and testing methods, concluding with successfully creating a new fabrication approach for origami-based assistive gloves.**

***Index terms*—** Assistive Glove, Origami robotics, self-folding robots, Fabrication, 3d printing, welding technique, heat-sealing plastic, heat-sealing elastic, 3d plotter machine.

I. INTRODUCTION

Hand functionality is paramount to our daily lives, enabling us to perform a multitude of tasks with precision and dexterity. From simple actions like grasping objects to complex manipulations required in professional or domestic settings, our hands are indispensable tools. However, when hand functionality is compromised due to injury, disability, or medical conditions such as stroke, it can profoundly impact an individual's independence and quality of life.

The loss of hand functionality can present significant challenges, ranging from difficulties in performing basic self-care tasks to limitations in engaging in work or recreational activities. Individuals who experience a decline in hand function often face barriers to independence [1] and may require assistance with even the most routine activities [2].

Various factors can contribute to the loss of hand functionality, including traumatic injuries, degenerative conditions like arthritis, and neurological disorders such as stroke. Among these, stroke stands out as a leading cause of disability worldwide with approximately $800,000$ Stroke-patients annually in the USA [3], frequently resulting in impairments in motor control and coordination, particularly on one side of the body, including the hand.

Assistive gloves represent a promising solution for individuals grappling with hand impairments [4], including those stemming from a stroke. These gloves leverage innovative technologies to support weakened hand muscles, enhance grip strength, and facilitate movement precision. Importantly, they serve as valuable tools in rehabilitation, helping individuals relearn essential motor skills and regain independence in daily activities.

For stroke survivors, assistive gloves play a crucial role in rehabilitation by providing targeted assistance to weakened hand muscles and promoting neural recovery. These gloves help stimulate the brain's plasticity by offering tactile feedback and guiding movements, facilitating the rewiring of neural pathways necessary for improved hand function [5]. Additionally, assistive gloves can assist individuals in performing repetitive exercises vital for motor skill relearning, contributing to more efficient and effective rehabilitation outcomes [6].

While an array of assistive gloves exists to aid patients with hand functionality impairment, they often grapple with challenges such as excessive bulk, weight, intricate designs, and discomfort, consequently resulting in suboptimal fit. Furthermore, these gloves are predominantly available in standardized sizes, rendering customization for individual patients unattainable [3], [9], [10], [11], [12], [13]. These limitations highlight the need for advancements in ergonomic design and manufacturing flexibility.

Origami, an ancient Japanese paper-folding art, has obtained a growing interest in the field of robotics, primarily due to its unique attributes. These distinctive features encompass the capacity to seamlessly transition from a 2D planar structure to a 3D configuration, adapt to various modes and shapes, execute tasks both autonomously and manually, possess a lightweight construction, enable straightforward manufacturing processes [14], and offer a versatile array of activation methods and control strategies. These exceptional qualities have shown in a new era of robotics, enabling the creation of previously unfeasible

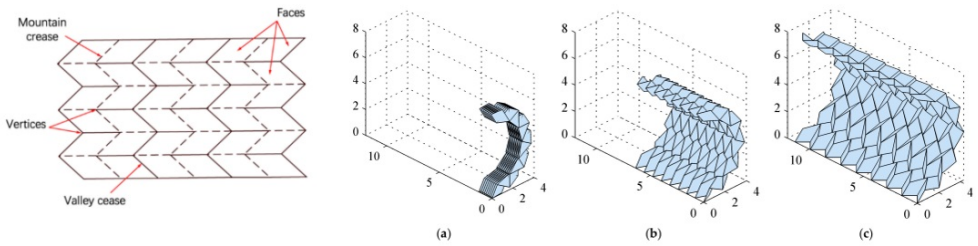


Fig. 1: Left figure: A Miura Ori folding pattern made on a single layer of paper. a) folded state b) partially folded state. Chen et al [7].

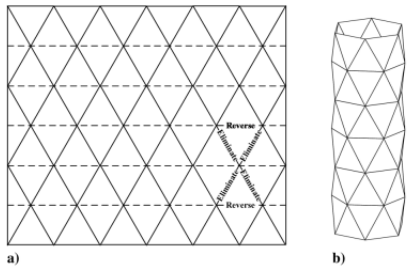


Fig. 2: a) Typical Yoshimura folding pattern. Solid and dashed lines represent mountain and valley folds. A typical Yoshimura pattern consists of uniform triangles that fold into a cylindrical shape and can expand and contract under uniaxial force, b) cylindrical structure made with Yoshimura pattern. This figure is taken and modified from Sue et al [8].

robots within the confines of conventional robotics, primarily rooted in the assembly of rigid components.

Moreover, Origami robots are designed to embrace the concept of compactness and flexibility, enabling them to perform a diverse array of tasks. Their design is an art form, relying on geometric principles and intricate folding patterns that transform flat sheets of material into three-dimensional marvels [15]. This transition is achieved by the purposeful utilization of mountain and valley folds, serving as pivotal mechanisms to generate precise folding actions that induce either the contraction or expansion of the origami robot's overall structure or specific components. Importantly, a significant proportion of origami robots adhere to universally recognized design principles when constructing these crease patterns such as [16]:

1) Miura crease pattern (Figure 1).
2) Yoshimura pattern (Figure 2).
3) Kresling pattern.
4) Waterbomb pattern.
5) Magic ball pattern.

Fabricating origami designs involves converting these designs into tangible structures, necessitating a thorough understanding of both materials and manufacturing techniques. The materials used can vary from traditional paper to advanced options like polymers or metals [18]. Similar to crafting an intricate origami piece, the fabrication process requires precision and meticulous attention to detail to ensure the final product functions as intended. Origami robotics also employs standard fabrication methods akin to those used in design. There are three primary methods for fabricating an origami design [16]:

1) Single layer fabrication (Figure 4).
2) Layer-by-layer fabrication (Figure 5)
3) 3D print (Figure 3)

Furthermore, the single-layer fabrication method as the name suggests makes use of only one single layer of material and employs advanced printing technology or laser-cut machinery to create the design patterns. This method has the limitation of not being able to merge multiple layers of the materials for designing more

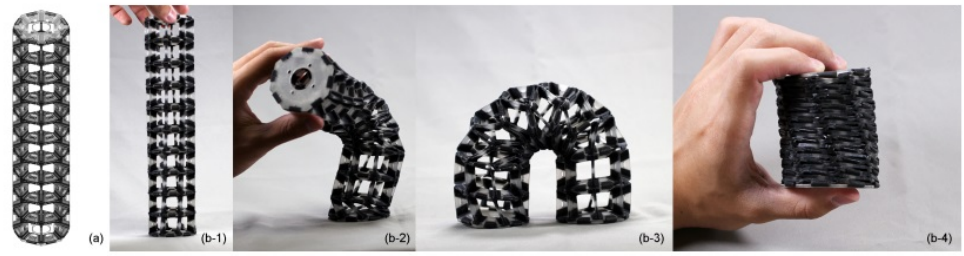


Fig. 3: 3d printing fabrication method. A 10-layer tower with an octagon as a base geometry: (a) CAD model and (b) 3D printed tower shown in different configurations [17].

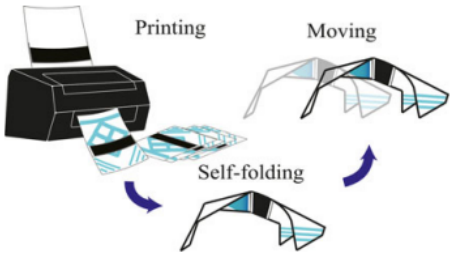


Fig. 4: Single layer fabrication method using an advanced printing technology. [19]

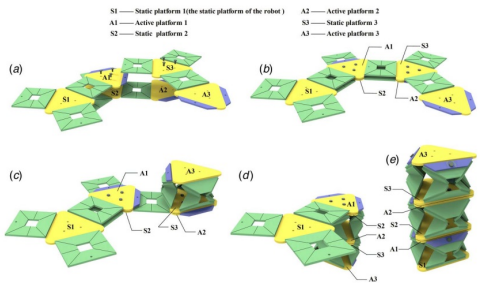


Fig. 5: Layer-by-layer manufacturing process an illustrative example. The assembly process of the segment: (a) cardboard in a regular arrangement, (b) connect the active platforms and static platforms, (c) fold the 3rd unit into shape, (d) fold the 2nd unit into shape, and (e) fold the 1st unit into shape [20].

complex origami robots and is constrained by the use of a single uniform material. In addition, this method restricts folding in a single direction and potentially weakens crease lines, which can lead to a reduced operational lifespan, as highlighted in [21]. While the layer-by-layer manufacturing method mitigates the limitations of the single-layer approach by allowing for multiple layers with diverse materials and crease patterns in various orientations, it necessitates the creation of individual crease patterns for each layer using laser-cut machinery. These layers are then assembled using techniques such as heat-press, pick-and-place automated robots, or manual assembly. Consequently, this method increases both the time required and the complexity of the fabrication process. While 3D-printing technology can use different materials to construct a given design pattern by adding the materials layer by layer, this method comes also with some limitations such as dimensional accuracy and precision, poor adhesion, residual stress, and printing time [22].

In summary, in light of the crucial role of hand functionality in daily life and the challenges individuals face when impaired, there is a pressing need for innovative solutions to enhance rehabilitation and improve quality of life. Existing assistive gloves, while promising, often fall short due to their bulkiness, complexity, and lack of customization options, hindering their effectiveness and user comfort. While, the origami concept provides features such as compactness, and lightweight to address the downsides of the existing assistive gloves, the current fabrication methods for origami design

Single-layer	Layer-by-layer	3D printing
One single layer	each layer should be made individually	Poor adhesion
Uniform material	need another method to attach the layers	Residual stress
uni-directional crease creation	time consuming	time consuming
Weak crease lines		

TABLE I: The limitations in the existing fabrication methods in constructing origami designs.

patterns present notable restrictions, including poor dimensional accuracy, time-consuming processes, and restricted material choices. Table I gives an overview of the limitations in the existing fabrication methods used in the field of origami robotics.

In the context of origami robots the need for a new fabrication method is clear that can use multiple layers of the material, does not require additional methods or mechanisms to put the final product together, can take at least two different or the same material types, has relatively high dimensional accuracy and precision, does not have complex fabrication process, is fast, has low residual stress and enables pneumatic activation.

Additionally, because the final goal of the proposed method in this paper is to fabricate origami assistive gloves, the product should generate sufficient force required to bend the fingers. I.e., the outcome of the machine should withstand the pressure needed to bend fingers. Also, it should enable the fabrication of customized, lightweight, and ergonomic (not bulky) gloves. Moreover, the produce origami glove should adhere to the concept of origami. That is to enable the transition from $2D$ to $3D$ upon activation and re-configuring to the initial $2D$ state when deactivated.

to sum up, the fabrication method should be able to:

1) Attach at least two different or the same material layers to fabricate a glove.
2) To achieve this in one single machine. I.e., it does not need any additional device.
3) Have a dimensional accuracy and precision, a root mean square error (RMSE) smaller equal to 0.5 $[mm]$.
4) Have a user-friendly interface. I.e., it is easy to work with.
5) Construct origami design patterns that can transfer from a $2D$ to a $3D$ state upon activation and
6) Re-configure to its initial state when deactivated.
7) Enable pneumatic activation of the produced origami design pattern.
8) Produce design patterns that can tolerate the activation force required to bend the fingers approximately equal to 120 $[kpa]$ or 1.2 $[bar]$ [23].
9) Make lightweight customized assistive gloves
10) Make ergonomic assistive gloves. I.e., the glove should be comfortable to wear.

Therefore, this study aims to develop a novel fabrication method for origami design patterns, striving to overcome the existing limitations in the current fabrication techniques. By addressing issues such as dimensional accuracy, material versatility, and production efficiency, this new method seeks to enable the creation of lightweight, customizable origami assistive gloves that are ergonomic and user-friendly. Through the integration of advanced manufacturing technologies and optimized design principles, we aim to streamline the fabrication process and improve the overall functionality and comfort of assistive gloves for individuals with hand impairments, particularly stroke survivors.

II. Device Design

A. structural Backbone

The Ultimaker 3 (UM3) 3D printer frame is given as the structural backbone, Figure 6 for the

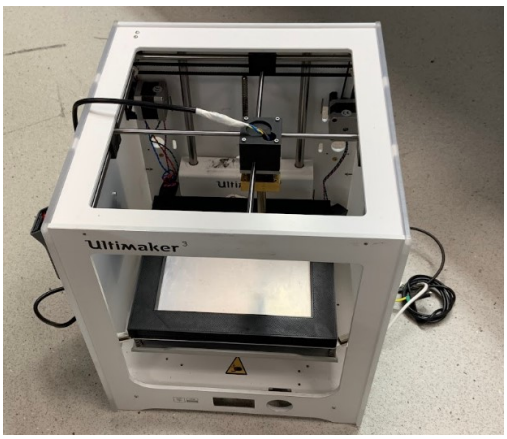


Fig. 6: The structural backbone of the machine.

FlexiFabricate system, owing to its robustness, adaptability, and availability. The UM3 uses three stepper motors, three hard limit switches, and a build plate.

1) Workspace: The workspace defines the machine's internal dimensions, indicating the area the machine can effectively reach. Therefore, these dimensions must be considered during pattern design. Notably, the new fabrication method, which involves using a soldering iron rod for material sealing, results in the original build dimensions of the UM3 being different from the workspace of the FlexiFabricate. This variation is primarily due to the necessity of a new frame design to support the soldering iron for the FlexiFabricate. Furthermore, the new soldering iron holder frame must meet specifications different from the original UM3 nozzle frame. For example, a frame designed for a nozzle that prints material onto a build plate does not need to counteract shear forces generated by the frame's movement in direct contact with material layers, a critical requirement for a soldering iron frame in heat-sealing plastic layers. Additionally, a soldering iron rod's longer length than a nozzle leads to a significantly reduced plate height. Table II presents the maximum travel distances along each axis for both the original UM3 and the new FlexiFabricate.

In addition, the machine's plate contacts the soldering rod tip at $\boxed{Z_{plate-height} = 136 \ [mm]}$ and starts exerting force on the spring. Therefore, it is essential to consider the thickness of the material while developing a G-code file. Since pressure is directly affected by the Z-level, equation 1 should be used to ensure the pressure does not exceed the optimum pressure.

	X [mm]	Y [mm]	Z [mm]
UM3 [24]	216	216	200
FlexiFabricate	212	212	145

TABLE II: Workspace of the machine.

$$Z_{contact} = Z_{plate-height} + \tau \tag{1}$$

Where τ refers to the thickness of the material.

It is pertinent to address that an enormous vertical pressure applied by the machine's plate to the soldering iron rod will diminish the fluidity of the machine's motion along the XY plane and may result in the loss of force by the z-stepper motor due to the fact that the stepper motors are in general not good at torque or force generation. So, they can hold a maximum force and if the force exceeds this maximum, the stepper motor will turn in the backward direction. Another downside is that this backward motion is not accounted for by the machine i.e., the real location of the rod tip is no longer the same as indicated by the machine, and a *homing* procedure is required for the machine to know its real position.

Another important feature is that the machine's workspace is defined in the negative space meaning that any given command to the machine should be in the negative direction. For example, the following command line will move the machine from its current position to $-100[mm]$ in all three axes:

```
; To move the machine 100 mm in the space.
     Note that the F 100 refers to the
    feed rate or the speed of the movement

G01 X-100 Y-100 Z-100 F 100
```

Listing 1: Gcode example

Finally, the XY-origin of the soldering iron tip is set to the front-left of the workspace, opposite to the machine's home position.

B. Hardware Setup

One of the most critical parameters in defining the performance of the FlexiFabricate system is ensuring precise control over the printer's movement, avoiding any unintended motion beyond the commanded path. To take control of the machine, the original UM3 controller board is replaced with an Arduino UNO, a highly regarded controller board in manufacturing CNC and 3D printer machines. The Arduino UNO board requires a CNC shield and three stepper motor drivers to operate the stepper motors.

The chosen CNC shield must be compatible with the UNO board and facilitate precise control of the stepper motors. Consequently, the Arduino CNC Shield V3.00 is selected due to its capability to accurately control four stepper motors, compatibility with the Arduino UNO, and support for micro-stepping of the stepper motors for enhanced path accuracy. Additionally, the A4988 stepper motor driver is selected to drive the stepper motors, primarily because it supports micro-stepping from $\frac{1}{2}$ to $\frac{1}{16}$ micro-steps [26] and is fully compatible with the Arduino CNC Shield V3.00. Figure 7 illustrates the mentioned Arduino UNO board, Arduino CNC Shield, the A4988 stepper motor drivers, and the USB cable to connect the UNO board to an operating system.

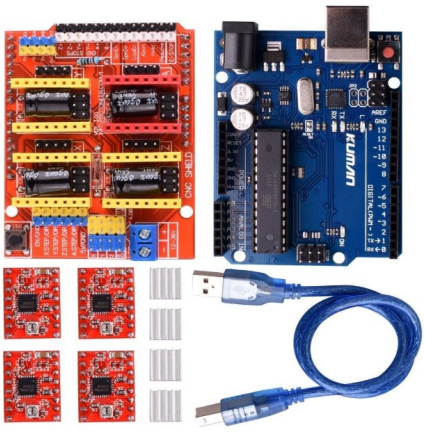


Fig. 7: Hardware components used in FlexiFabricate. The top left image shows the Arduino CNC Shield, the top right image depicts the Arduino UNO board, the bottom left one demonstrates four A4988 stepper motor drivers and the bottom right shows a USB cable to connect the Arduino UNO board to the operating system (laptop or PC).

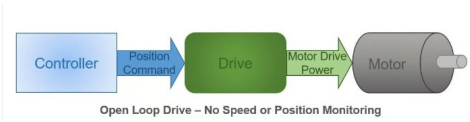


Fig. 8: The open loop system used in the CNC machines to drive the stepper motors. The controller sends the incremental steps as pulse signals to the stepper motor drivers. Consequently, the drivers convert these pulses into the steps for the stepper motors to run toward the target position. The figure is taken and modified from [25]

It is also worth noting that the Arduino UNO controller board, the Arduino CNC Shield V3.0 and stepper motor drivers follow an open-loop system to drive the stepper motors as shown in Figure 8. In an open-loop system, the distance from the current location to the target location is divided into a number of steps for precise and incremental positioning of the machine by the controller (Arduino UNO and Arduino CNC Shield). Furthermore, these incremental steps, also commonly referred to as micro-steps, are then sent to the stepper motor drivers in the form of pulse signals. Afterwards, the stepper motor drivers convert these pulse signals into the derivable steps for the stepper motors [25].

Figure 9 shows how to connect the Arduino UNO board, Arduino CNC Shield V3.00, and A4988 the stepper motor drivers. A detailed assembly explanation can be found in [26].

In addition, Arduino CNC Shield V3.00 allows using three jumpers to enable micro-stepping of

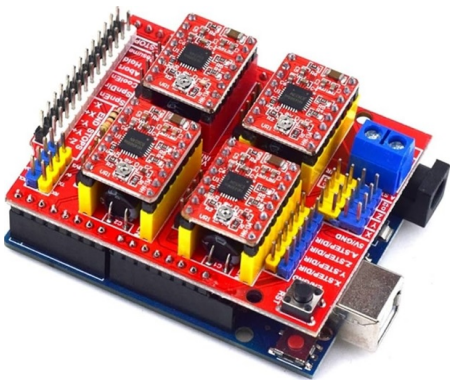


Fig. 9: Connecting Arduino UNO board to Arduino CNC Shield and A4988 stepper motor drivers.

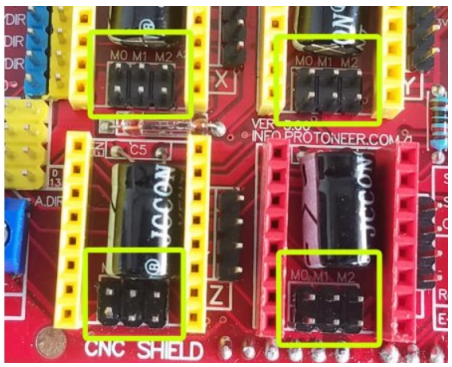


Fig. 10: Inserting jumpers in the M0, M1, and M2 pins on Arduino CNC shield v3.00 to enable micro stepping of the stepper motors [27]. Table III gives an overview of the possible micro-steps of the A4988 stepper motor driver.

the A4988 stepper motor driver for enhancing path-following accuracy and smooth motion of each stepper motor as shown in Figure 10. To enable the micro-stepping a jumper can be inserted into each pin. Table III shows how to enable different micro steps from the full to the sixteenth step for each stepper motor. Furthermore, micro-stepping is used in the controller system of the CNC machines to divide the number of steps into smaller steps to enhance the path-following capability. For example, if a stepper motor uses a full-step of 1.8°, with half micro-step activation it will step in an increment of $\frac{1.8}{2} = 0.9°$ which results in doubling the precision of the machine. However, enabling the micro-stepping technology has the drawback that it reduces the maximum torque generation by the stepper motor. As a consequence, the stepper motors can deliver less power. However, this Pitfall may not be of much importance for 3d printer machines, it plays a very critical role in FlexiFabricate. Primarily, because, as mentioned earlier, in FlexiFabricate the stepper motors must overcome shear forces due to the direct contact with the material for enhanced heat flux. Therefore, an attempt should be made to achieve an optimum balance between path-following precision and torque generation capacity.

