CORNELL TECH

INFO 5380 - Digital Fabrication
Homework 1

Team Members:

William J. Reid (NetID: wjr83, GitHub_ID: wjr83)
Harshini Donepudi (NetID: hsd39, GitHub_ID: HarshiniDonepudi)
Paul-André Bisson, (NetID: pb583, GitHub_ID: bisson2000)
Khushi Bhansali (NetID: kb737, GitHub_ID: Khushibhansali)

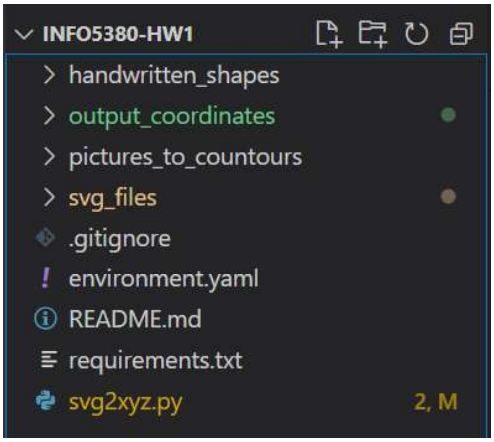
February 23rd, 2024

The maximum number of points is 100, for each of the 5 categories you can earn 16 points (total = 80) and 20 points for well-reported project documentation. If you do an exceptional job in any of the 5 categories we may give up to +4 bonus points for each (to compensate for point losses elsewhere). But we cap the total point count to 100.

[20 points] Project Documentation and Description: either pictures with text were needed, video (link to video make sure instructors have full access -5 points if we need to ask for permission), screenshots of digital design, screencast of the interactive code base, link to code repo (-5 points if we have to ask for permission, clearly mark input and output files).

Description:
The goal of our project was to convert hand-drawn shapes into a readable format that is available on the wire bender. To achieve this, the team created a program that converts images, such as `svg`, `png` and `jpg` to a `csv` that can be understood by the wire bending machine.

Documentation:
The project code is available online on GitHub: https://github.com/bisson2000/INFO5380-HW. In the project Readme, the user can follow the steps in order to set up the project and start using it. Sample handwritten shapes, pictures and svg files are provided in the repository than can be tested. These were specifically chosen to show the capabilities and limitations of the current status of the code (these are outlined in detail later in this document).

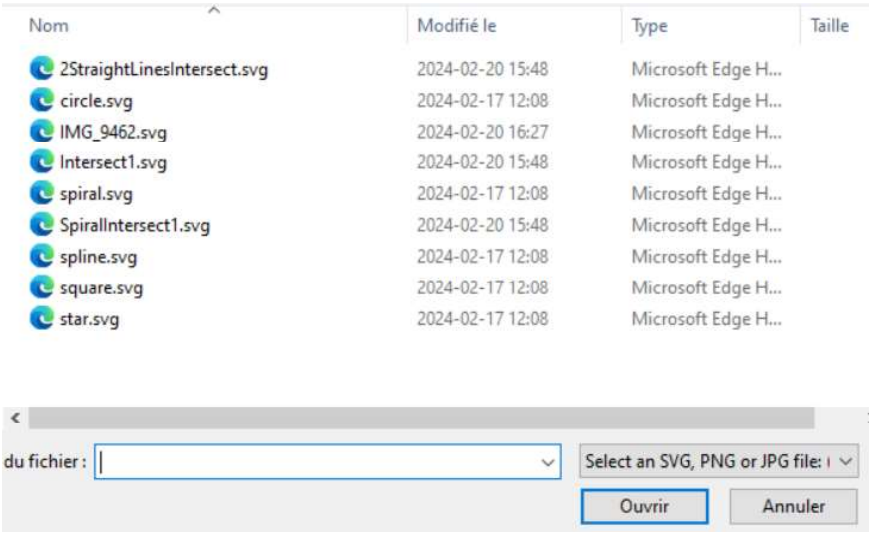


Source Code Structure

To set up the project, the user must first clone the available code on the repository and install the required packages (as outlined in the `requirements.txt`) with Python. Once the project is set up, the user must open a console in the root repository and run the following command: `python svg2xyz.py`

```
Paul@DESKTOP-GLVLR24 MINGW64 ~/Documents/Cornell/Spring2024/INFO5380/INFO5380-HW1 (main)
$ python svg2xyz.py
```