M0	M1	M2	Microstep resolution
LOW	LOW	LOW	FULL STEP
HIGH	LOW	LOW	HALF STEP
LOW	HIGH	LOW	QUARTER STEP
HIGH	HIGH	LOW	EIGHTH STEP
HIGH	HIGH	HIGH	SIXTEENTH STEP

TABLE III: Enabling micro-steps of the A4988 stepper motor driver [26]. To set a pin HIGH a jumper should be inserted into that specific pin on the Arduino CNC Shield v3.00 as shown in Figure 10.

C. Software Selection

In the field of software architecture for advanced manufacturing systems, a crucial decision involves selecting the appropriate framework: developing a bespoke solution tailored specifically to the machine's needs or adopting a universal framework with broader applicability. Opting for the latter one, the GRBL library [28] is chosen as the cornerstone of our software integration strategy. This decision is driven by GRBL's renowned versatility and compatibility with G-code, making it an ideal choice for precise communication with CNC machines. This capability is essential for executing intricate fabrication tasks and ensuring exact machine movements.

The GRBL library offers extensive control over critical machine parameters, including:

- **Homing Procedures**
- **Soft and Hard Limits**
- **Maximum Travel Distances**
- **Step Resolution**

Next, to enhance communication with the machine, the Universal G-code Sender (UGS) platform [29] is chosen mainly for its intuitive interface and seamless compatibility with GRBL. Key features of UGS include:

- **G-code Testing and Visualization**: Allows preemptive identification of potential issues such as exceeding maximum travel distances along the X and Y axes.
- **User-friendly Interface**: Simplifies communication with the FlexiFabricate system.

This pre-execution validation is crucial for ensuring smooth and error-free operation during production runs, significantly enhancing system reliability and operational efficiency.

The UGS platform can also be used to set and store important parameters such as *maximum travel distance, maximum feed rate, and steps per millimetre* to the EEPROM of the Arduino UNO controller board. For example, the following command lines are used to change and store the maximum travel distance in X, Y, and Z directions to the value in table II.

```
$130 = 212; command to change the maximum
    travel distance in the X-axis

$131 = 212; command to change the maximum
    travel distance in the Y-axis

$132 = 145; command to change the maximum
    travel distance in the Z-axis
```

Listing 2: An example of the commands to change and save the maximum travel distance in UGS platform.

1) Homing, soft, and hard limit switches: Next, the Homing feature provided by the grbl library is enabled. Homing is important for the machine to know its position especially when the machine's power is off or the users move the motors by accident. Activating *Homing* requires other important features, namely *soft* and *hard limits*. As the names suggest, these features ensure that the machine will not move beyond its workspace limits. For example, if a command tries to send the machine outside of its maximum travel distance, soft limits will not start the stepper motors and will show an alarm to check and correct the command. Similarly, hard limits will stop the stepper motors from running if they are pressed with an alarm that the machine coordinates are incorrect and a Homing is required for the machine to know its actual position.

To activate the *hard limits* the outputs of the limit switches are inserted into the corresponding pins of the Arduino CNC Shield V3.00. That are pins 9, 10, 11 for x, y, z direction respectively.

D. Operating Procedures

Communication with the machine is conducted via the UGS platform. To command the machine using the UGS platform first, the Arduino UNO board should be connected to the operating system (laptop, PC) through the provided USB cable, Figure 7. Then, the UGS platform can take control of the machine. A detailed explanation of how to use the UGS platform is provided in [30] (steps 1 to 3). After successfully connecting the UGS platform to the controller board any G-code commands and files can be run through the UGS interface.

It is important to highlight that G-code is the only programming language the machine can understand. Therefore, writing a proper G-code file is essential to obtain the desired result. Different platforms can be used to write G-code. For example, *Notepad ++* [31] can be used to write a G-code which also provides a feature to visualize a G-code, e.g., *NCentic gcode plugin* [32] while developing.

Another approach to designing crease patterns can be achieved through programming languages like Python or MATLAB. The generated design can be stored in a two-dimensional array, representing the X and Y coordinates. These coordinates can then be converted into a G-code file via the provided Python script (Appendix

A). When employing this method, it is crucial to consider that it presumes the design pattern is uniform, necessitating the machine to level the Z-height only once. For designs that include unconnected parts, it is strongly recommended to separate the coordinates from the brake lines to ensure proper execution.

An alternative method for utilizing the machine entails using Computer-Aided Software Design (CAD) such as SolidWorks to develop a design pattern. Since origami design patterns are created in a two-dimensional format, the pattern can be saved in the SVG file format and converted into G-code via Inkscape [33]. Instructions for this conversion are provided in [34], [35] and Appendix K (Figures 36(d), 36(e), and 36(f)). Additionally, a Python script (Appendix B) is available to convert the Inkscape output into a format that meets the Universal Gcode Sender (UGS) requirements, thus making it suitable for FlexiFabricate. To verify that the converted file complies with the machine's specifications, the G-code file can be visualized and checked using Notepad++ prior to loading into the machine.

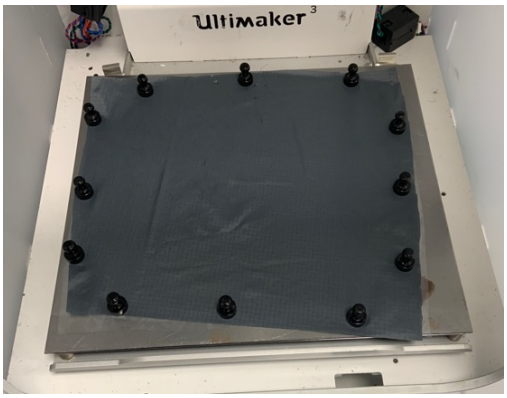


Fig. 11: Magnetic clamps used to fix the material layers on the build plate of the machine.

An important factor in operating the machine is the fixation of the material layers to the machine's plate. To do this a series of 12 magnets are provided to fix the material layers, shown in Figure 11, to the build plate. Because the original build plate of the machine is made from diamagnetic Aluminium, a ferromagnetic material with a thickness of 2 $[mm]$ is stuck to the original build plate, using double sided tapes.

Once the material layers are securely fixed, the soldering iron station can be activated, and the operating temperature set. It is important to note that the soldering station requires approximately 5 to 7 minutes to heat the soldering iron tip to the desired temperature. Additionally, it is worth mentioning that altering the temperature setting, such as adjusting from 200°C to 300°C, takes about 3 minutes.

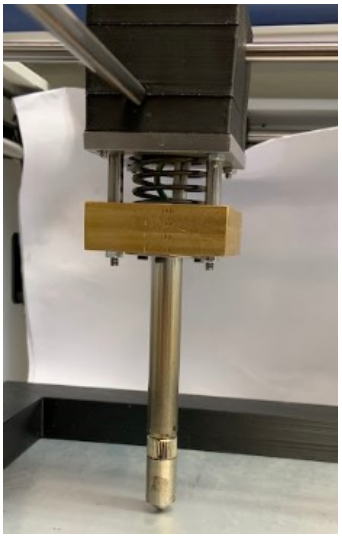


Fig. 12: A complete view of the soldering iron frame.

E. Soldering Iron Holder Frame

Considering the approach introduced by FlexiFabricate, which leverages a soldering iron for material fusion, the traditional extrusion nozzle of the Ultimaker 3 has been replaced with a soldering iron assembly. Although many different soldering irons exist in the market, the Velleman solder station (*VTSS4N 5410329442477*) [36] is chosen to replace the original nuzzle of the UM3. This choice is mainly supported by its high-temperature range from 150 to 450°*C* [36] and low cost compared to other soldering stations. It is worth noting that this high-temperature range enables using materials with different melting points enlarging

the application scope of the FlexiFabricate. This alteration necessitates the development of a tailored frame solution to accommodate the soldering iron and optimize its functionality for the specific requirements of heat-welding plastic materials.

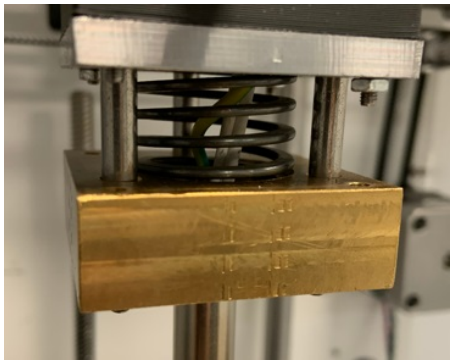


Fig. 13: The guidelines used in the soldering iron frame. The guidelines are designed with high precision to allow smooth vertical movement of the soldering iron while restricting non-intentional movement in the XY plane.

To design a robust frame capable of carrying the soldering iron, it is essential to decide whether to create a rigid frame with a fixed tip position or a frame that allows the soldering iron tip to move along the Z-axis. Due to the importance of applying vertical pressure to achieve high-quality sealing strength in heat sealing plastic techniques [37], the latter design approach was chosen. This involves designing a frame that permits the plate to move higher than the contact point with the soldering iron tip, specifically moving above $Z_{contact}$ as outlined in equation 1. Furthermore, a frame that allows controlled motion along the Z-axis while preventing any unintended movement in the XY plane ensures the ability to determine the optimal pressure required for achieving maximum heat sealing strength.

The frame design, depicted in Figure 12, is engineered with high precision to ensure optimal performance during the soldering process. The frame is configured to apply and maintain vertical pressure during sealing to streamline an efficient heat transfer to the material layers. The application of steady vertical pressure ensures the obtaining of stable sealing lines.

Additionally, the frame is fortified to withstand potential perturbations along the z-axis, acting as a buffer to absorb and mitigate any disturbances that could damage the soldering iron components or the materials under processing.

To realize this, the soldering iron frame utilises a spring, as depicted in Figure 12. Including a spring in the frame structure allows the soldering iron to exert and regulate the pressure on the soldering iron ensuring the desired vertical contact force and an effective heat transfer from the soldering iron to the material layers are retained. Likewise, a spring in the frame body absorbs vertical perturbations, thereby preventing damage to the soldering iron.

To facilitate the movement of the soldering iron in the Z-direction, the frame utilizes three guidelines, as depicted in Figure 13. These guidelines are secured to the soldering iron holder frame at the top and the soldering iron tip at the bottom, with a maximum allowable movement distance constrained to the zero-length of the spring (30 $[mm]$). The guidelines are fabricated with high precision to minimize friction during vertical movement.

Finally, accurate control over the soldering iron's positioning is imperative for maintaining the fidelity of the fabricated structures. To this end, the frame design incorporates mechanisms to constrain undesired motion within the XY plane, thereby ensuring that the soldering iron's movements are exclusively dictated by the commanded motion of the machine's control system.

F. Soldering Iron Tip Design

The diameter and shape of the soldering iron tip are critical design parameters that significantly impact the heat sealing quality. Specifically, the tip diameter determines the weld thickness, necessitating the complete closure of weld lines, thus making selecting an optimal diameter essential.

Fig. 14: Original soldering iron tip of Velleman Soldering station. The soldering iron has a sharp head and a small tip diameter.

While a very small tip diameter may result in insufficient heat sealing strength, an excessively large diameter impairs the machine's capability to produce fine crease lines, which are crucial for origami-designed assistive gloves. Therefore, achieving a balance between sufficient weld thickness and the ability to fabricate small crease lines is imperative.

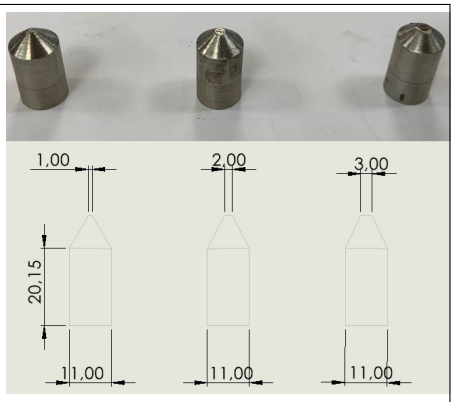


Fig. 15: The rounded soldering iron tip used in FlexiFabricate to weld the materials. A rounded tip form ensures smooth contact with materials and minimizes the shear forces generated by the movement of the soldering iron frame.

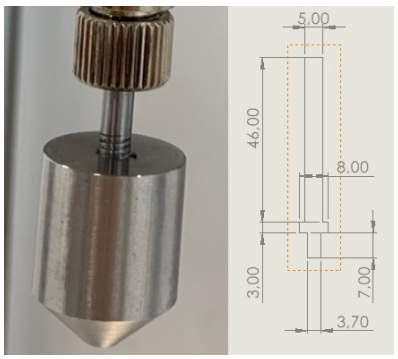
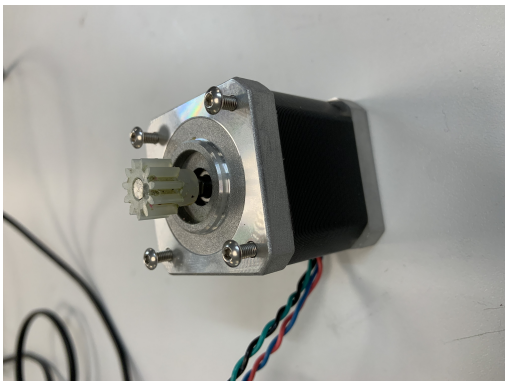


Fig. 16: The tip contains a nut where the soldering iron rod (heat element) can be screwed in, enabling the use of multiple soldering iron tips. A bolt and nut mechanism ensures the tip and rod's position remains fixed and supports sufficient heat transfer from the soldering heat element to reach the desired sealing temperature. The tip displayed in this figure has a contact diameter of 1 [*mm*].

Determining the minimum and maximum thicknesses the machine can handle is essential for optimizing performance. Experimental trials with various tip diameters, validated through pneumatic testing for weld closure, help establish the minimum effective thickness. Conversely, the maximum diameter is constrained by the precision required for crease patterns. Multiple soldering tips with varying diameters are recommended to accommodate these diverse requirements. Consequently, tips with diameters of 1 mm, 2 mm, and 3 mm are produced, offering flexibility in welding thickness to meet a range of design specifications. In addition, the availability of multiple soldering iron tips enables designers to create patterns with varying levels of precision, facilitating the fabrication of intricate designs and allowing for tailored solutions to specific project requirements.

The shape of the tip is crucial for ensuring smooth adherence to design patterns without damaging the material. For instance, sharp-edged tips can induce shear forces that misalign or tear layers along the motion path. Therefore, a tip with a rounded edge is preferred since it diminishes the chance to pull and rupture the

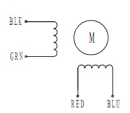


(a) Nema 17 stepper motor used in Ultimaker 3.

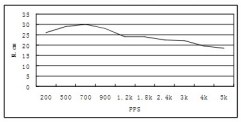
HIGH TORQUE HYBRID STEPPING MOTOR SPECIFICATIONS

General specifications		Electrical specifications	
Step Angle (°)	1.8	Rated Voltage (V)	2.8
Temperature Rise (℃)	80 Max (rated current, 2 phase on)	Rated Current (A)	1.68
Ambient Temperature (℃)	-20~+50	Resistance Per Phase (±10% Ω)	1.65
Number of Phase	2	Inductance Per Phase (±20% mH)	3.2
Insulation Resistance (MΩ)	100 Min (500VDC)	Holding torque(N.cm)	36
Insulation Class	Class B	Detent Torque(N.cm)	150
Max.radial force (N)	28 (20mm from the flange)	Rotor Torque (N.cm)	54
Max.axial force (N)	10	Weight (kg)	0.28

Wiring Diagram :



PULL out turque curve :
VOLTAGE: 24VDC CONSTANT CURRENT : 1.68A HALF STEP



(b) Technical features of the Nema 17 stepper motor used in Ultimaker 3.

Fig. 17: The stepper motor employed in UM3.

material and ensures smooth movement. Besides, a rounded-edge tip allows for gentle contact with the material, minimizing unintended forces and ensuring optimal performance.

The original soldering iron tip shown in Figure 14 has a sharp head and small diameter of 0.5 $[mm]$ therefore, it necessitates implementing a new soldering iron tip design with a flat surface and rounded edges as illustrated in Figure 15. A tip with flat surface ensures uniform contact between the soldering iron and the material layers resulting in uniform heat transfer through the sealing thickness to the materials and improved sealing quality.

To allow using different sealing thicknesses, the new soldering iron heating element and tip are designed to interlock precisely, akin to the fit between a bolt and a nut as shown in Figure 16. This approach permits multiple iron tips to be used with a single heating element, thereby decreasing material consumption. Likewise, the bolt-and-nut design ensures a stable connection between the soldering iron rod and tip, enhancing heat transfer due to the larger contact area.

Another important consideration in tip design is the material selection. The material used for the soldering iron tip should be chosen to minimize the task execution time and prevent large heat loss to the surroundings. while a low thermal conductivity material increases the task completion time, a material with very high thermal conductivity will excessively consume heat. Therefore, a relatively low thermal conductivity material is chosen to limit the heat transfer to the contact area of the tip and reduce undesired heat loss with the ambient air. Hence, the three soldering iron tips are made of a steel rod. Stainless steel, with a thermal conductivity of $15\frac{W}{m \cdot K}$ [38], is significantly efficient and ensures fast heat transfer to the material for improved dwell-time.

G. Parameter tuning

To optimize the machine's performance it is important to critically tune the parameters that play a major role in the criteria that define the performance. Therefore, this section aims to clarify the reasons behind the chosen parameters. It is important to mention that three parameters "sealing temperature", "dwell time" and "vertical pressure" are the most important ones that define the final performance of the machine. Accordingly, these parameters will be chosen based on the evidence to ensure optimum performance. Hence, tuning these parameters will mostly be based on the work provided by Cheng et al. (2007). [37].

1) Steps per Millimetre: The number of steps per millimetre (SPM) for each stepper motor is

a key parameter that affects the machine's performance and accuracy. Therefore, having a good understanding of the stepper motor type and tools used to transfer the rotational movement of the motors into linear motion such as belts and pulleys is necessary. Moreover, to correctly tune the number of steps per millimetre, the following equation has been used:

$$SPM = \frac{SPR \times MS}{MPR} \tag{2}$$

SPR refers to steps per revolution, *MS* to micro steps and *MPR* to millimetres per revolution.

It is important to note that UM3 employs Nema 17 stepper motors, depicted in Figure 17(a). These motors feature a full step size of 1.8°, requiring 200 steps to achieve a complete revolution. Detailed characteristics of the Nema 17 stepper motors are provided in Figure 17(b).

With the type of belts and pulleys used in UM3, it takes 5 steps for the machine to move the soldering iron 1 $[mm]$ in the XY direction. Which gives a total of $\frac{200}{5} = 40$ $[mm]$ movement for one revolution. Finally, the number of micro-steps used in this machine equals 8. Using these values the steps per millimetre in the XY-plane can be determined and is equal to:

$$SPM = \frac{200 \times 8}{40} = 40$$

Similarly, we can define the SPM in the Z-direction. Since the Z-axis uses lead-screw instead of a belt and pulley, it takes 8.5 MPR. Also, the number of micro-steps for the Z-stepper motor equals 16. Hence, we obtain:

$$SPM = \frac{200 \times 16}{8.5} = 378.5$$

2) Sealing temperature: The temperature at which sealing occurs is paramount for achieving high sealing strength. While no singular temperature guarantees robust material binding, a strong bond is typically achieved within a specific temperature range. Through experiments, Cheng et al. (2007). [37] demonstrated that this temperature range extends from below the melting point to several degrees above it. Figure 18 provides a visual representation of how temperature variation influences sealing strength across different materials, offering valuable insights into the optimal temperature parameters for effective sealing.

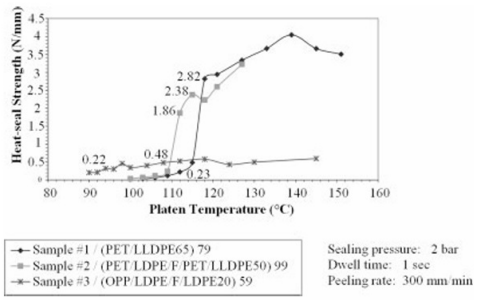


Fig. 18: The change in the sealing strength as a function of the sealing temperature for three materials. Sample #1 and #2 are linear low-density polyethylene with 104.6° melting point. Sample #3 is low-density polythene with a melting point of 88.9°.[37]

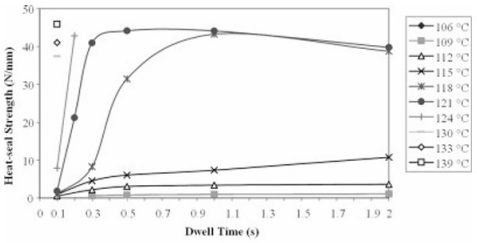


Fig. 19: Heat-sealing strength at different temperatures as a function of dwell time. The heat-sealing strength increases significantly for a dwell time of $1[s]$ and remains almost constant. Another important observation is that for the temperature below the melting point $(106-115\ ^\circ)$ dwell time has almost no effect on the sealing-heat strength, as this would be expected since the materials are not melted enough to bind together and as a result insufficient heat-sealing stress is achieved. [37]

3) Dwell time and sealing speed: As stated in [37] dwell time is the time duration that the material is in direct contact with the heat-sealing rod. I.e., the welding time. Dwell time is a function of sealing temperature. I.e., as the sealing temperature increases the dwell time decreases. An optimum dwell time is significant for achieving a highly qualified heat-sealing strength. As proven by Cheng et al. (2007) [37] and demonstrated in Figure 19 the most efficient dwell time is around $1[s]$ around the melting point temperature. This means that the velocity at which the machine should move the heating rod is constrained by the dwell time to achieve the best performance.

Despite the differing materials used in [37], Figures 18 and 19 can act as a reference for comparison, supporting the validity of the results to be obtained in this study

4) Vertical pressure: A steady contact between the rod and the material must be maintained to ensure a proper heat flow rate from the sealing rod to the material. However, as shown in Figure 20 increasing the pressure does not have a large effect on the heat-sealing strength [37], an optimum vertical pressure is needed to press the sealing rod on the material to reach sufficient heat flux from the rod and the material layers.

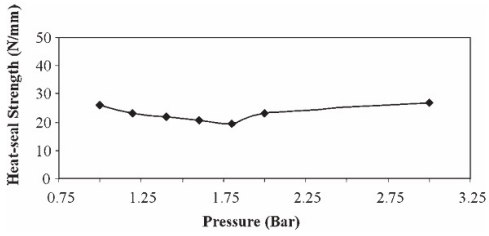


Fig. 20: Vertical pressure effect on the heat-sealing strength. This graph shows that vertical pressure does not considerably affect the heat-sealing stress [37].

To obtain the vertical pressure, first, the spring force used in the sealing rod frame should be calculated as follows:

$$F = KZ \tag{3}$$

Where K is the stiffness of the spring and Z is the length change of the spring. Next, the pressure can be computed as the force divided by the soldering iron tip's area (A). That is:

$$P = \frac{F}{A} \tag{4}$$

Because the soldering iron has a circular tip, its area can be calculated using the tip diameter (d):

$$A = \frac{\pi d^2}{4} \tag{5}$$

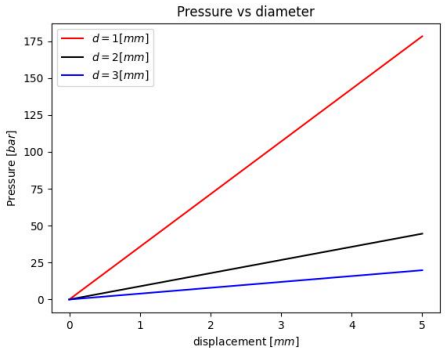


Fig. 21: Vertical pressure vs displacement. The vertical pressure on the material increases with displacement. Also, the pressure increases noticeably with the decreasing tip diameter. Therefore, to maintain an optimum pressure, the vertical displacement of the spring used in the soldering iron holder frame should be altered for different tip diameters.

Figure 21 illustrates the pressure difference for each tip diameter. As expected, as the contact diameter increases, the pressure from the spring in the soldering iron holder frame on the machine's plate (or vice versa) decreases. This Figure suggests that to maintain the desired optimum pressure, the $Z_{contact}$ in equation 1 should be adjusted depending on the tip diameter.

III. Method

To assess the machine's operational capabilities, three distinct tests will be performed. These tests

aim to offer thorough insights into different aspects of the machine's performance and functionality. By examining specific operational components, a complete evaluation of the machine's overall efficiency and effectiveness will be achieved.

The first series of tests will focus on the fabrication of an origami design pattern, aimed at demonstrating the machine's capacity to create origami structures. This test will underscore the machine's ability to execute detailed and intricate design patterns accurately. The second test will be dedicated to the production of a glove, intended to display the machine's proficiency in manufacturing gloves using the provided materials. This will showcase the machine's practical application in producing functional and wearable items, emphasizing its versatility and efficiency in diverse manufacturing processes. The final test will involve evaluating the machine's technical performance by assessing its path-following precision, the quality of the seals for secure joining of material layers, and its capability to process various material types, such as plastic and elastic materials. This evaluation will provide detailed insights into the machine's accuracy and material handling capabilities.

Conducting the first test aims to demonstrate the machine's competence in creating origami structures, and the second test will verify its capability to fabricate gloves. The third test will illustrate the machine's ability to generate reliable sealing lines, sufficient to achieve the necessary pneumatic force for finger bending. The focus will be on evaluating the integrity and consistency of these seals to ensure that they fulfil the operational requirements for pneumatic functionality.

Collectively, successful completion of these tests will confirm the machine's proficiency in integrating an origami design pattern within a glove, which is essential for the production of origami assistive gloves. Furthermore, these evaluations will offer a detailed understanding of the machine's capabilities and will help identify areas that may require improvement or optimization. This comprehensive evaluation process will be instrumental in enhancing the machine's performance and ensuring its adaptability for a wide range of manufacturing applications.

A. Origami structure fabrication

This test is designed to thoroughly assess the machine's capability to produce origami designs that exhibit dynamic behaviour, transforming into an altered state upon activation and reverting to its initial state when deactivated. To conduct this evaluation, three specific origami design patterns, illustrated in Figure 22, have been selected for welding onto two plastic layers. The selection of these patterns allows for the examination of a range of design complexities and folding mechanisms. After securely sealing the patterns on the plastic layers, the structures will undergo activation using a pneumatic pressure pump.

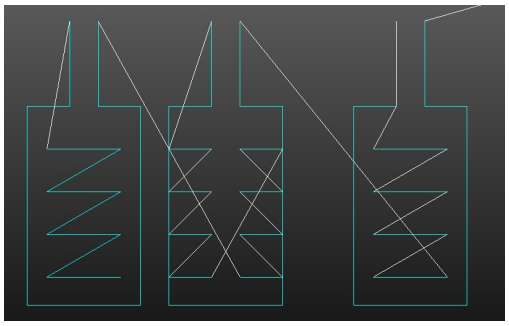
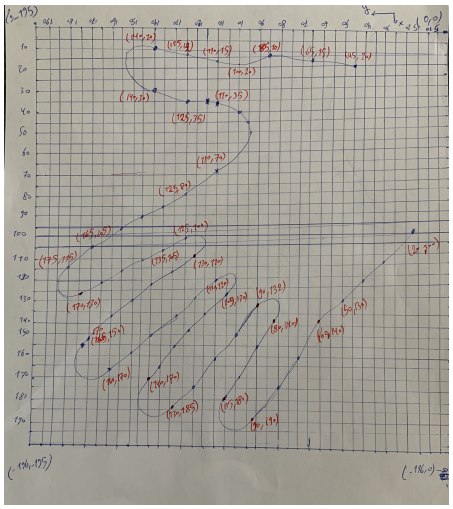


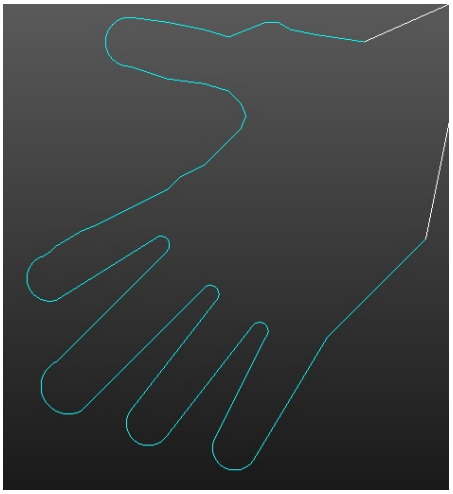
Fig. 22: Origami design patterns to assess the power of the machine in generating origami structures. Left: *zigzag* pattern. Middle: *sideline pattern*. Right: *midline* pattern.

This process is crucial for testing the functionality and precision of the machine in creating origami designs that perform as intended under pneumatic actuation.

The G-code file (Appendix I) required for the generation of the origami design patterns was meticulously written using the *Notepad++* software. The first design pattern, identified as the *zigzag* pattern, is carefully situated in the middle of a square block to enable a distinct folding mechanism. The second design termed the *sideline pattern*, features creases along both sides



(a) Hand trajectory generated on an A4 paper. The paper is converted into a grid of 5 *[mm]*, a hand is placed on the paper, and the hand trajectory is drawn. Specific locations are marked in red to convert into G-code for the machine to seal two layers of plastic for fabricating a glove as proof of concept.



(b) The resulting hand trajectory after converting into G-code. This trajectory serves only as a proof of concept example to demonstrate the ability of the machine to fabricate gloves. The corresponding G-code is provided in Appendix J.

of a square block, thereby illustrating the folding properties along its edges. The third pattern, named the *midline pattern*, incorporates crease lines centrally located within the structure, aimed at demonstrating the middle folding mechanism. All patterns have been designed to include an opening to facilitate airflow, which is crucial for pneumatic actuation. The selection of these three patterns is intended to showcase the machine's versatility in fabricating various origami designs and their respective folding behaviours. It is significant to mention that the square blocks have been dimensioned to represent the size of a finger, thereby ensuring the practical applicability of the designs.

B. Glove fabrication

As the machine's final goal is to create origami assistive gloves, it is important to confirm that it is qualified to seal two material layers to fabricate a glove. To do this, a hand trajectory is required. As depicted in Figure 23(a), the hand trajectory is delineated using an A4 paper. Initially, a grid is established on the paper, with a grid size of 5 [*mm*], enhancing the accuracy of the drawn hand trajectory. Subsequently, the trajectory is drawn with a hand positioned on the gridded paper. Following this, specific coordinates (in red) from the drawn hand trajectory are picked for conversion into G-code (Appendix 23(b)) for the machine to execute.

The objective of this test is to determine the machine's glove-making capability. Accordingly, the glove's quality is not a major factor. To create a more visually appealing glove, a more precise procedure can be implemented.

C. Technical performance

1) Precision test: The first test will assess the machine's precision and smoothness in executing predefined paths. This assessment is essential for determining the machine's ability to accurately follow instructions and maintain consistency in its movements. By analyzing how well the machine adheres to specified trajectories, we can gain valuable insights into its overall accuracy and reliability. Moreover, this test aims to show that the machine meets the precision requirement of being able to execute any trajectory with an error

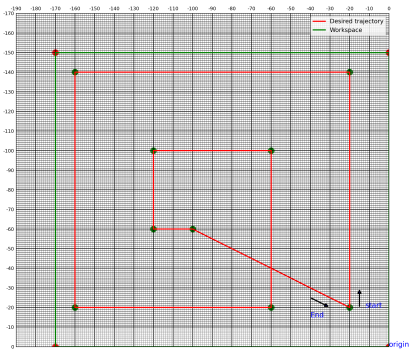


Fig. 23: The method to assess the precision of the machine in executing pre-defined trajectories. The trajectory is made in Python programming language (Appendix C) to ensure high precision and prevent any human-made error. Furthermore, the trajectory is made compatible with A4 paper size due to the high accessibility of A4 paper printer machines. Also, the paper is girded with a grid size of 1 $[mm]$ to enhance error identification. Specific locations (green dots) are marked as comparison points to detect possible errors.

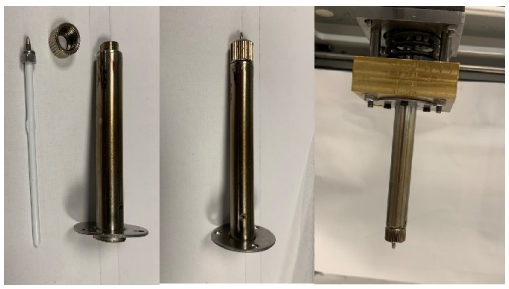


Fig. 24: The visualization of the steps to convert the machine from a sealing into a plotter machine. The sealing rod is replaced with a pen to plot a trajectory on paper instead of sealing on a material. This conversion simplifies the comparison process for error detection as the machine path can in the new setup directly be drawn on or next to the actual trajectory. An advantage of this is that it eliminates human error from the comparison process as there is no need for a human to align the machine path with the desired path. In conclusion, a pen-plotter machine will enhance error detection significantly.

smaller equal to 0.5 $[mm]$ accuracy.

To start the first test, a trajectory will be drawn on paper and converted into G-code for FlexiFabricate to execute. Once the machine plots the trajectory, the machined path will be compared to the original drawing. This comparison will assess the precision of FlexiFabricate's movements, reveal any deviations from the intended path, and visually demonstrate its accuracy. Although it is possible to manually draw the trajectory shown in Figure 23, the Python programming language (Appendix C) will be used. Mainly, because a programming language will prevent any human-made error and ensure high precision is retained. Furthermore, a conversion from pixel to millimetre is used to draw the trajectory in real-world dimensions and ensure the drawing is compatible with A4 paper. The choice of making the dimensions compatible with A4 paper size is due to the high availability of A4 paper printer machines to print the trajectory for comparison with the FlexiFabricate executed trajectory. Moreover, a 2d grid with a grid size of $1[mm]$ is drawn on the A4 paper to simplify error detection. Also, specific locations (green dots) are chosen as the measurement points to check the precision.

It is important to note that to perform the precision test first, the machine is turned into a pen-plotter machine as shown in Figure 24. To convert the FelxiFabricate into a plotter machine, the soldering iron tip is changed with a pen. The conversion to a plotter machine considerably improves the trajectory comparison process as the actual trajectory and machine path can now be plotted on the same paper and visually compare the results. Moreover, a pen-plotter machine reduces the duration time for performing the precision test as it can directly plot the trajectory on or next to the actual trajectory on the same paper compared to using heat sealing to seal two material layers and subsequently align the sealing lines with the printed trajectory on an A4 paper. On account of this, a pen-plotter machine

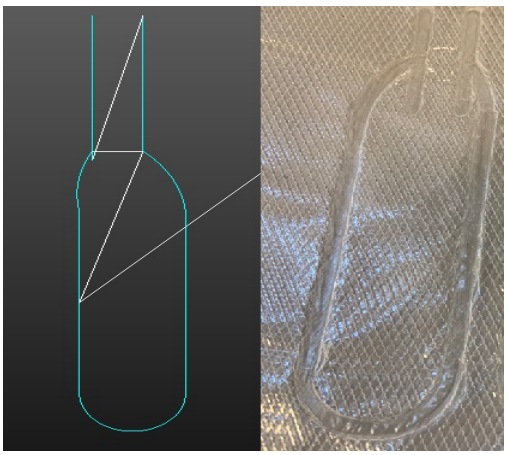


Fig. 25: The cylinder shape seals two layers to each other and allows a pneumatic pressure test for determining the sealing strength. The figure on the left is made in Notepad++ for better visualization and the image on the right is a machine-made example.

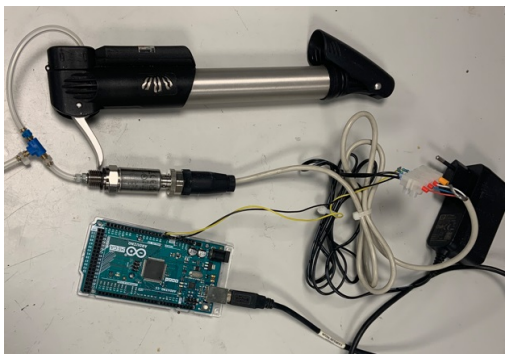


Fig. 26: Sealing quality measurement. A bike pump, pressure sensor and Arduino Mega Rev3 controller are used to measure and record the values from the pressure sensor. To obtain and store the values a Python (Appendix E) python and a C++ (Appendix F) script were used.

expunges the chance of human-made error as the comparison process can be done automatically.

Next, The trajectory shown in Figure 23 is converted to the G-code (Appendix G) and subsequently, the machine was tasked to draw the trajectory on the same paper for error detection. To draw the trajectory on the same paper, the zero location of the machine in XY-plane was aligned with the origin of the paper with Z-level at $Z_{contact} - 1$. Then, the machine is commanded to move to the red dots along the green lines in the figure 23. After aligning the pen at each red-dots a clamp is used to fix the paper on the build plate to avoid dislocation of the paper with the workspace.

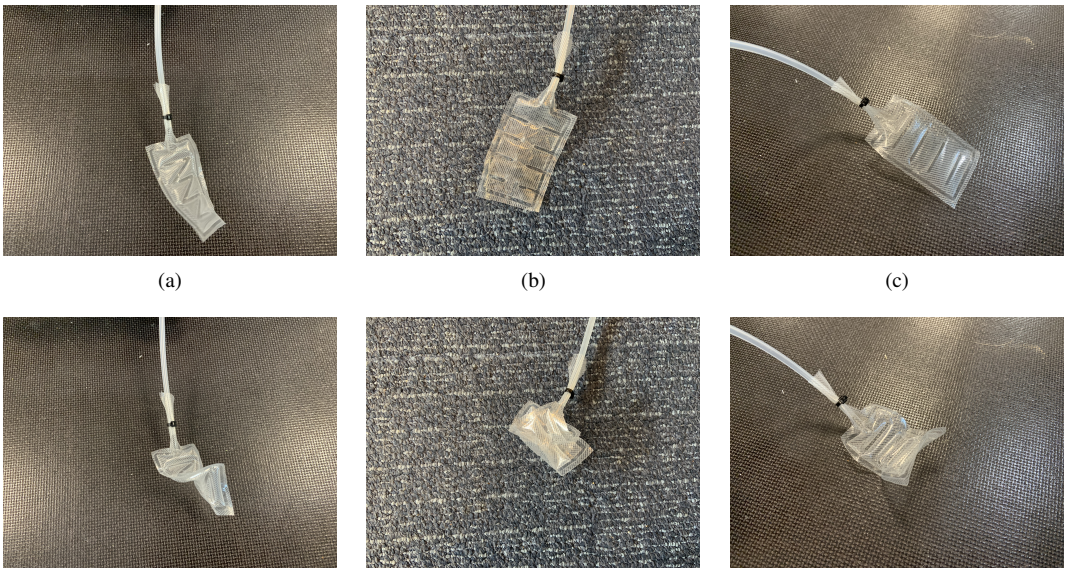
Finally, to evaluate the machine's precision, the evaluation metric will determine the *Root Mean Squared Error (RMSE)* between the actual and machine path at the specified locations.

2) Sealing quality γ: The second evaluation will concentrate on the machine's welding quality following the precision test. This assessment is particularly crucial for applications involving pneumatic or hydraulic actuators, where the integrity of welds plays a critical role in ensuring operational efficiency and safety. By examining the quality of sealing lines produced by the machine, we can verify whether materials are joined securely and reliably, meeting the requisite standards for subsequent use.

$$\gamma\ [bar]$$
$$\Big|$$
$$d = i \text{ for } i = 1,2,3\ [mm]$$
$$\Big|$$
$$T = j \text{ for } j = 200,250,300\ ^{\circ}C$$
$$\Big|$$
$$Z = k \text{ for } k = 1,2,3\ [mm]$$
$$\Big|$$
$$V = l \text{ for } l = 20,50,100,200,300\ [\tfrac{mm}{min}]$$

Fig. 27: The tree shows the different settings that the machine will be assessed for to analyse the heat sealing quality. Because the pressure is a linear multiplication with the machine's plate level, only Z-displacement is shown here instead of the pressure.

To measure the sealing line quality, the machine



(a) (b) (c)

(d) (e) (f)

Fig. 28: Origami patterns in unfold and folded state. The *zigzag* pattern shows a twisting motion when activated. The *sideline* pattern and the *midline* patterns result in the bending of the structure. a) unfold zigzag pattern, b) unfold *side line* pattern, c) unfold *mid line* pattern, d) folded *zigzag* pattern, e) folded *sideline* pattern, f) folded *midline* pattern.

will be tasked with sealing two layers of material together to make a cylinder shape shown in Figure 25. Subsequently, a pneumatic pump will blow air into the sealed cylinder. In addition, an Arduino Mega 2560 REV controller board and a pressure sensor will be used to record the values from the pressure sensor and store them for further analysis. A complete view of the pressure test setup is shown in Figure 26.