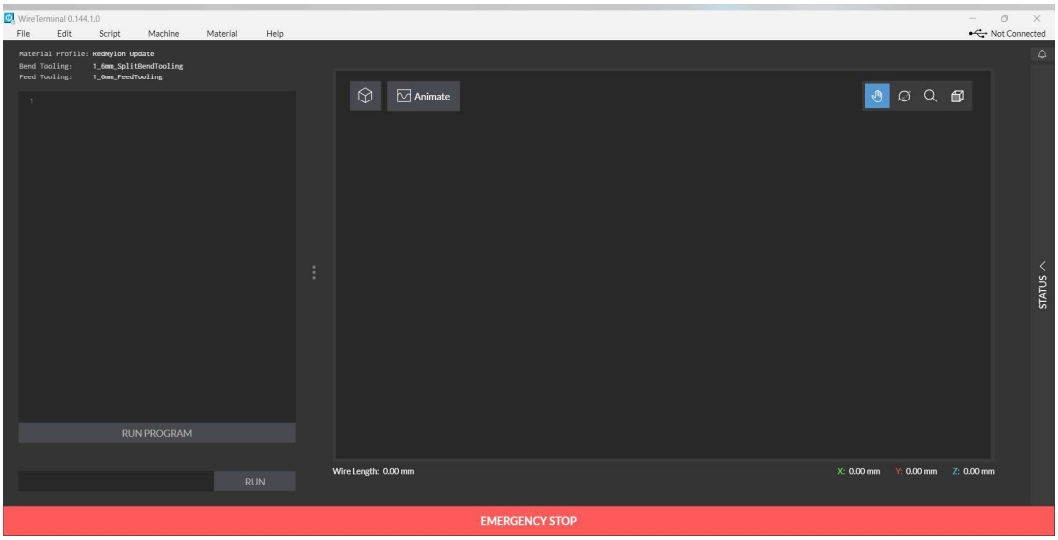
This command will effectively open the file explorer on the user's machine and prompt them to choose a file of type `svg`, `png` or `jpg/jpeg`.



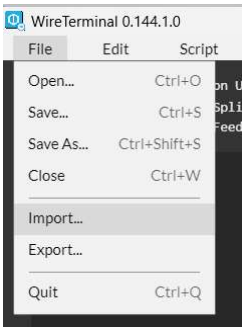
Once the file has been selected, it is then processed by the application, and an output `csv` file is created with the appropriate coordinates to represent the input shape. The output `csv` file will have the same name as the original file except for the extension (as the output file will always be a `csv` file). The newly generated output file will be saved to the output_coordinates folder

```
(cs5670_python_env) C:\Users\Windows\Documents\Cornell Tech\Spring 2024\INFO 5380 - Digital Fabrication\HW1\Git-Repository\INFO5380-HW1>python svg2xyz.py
The selected file is not an SVG file, converting to SVG...
Conversion to SVG completed. New SVG saved to: svg_files/flower.svg
Conversion to XYZ coordinates completed.
Output saved to: output_coordinates\flower.csv
```

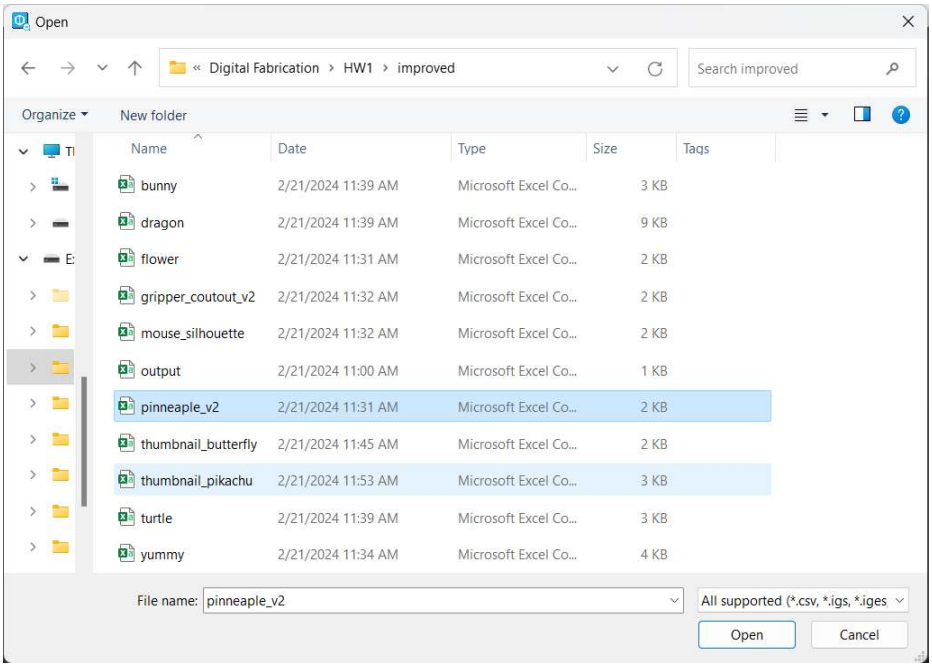
Once this file has been created, it can then be imported into the WireTerminal (the wire bender software). Refer to the screenshots below:

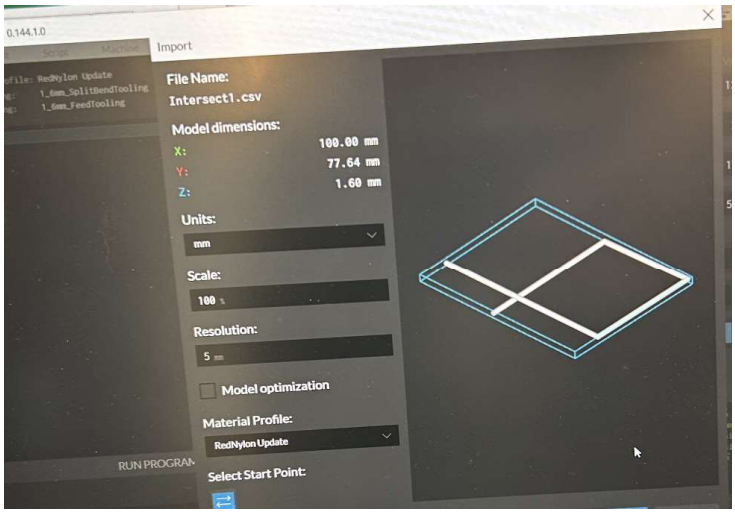


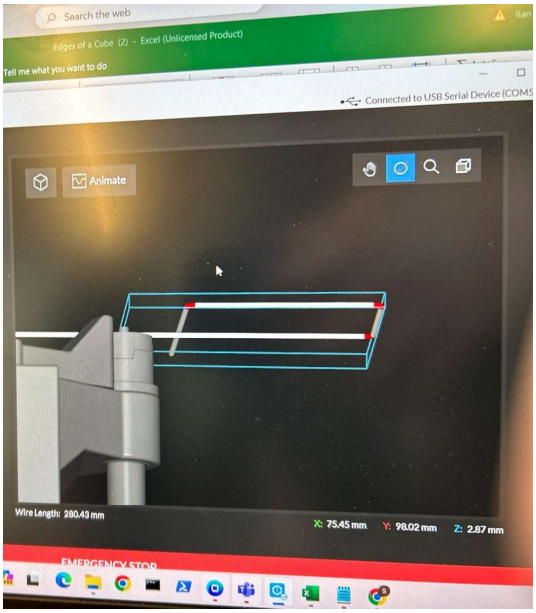
Click on `File` → `Import`:



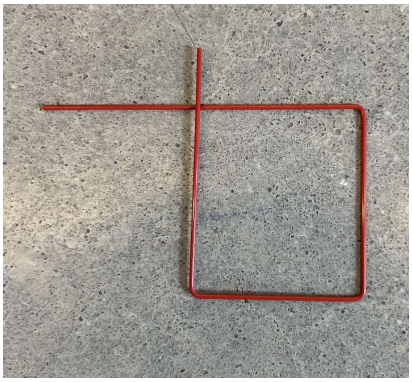
Then, select the desired file:







Once the file has been imported, a preview of the result is shown, and the user can start the wire bender in order to create the shape.



Explanation of the steps in the algorithm:

The computer vision steps used to process hand-drawn shapes involve several key processes. The color image is first converted into a grayscale image to simplify processing and reduce computational complexity. A Gaussian blur filter is then applied to reduce noise and smoothen the image, aiding in accurate edge detection. The Canny edge detection algorithm identifies the edges of the shapes by detecting areas of high-intensity variation, typically corresponding to object boundaries. These detected edges are dilated to ensure better connectivity and achieve a single-pixel width, preserving the integrity of the shape's outline. Contours are extracted from the dilated edges, representing the shape boundaries and essential for further processing.