The Python (Appendix E) and C++ (Appendix F) programming languages were used to communicate with the controller and read the values from the sensor. Also, data were collected as *.CSV* file to compare the results obtained under different machine settings.

Furthermore, the sealing quality test will be performed under various settings to analyse, and compare the performance of the machine for each condition and find an optimum state that results in the best sealing quality. These settings will cover the heat sealing thickness, the sealing speed, temperature, pressure, and dwell time. I.e., the sealing quality will be assessed for different sealing iron tip diameters, speeds, temperature, and sealing pressure as shown in Figure 27. In addition, every test set will be performed 2 times to limit bias, reduce the risk of errors and possibly eliminate the potential for mistakes. Finally, the average outcome of each test set will be calculated to increase the validity of the results.

3) Material test: Lastly, the third test will focus on evaluating the machine's performance across a range of different materials. This assessment will provide valuable data regarding the machine's versatility and adaptability in handling diverse fabrication scenarios. By subjecting the machine to various materials, we can assess its ability to consistently deliver high-quality results across different material properties.

Since a material with plastic properties was

used to perform the sealing quality test, the material test will assess the ability of the machine to seal elastic-type materials. Therefore, the TPU (Thermoplastic Polyurethane) will be selected for assessment. Mainly because the TPU's inherent flexibility and widespread use in origami robots render it an apt candidate for testing the machine's performance across varied material properties.

IV. Result

A. Origami structure fabrication

Figure 28 provides a visual representation of the three design patterns in their respective unfolded and folded configurations. It is noted that the *zigzag* pattern tends to exhibit rotational movement when subjected to bending forces, whereas both the *sideline* and *midline* patterns result in the bending of the structures without rotation. All patterns are designed to fold along their specified crease lines. The test outcomes validate the machine's ability to fabricate origami structures that are functional in bending the fingers, thereby confirming its effectiveness for applications involving precise and controlled bending mechanisms.

Fig. 29: The produced glove by the machine. The machine can fabricate custom-sized gloves to meet the individual's needs.

B. Glove Fabrication

Figure 29 presents a comprehensive view of the glove fabricated by the machine. The glove is shown to be securely sealed, which ensures its effectiveness for use in pneumatic activation. The size of the glove matches exactly with the specifications outlined in the corresponding G-code, thereby validating the machine's ability to fabricate gloves with accurate dimensions. This result successfully demonstrates the machine's competence in glove production. Additionally, since the machine is capable of plotting any trajectory on the provided material layers, it can produce gloves in various sizes, thus enabling the customization of gloves to fit specific size requirements.

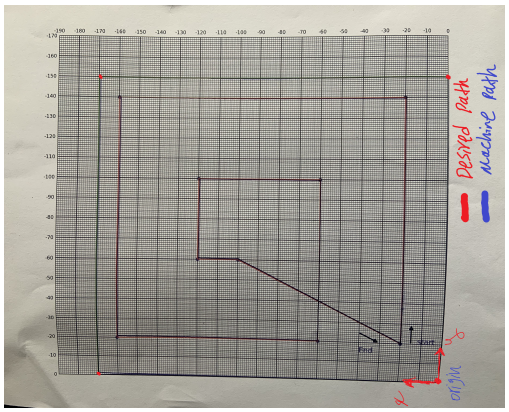


Fig. 30: Visual demonstration of the difference between the machine and the desired path. The result shows that the machine can precisely follow any trajectory. Also, there is no systematic error such as vibration meaning that the machine smoothly tracks a trajectory.

C. Technical performance

It is important to mention that the results obtained for the technical performance test belong to the pressure of 1 $[mm]$ plate displacement. This is because the stepper motors used in the structural backbone of FlexiFabricate are mainly suitable for 3d printing tasks where the stepper motors do not need to overcome the shear force exerted by the movement of the soldering iron rod tip while it is in direct contact with the material. Therefore, the stepper motors are not

able to generate enough force and pressure values bigger than the value obtained for $Z = 1$ $[mm]$ are not examined in this paper.

1) Precision test: Figure 30 offers a visual comparison between the intended path and the path achieved by the machine. The machine's actual trajectory closely mirrors the desired path, indicating a strong alignment with the planned design. This close correspondence highlights the machine's proficiency in accurately following the prescribed path, thereby validating its capability to execute complex trajectories with precision.

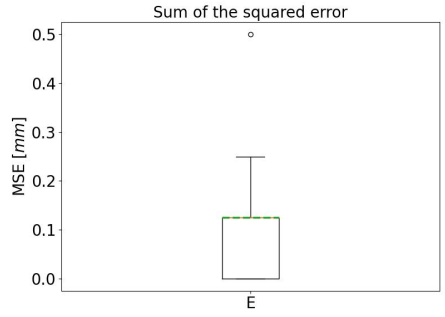


Fig. 31: Mean Square Error between the actual and machine trajectory. The machine shows a root mean square error equal to 0.35 $[mm]$ that is smaller than the required precision.

To gain a more nuanced understanding of the machine's accuracy concerning the desired path, the Root Mean Square Error (RMSE) of the discrepancy between the actual and intended trajectories is utilized as a comparative metric. Figure 31 illustrates that the machine's precision, with an RMSE of 0.35 mm, exceeds the required precision threshold of $RMSE \leq 0.5mm$. This finding indicates that the machine demonstrates a high level of accuracy in following trajectories, suggesting its capability to effectively and precisely execute any origami-designed pattern.

In addition, a detailed comparison along each axis, specifically the X and Y axes, is conducted to assess the machine's directional accuracy. As depicted in Figure 32, the machine displays a comparable level of precision along both the

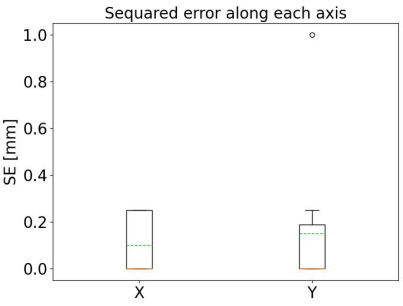


Fig. 32: Square Error between the actual and machine trajectory along each axis. According to the figure the machine has a similar precision along the X-direction and the Y-direction. The dashed lines show the mean value of the error in each direction.

X-axis and Y-axis. This observation highlights the machine's consistent accuracy across both directional dimensions, confirming its ability to perform with equal precision in multiple axes.

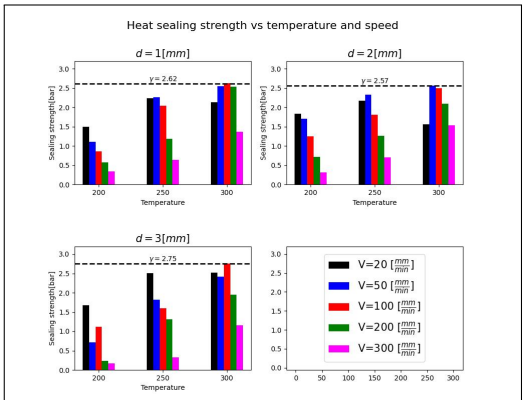


Fig. 33: Sealing strength for different conditions. At lower temperatures sealing quality degrades with increasing speed. on the contrary, at higher temperatures, faster task execution improves the sealing strength. Overall, a greater heat sealing strength is obtained at higher temperatures. In addition, the sealing strength is independent of the sealing line thickness.

2) Sealing quality: Figure 33 illustrates the relationship between heat sealing strength and the variables of temperature and speed. The graph shows that, with a constant sealing thickness, the strength of the seal is substantially affected by both the sealing speed and temperature. This indicates that variations in these parameters can lead to significant changes in the sealing quality, highlighting the importance of optimizing both temperature and speed to achieve the desired sealing strength.

For instance, when the tip diameter is set to $d = 1\,\text{mm}$ and the temperature is maintained at $T = 200°\text{C}$, it is observed that the sealing strength diminishes as the sealing speed increases. Similarly, at a higher temperature of $T = 250°\text{C}$, the sealing strength shows a decrease as the speed is varied from 20 to 300 $[\frac{\text{mm}}{\text{min}}]$. Conversely, at an elevated temperature of $T = 300°C$ the sealing quality initially improves with increasing speed from 20 to 100 $[\frac{mm}{min}]$ but then deteriorates at higher speeds. Additionally, it is noted that, irrespective of the sealing thickness, the sealing strength exhibits a general increase with rising temperature from 200 to $300°C$. This indicates that higher temperatures lead to stronger material bonding, thereby enhancing the overall quality of the seal.

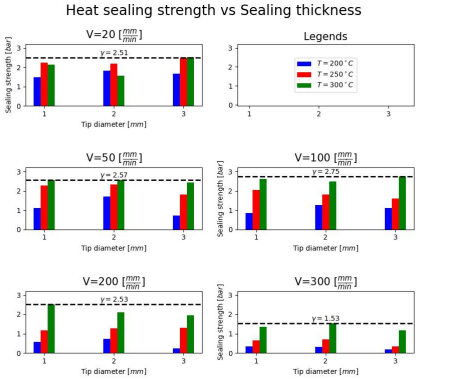


Fig. 34: Sealing strength vs sealing line thickness. The figure shows that the sealing thickness does not significantly impact the sealing quality.

Figure 34 presents an analysis of how sealing thickness impacts the quality of the seal. A significant observation from this analysis is that variations in sealing thickness have a negligible effect on sealing strength. Specifically, increasing the sealing line thickness from 1 to 3 $[mm]$ does not produce a noticeable change in the strength of the seal when both temperature and speed are maintained constant. This observation effectively distinguishes the dominant roles of temperature and speed in affecting sealing quality, while indicating that sealing thickness has a comparatively minor effect on the strength of the seal.

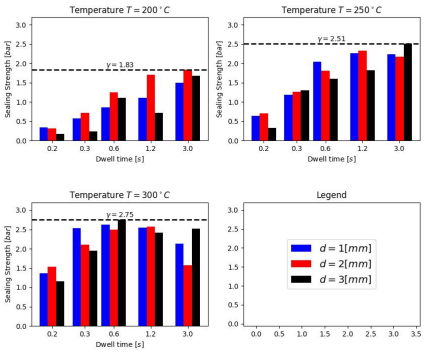
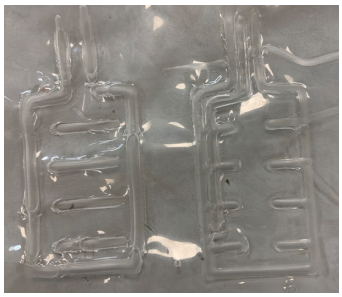


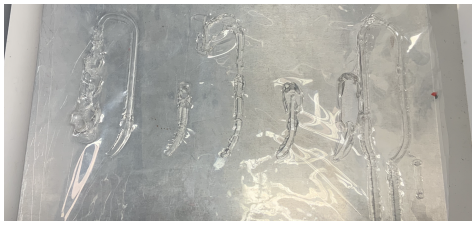
Fig. 35: Dwell-time for different sealing thickness and temperature. The sealing quality increases with increasing dwell time from 0.2 to 3 $[s]$ at temperatures $T = 200$, and $T = 250°C$ for all sealing iron tip diameters. However, at sealing temperature $T = 300°C$ the maximum sealing strength is achieved at dwell time of $0.6[s]$. In conclusion, dwell time decreases with increasing the temperature regardless of the sealing thickness.

Figure 35 illustrates the effect of dwell time on sealing quality for three different tip diameters across various temperatures. t is evident that, for temperatures of $T = 200$ and $T = 250°C$ an increase in dwell time leads to an enhancement in sealing quality across all tested sealing thicknesses. In contrast, at $T = 300°C$ the sealing strength reaches a maximum at dwell time of 0.6 $[s]$. The data also highlight an interdependence between dwell time and temperature, with dwell time being inversely related to temperature.



(a) Testing of materials: The device can effectively seal elastic-type materials like TPU. Using a protective cover, as illustrated in Figure 11, is essential to safeguard the layers from rupturing during the sealing process. This test was performed at $T = 300°$, $V = 10[\frac{mm}{min}]$, $d = 2[mm]$, *and* $Z = 1[mm]$ *above* $Z_{contact}$.

(b) The midline pattern in folded state sealed on TPU layers.



(c) Testing the machine's ability to seal two layers of TPU material. Due to the elastic behaviour of TPU, the material attaches to and stretches with the soldering rod resulting in rupture of the material if the material layers and soldering iron tip are in direct contact with each other.

This indicates that as temperature increases, the required dwell time decreases to achieve optimal sealing quality.

To summarize, both speed and temperature are identified as crucial factors that significantly affect the quality of sealing. The results indicate that at higher temperatures, it is feasible to achieve similar heat sealing strength at increased speeds, which translates to shorter dwell times. This is an important observation as it can lead to reduced processing times, thereby improving task efficiency. Furthermore, the findings reveal that variations in sealing thickness do not significantly impact the quality of the material binding during the heat-sealing of plastic materials. This highlights that while speed and temperature are key determinants of sealing performance, the thickness of the sealing line does not have a substantial effect on the binding quality.

3) Material test: The material test aimed to determine the machine's effectiveness with different materials, with Thermoplastic Polyurethane (TPU) being the focus of the final evaluation. Elastic materials differ from plastic-like ones in that they stretch under pressure when the soldering iron presses and moves, which can cause rupturing. To avoid this, it is important to add a thin layer of non-stretchable and non-sealable material over the elastic layers. Moreover, extending the dwell time is essential for achieving optimal results with elastic materials.

Figure 36(a) demonstrates that, under suitable conditions, the machine can effectively seal both plastic layers and more challenging materials such as TPU, which have natural elasticity. Conversely, as shown in Figure 36(c), sealing with the soldering iron tip in direct contact with TPU layers can lead to the rupture of these elastic materials.

In summary, the machine demonstrates exceptional capability in sealing layers of plastic materials, achieving a binding strength that surpasses the inherent strength of the materials themselves. However, challenges persist when attempting to bind two elastic materials using the sealing technique. The primary issue arises from the elastic properties of the materials, which cause them to stretch and shift in response to the movement of the soldering tip. This behaviour impairs the sealing process, leading to difficulties in achieving a reliable and consistent bond between elastic materials. Nevertheless, these challenges can be addressed by incorporating an additional material layer to shield and protect the elastic layers.

V. Discussion

The objective of this paper is to introduce a novel fabrication technique aimed at producing origami assistive gloves. This method integrates 3D printing technology with a 2D welding approach to create intricate origami design patterns. The fabrication process involves the machine applying two layers of material and sealing the specified pattern onto these layers. The sealing operation is executed using a specialized soldering iron station, which features a custom-designed soldering iron tip and an accompanying soldering iron holder frame. This combination of technologies is intended to optimize the creation of detailed origami patterns, facilitating the development of assistive gloves with enhanced functionality.

In constructing the machine, the UliMaker3 3D printing device was utilized as the structural backbone, primarily due to its availability. While the use of this 3D printer simplifies the design process for the new fabrication method, it also imposes certain constraints. These limitations are associated with the specific capabilities and design parameters of the UliMaker3, which may impact the overall functionality and adaptability of the new fabrication technique.

Particularly, the UM3 utilizes stepper motors designed for 3D printing tasks, where the motors are not subjected to shear forces resulting from the direct contact between the soldering iron tip and the material during the sealing process. Consequently, the stepper motors in the UM3 possess a maximum torque generation capacity insufficient to overcome shear forces in the XY plane, which are exerted by the machine's plate when it moves more than $1,\text{mm}$ above the contact point between the soldering iron tip and the plate. As a result, it was not possible to analyze the heat sealing quality under varying vertical pressures.

Additionally, the FlexiFabricate uses a soldering iron to seal the material layers with direct contact between the soldering iron and the material on the plate. The UM3 Z-plate is designed to resist 3d printed objects which are in general lightweight. This means that the UM3 plate in the new function undergoes relatively large pressure at the contact point between the soldering iron tip and the plate, as a consequence, the plate buckles at the contact point resulting in an imbalance in the sealing process that degrades the smooth path-following of the machine and steady sealing lines during the sealing process.

The UM3 operates within a fixed environment, which imposes limitations on the maximum size of the origami assistive gloves. This constraint consequently restricts the range of origami design patterns that can be accommodated. Moreover, the enclosed nature of this environment presents challenges in securely fixing the material layers within the machine.

Besides, the precision test covers the machine's accuracy in the XY plane since fabricating origami design patterns is done in the 2d plane. Therefore, it is assumed here that the precision along the Z-direction is of less interest. However, owing to the fact that the micro-steps used in the Z-stepper motor equal $\frac{1}{16}$ which is more precise than micro-steps used in XY plane $(\frac{1}{8})$, it can be concluded that the machine acquires a similar or even better RMSE along the Z-axis.

The results for the sealing quality test obtained in this paper are at constant vertical pressure achieved at 1 $[mm]$ plate displacement. Therefore, the results can differ if the vertical pressure can also be modified.

The maximum sealing strength acquired in the sealing quality test was affected by the maximum pressure material layers could tolerate. Meaning that the plastic layers used to obtain the sealing strength could endure a maximum of approximately 2.75 $[bar]$. Hence, a material with stronger inherent properties can yield a higher sealing quality.

It should be emphasized that the highest sealing strength recorded in the technical performance analysis was attained through the application of concurrently increasing pneumatic pressure. The potential effects of leakage for a duration of time were not addressed within the scope of the

technical performance setup.

The UM3 controller board is replaced with an Arduino UNO, Arduino CNC Shield V3.00 and A4988 stepper motors, to steer the machine's operations in three-dimensional coordinates. Alongside this, the GRBL library and the Universal G-code Sender (UGS) are selected as the software tools to make a user-friendly interface.

To further simplify working with the machine, a Python script is provided which takes the coordinates of a 2d design pattern and converts it into the G-code, the operating language of the machine.

To thoroughly evaluate the machine's performance, three separate tests were administered. The initial test was conducted to verify the machine's ability to produce origami design patterns, thereby assessing the effectiveness of the fabrication method for intricate designs. The second test was aimed at demonstrating the machine's capability to fabricate custom-sized gloves using two layers of material, highlighting its adaptability in glove production. The final test was designed to measure the machine's technical performance, encompassing aspects such as precision and efficiency. These tests collectively offer a comprehensive assessment of the machine's operational capabilities and its effectiveness in diverse manufacturing tasks.

To fabricate origami structures, three specific design patterns—*zigzag*, *sideline*, and *midline*—were chosen. The fabrication results demonstrate that the machine can successfully seal these origami design patterns onto two material layers. Upon pneumatic activation, the structures achieve their desired folded states. Additionally, these structures are capable of returning to their initial configurations when the air pressure is released. This confirms the machine's effectiveness in creating functional and reconfigurable origami structures.

The outcome of the glove fabrication test validates the machine's capability to produce custom-sized gloves with flexibility. This adaptability is attributed to the machine's ability to interpret any 2D pattern provided as a G-code file and accurately seal it onto the material layers. This feature allows for precise customization of glove sizes, demonstrating the machine's versatility in accommodating various design specifications.

The machine shows a dimensional accuracy and precision of $0.35[mm]$, which is smaller than the required precision of 0.5 $[mm]$ in executing motions that are within the precision requirement for the machine. The root mean square error is chosen as the performance metric to assess the precision of the machine

Likewise, the results from the heat sealing quality assessment show that speed and temperature are critical parameters that can greatly impact sealing quality. Additionally, higher temperatures result in similar heat-sealing strength at faster speeds and shorter dwell times. Moreover, it has been demonstrated that sealing thickness has an insignificant impact on the quality of material binding in heat-sealed plastic. Also, the results show that the sealing line strength obtained using FlexiFabricate under certain sealing settings can yield a sealing strength equal to or stronger than the material's inherent strength. To this end, the maximum pneumatic pressure achieved in the sealing quality test equals $2.75[ba]$ which is much more than it is required to bend a finger. Therefore, it can be concluded that the proposed method can successfully fabricate origami assistive gloves.

The results from the material test indicate that heat-sealing elastic materials such as TPU remain challenging for the inherent elastic behaviour of these materials if the material is in direct contact with the soldering iron tip. However, it is also shown that if another third party material is used to prevent the TPU layers from directly contacting the soldering iron tip, the machine is able to seal TPU layers.

VI. Conclusion

In summary, this paper introduced a new method that combines the 2d plastic welding technique and 3d printing technology to fabricate origami design patterns that are fast, cost-effective, can use two material layers and have high dimensional accuracy, a path-following RMSE equal to $0.35[mm]$ in. Through experiments, it is shown that the proposed method can generate sealing lines that can withstand a pneumatic pressure equal to or greater than the inherent strength of the material individual layers. Moreover, using two layers of sealing plastic, the maximum pneumatic pressure obtained equals $2.75[bar]$ which is more than needed to flex a finger.

Because of the compactness, and inherent flexibility of the origami design, it provides a promising alternative to the existing assistive gloves to be lightweight, flexible and can be produced in custom size. Therefore, the machine aims to produce origami assistive gloves by sealing an origami design pattern onto two material layers suitable for use with a pneumatic actuator.

In conclusion, this paper proposes a new fabrication method to integrate the origami design principle with assistive gloves and produce compact, custom-sized, flexible, and lightweight prototypes for origami assistive gloves. Furthermore, through experiments, it is shown that the new fabrication method can successfully seal an origami design pattern on two layers of material suitable for use with pneumatic actuators. It is also demonstrated that the machine has the ability to fabricate custom-sized gloves.

VII. Future recommendations

As mentioned earlier, the stepper motors used in the UM3 are specifically intended for 3D printing applications, where the motors function in an unencumbered environment without having to contend with forces like friction from the machine's plate resulting from vertical pressure and material roughness. Therefore, the implementation of more powerful stepper motors with higher torque capacity would significantly enhance the machine's robustness and overall performance.

Likewise, a considerable reduction in friction within the XY-plane can be achieved by substituting the rod's contact surface with a roller ball mechanism. This approach would enable the ball to roll unimpeded while sealing the material, thereby minimizing friction and enhancing the machine's efficiency.

A notable design constraint lies in the UM3 bed, which is not constructed to withstand high levels of pressure. Accordingly, substituting the current bed with a more robust or reinforced version would substantially increase the machine's stability and the quality of the produced design patterns.

Using the UM3 frame as the structural backbone constrains the production of the maximum possible glove size since the UM3 has a fixed dimension and is made for relatively small-sized 3d printing objects. In addition, the UM3 closed environment complicates the insertion of the materials on the Z-plate. As a consequence, the material layers do not perfectly lay on the bed and exert unintended pressure on the stepper motors at certain points affecting the sealing quality. Therefore, having a structural backbone with an open environment will not only eliminate the constraint on the glove size but also considerably simplify the fixation of the material layers resulting in an enhanced performance.

By addressing these limitations, the machine would be better equipped to handle the demands of more precise applications, leading to more consistent and reliable results.

References

[1] P. Maciejasz, J. Eschweiler, K. Gerlach-Hahn, A. Jansen-Troy, and S. Leonhardt, "Jner journal of neuroengineering and rehabilitation review open access a survey on robotic devices for upper limb rehabilitation," p. 3, 2014. [Online]. Available: http://www.jneuroengrehab.com/content/11/1/3

[2] A. Wege and G. Hommel, "Development and control of a hand exoskeleton for rehabilitation of hand injuries," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2005, pp. 3046–3051.

[3] C. N. Schabowsky, S. B. Godfrey, R. J. Holley, and P. S. Lum, "Development and pilot testing of hexorr: Hand exoskeleton rehabilitation robot," 2010. [Online]. Available: http://www.jneuroengrehab.com/content/7/1/36

[4] T. Worsnopp, M. Peshkin, J. Colgate, and D. Kamper, "An actuated finger exoskeleton for hand rehabilitation following stroke," in *2007 IEEE 10th International Conference on Rehabilitation Robotics*, 2007, pp. 896–901.

[5] F. Nasrallah, A. Mohamed, H. Yap, H. Lai, C.-H. Yeow, and J. Lim, "Effect of proprioceptive stimulation using a soft robotic glove on motor activation and brain connectivity in stroke survivors," *Journal of Neural Engineering*, vol. 18, 2021.

[6] M. Liu, S. Wilder, S. Sanford, M. Glassen, S. Dewil, S. Saleh, and R. Nataraj, "Augmented feedback modes during functional grasp training with an intelligent glove and virtual reality for persons with traumatic brain injury," *Frontiers in Robotics and AI*, vol. 10, 2023.

[7] Y. Chen, J. Yan, and J. Feng, "Geometric and kinematic analyses and novel characteristics of origami-inspired structures," 2019.

[8] J. E. Suh, T. H. Kim, and J. H. Han, "New approach to folding a thin-walled yoshimura patterned cylinder," *Journal of Spacecraft and Rockets*, vol. 58, pp. 516–530, 2021.

[9] M. Chen, S. Ho, H. Zhou, P. Pang, X. Hu, D. Ng, and K. Tong, "Interactive rehabilitation robot for hand function training," in *2009 IEEE International Conference on Rehabilitation Robotics, ICORR 2009*, 2009, pp. 777–780.

[10] I. Ertas, E. Hocaoglu, D. Barkana, and V. Patoglu, "Finger exoskeleton for treatment of tendon injuries," in *2009 IEEE International Conference on Rehabilitation Robotics, ICORR 2009*, 2009, pp. 194–201.

[11] *2011 IEEE International Conference on Rehabilitation Robitics.* IEEE, 2011.

[12] M. F. Rotella, K. E. Reuther, C. L. Hofmann, E. B. Hage, and B. F. BuSha, "An orthotic hand-assistive exoskeleton for actuated pinch and grasp," in *2009 IEEE 35th Annual Northeast Bioengineering Conference*, 2009, pp. 1–2.

[13] K. Tong, S. Ho, P. Pang, X. Hu, W. Tam, K. Fung, X. Wei, P. Chen, and M. Chen, "An intention driven hand functions task training robotic system," in *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*, 2010, pp. 3406–3409.

[14] W. Yan, S. Li, M. Deguchi, Z. Zheng, D. Rus, and A. Mehta, "Origami-based integration of robots that sense, decide, and respond," *Nature Communications*, vol. 14, 12 2023.

[15] Z. Zhakypov and J. Paik, "Design methodology for constructing multimaterial origami robots and machines," *IEEE Transactions on Robotics*, vol. 34, pp. 151–165, 2 2018.

[16] H. A. Beigi and A. H. A. Stienen, "Robotic origami: Design, fabrication and actuation," 2023.

[17] Y. Wang and K. Lee, "3d-printed semi-soft mechanisms inspired by origami twisted tower," 2017, pp. 161–166.

[18] J. Jovanova, M. Anachkova, V. Gavriloski, D. Petrevski, F. Grazhdani, and D. Pecioski, "Modular origami robot inspired by a scorpion tail," vol. 2, 2018.

[19] Q. Chen, F. Feng, P. Lv, and H. Duan, "Origami spring-inspired shape morphing for flexible robotics," *Soft Robotics*, vol. 9, pp. 798–806, 2022.

[20] Y. Guan, Z. Zhuang, Z. Zhang, and J. S. Dai, "Design, analysis, and experiment of the origami robot based on spherical-linkage parallel mechanism," *Journal of Mechanical Design, Transactions of the ASME*, vol. 145, 8 2023.

[21] C. D. Onal, R. J. Wood, and D. Rus, "An origami-inspired approach to worm robots," *IEEE/ASME Transactions on Mechatronics*, vol. 18, pp. 430–438, 2013.

[22] S. Singh, G. Singh, C. Prakash, and S. Ramakrishna, "Current status and future directions of fused filament fabrication," pp. 288–306, 7 2020.

[23] H. Yap, J. Lim, F. Nasrallah, and C.-H. Yeow, "Design and preliminary feasibility study of a soft robotic glove for hand function assistance in stroke survivors," *Frontiers in Neuroscience*, vol. 11, 2017.

[24] UltiMaker, "Ultimaker3 3d printer," https://www.dimensions.com/element/ultimaker-3-3d-printer#:~:text=The%20overall%20build%20volume%20is,sophisticated%20models%20with%20intricate%20geometries.

[25] ALL3DP, "Stepper motor driver: All you need to know," https://all3dp.com/2/what-s-a-stepper-motor-driver-why-do-i-need-it/.

[26] M. Hardware, "Cnc shield," https://makerhardware.net/wiki/doku.php?id=electronics:cnc_shield/.

[27] E. Clinic, "Arduino cnc shield v3.0 and a4988 hybrid stepper motor driver + joystick," https://www.electroniclinic.com/arduino-cnc-shield-v3-0-and-a4988-hybrid-stepper-motor-driver-joystick/.

[28] GRBl, "Grbl library," https://github.com/grbl/grbl/.

[29] UGS, "Universal g-code sender," https://github.com/winder/Universal-G-Code-Sender/.

[30] SainSmart, "How to set up universal gcode sender (ugs) for windows operating system," https://docs.sainsmart.com/article/xevbm8qufa-how-to-install-universal-gcode-sender-ugs-for-windows-operating-system.

[31] NotePad++, "Notepad++," https://notepad-plus-plus.org/downloads/.

[32] Ncentic, "Notepad++ gcode plugin," https://ncnetic.com/notepad-gcode-plugin/.

[33] inkscape, "inkscape," https://inkscape.org/.

[34] Maslow, "Inkscape for gcode generation - quick instructions to get started," https://forums.maslowcnc.com/t/inkscape-for-gcode-generation-quick-instructions-to-get-started/12049/12.

[35] YouTube, "A guick guid to make gcode in inkscape," https://www.youtube.com/watch?v=JkVj2MAyj18.

[36] Bol.com, "Solder station," http://www.bol.com/nl/nl/p/velleman-soldeerstation-instelbaar-40-48-w-temperatuurbereik-150-450-c-grijs/9200000042746360/?s2a=/.

[37] S. Y. Cheng, A. Hassan, M. I. H. Ghazali, and A. F. Ismail, "Heat sealability of laminated films with lldpe and ldpe as the sealant materials in bar sealing application," *Journal of Applied Polymer Science*, vol. 104, pp. 3736–3745, 6 2007.

[38] Thermtest, "Thermal conductivity of steel," https://thermtest.com/thermal-conductivity-of-steel#:~:text=The%20thermal%20conductivity%20of%20steel,235%20W%2F(mK)%20respectively.

APPENDIX

A. Python script to convert coordinates to G-code

```
'''
    This file takes XY-coordinates stored in  .csv file
    and converts it to a gcode file.

    input_file : the file to with .csv file to be converted. E.g., "myfile.csv"
    output: .gcode file

    'Run the function general()'

'''

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


def load_data(file_path):
    # myfile = pd.read_csv(file_path)
    data = pd.read_csv(file_path).values
    data = data[28:1024, :]
    data=np.round(data/6.5,4)
    return data
def convert_to_gcode(coordinates):
    gcode_commands = []
    x,y = coordinates[0,:]
    gcode_commands.append(f"G17 ; Select XY plane for circular interpolation\n")
    gcode_commands.append(f"G21 ; set the units to milimeters\n")
    gcode_commands.append(f"G54 ; select the coordinate system 1\n")
    gcode_commands.append(f"G80 ;cancle motion\n")
    gcode_commands.append(f"G90 ; non-incremental motion\n")
    gcode_commands.append(f"G94 ;feed/minute mode\n")
    gcode_commands.append(f"G00 F1000 ; set the feed rate for fast movement\n")
    gcode_commands.append(f"G01 F100 ; set the feed rate that machine should move with\
    n")
    gcode_commands.append(f"G00 Z{-160}\n")
    gcode_commands.append(f"G00 X{x} Y{y}\n")
    gcode_commands.append(f"G01 Z{-162}\n")
    for point in coordinates[1:,:]:
        x, y = point
        gcode_commands.append(f"G01 X{x} Y{y}\n")  # Assuming linear interpolation (G01
    ) for simplicity

    gcode_commands.append(f"G00 Z{-150}\n")
    gcode_commands.append(f"G04 P{1000} ;pause the machine for 1000 ms\n")
    gcode_commands.append(f"G28 ; Home the machine\n")
    return gcode_commands

def transfer_coordinates_into_machine_workspace(coordinates):
    l = len(coordinates)
    new_coordinates = np.zeros((l,2))
    x_data = coordinates[:,0]
    y_data = coordinates[:,1]
    x_max = np.max(x_data)
    y_max = np.max(y_data)
    y_offset = 10
    x_offset = 10
    if x_max >= 0:
        x_data -= x_max
        x_data -= x_offset
    if y_max >= 0:
        y_data -= y_max
```

```
        y_data -= y_offset
    if x_max <= -200:
        x_min = np.min(x_data)
        x_exceed = x_min + 200
        x_offset = x_exceed - 5
        x_data -= x_offset
    if y_max <= -200:
        y_min = np.min(y_data)
        y_exceed = y_min + 200
        y_offset = y_exceed - 5
        y_data -= y_offset
    new_coordinates[:,0] = np.round(x_data,3)
    new_coordinates[:,1] = np.round(y_data,3)
    return new_coordinates

def plot_data(data):
    color = ['blue', 'red']
    labels = ['original', 'converted']
    j=0
    i=0
    col = np.shape(data)[1]
    fig, ax = plt.subplots(1, 1, figsize=(10, 8))
    while i!=col:
        ax.plot(data[:, i], data[:, i+1], color=color[j], label=labels[j], linewidth
    =4-(i))
        i+=2
        j+=1
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.legend()
    fig.show()

def write_file(file_path, gcode_commands):
    with open(file_path, "w") as file:
        file.writelines(gcode_commands)
    # return file
def general():
    input_data = input("Insert the input file name, e.g: mydata.csv: ")
    file_name = input_data.split('.')[0]
    data = load_data(input_data)
    # Convert NumPy array to G-code commands
    new_data = transfer_coordinates_into_machine_workspace(data)
    gcode_commands = convert_to_gcode(new_data)
    # Write G-code commands to a file
    file_path = "%s.gcode"%file_name
    write_file(file_path, gcode_commands)

    data_base = np.zeros((len(data),4))
    data_base[:,0:2] = data
    data_base[:,2:4] = new_data

    plot_data(data_base)

    print("proces finished. A G-code file (%s.gcode) is created in "
      "the same directory where the input file is."%file_name)


#general() # uncomment and run the script

'''
End of converting np.arrays to G-code
'''
```

B. Python script to check the output G-code file from Inkscape

```
'''
This file takes a gcode file from inkscape Gcode-tool result and convert it into
a suitable gcode file that is readable by the Universal G-code Sender.

input: G-code file myfile.gcode --> COPY THE FILE INTO THE SAME DIRECTORY WITH THIS
    FILE.
output: new_myfile.gcode

Hence: To convert the output of the linkscape file (which is in .NGC formate) to .gcode
open the file in "Notepad++" and rename it for example "myfile.gcode".

Make sure you type '.gcode' at the end of file name. This will make sure you do not get
     any error.
'''

import numpy as np
import convert_to_grbl as ctg
import time

def read_gcode_file(file_path):
    print('Process started...')
    with open(file_path, 'r') as file:
        f = file.read()
        f0 = f.split('\n')
        motion_commands = []
        coordinates = np.zeros((len(f0), 6)) # X,Y,Z,I,J,F
        f1 = []
        ii=0
        for line in f0:
            if line != '':
                f1.append(line)
            ii+=1
        i=0
        for line in f1:
            command = line.strip()
            command = command.split(' ')
            # print(command)
            if command[0].startswith('%') or command[0].startswith('('):
                pass
            elif command[0].startswith('M') or command[0]=='G21':
                pass
            else:
                # if command[0]=='G00' or command[0]=='G01':
                if command[1].startswith("Z"):
                    motion_commands.append(command[0])
                    coordinates[i,2] = round(float(command[1][1:]),4)
                elif command[1].startswith("X"):
                    motion_commands.append(command[0])
                    L = len(command[1:])
                    for j in range(2):
                        if L>3:
                            if command[4].startswith('F'):
                                coordinates[i, j] = round(float(command[j + 1][1:]), 4)
                            else:
                                coordinates[i,j] = round(float(command[j+1][1:]),3) #
    coordinate
                                coordinates[i, j+3] = round(float(command[j + 4][1:]),
    4)
                        else:
                            # strToint = str.
                            coordinates[i, j] = round(float(command[j + 1][1:]), 4)

                i+=1
```

```
        coordinates = coordinates[:i,:5]
    return motion_commands, coordinates

def write_proper_gcode(motion_commands, coordinates):
    gcode_commands = []
    # x, y, z = coordinates[0, :3]
    gcode_commands.append(f"G17 ; Select XY plane for circular interpolation\n")
    gcode_commands.append(f"G21 ; set the units to milimeters\n")
    gcode_commands.append(f"G54 ; select the coordinate system 1\n")
    gcode_commands.append(f"G80 ;cancle motion\n")
    gcode_commands.append(f"G90 ; non-incremental motion\n")
    gcode_commands.append(f"G94 ;feed/minute mode\n")
    gcode_commands.append(f"G00 F1000 ; set the feed rate for fast movement\n")
    gcode_commands.append(f"G01 F100 ; set the feed rate that machine should move with\
    n")
    gcode_commands.append(f"G02 F100 ; set the feed rate that machine should move with\
    n")
    gcode_commands.append(f"G03 F100 ; set the feed rate that machine should move with\
    n")
    i=0
    for point in coordinates[:, :]:
        G0 = motion_commands[i]
        x, y, z, I, J = point
        if z==0:
            if G0 == 'G00':
                gcode_commands.append(f"{G0} X{x} Y{y}\n")
            elif G0=='G01':
                gcode_commands.append(f"{G0} X{x} Y{y}\n")
            elif G0=='G02':
                gcode_commands.append(f"{G0} X{x} Y{y} I{I} J{J}\n")
            else:# G0 == 'G02':
                gcode_commands.append(f"{G0} X{x} Y{y} I{I} J{J}\n")
        if z != 0:
            gcode_commands.append(f"{G0} Z{z}\n")
        i+=1
    gcode_commands.append(f"G28 ; Home the machine\n")
    return gcode_commands

def main():
    ts = time.time()
    file_path = input('Provide the input .gcode file: ')
    # file_path = 'sideline_1.gcode'
    motion_commands, coordinates = read_gcode_file(file_path)
    coordinates[:,:2] = ctg.transfer_coordinates_into_machine_workspace(coordinates
    [:,:2])
    gcode_commands = write_proper_gcode(motion_commands=motion_commands, coordinates=
    coordinates)
    file_name = file_path.split('/')[-1]
    file_name = file_name.split('.')[0]
    file_path = "new_%s.gcode" % file_name
    ctg.write_file(file_path, gcode_commands)

    te = time.time()

    t = round(te - ts,2)

    print('Process finished in %s seconds.'%t)
    print('File has been converted. Check the file using Notepad++ as instructed in the
     paper.')
```

C. Python script for precision test

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

def save_csv(coordinates, filename=''):
    # coordinates = coordinates.reshape(len(x_coor), 2)
    df = pd.DataFrame(coordinates)
    # df = pd.DataFrame(list(coordinates.items()), columns=['Key', 'Value'])
    filename = filename + '.csv'
    df.to_csv(filename, sep=',', encoding='utf-8', index=False)
def create_sine(X, A, F, y0):
    Y = A*np.sin(2*np.pi*F*X) + y0
    return Y

X = np.linspace(-160, -20, 70)
Y = create_sine(X,60, 1, -80)

file = np.zeros((len(X),2))
file[:,0] = X
file[:,1]=Y

save_csv(coordinates=file, filename='sin')
# plt.plot(X,Y)
# plt.show()
machine_path = np.array([
    [-20, -19.5, -159.5, -160, -60, -60, -119.5,-119.5, -100, -20],
    [-20, -140, -140, -21, -20, -100, -100, -60.5, -60.5, -20]
])
machine_path = np.transpose(machine_path)
coordiates = np.array([
    [-20, -20, -160, -160, -60, -60, -120, -120, -100, -20],
    [-20, -140, -140, -20, -20, -100, -100, -60, -60, -20]])
# Define the dimensions of the environment in mm
min_x = -221#190  # mm
min_y = -201#170  # mm
max_x = 0  # mm
max_y = 0  # mm
grid_size_mm = 1  # Grid size in mm

# Define A4 paper size in inches (1 inch = 25.4 mm)
a4_width_mm = 297
a4_height_mm = 210
a4_width_in = a4_width_mm / 25.4
a4_height_in = a4_height_mm / 25.4

# Calculate the width and height of the environment
width_mm = abs(min_x)
height_mm = abs(min_y)

# Calculate the necessary margins to center the plot on A4 paper
margin_x = (a4_width_mm - width_mm) / 2
margin_y = (a4_height_mm - height_mm) / 2

# Create a figure with A4 size
fig = plt.figure(figsize=(a4_width_in, a4_height_in))  # Set figure size to A4

# Create an axis with limits set to the dimensions of the environment
ax = fig.add_axes([margin_x / a4_width_mm, margin_y / a4_height_mm, width_mm /
    a4_width_mm, height_mm / a4_height_mm])
# ax1 = fig.add_axes([margin_x / a4_width_mm, margin_y / a4_height_mm, width_mm /
    a4_width_mm, height_mm / a4_height_mm])