To achieve the most faithful representation of the hand-drawn shapes while minimizing unnecessary complexity, various epsilon values were tested during the polygon approximation process. The epsilon parameter controls the approximation accuracy, with smaller values resulting in more detailed representations and larger values yield simpler approximations. By systematically testing different epsilon values (0.0004, 0.0002, 0.0001, 0.0015), we aimed to find the optimal balance between accuracy and simplicity in the resulting SVG paths. Each epsilon value corresponds to a different level of smoothing applied to the contours, influencing the fidelity of the shape representation. Through this iterative testing process, we sought to identify the epsilon value that best preserved the essential features of the hand-drawn shapes while minimizing unnecessary intricacies in the SVG paths. This optimization step ensures that the final SVG representations closely match the original shapes while maintaining a manageable level of complexity for downstream processing and manufacturing. See screenshots further in this document as an example.

The extracted contours are then approximated with simpler polygonal curves to reduce complexity while preserving essential features. This polygonal approximation is converted into SVG path data format, embedded within a new SVG file, serving as the machine-readable representation of the hand-drawn shape. To ensure compatibility with the wire bender, an algorithm is implemented to address intersections and close elements. If intersections are detected, subsequent points are adjusted in the Z-axis to remove them, ensuring proper

operation of the wire bender. While this adjustment may alter the design's appearance from a horizontal view, the result remains the same when viewed from the top.

[16 points] Ambition: describe in a paragraph (10 lines max) what you set out to achieve and why this is challenging (ok if you did not accomplish all of it) 8 points you gain by submitting an approved project proposal 2 weeks before the homework!

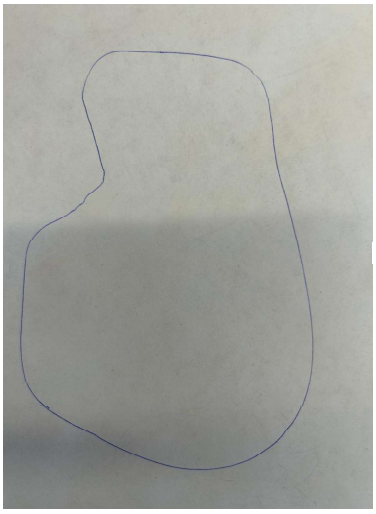
We aimed to create a user-friendly wire bender plugin for generating machine code from hand-drawn wire shapes, bridging the gap between manual sketching and wire-bending machinery. Challenges included adding depth, handling overlapping lines, handling gaps in images, considering wire type and tension, and ensuring compatibility with the required file format. The scalability of hand-drawn sketches involved a detailed workflow with steps like scanning, cropping, SVG conversion, raster-to-vector path transformation, and path rendering optimization. Each step required careful implementation to address the complexities effectively. We were able to overcome most of these challenges by only focusing on 1 item in the image, by removing overlapping lines, scaling images to an established value, eliminating images with a gap, and testing images on the wire bender with multiple iterations to make sure it doesn't collide with the machine. However, when looking at hand-drawn letters, the code isn't able to detect the letters correctly leading to skewed images and inaccurate results. Lastly, the only challenge we have yet to conquer is ensuring that the wirebender correctly bends the wire. Despite feeding precise coordinates, the wire gets misaligned and the machine isn't able to bend the wire to the best of its ability.

[16 points] Depth/Execution: describe in a paragraph (10 lines max) how much of the ambition you managed to achieve and what elements you are particularly proud of + why.

The script serves the purpose of converting hand-drawn shapes, captured as image files (JPG/JPEG and PNG), or vector graphics (e.g., SVG files) into machine-readable XYZ coordinates. It begins with the user selecting a file using a file explorer prompt. If the chosen file is an SVG, the script directly extracts the path data and converts it into XYZ coordinates, saving them to a CSV file. For image files, the script undergoes a two-step process. First, the image is transformed into an SVG file by detecting edges, smoothing the shape, and approximating it with polygonal curves. Then, this SVG file is processed to obtain the XYZ coordinates, which are subsequently saved to a CSV file (X is column 1, Y is column 2, and Z is column 3). The script handles various types of curves encountered in SVG files, including straight lines, arcs, and Bezier curves, while addressing challenges like overlapping lines and unconnected shapes. Its ultimate goal is to streamline the generation of wire-bent products by bridging the gap between hand-drawn sketches and wire design manufacturing. We are proud of having created a working pipeline that can take hand-drawn & digital images and convert them into a format understandable by the WireTerminal. We are also proud of the fact that our software reduces the barrier to entry and allows new users to test the Wire Bender with little domain and/or machine knowledge.