# Plot grid lines
```

```
i=10
linewidth=0.5
for x in range(0, width_mm + 1, grid_size_mm):
    if x==i:
        linewidth=1
        col = 'black'
        i+=10
    else:
        linewidth=0.5
        col = 'gray'
    ax.axvline(-x, color=col, linewidth=linewidth)
i=10
for y in range(0, height_mm + 1, grid_size_mm):
    if y==i:
        linewidth=1
        col = 'black'
        i+=10
    else:
        linewidth=0.5
        color = 'gray'
    ax.axhline(-y, color=col, linewidth=linewidth)

min_x = -190
min_y = -170
# Add a marker (star or dot) at a specific location
marker_x, marker_y = coordiates[0,:], coordiates[1,:]  # Coordinates for the marker
marker_XY = np.array([
    [0, 0, -170, -170, 0],
    [0, -150, -150, 0, 0]
])
marker_size = 100  # Size of the marker
ax.scatter(marker_x, marker_y, s=marker_size, color='green', marker='o')  # Star marker
ax.scatter(marker_XY[0,:], marker_XY[1,:], s=marker_size, color='red', marker='o')  #
    Star marker
ax.plot(marker_x, marker_y, color='red', linewidth=2, label='Desired trajectory')
ax.plot(machine_path[:,0], machine_path[:,1], color='blue', linewidth=1,label='Machine
    trajectory')
ax.plot(marker_XY[0,:], marker_XY[1,:], color='green', linewidth=2, label='Workspace')
# ax.scatter(marker_x, marker_y, s=20, color='blue', marker='o')  # Star marker
# ax.plot(X,Y, color='brown',linewidth=4)
# Set the limits and aspect ratio
ax.set_xlim(min_x, max_x)
ax.set_ylim(min_y, max_y)
ax.set_aspect('equal')

# Invert the Y-axis to have zero at the bottom
ax.invert_yaxis()

# Label the axes with numbers
ax.set_xticks(range(min_x, max_x + 1, 10))  # Adjust the step size as needed
ax.set_yticks(range(min_y, max_y + 1, 10))  # Adjust the step size as needed
ax.set_xticklabels(range(min_x, max_x + 1, 10), fontsize=8)
ax.set_yticklabels(range(min_y, max_y + 1, 10), fontsize=8)

# Move the y-axis ticks and labels to the right side
ax.yaxis.set_label_position("right")
ax.yaxis.tick_left()
ax.xaxis.tick_top()

# Remove axes for a cleaner look but keep the ticks and labels
ax.tick_params(axis='both', which='both', length=0)  # Remove tick lines

# Draw an arrow from (1.5, 6) to (2.5, 7.5)
ax.annotate('', xy=(marker_x[0]+5, marker_y[0]), xytext=(marker_x[0]+5, marker_y[0]-10)
    ,
```

```
            arrowprops=dict(facecolor='blue', arrowstyle='<|-', lw=2),
            fontsize=12)
# Add text near the arrow
ax.text(marker_x[0]+8, marker_y[0], 'start', fontsize=12, color='blue')


ax.annotate('', xy=(marker_x[0]-20, marker_y[0]-5), xytext=(marker_x[0]-10, marker_y
    [0]),
            arrowprops=dict(facecolor='blue', arrowstyle='<|-', lw=2),
            fontsize=12)
ax.text(marker_x[0]-20, marker_y[0]+5, 'End', fontsize=12, color='blue')
ax.text(0,0, 'origin', color='blue', fontsize=12)

ax.legend()
# Save the plot as a high-resolution image
# output_file = 'image/final_figures/grid_plot_compatible_with_a4.png'
output_file = 'image/final_figures/grid_plot_compatible_with_a4_result.png'
fig.savefig(output_file, dpi=254)  # 254 dpi ensures that 1 pixel = 0.1 mm (since 1
    inch = 25.4 mm)
plt.show()
# Close the plot
plt.close(fig)

print(f"Plot saved as {output_file}. Open this file and print it at 100% scale to
    ensure correct dimensions.")
```

D. Precision test analysis

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os
import math
import sklearn.metrics


def load_data(file_path):
    # myfile = pd.read_csv(file_path)
    data = pd.read_csv(file_path).values
    # data = data[28:1024, :]
    # data=np.round(data/6.5,4)
    return data

def calculate_rmse_2d(actual, prediction):
    error = actual - prediction
    squared_error = np.square(error)
    N = np.shape(error)[1]
    # Calculate the  sum of the squared differences in both dimensions
    sum_squared_diffs = np.sum(squared_error, axis=1)/N
    # Calculate the mean of the squared differences
    mean_squared_diffs = np.mean(sum_squared_diffs)
    # Calculate the RMSE
    rmse = np.sqrt(mean_squared_diffs)
    return rmse, sum_squared_diffs

# data = load_data('precision_test/precision_points.csv')
# data = data[:,0]


actual_path = np.array([
    [-20, -20, -160, -160, -60, -60, -120, -120, -100, -20],
    [-20, -140, -140, -20, -20, -100, -100, -60, -60, -20]

])

machine_path = np.array([
    [-20, -19.5, -159.5, -160, -60, -60, -119.5,-119.5, -100, -20],
    [-20, -140, -140, -21, -20, -100, -100, -60.5, -60.5, -20]
])

actual_path = np.transpose(actual_path)
machine_path = np.transpose(machine_path)

rmse, sum_squar_error = calculate_rmse_2d(actual_path, machine_path)

root_sum_squar_error = np.sqrt(sum_squar_error)
print('root mean squared error = ',rmse)

error = actual_path - machine_path
squar_error = error**2


plt.plot(actual_path[:,0], actual_path[:,1], color='blue', label='actual path')
plt.plot(machine_path[:,0], machine_path[:,1],color='red', label='machine path')
plt.title('Precision test')
plt.xlabel('x [mm]')
plt.ylabel('y [mm]')
plt.legend()

plt.savefig('image/final_figures/precision_result.jpg')
# plt.show()
```

```
'''
plotting the sum of the squared error
'''
figbox_rms, box_rms = plt.subplots(figsize=(8, 6))
boxplot_rms = box_rms.boxplot(sum_squar_error, labels='E',showmeans=True, meanline=True
    )
# boxplot_rms = box_rms.boxplot(root_sum_squar_error, showmeans=True, meanline=True)

# Customize the mean line thickness
for line in boxplot_rms['means']:
    line.set_linewidth(2.5)


plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
box_rms.set_title('Sum of the squared error', fontsize=20)
box_rms.set_ylabel(r'MSE $[mm]$', fontsize=20)
box_rms.set_xlabel(' ')
plt.tight_layout()
plt.savefig('image/final_figures/rms_box.jpg')
figbox_rms.show()


fig, box = plt.subplots(figsize=(8, 6))
box.boxplot(squar_error, labels=['X', 'Y'], showmeans=True, meanline=True)
box.set_title('Sequared error along each axis', fontsize=20)
box.set_ylabel('SE [mm]', fontsize=20)

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)

# box.set_ylable('')
plt.savefig('image/final_figures/error_box.jpg')
fig.show()
```

E. Sealing quality Python code

```
import time

import serial
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def pressure_reading():
    def save_csv(coordinates,filename=''):
        # coordinates = coordinates.reshape(len(x_coor), 2)
        df = pd.DataFrame(coordinates)
        df.to_csv(filename, sep=',', encoding='utf-8', index=False, header=False)

    inp = input('Enter the file name : ')
    N_test = input('Enter the number of test: ')
    st_time=time.time()
    ser = serial.Serial('COM4', 115200)
    run = True
    N=100;
    i=0
    # Data = []
    data = np.zeros((N,1))
    command = '1'.encode()
    # ser.write(command)
    while run:
        ser.write(command)
        # time.sleep(0.5)
        message = ser.readline()
        # message.decode('utf-8').rstrip()
        message = str(message, 'utf-8')
        message = message.strip('\r\n')
        print(message)
        data[i, 0] = float(message)
        i += 1
        end_time= time.time()
        elapse_time = round(end_time-st_time,0)
        # print(elapse_time)
        if elapse_time >= 60.0:
            print("Done")
            command = "0".encode()
            ser.write(command)
            run = False
            ser.close()



    # data = np.zeros((len(Data),1))
    #
    # for i in range(len(Data)):
    #     data[i,0]=Data[i]
    filename = 'result/plastic/individual_data/%s/'%N_test+ inp +'.csv'
    # filename = 'result/TPU/individual_data/%s/'%N_test+ inp +'.csv'
    save_csv(data, filename=filename)
    print("file saved.")
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os

def concatenate_data():
    def load_data(file_path):
```

```
        # myfile = pd.read_csv(file_path)
        data = pd.read_csv(file_path).values
        # data = data[28:1024, :]
        # data=np.round(data/6.5,4)
        return data
    # Data = np.zeros((100,27))

    def save_csv(coordinates, filename=''):
        # coordinates = coordinates.reshape(len(x_coor), 2)
        # df = pd.DataFrame(coordinates)
        df = pd.DataFrame(list(coordinates.items()), columns=['Key', 'Value'])
        filename = filename + '.csv'
        df.to_csv(filename, sep=',', encoding='utf-8', index=False)

    # save_csv(Data,"Data.csv")
    material = input('Specify the material: ')
    N_test = input('Specify the number of tests: ')
    # data = load_data("result/Data.csv")[:,0]
    path_plastic = "result/plastic/individual_data/%s"%N_test
    path_tpu = "result/tpu/individual_data/%s"%N_test
    if material == 'plastic':
        path = path_plastic
        directory = os.listdir(path)
    elif material == 'TPU':
        path = path_tpu
        directory = os.listdir(path)
    else:
        print("Invalid material. Please choose between 'plastic' and 'TPU'")
        # material = input('Specify the material: ')
    # directory = os.listdir(path)
    # print(directory)
    N = len(directory)
    scale = 80 # 1 bar = 80 in arduino output
    Data = []
    # Data = np.zeros()
    # header =
    max_values = {

    }
    # Material = []
    i=0
    for file in directory:
        # Material.append(material)
        filename = file.split('.')[0]
        # header.append(filename)
        new_data = load_data(path + '/' + file) #/ scale
        data = new_data[:,0]/scale
        # max_values.append(max(data))
        max_values[filename] = max(data)
        Data.append(new_data[:,0])
        i+=1

    print(len(max_values))
    if material=='plastic':
        save_csv(coordinates=max_values, filename='result/data_plastic_%s'%N_test)
    elif material=='TPU':
        save_csv(coordinates=max_values, filename='result/data_tpu_%s'%N_test)
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

def plot_data():
    def load_data(file_path):
        # myfile = pd.read_csv(file_path)
```

```
    data = pd.read_csv(file_path)  # .values
    data_dict = data.set_index('Key').to_dict()['Value']
    # data = data[28:1024, :]
    # data=np.round(data/6.5,4)
    return data_dict

material = 'plastic'
file_path_1 = 'result/data_plastic_1.csv'
file_path_2 = 'result/data_plastic_2.csv'
file_path_3 = 'result/data_plastic_3.csv'

data_1 = load_data(file_path_1)
data_2 = load_data(file_path_2)
data_3 = load_data(file_path_3)

keys = data_2.keys()
keys = list(keys)


D = [1, 2, 3]
T1_20 = []
T1_50 = []
T1_100 = []
T1_200 = []
T1_300 = []
Max_vals1_20 = []
Max_vals1_50 = []
Max_vals1_100 = []
Max_vals1_200 = []
Max_vals1_300 = []

T2_20 = []
T2_50 = []
T2_100 = []
T2_200 = []
T2_300 = []
Max_vals2_20 = []
Max_vals2_50 = []
Max_vals2_100 = []
Max_vals2_200 = []
Max_vals2_300 = []

T3_20 = []
T3_50 = []
T3_100 = []
T3_200 = []
T3_300 = []
Max_vals3_20 = []
Max_vals3_50 = []
Max_vals3_100 = []
Max_vals3_200 = []
Max_vals3_300 = []
average_values = {}
# print(data_2[key])
# val_3 = 0
for key in keys:
    # val_1 = data_1[key]
    val_2 = data_2[key]
    val_3 = data_3[key]
    av_val = (val_2 + val_3)/2
    average_values[key] = av_val

max_values = average_values
# max_values = data_2
# max_values = data_3
```

```
for i in range(len(keys)):
    label = keys[i]
    d = int(label.split(',')[2].split('=')[1])  # diameter
    v = int(label.split(',')[3].split('=')[1])  # speed

    if d == 1:
        if v == 20:
            Max_vals1_20.append(max_values[label])
            T1_20.append(int(label.split(',')[0].split('=')[1]))
        if v == 50:
            Max_vals1_50.append(max_values[label])
            T1_50.append(int(label.split(',')[0].split('=')[1]))
        if v == 100:
            Max_vals1_100.append(max_values[label])
            T1_100.append(int(label.split(',')[0].split('=')[1]))
        if v == 200:
            # print(label)
            Max_vals1_200.append(max_values[label])
            T1_200.append(int(label.split(',')[0].split('=')[1]))
        if v == 300:
            Max_vals1_300.append(max_values[label])
            T1_300.append(int(label.split(',')[0].split('=')[1]))
    if d == 2:
            if v == 20:
                Max_vals2_20.append(max_values[label])
                T2_20.append(int(label.split(',')[0].split('=')[1]))
            if v == 50:
                Max_vals2_50.append(max_values[label])
                T2_50.append(int(label.split(',')[0].split('=')[1]))
            if v == 100:
                Max_vals2_100.append(max_values[label])
                T2_100.append(int(label.split(',')[0].split('=')[1]))
            if v == 200:
                Max_vals2_200.append(max_values[label])
                T2_200.append(int(label.split(',')[0].split('=')[1]))
            if v == 300:
                # print(max_values[i])
                Max_vals2_300.append(max_values[label])
                T2_300.append(int(label.split(',')[0].split('=')[1]))
    if d == 3:
        if v == 20:
            Max_vals3_20.append(max_values[label])
            T3_20.append(int(label.split(',')[0].split('=')[1]))
        if v == 50:
            Max_vals3_50.append(max_values[label])
            T3_50.append(int(label.split(',')[0].split('=')[1]))
        if v == 100:
            Max_vals3_100.append(max_values[label])
            T3_100.append(int(label.split(',')[0].split('=')[1]))
        if v == 200:
            # print(max_values[i])
            Max_vals3_200.append(max_values[label])
            T3_200.append(int(label.split(',')[0].split('=')[1]))
        if v == 300:
            # print(max_values[i])
            Max_vals3_300.append(max_values[label])
            T3_300.append(int(label.split(',')[0].split('=')[1]))


for i in range(3):
    T1_20[i] -= 10
    T1_50[i] -= 5
    T1_200[i] += 5
    T1_300[i] += 10
```

```
        T2_20[i]-=10
        T2_50[i]-=5
        T2_200[i]+=5
        T2_300[i]+=10

        T3_20[i]-=10
        T3_50[i] -= 5
        T3_200[i] +=5
        T3_300[i] += 10


    tick_label = ['200', '250', '300']

    fig, axs = plt.subplots(2, 2, figsize=(10, 8))
    fig.suptitle('Heat sealing strength vs temperature and speed', fontsize=16)
    # Plot on the first subplot (0, 0)
    axs[0, 0].bar(T1_20, Max_vals1_20, width=5, label=r'V=20 $[\frac{mm}{min}]$', color
    ='black')
    axs[0, 0].bar(T1_50, Max_vals1_50, width=5, label=r'V=50 $[\frac{mm}{min}]$', color
    ='blue')
    axs[0, 0].bar(T1_100, Max_vals1_100, tick_label=tick_label, width=5, label=r'V=100
    $[\frac{mm}{min}]$', color='red')
    axs[0, 0].bar(T1_200, Max_vals1_200, width=5, label=r'V=200 $[\frac{mm}{min}]$',
    color='green')
    axs[0, 0].bar(T1_300, Max_vals1_300, width=5, label=r'V=300 $[\frac{mm}{min}]$',
    color='magenta')
    axs[0, 0].set_title(r'$d=1 [mm]$',fontsize=15)
    axs[0, 0].set_ylabel('Sealing strength[bar]', fontsize=10)
    axs[0, 0].set_xlabel('Temperature', fontsize=10)
    axs[0, 0].set_ylim(top=3.2)

    d1_eff = [Max_vals1_20, Max_vals1_50, Max_vals1_100, Max_vals1_200, Max_vals1_300]
    d2_eff = [Max_vals2_20, Max_vals2_50, Max_vals2_100, Max_vals2_200, Max_vals2_300]
    d3_eff = [Max_vals3_20, Max_vals3_50, Max_vals3_100, Max_vals3_200, Max_vals3_300]
    # Add a horizontal line to show the maximum

    mx = round(np.max(d1_eff),2)
    axs[0, 0].axhline(y=mx, color='black', linestyle='--', linewidth=2)
    # Get the x-axis limits to calculate the midpoint
    x_min, x_max = axs[0, 0].get_xlim()
    x_mid = (x_min + x_max) / 2
    # Add text in the middle of the horizontal line
    axs[0, 0].text(x_mid, mx, r'$\gamma = %s $'%mx, ha='center', va='bottom', color='
    black', fontsize=10)


    # Plot on the first subplot (0, 1)
    axs[0, 1].bar(T2_20, Max_vals2_20, width=5, label=r'V=20 $[\frac{mm}{min}]$', color
    ='black')
    axs[0, 1].bar(T2_50, Max_vals2_50, width=5, label=r'V=50 $[\frac{mm}{min}]$', color
    ='blue')
    axs[0, 1].bar(T2_100, Max_vals2_100, tick_label=tick_label, width=5, label=r'V=100
    $[\frac{mm}{min}]$', color='red')
    axs[0, 1].bar(T2_200, Max_vals2_200, width=5, label=r'V=200 $[\frac{mm}{min}]$',
    color='green')
    axs[0, 1].bar(T2_300, Max_vals2_300, width=5, label=r'V=300 $[\frac{mm}{min}]$',
    color='magenta')
    axs[0, 1].set_title(r'$d=2 [mm]$', fontsize=15)
    axs[0, 1].set_ylabel('Sealing strength[bar]', fontsize=10)
    axs[0, 1].set_xlabel('Temperature', fontsize=10)
    axs[0, 1].set_ylim(top=3.2)

    mx = round(np.max(d2_eff),2)
    axs[0, 1].axhline(y=mx, color='black', linestyle='--', linewidth=2)
    # Get the x-axis limits to calculate the midpoint
```

```
x_min, x_max = axs[0, 1].get_xlim()
x_mid = (x_min + x_max) / 2
# Add text in the middle of the horizontal line
axs[0, 1].text(x_mid, mx, r'$\gamma = %s $'%mx, ha='center', va='bottom', color='
black', fontsize=10)


# Plot on the first subplot (1, 0)
axs[1, 0].bar(T3_20, Max_vals3_20, width=5, label=r'V=20 $[\frac{mm}{min}]$', color
='black')
axs[1, 0].bar(T3_50, Max_vals3_50, width=5, label=r'V=50 $[\frac{mm}{min}]$', color
='blue')
axs[1, 0].bar(T3_100, Max_vals3_100, tick_label=tick_label, width=5, label=r'V=100
$[\frac{mm}{min}]$', color='red')
axs[1, 0].bar(T3_200, Max_vals3_200, width=5, label=r'V=200 $[\frac{mm}{min}]$',
color='green')
axs[1, 0].bar(T3_300, Max_vals3_300, width=5, label=r'V=300 $[\frac{mm}{min}]$',
color='magenta')
axs[1, 0].set_title(r'$d=3 [mm]$',fontsize=15)
axs[1, 0].set_ylabel('Sealing strength[bar]', fontsize=10)
axs[1, 0].set_xlabel('Temperature', fontsize=10)
axs[1, 0].set_ylim(top=3.2)

mx = round(np.max(d3_eff),2)
axs[1, 0].axhline(y=mx, color='black', linestyle='--', linewidth=2)
# Get the x-axis limits to calculate the midpoint
x_min, x_max = axs[1, 0].get_xlim()
x_mid = (x_min + x_max) / 2
# Add text in the middle of the horizontal line
axs[1, 0].text(x_mid, mx, r'$\gamma = %s $'%mx, ha='center', va='bottom', color='
black', fontsize=10)


# Plot on the first subplot (1, 1)
axs[1, 1].bar([200], [0], width=5, label=r'V=20 $[\frac{mm}{min}]$', color='black')
axs[1, 1].bar([250], [0], width=5, label=r'V=50 $[\frac{mm}{min}]$', color='blue')
axs[1, 1].bar([300], [0], width=5, label=r'V=100 $[\frac{mm}{min}]$', color='red')
axs[1, 1].bar([0], [0], width=5, label=r'V=200 $[\frac{mm}{min}]$', color='green')
axs[1, 1].bar([100], [0], width=5, label=r'V=300 $[\frac{mm}{min}]$', color='
magenta')
axs[1, 1].set_ylim(top=3.2)
axs[1, 1].legend(loc='center', fontsize=15)
plt.subplots_adjust(hspace=0.5)
plt.savefig('image/final_figures/sealing_strength_bar_%s.jpg'%material)
# fig.show()


'''
Next we plot the Dwell time: the time material layers are in direct contact
with the heat Source.
'''

V = [300, 200, 100, 50, 20]
Dt = np.zeros(len(V))
# dwell_t = np.reshape(dwell_t, (1,len(V)))
# print(dwell_t[0])
dwell_t = []
dwell_t2 = np.zeros(len(V))
# dwell_t3 = []
k=-0.2
for i in range(len(V)):
    V[i]=V[i]/60 # convert to mm/seconds
    dwt = np.round(1/V[i],1) # get the dwell time it takes for the machine to move
1 mm
    # Dt[i]=dwt
```

```
    dwell_t2[i] = dwt
    dwell_t.append(dwt) # append the dwell time for the labels tick
dwell_t2[0] -= 0.5
dwell_t2[1] += 0.2
dwell_t2[2] += 0.6
dwell_t2[3] += 0.9
Dt = dwell_t2 - 0.2
dwell_t3 = dwell_t2 + 0.2

# Dt[0] -= 0.5
# Dt[1] += 0.2
# Dt[2] += 0.6
# Dt[3] += 0.9
# dwell_t2 = Dt-0.2
# dwell_t3 = Dt + 0.2


'''
Dwell time for d=1mm at 3 different temperatures 200, 250, and 300 respectively.
'''

fig_dwell_1, axs_dwell_1 = plt.subplots(2, 2, figsize=(10, 8))
fig_dwell_1.suptitle(r'Heat sealing strength vs dwell time for sealing thickness
of $d=1mm$', fontsize=16)

T200_1_eff = [Max_vals1_300[0], Max_vals1_200[0], Max_vals1_100[0], Max_vals1_50
[0], Max_vals1_20[0]] # @T=200
T250_1_eff = [Max_vals1_300[1], Max_vals1_200[1], Max_vals1_100[1], Max_vals1_50
[1], Max_vals1_20[1]] # @T=250
T300_1_eff = [Max_vals1_300[2], Max_vals1_200[2], Max_vals1_100[2], Max_vals1_50
[2], Max_vals1_20[2]] # @T=300


# Plot on the first subplot (0, 0)
axs_dwell_1[0, 0].bar(Dt, T200_1_eff, tick_label=dwell_t, color='blue', width=0.2,
label=r'$T=200^\circ C$')
axs_dwell_1[0, 0].plot(Dt, T200_1_eff, color='blue', linewidth=2)
axs_dwell_1[0, 0].set_xlabel(r'Dwell time $[s]$', fontsize=10)
axs_dwell_1[0, 0].set_ylabel('Sealing Strength $[bar]$', fontsize=10)
axs_dwell_1[0, 0].set_ylim(top=3.2)

axs_dwell_1[0, 1].bar(Dt, T250_1_eff, tick_label=dwell_t, color='red', width=0.2,
label=r'$T=250^\circ C$')
axs_dwell_1[0, 1].plot(Dt, T250_1_eff, color='red', linewidth=2)
# axs_dwell_1[0, 1].set_title('$T=250^\circ C$ $[mm]$', fontsize=15)
axs_dwell_1[0, 1].set_xlabel(r'Dwell time $[s]$', fontsize=10)
axs_dwell_1[0, 1].set_ylim(top=3.2)

axs_dwell_1[1, 0].bar(Dt, T300_1_eff, tick_label=dwell_t, color='black', width=0.2,
 label=r'$T=300^\circ C$')
axs_dwell_1[1, 0].plot(Dt, T300_1_eff, color='black', linewidth=2)
# axs_dwell_1[1, 0].set_title('$T=300^\circ C$ $[mm]$', fontsize=15)
axs_dwell_1[1, 0].set_xlabel(r'Dwell time $[s]$', fontsize=10)
axs_dwell_1[1, 0].set_ylabel('Sealing Strength $[bar]$', fontsize=10)
axs_dwell_1[1, 0].set_ylim(top=3.2)

axs_dwell_1[1, 1].bar([0.3],[0], label=r'$T=200^\circ C$', color='blue')
axs_dwell_1[1, 1].bar([0.6],[0], label=r'$T=250^\circ C$', color='red')
axs_dwell_1[1, 1].bar([3],[0], label=r'$T=300^\circ C$', color='black')
axs_dwell_1[1, 1].set_ylim(top=3.2)
plt.subplots_adjust(hspace=0.5)  # Increase hspace as needed
axs_dwell_1[1, 1].legend(loc='center', fontsize=15)
plt.savefig('image/final_figures/dwellTime_d1.jpg')
# fig_dwell_1.show()
```

```
'''
Dwell time for d=2 mm at 3 different temperatures 200, 250, and 300 respectively.
'''
fig_dwell_2, axs_dwell_2 = plt.subplots(2, 2, figsize=(10, 8))
fig_dwell_2.suptitle(r'Heat sealing strength vs dwell time for sealing thickness
of $d=2mm$', fontsize=16)

T200_2_eff = [Max_vals2_300[0], Max_vals2_200[0], Max_vals2_100[0], Max_vals2_50
[0], Max_vals2_20[0]]
T250_2_eff = [Max_vals2_300[1], Max_vals2_200[1], Max_vals2_100[1], Max_vals2_50
[1], Max_vals2_20[1]]
T300_2_eff = [Max_vals2_300[2], Max_vals2_200[2], Max_vals2_100[2], Max_vals2_50
[2], Max_vals2_20[2]]


# for i in range(3):
#     dwell_t[i] = dwell_t[i]*2
# print(dwell_t)
# Plot on the second suplot (0, 1)
axs_dwell_2[0, 0].bar(dwell_t2, T200_2_eff, tick_label=dwell_t, color='blue', width
=0.2, label=r'$T=200^\circ C$')
axs_dwell_2[0, 0].plot(dwell_t2, T200_2_eff, color='blue', linewidth=2)
axs_dwell_2[0, 0].set_xlabel(r'Dwell time $[s]$', fontsize=10)
axs_dwell_2[0, 0].set_ylabel('Sealing Strength $[bar]$', fontsize=10)
axs_dwell_2[0, 0].set_ylim(top=3.2)

axs_dwell_2[0, 1].bar(dwell_t2, T250_2_eff, tick_label=dwell_t, color='red', width
=0.2, label=r'$T=250^\circ C$')
axs_dwell_2[0, 1].plot(dwell_t2, T250_2_eff, color='red', linewidth=2)
axs_dwell_2[0, 1].set_xlabel(r'Dwell time $[s]$', fontsize=10)
axs_dwell_2[0, 1].set_ylabel('Sealing Strength $[bar]$', fontsize=10)
axs_dwell_2[0, 1].set_ylim(top=3.2)

axs_dwell_2[1, 0].bar(dwell_t2, T300_2_eff, tick_label=dwell_t, color='black',
width=0.2, label=r'$T=300^\circ C$')
axs_dwell_2[1, 0].plot(dwell_t2, T300_2_eff, color='black', linewidth=2)
axs_dwell_2[1, 0].set_xlabel(r'Dwell time $[s]$', fontsize=10)
axs_dwell_2[1, 0].set_ylabel('Sealing Strength $[bar]$', fontsize=10)
axs_dwell_2[1, 0].set_ylim(top=3.2)

axs_dwell_2[1, 1].bar([0.3],[0], label=r'$T=200^\circ C$', color='blue')
axs_dwell_2[1, 1].bar([0.6],[0], label=r'$T=250^\circ C$', color='red')
axs_dwell_2[1, 1].bar([3],[0], label=r'$T=300^\circ C$', color='black')
axs_dwell_2[1, 1].legend(fontsize='x-large', loc='center', borderaxespad=0.5)
axs_dwell_2[1, 1].set_ylim(top=3.2)
plt.subplots_adjust(hspace=0.5)  # Increase hspace as needed

plt.savefig('image/final_figures/dwellTime_d2.jpg')
# fig_dwell_2.show()
'''
Dwell time for d=2mm at 3 different temperatures 200, 250, and 300 respectively.
'''

fig_dwell_3, axs_dwell_3 = plt.subplots(2, 2, figsize=(10, 8))
fig_dwell_3.suptitle(r'Heat sealing strength vs dwell time for sealing thickness
of $d=3mm$', fontsize=16)


T200_3_eff = [Max_vals3_300[0], Max_vals3_200[0], Max_vals3_100[0], Max_vals3_50
[0], Max_vals3_20[0]]
T250_3_eff = [Max_vals3_300[1], Max_vals3_200[1], Max_vals3_100[1], Max_vals3_50
[1], Max_vals3_20[1]]
T300_3_eff = [Max_vals3_300[2], Max_vals3_200[2], Max_vals3_100[2], Max_vals3_50
[2], Max_vals3_20[2]]
# for i in range(3):
```

```
#     dwell_t[i] = dwell_t[i]*3/2
# Plot on the third subplot (1, 0)
axs_dwell_3[0, 0].bar(dwell_t3, T200_3_eff, tick_label=dwell_t, color='blue', width
=0.2, label=r'$T=200^\circ C$')
axs_dwell_3[0, 0].plot(dwell_t3, T200_3_eff, color='blue', linewidth=2)
axs_dwell_3[0, 0].set_xlabel(r'Dwell time $[s]$', fontsize=10)
axs_dwell_3[0, 0].set_ylabel('Sealing Strength $[bar]$', fontsize=10)
axs_dwell_3[0, 0].set_ylim(top=3.2)

axs_dwell_3[0, 1].bar(dwell_t3, T250_3_eff, tick_label=dwell_t, color='red', width
=0.2, label=r'$T=250^\circ C$')
axs_dwell_3[0, 1].plot(dwell_t3, T250_3_eff, color='red', linewidth=2)
axs_dwell_3[0, 1].set_xlabel(r'Dwell time $[s]$', fontsize=10)
axs_dwell_3[0, 1].set_ylabel('Sealing Strength $[bar]$', fontsize=10)
axs_dwell_3[0, 1].set_ylim(top=3.2)

axs_dwell_3[1, 0].bar(dwell_t3, T300_3_eff, tick_label=dwell_t, color='black',
width=0.2, label=r'$T=300^\circ C$')
axs_dwell_3[1, 0].plot(dwell_t3, T300_3_eff, color='black', linewidth=2)
axs_dwell_3[1, 0].set_xlabel(r'Dwell time $[s]$', fontsize=10)
axs_dwell_3[1, 0].set_ylabel('Sealing Strength $[bar]$', fontsize=10)
axs_dwell_3[1, 0].set_ylim(top=3.2)

axs_dwell_3[1, 1].bar([0.3],[0], label=r'$T=200^\circ C$', color='blue')
axs_dwell_3[1, 1].bar([0.6],[0], label=r'$T=250^\circ C$', color='red')
axs_dwell_3[1, 1].bar([3],[0], label=r'$T=300^\circ C$', color='black')
axs_dwell_3[1, 1].legend(fontsize='x-large', loc='center')
axs_dwell_3[1, 1].set_ylim(top=3.2)
plt.subplots_adjust(hspace=0.5)  # Increase hspace as needed

plt.savefig('image/final_figures/dwellTime_d3.jpg')
# fig_dwell_3.show()


'''
################   Dwell time  ######################
'''
vals200 = [T200_1_eff,T200_2_eff,T200_3_eff]
vals250 = [T250_1_eff, T250_2_eff, T250_3_eff]
vals300 = [T300_1_eff, T300_2_eff, T300_3_eff]


max_1 = []
max_2 = []
max_3 = []
for i in range(3):
    max_1.append(max(vals200[i]))
    max_2.append(max(vals250[i]))
    max_3.append(max(vals300[i]))


fig_dwell, axs_dwell = plt.subplots(2, 2, figsize=(10, 8))
fig_dwell.suptitle(r'Heat sealing strength vs dwell time', fontsize=16)

#Plot on the first subplot (0, 0)
axs_dwell[0, 0].bar(Dt, T200_1_eff, color='blue', width=0.2, label=r'$d=1 [mm]$')
# axs_dwell[0, 0].plot(Dt, T200_1_eff, color='blue', linewidth=2)

axs_dwell[0, 0].bar(dwell_t2, T200_2_eff, tick_label=dwell_t, color='red', width
=0.2, label=r'$d=2 [mm]$')
# axs_dwell[0, 0].plot(dwell_t2, T200_2_eff, color='red', linewidth=2)

axs_dwell[0, 0].bar(dwell_t3, T200_3_eff, color='black', width=0.2, label=r'$d=3 [
mm]$')
# axs_dwell[0, 0].plot(dwell_t3, T200_3_eff, color='black', linewidth=2)
```

```
axs_dwell[0, 0].set_title(r'Temperature $T=200^\circ C$')
axs_dwell[0, 0].set_xlabel(r'Dwell time $[s]$', fontsize=10)
axs_dwell[0, 0].set_ylabel('Sealing Strength $[bar]$', fontsize=10)
axs_dwell[0, 0].set_ylim(top=3.2)

# Plot on the first subplot (0, 01)
axs_dwell[0, 1].bar(Dt, T250_1_eff, color='blue', width=0.2, label=r'$d=1 [mm]$')
# axs_dwell[0, 1].plot(Dt, T250_1_eff, color='blue', linewidth=2)

axs_dwell[0, 1].bar(dwell_t2, T250_2_eff, tick_label=dwell_t, color='red', width
=0.2, label=r'$d=2 [mm]$')
# axs_dwell[0, 1].plot(dwell_t2, T250_2_eff, color='red', linewidth=2)

axs_dwell[0, 1].bar(dwell_t3, T250_3_eff, color='black', width=0.2, label=r'$d=3 [
mm]$')
# axs_dwell[0, 1].plot(dwell_t3, T250_3_eff, color='black', linewidth=2)

axs_dwell[0, 1].set_title(r'Temperature $T=250^\circ C$')
axs_dwell[0, 1].set_xlabel(r'Dwell time $[s]$', fontsize=10)
axs_dwell[0, 1].set_ylabel('Sealing Strength $[bar]$', fontsize=10)
axs_dwell[0, 1].set_ylim(top=3.2)

# Plot on the first subplot (1, 0)
axs_dwell[1, 0].bar(Dt, T300_1_eff, color='blue', width=0.2, label=r'$d=1 [mm]$')
# axs_dwell[1, 0].plot(Dt, T300_1_eff, color='blue', linewidth=2)

axs_dwell[1, 0].bar(dwell_t2, T300_2_eff, tick_label=dwell_t, color='red', width
=0.2, label=r'$d=2 [mm]$')
# axs_dwell[1, 0].plot(dwell_t2, T300_2_eff, color='red', linewidth=2)

axs_dwell[1, 0].bar(dwell_t3, T300_3_eff, color='black', width=0.2, label=r'$d=3 [
mm]$')
# axs_dwell[1, 0].plot(dwell_t3, T300_3_eff, color='black', linewidth=2)

axs_dwell[1, 0].set_title(r'Temperature $T=300^\circ C$')
axs_dwell[1, 0].set_xlabel(r'Dwell time $[s]$', fontsize=10)
axs_dwell[1, 0].set_ylabel('Sealing Strength $[bar]$', fontsize=10)
axs_dwell[1, 0].set_ylim(top=3.2)

axs_dwell[1, 1].bar([0.3], [0], label=r'$d=1 [mm]$', color='blue')
axs_dwell[1, 1].bar([0.6], [0], label=r'$d=2 [mm]$', color='red')
axs_dwell[1, 1].bar([3], [0], label=r'$d=3 [mm]$', color='black')
axs_dwell[1, 1].legend(fontsize='x-large', loc='center')
axs_dwell[1, 1].set_title('Legend')
axs_dwell[1, 1].set_ylim(top=3.2)

mx = round(max(max_1), 2)
axs_dwell[0, 0].axhline(y=mx, color='black', linestyle='--', linewidth=2)
# Get the x-axis limits to calculate the midpoint
x_min, x_max = axs_dwell[0, 0].get_xlim()
x_mid = (x_min + x_max) / 2
# Add text in the middle of the horizontal line
axs_dwell[0, 0].text(x_mid, mx, r'$\gamma = %s $' % mx, ha='center', va='bottom',
color='black', fontsize=10)

mx = round(max(max_2), 2)
axs_dwell[0, 1].axhline(y=mx, color='black', linestyle='--', linewidth=2)
# Get the x-axis limits to calculate the midpoint
x_min, x_max = axs_dwell[0, 1].get_xlim()
x_mid = (x_min + x_max) / 2
# Add text in the middle of the horizontal line
axs_dwell[0, 1].text(x_mid, mx, r'$\gamma = %s $' % mx, ha='center', va='bottom',
color='black', fontsize=10)
```

```
mx = round(max(max_3), 2)
axs_dwell[1, 0].axhline(y=mx, color='black', linestyle='--', linewidth=2)
# Get the x-axis limits to calculate the midpoint
x_min, x_max = axs_dwell[1, 0].get_xlim()
x_mid = (x_min + x_max) / 2
# Add text in the middle of the horizontal line
axs_dwell[1, 0].text(x_mid, mx, r'$\gamma = %s $' % mx, ha='center', va='bottom',
color='black', fontsize=10)

plt.subplots_adjust(hspace=0.5)  # Increase hspace as needed
plt.savefig('image/final_figures/dwellTime.jpg')
fig_dwell.show()

'''
                    ***********

calculating and ploting the heat sealing strength vs sealing thickness

                    ***********
'''

d0_eff = [Max_vals1_20[0], Max_vals2_20[0], Max_vals3_20[0]] # @T=200
d1_eff = [Max_vals1_20[1], Max_vals2_20[1], Max_vals3_20[1]] #sealing strength at T
=250 for tip diameters
d2_eff = [Max_vals1_20[2], Max_vals2_20[2], Max_vals3_20[2]] # @T=300

D1 = []
D2 = []
for i in range(len(D)):
    D1.append(D[i] - 0.1)
    D2.append(D[i] + 0.1)
#

fig_tip_d, axs_tip_d = plt.subplots(3, 2, figsize=(10, 8))
fig_tip_d.suptitle(r'Heat sealing strength vs Sealing thickness', fontsize=20)

# Plot on the first subplot (0, 0)
axs_tip_d[0, 0].bar(D1, d0_eff, label=r'T=200', color='blue',width=0.1)
axs_tip_d[0, 0].bar(D, d1_eff, tick_label=D, label='T=250', color='red',width=0.1)
axs_tip_d[0, 0].bar(D2, d2_eff, label='T=300', color='green',width=0.1)
axs_tip_d[0, 0].set_title(r'V=20 $[\frac{mm}{min}]$', fontsize=15)
axs_tip_d[0, 0].set_ylabel(r'Sealing strength $[bar]$', fontsize=10)
axs_tip_d[0, 0].set_xlabel(r'Tip diameter $[mm]$', fontsize=10)
axs_tip_d[0, 0].set_ylim(top=3.2)
# axs_tip_d[0, 0].legend(loc='lower center')

mx = round(max(max(d0_eff, d1_eff, d2_eff)),2)
axs_tip_d[0, 0].axhline(y=mx, color='black', linestyle='--', linewidth=2)
# Get the x-axis limits to calculate the midpoint
x_min, x_max = axs_tip_d[0, 0].get_xlim()
x_mid = (x_min + x_max) / 2

# Add text in the middle of the horizontal line
axs_tip_d[0, 0].text(x_mid, mx, r'$\gamma = %s $'%mx, ha='center', va='bottom',
color='black', fontsize=10)

# ****  legend in subfigure (0, 1) --> beter for visualizing
axs_tip_d[0, 1].bar(D, [0,0,0], label=r'$T=200^\circ C$', color='blue',width=0.1)
axs_tip_d[0, 1].bar(D1, [0,0,0], tick_label=D, label=r'$T=250^\circ C$', color='red
',width=0.1)
axs_tip_d[0, 1].bar(D2, [0,0,0], label=r'$T=300^\circ C$', color='green',width=0.1)
axs_tip_d[0, 1].set_title('Legends', fontsize=15)
axs_tip_d[0, 1].set_ylim(top=3.2)
axs_tip_d[0, 1].legend(loc='center')
```

```
d0_eff = [Max_vals1_50[0], Max_vals2_50[0], Max_vals3_50[0]] # @T=200
d1_eff = [Max_vals1_50[1], Max_vals2_50[1], Max_vals3_50[1]] #sealing strength at T
=250 for tip diameters
d2_eff = [Max_vals1_50[2], Max_vals2_50[2], Max_vals3_50[2]] # @T=300

# Plot on the second subplot (0, 1)
axs_tip_d[1, 0].bar(D1, d0_eff, label='T=200', color='blue',width=0.1)
axs_tip_d[1, 0].bar(D, d1_eff, tick_label=D, label='T=250', color='red',width=0.1)
axs_tip_d[1, 0].bar(D2, d2_eff, label='T=300', color='green',width=0.1)
axs_tip_d[1, 0].set_title(r'V=50 $[\frac{mm}{min}]$', fontsize=15)
# axs_tip_d[1, 0].set_ylabel(r'Sealing strength $[bar]$', fontsize=15)
axs_tip_d[1, 0].set_xlabel(r'Tip diameter $[mm]$', fontsize=10)
axs_tip_d[1, 0].set_ylim(top=3.2)

# Add a horizontal line to show the maximum
mx = round(max(max(d0_eff, d1_eff, d2_eff)),2)
axs_tip_d[1, 0].axhline(y=mx, color='black', linestyle='--', linewidth=2)
# Get the x-axis limits to calculate the midpoint
x_min, x_max = axs_tip_d[1, 0].get_xlim()
x_mid = (x_min + x_max) / 2
# Add text in the middle of the horizontal line
axs_tip_d[1, 0].text(x_mid, mx, r'$\gamma = %s $'%mx, ha='center', va='bottom',
color='black', fontsize=10)


d0_eff = [Max_vals1_100[0], Max_vals2_100[0], Max_vals3_100[0]] # @T=200
d1_eff = [Max_vals1_100[1], Max_vals2_100[1], Max_vals3_100[1]] #sealing strength
at T=250 for tip diameters
d2_eff = [Max_vals1_100[2], Max_vals2_100[2], Max_vals3_100[2]] # @T=300

# Plot on the third subplot (1, 0)
axs_tip_d[1, 1].bar(D1, d0_eff, label='T=200', color='blue',width=0.1)
axs_tip_d[1, 1].bar(D, d1_eff, tick_label=D, label='T=250', color='red',width=0.1)
axs_tip_d[1, 1].bar(D2, d2_eff, label='T=300', color='green',width=0.1)
axs_tip_d[1, 1].set_title(r'V=100 $[\frac{mm}{min}]$', fontsize=15)
axs_tip_d[1, 1].set_ylabel(r'Sealing strength $[bar]$', fontsize=10)
axs_tip_d[1, 1].set_xlabel(r'Tip diameter $[mm]$', fontsize=10)
axs_tip_d[1, 1].set_ylim(top=3.2)

# Add a horizontal line to show the maximum
mx = round(max(max(d0_eff, d1_eff, d2_eff)),2)
axs_tip_d[1, 1].axhline(y=mx, color='black', linestyle='--', linewidth=2)
# Get the x-axis limits to calculate the midpoint
x_min, x_max = axs_tip_d[1, 1].get_xlim()
x_mid = (x_min + x_max) / 2
# Add text in the middle of the horizontal line
axs_tip_d[1, 1].text(x_mid, mx, r'$\gamma = %s $'%mx, ha='center', va='bottom',
color='black', fontsize=10)


d0_eff = [Max_vals1_200[0], Max_vals2_200[0], Max_vals3_200[0]] # @T=200
d1_eff = [Max_vals1_200[1], Max_vals2_200[1], Max_vals3_200[1]] #sealing strength
at T=250 for tip diameters
d2_eff = [Max_vals1_200[2], Max_vals2_200[2], Max_vals3_200[2]] # @T=300


# Plot on the first subplot (0, 0)
axs_tip_d[2, 0].bar(D1, d0_eff, label='T=200', color='blue',width=0.1)
axs_tip_d[2, 0].bar(D, d1_eff, tick_label=D, label='T=250', color='red',width=0.1)
axs_tip_d[2, 0].bar(D2, d2_eff, label='T=300', color='green',width=0.1)
axs_tip_d[2, 0].set_title(r'V=200 $[\frac{mm}{min}]$', fontsize=15)
# axs_tip_d[2, 0].set_ylabel(r'Sealing strength $[bar]$', fontsize=15)
axs_tip_d[2, 0].set_xlabel(r'Tip diameter $[mm]$', fontsize=10)
axs_tip_d[2, 0].set_ylim(top=3.2)
```

```
    # Add a horizontal line to show the maximum
    mx = round(max(max(d0_eff, d1_eff, d2_eff)),2)
    axs_tip_d[2, 0].axhline(y=mx, color='black', linestyle='--', linewidth=2)
    # Get the x-axis limits to calculate the midpoint
    x_min, x_max = axs_tip_d[2, 0].get_xlim()
    x_mid = (x_min + x_max) / 2
    # Add text in the middle of the horizontal line
    axs_tip_d[2, 0].text(x_mid, mx, r'$\gamma = %s $'%mx, ha='center', va='bottom',
    color='black', fontsize=10)


    d0_eff = [Max_vals1_300[0], Max_vals2_300[0], Max_vals3_300[0]] # @T=200
    d1_eff = [Max_vals1_300[1], Max_vals2_300[1], Max_vals3_300[1]] #sealing strength
    at T=250 for tip diameters
    d2_eff = [Max_vals1_300[2], Max_vals2_300[2], Max_vals3_300[2]] # @T=300


    # Plot on the first subplot (0, 0)
    axs_tip_d[2, 1].bar(D1, d0_eff, label='T=200', color='blue',width=0.1)
    axs_tip_d[2, 1].bar(D, d1_eff, tick_label=D, label='T=250', color='red',width=0.1)
    axs_tip_d[2, 1].bar(D2, d2_eff, label='T=300', color='green',width=0.1)
    axs_tip_d[2, 1].set_title(r'V=300 $[\frac{mm}{min}]$', fontsize=15)
    axs_tip_d[2, 1].set_ylabel(r'Sealing strength $[bar]$', fontsize=10)
    axs_tip_d[2, 1].set_xlabel(r'Tip diameter $[mm]$', fontsize=10)
    axs_tip_d[2, 1].set_ylim(top=3.2)

    # Add a horizontal line to show the maximum
    mx = round(max(max(d0_eff, d1_eff, d2_eff)),2)
    axs_tip_d[2, 1].axhline(y=mx, color='black', linestyle='--', linewidth=2)
    # Get the x-axis limits to calculate the midpoint
    x_min, x_max = axs_tip_d[2, 1].get_xlim()
    x_mid = (x_min + x_max) / 2
    # Add text in the middle of the horizontal line
    axs_tip_d[2, 1].text(x_mid, mx, r'$\gamma = %s $'%mx, ha='center', va='bottom',
    color='black', fontsize=10)


    plt.subplots_adjust(hspace=1)
    plt.savefig('image/final_figures/sealing_thickness.jpg')
    plt.figure(fig_tip_d.number)
    # fig_tip_d.show()
    # plt.show()
    print("***Run completed. Data are saved.***")

plot_data()
```