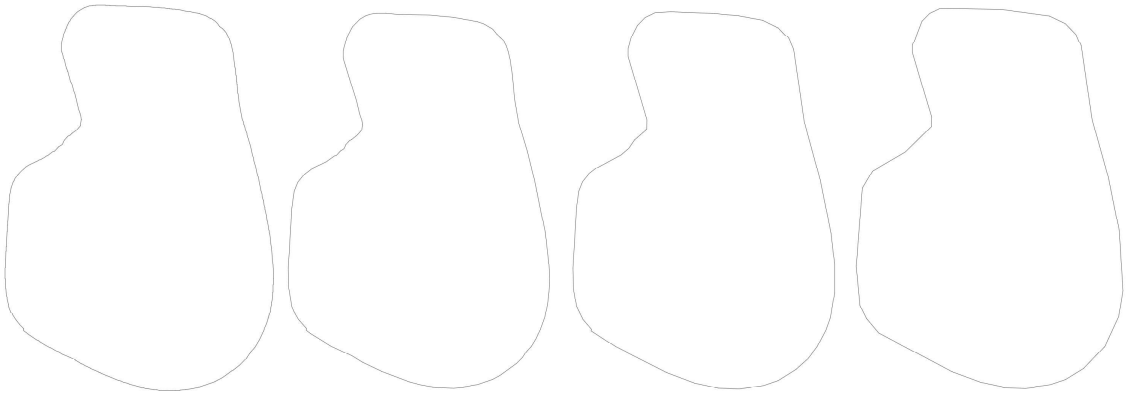
In summary, the script we developed simplifies the conversion of hand-drawn shapes into machine-readable XYZ coordinates, thereby facilitating the manufacturing process of wire-bent products. By seamlessly handling both image and SVG files, it offers a versatile solution accessible to users with varying levels of technical expertise. We take pride in the creation of a functional pipeline capable of translating hand-drawn and digital images into a format compatible with the WireTerminal, effectively lowering the barriers to entry for new users. Our accomplishment lies in the integration of computer vision algorithms and 2D mathematical concepts to achieve this outcome. Several examples are detailed below.

Hand-Drawn Images



Hand-drawn contour of computer mouse.

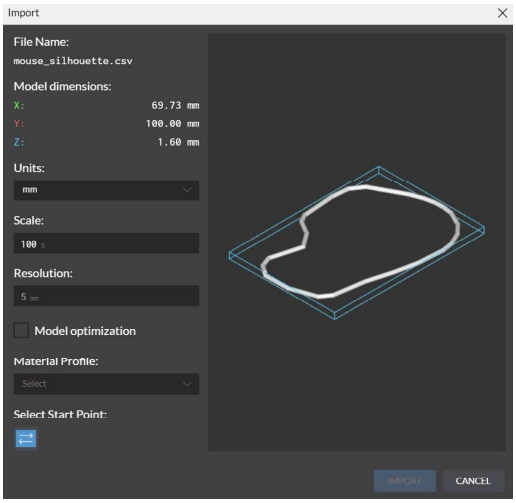
Conversion of hand-drawn contour of computer mouse to `svg` *format:*

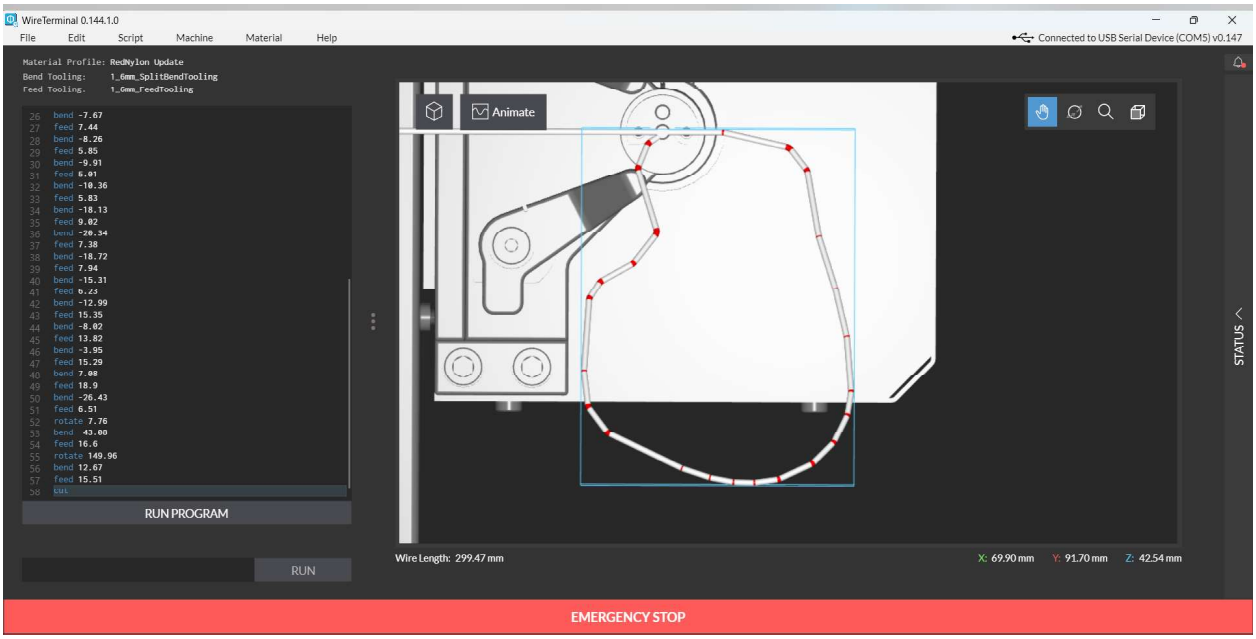


epsilon = 0.0004 epsilon = 0.0002 epsilon = 0.0001 epsilon = 0.0015

Conversion of hand-drawn contour of computer mouse to svg format: here, various values of epsilon are used to determine the precision at which the image is detected. A higher value will create more “square” edges, while a lower value will increase the amount of detail. After testing many different shapes, the default value was set to epsilon = 0.0015 to remove “fuzzy lines” for straighter lines.

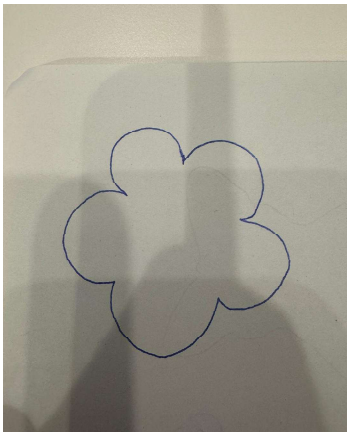
Wire Terminal Import:





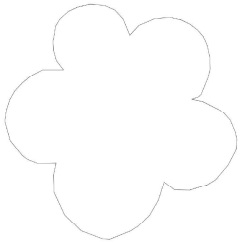
As we can see, the import of this shape yields the expected results in the WireTerminal software. Next, we detail several of the shapes tested.

Shape: flower



(A)

(B)