F. Sealing quality C++ code

```
const int pressureSensorPin = A0; // Pin connected to the pressure sensor
const int numReadings = 10;     // Number of readings to store
int max_pressure = 0;
int readings[numReadings];       // Array to store pressure readings
int index = 0;                   // Index for the current reading
bool dataCollected = false;      // Flag to indicate if data collection is complete
int Input=1;
void setup() {
  Serial.begin(115200);            // Initialize serial communication at 9600 bits per
    second
  for (int i = 0; i < numReadings; i++) {
    readings[i] = 0;             // Initialize all readings to 0
  }
}

void loop() {
  if (Serial.available()){
    String input = Serial.readString();
    Input = input.toInt();
  }
  if (Input==1) {
    int pressureValue = analogRead(pressureSensorPin);
    Serial.println(pressureValue);
    delay(200);                                        // Wait for 0.2 second before
    taking the next reading
  }
  else {
      Serial.println("data collected");
  }
}

// void printCollectedData() {
//   Serial.println("Collected Pressure Sensor Readings:");
//   for (int i = 0; i < numReadings; i++) {
//     Serial.print("Reading [");
//     Serial.print(i);
//     Serial.print("]: ");
//     Serial.println(readings[i]);
//   }
// }
```

G. Precision test Trajectory G-code

```
M3

G17  ; Select XY plane for circular interpolation
G21  ; Set units to millimeters
;G40 ; cancel diameter compensation
;G49 ; cancel length offset
G54 ; coordinate system 1
G80 ; cancel motion
G90 ; non-incremental motion
G94 ; Feed/minute mode
; Set initial position and feed rate
G00 F1000
G01 F600

G00 X0 Y0

G0 Z-145

G1 Z-149

G01 X0 Y-150
G01 X-170 Y-150
G01 X-170 Y0
G01 X0 Y0

G0 Z-145

G00 X-20 Y-20
G0 Z-145
G01 Z-149

G01 X-20 Y-140
G01 X-160 Y-140
G01 X-160 Y-20
G01 X-60 Y-20
G01 X-60 Y-100
G01 X-120 Y-100
G01 X-120 Y-60
G01 X-100 Y-60

G01 X-20 Y-20

G28
M5
```

H. Sealing quality G-code

```
; THIS GCODE FILE IS MENT TO TEST THE MACHINE PERFORMANCE FOR HEAT SEALING STRENGTH VS
    SPEED, TEMPERATURE AND PRESSURE
; PARTS 1 TO 3 --> TESTS THE MACHINE PERFORMANCE FOR VERTICAL PRESSURE ASCHIEVED AT X=1
     MM AND AT SPEEDS V=100, 200, 300 MM/MIN RESPECTIVELY
; PARTS 4 TO 6 --> TESTS THE MACHINE PERFORMANCE FOR VERTICAL PRESSURE ASCHIEVED AT X=2
     MM AND AT SPEEDS V=100, 200, 300 MM/MIN RESPECTIVELY
M3

G17  ; Select XY plane for circular interpolation
G21  ; Set units to millimeters
;G40 ; cancel diameter compensation
;G49 ; cancel length offset
G54 ; coordinate system 1
G80 ; cancel motion
G90 ; non-incremental motion
G94 ; Feed/minute mode
; Set initial position and feed rate
G00 F1000
G01 F50           ; Set feed rate to 100 mm/min

G00 X-100 Y-100
G00 Z-133 ;F1000


G00 X-15 Y-50

G01 Z-137  ; connect the bed to the soldering iron
G02 X-5 Y-70 I-15 J-20
G01 X-5 Y-150
G02 X-30 Y-150 I-12.5 J-0
G01 X-30 Y-70
G02 X-27 Y-50 R30


G00 Z-133 ;F1000


G00 X-15 Y-50

G01 Z-137
G01 X-15 Y-5
G0 Z-133
G00 X-27 Y-50

G01 Z-137
G01 X-27 Y-5

G0 Z-133 ;F1000

;;; END PART 1
G01 F50

G00 X-100 Y-100
G00 Z-133 ;F1000

G00 X-50 Y-50 ;F1000
G01 Z-137  ; connect the bed to the soldering iron
G02 X-40 Y-70 R30
G01 X-40 Y-150
G02 X-65 Y-150 I-12.5 J-0
G01 X-65 Y-70
G02 X-63 Y-50 R30
```

```
G0 Z-133 ;F1000

G00 X-50 Y-50

G01 Z-137
G01 X-50 Y-5
G0 Z-133
G00 X-63 Y-50

G01 Z-137
G01 X-63 Y-5

G00 Z-133

;;; END PART 2


G01 F100

G00 X-100 Y-100
G00 Z-133 ;F1000


G00 X-85 Y-50

G01 Z-137
G02 X-75 Y-70 R30
G01 X-75 Y-150
G02 X-100 Y-150 I-12.5 J-0
G01 X-100 Y-70
G02 X-97 Y-50 R30

G00 Z-133


G00 X-85 Y-50

G01 Z-137  ; connect the bed to the soldering iron
G01 X-85 Y-5
G0 Z-133 ;F1000
G00 X-97 Y-50

G01 Z-137  ; connect the bed to the soldering iron
G01 X-97 Y-5


G00 Z-133

;;; END PART 3

; G01 F100

G00 X-100 Y-100
G00 Z-133 ;F1000


G00 X-120 Y-50

G01 Z-137
G02 X-110 Y-70 R30
G01 X-110 Y-150
G02 X-135 Y-150 I-12.5 J-0
G01 X-135 Y-70
G02 X-132 Y-50 R30
```

```
G00 Z-133


G00 X-120 Y-50

G01 Z-137
G01 X-120 Y-5
G0 Z-133
G00 X-132 Y-50

G01 Z-137
G01 X-132 Y-5


G00 Z-133

;;; END PART 4


G01 F200           ; Set feed rate to 100 mm/min


G00 X-100 Y-100
G00 Z-133 ;F1000


G00 X-155 Y-50

G01 Z-137
G02 X-145 Y-70 R30
G01 X-145 Y-150
G02 X-170 Y-150 I-12.5 J-0
G01 X-170 Y-70
G02 X-167 Y-50 R30

G00 Z-133


G00 X-155 Y-50

G01 Z-137 F200
G01 X-155 Y-5 F200
G0 Z-133
G00 X-167 Y-50

G01 Z-137 F200
G01 X-167 Y-5 F200


G00 Z-133


;;; END PART 5

; G01 F200           ; Set feed rate to 100 mm/min


G00 X-100 Y-100
G00 Z-133 ;F1000


G00 X-190 Y-50

G01 Z-137
G02 X-180 Y-70 R30
```

```
G01 X-180 Y-150
G02 X-205 Y-150 I-12.5 J-0
G01 X-205 Y-70
G02 X-202 Y-50 R30

G00 Z-133


G00 X-190 Y-50

G01 Z-137  ; connect the bed to the soldering iron
G01 X-190 Y-5
G0 Z-133 ;F1000
G00 X-202 Y-50

G01 Z-137  ; connect the bed to the soldering iron
G01 X-202 Y-5


G00 Z-133

;;; END PART 6

G28

M5
```

I. Origami pattern G-code

```
; This is the G-code template for designing Origami pattern to test the machine's
    perfromance
; in fabricating Origami structures

G21 ; millimeters
G90 ; absolute coordinate
G17 ; XY plane
G94 ; units per minute feed rate mode
M3 S1000 ; Turning on spindle


G00 F1000
G01 F100

; Midline pattern
G00 X-35 Y-10
G00 Z-133
G01 Z-137

G01 X-35 Y-40
; Create rectangle
G01 X-20 Y-40
G01 X-20 Y-110
G01 X-60 Y-110
G01 X-60 Y-40
G01 X-45 Y-40

G00 Z-133
G00 X-45 Y-10
G01 Z-137
G01 X-45 Y-40

G00 Z-133
G00 X-53 Y-55

G01 Z-137
G01 X-27 Y-55

G00 Z-133
G00 X-53 Y-70
G01 Z-137
G01 X-27 Y-70

G00 Z-133
G00 X-53 Y-85
G01 Z-137
G01 X-27 Y-85

G00 Z-133
G00 X-53 Y-100
G01 Z-137
G01 X-27 Y-100

G00 Z-133


;;; New origami pattern: Sideline pattern

G00 X-100 Y-10
G01 Z-137
G01 X-100 Y-40
G01 X-85 Y-40
G01 X-85 Y-110
```

```
G01 X-125 Y-110
G01 X-125 Y-40
G01 X-110 Y-40
G01 X-110 Y-10

G00 Z-133
G00 X-125 Y-55
G01 Z-137
G01 X-110 Y-55

G00 Z-133
G00 X-125 Y-70
G01 Z-137
G01 X-110 Y-70


G00 Z-133
G00 X-125 Y-85
G01 Z-137
G01 X-110 Y-85

G00 Z-133
G00 X-125 Y-100
G01 Z-137
G01 X-110 Y-100

G00 Z-133
G00 X-85 Y-55
G01 Z-137
G01 X-100 Y-55

G00 Z-133
G00 X-85 Y-70
G01 Z-137
G01 X-100 Y-70

G00 Z-133
G00 X-85 Y-85
G01 Z-137
G01 X-100 Y-85

G00 Z-133
G00 X-85 Y-100
G01 Z-137
G01 X-100 Y-100

G0 Z-133

; 3th design ZIG_ZAG pattern


G00 X-150 Y-10
G01 Z-137

G01 X-150 Y-40

G01 X-135 Y-40
G01 X-135 Y-110
G01 X-175 Y-110
G01 X-175 Y-40
G01 X-160 Y-40
G01 X-160 Y-10

G00 Z-133
G00 X-168 Y-55
```

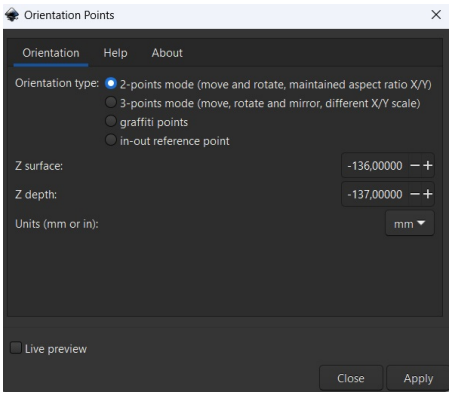
```
G01 Z-137
G01 X-142 Y-55
G01 X-168 Y-70
G01 X-142 Y-70
G01 X-168 Y-85
G01 X-142 Y-85
G01 X-168 Y-100
G01 X-142 Y-100

G00 Z-100
; Turning off spindle
M5
```

J. Hand trajectory G-code

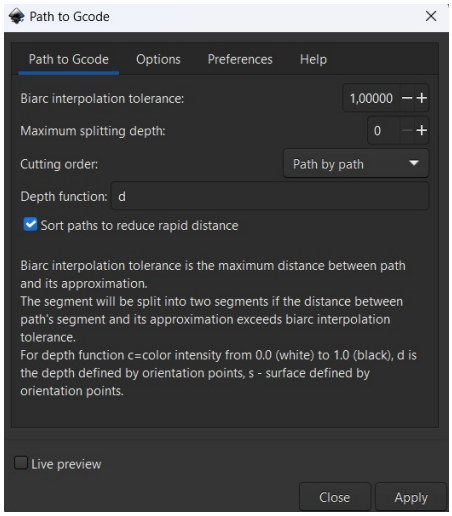
```
G21  ; Set units to millimeters
G17  ; Select XY plane for circular interpolation
; Set feed rate to -- mm/min
G00 F1000
G01 F200
; Set initial position and feed rate
G00 X-45 Y-20   ; Rapid move to starting point (X0, Y0, Z5)

G00 Z-135  ; connect the bed to the soldering iron

G01 Z-137
G01 X-65 Y-17
G01 X-75 Y-15
G01 X-80 Y-12
; G02 X-100 Y-18 R20
G01 X-85 Y-12
G01 X-100 Y-18
G01 X-110 Y-15

G01 X-125 Y-12

G01 X-140 Y-10

G03 X-140 Y-30 R10
G01 X-125 Y-35
G01 X-110 Y-37
G01 X-100 Y-40
G01 X-95 Y-45
G01 X-93 Y-50
G01 X-95 Y-55
G01 X-100 Y-60
G01 X-105 Y-65
G01 X-110 Y-70
G01 X-120 Y-75
G01 X-125 Y-80
G01 X-135 Y-85
G01 X-145 Y-90
G01 X-155 Y-95
G01 X-160 Y-97
G01 X-165 Y-100
G01 X-170 Y-103
G01 X-172 Y-105
G01 X-175 Y-107

G03 X-170 Y-125 I2.5 J-9

; G01 X-135 Y-105

G01 X-130 Y-100

G02 X-125 Y-105 I2.5 J-2.5

G01 X-170 Y-150

G03 X-160 Y-170 I5 J-10
G01 X-140 Y-150
G01 X-110 Y-120

G02 X-105 Y-125 I2.5 J-2.5

G01 X-140 Y-170
```

```
G03 X-125 Y-180 I7.5 J-5

G01 X-90 Y-135

G02 X-85 Y-140 I2.5 J-2.5

G01 X-100 Y-175

G03 X-90 Y-185 I5 J-5

G01 X-60 Y-140

G01 X-50 Y-130

G01 X-20 Y-100

G00 Z-100
G00 X0 Y0

M30
```

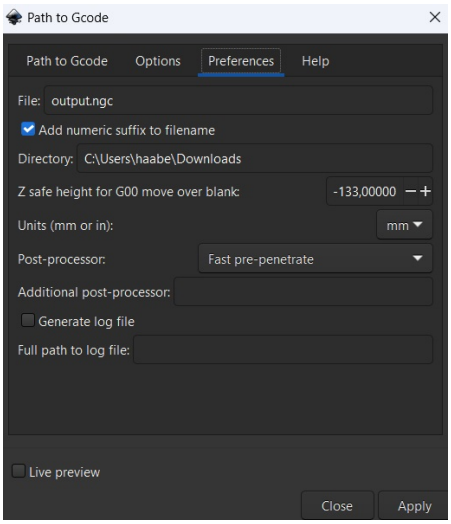
K. Additional Figures



(d) Orientation Points to be used in Inkscape. Adjust the *Z surface* and *Z dept* as shown in this figure.



(e) In the *Path to Gcode* adjust the values to be the same as shown in this figure.



(f) In the *Path to Gcode* navigate to the *references* tab and adjust the values to be the same as shown in this figure.
Hence, you can play with the *Post-processor* tab to obtain the best resulting G-code file.