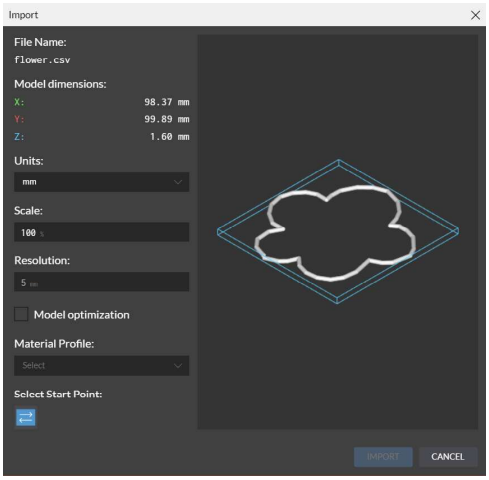


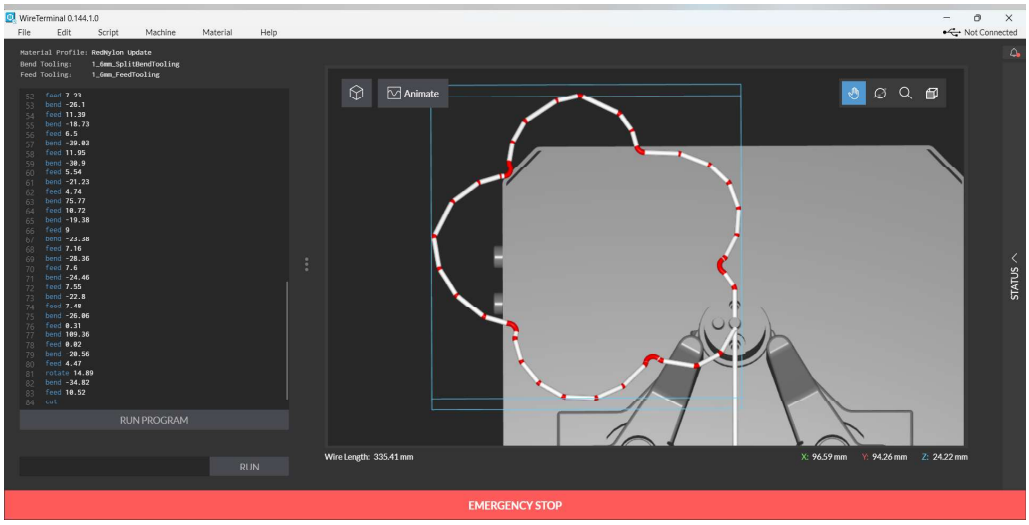
(C)

Key:

A. Hand-drawn flower.
B. Shape extracted using computer vision algorithms.
C. Shape squeezed to single pixel and converted to `svg` format.

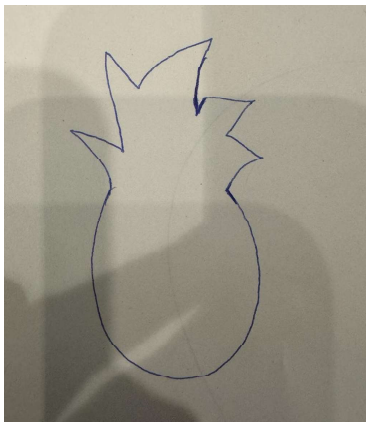
Wire Terminal Import:

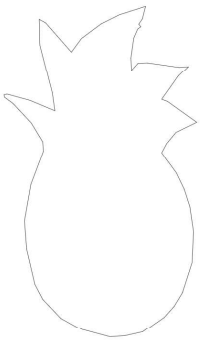




From the results of the image above, we can see there are many curves created in the output result. The final shape still resembles the original one.

Shape: pineapple



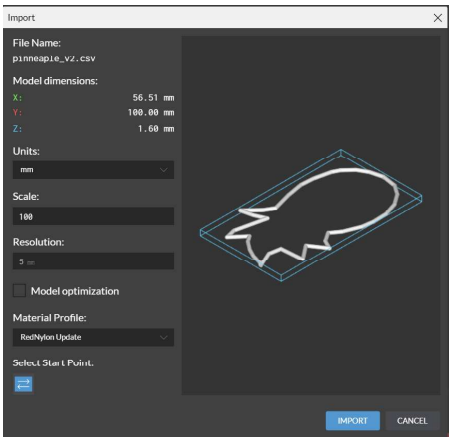


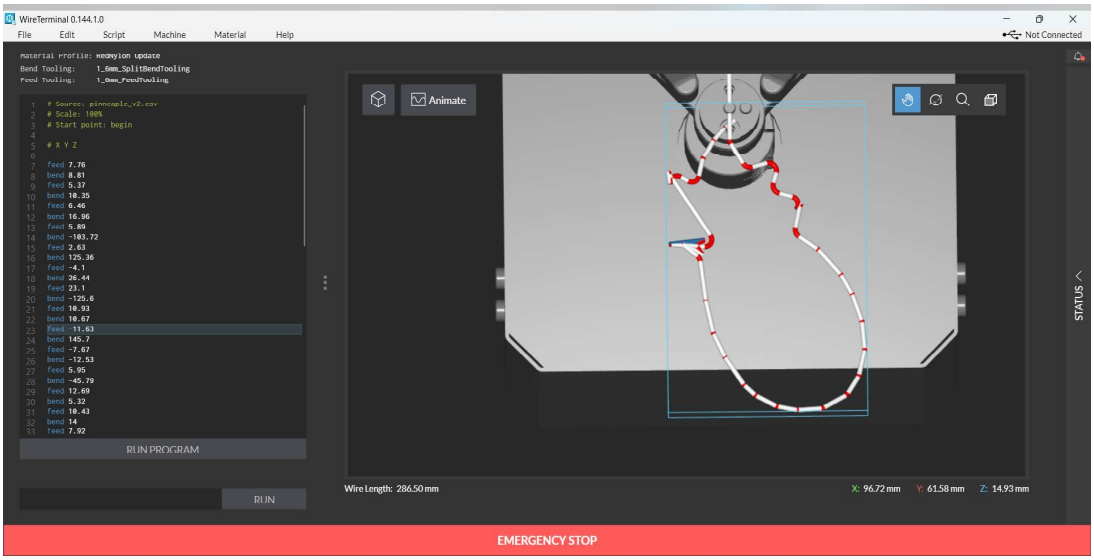
(D) (E) (F)

Key:

D. Hand-drawn pineapple.
E. Shape extracted using computer vision algorithms.
F. Shape squeezed to single pixel connected path and converted to `svg` format.

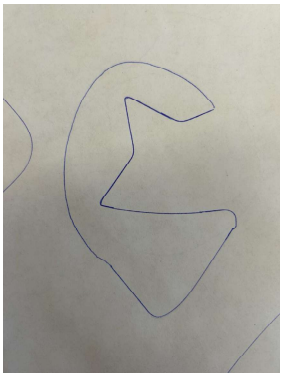
Wire terminal import:





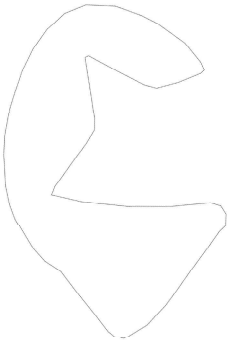
Although the image appears to be correctly imported, there are still some errors within the result shown on the screen. As one can see, some of the leaves of the pineapple are missing and have been misinterpreted.

<u>Shape:</u> gripper cutout



(G)

(H)

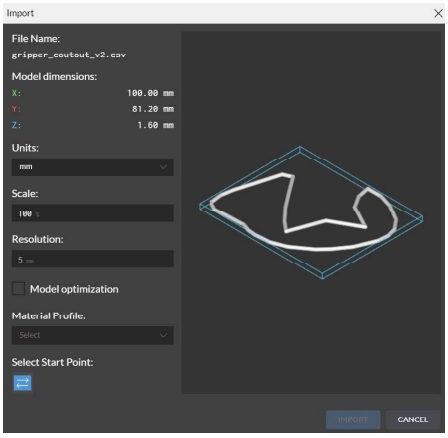


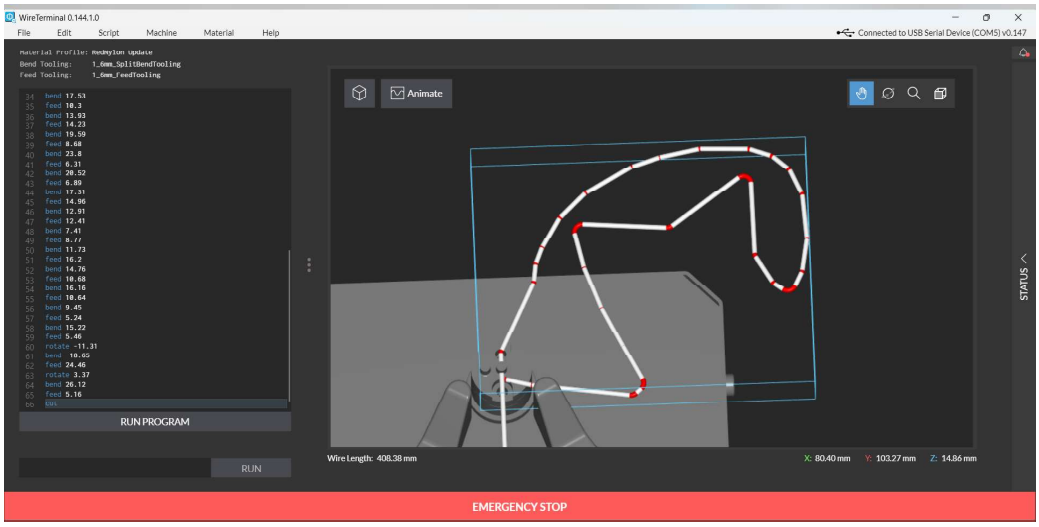
(I)

Key:

G. Hand-drawn gripper.
H. Shape extracted using computer vision algorithms.
I. Shape squeezed to single pixel connected path and converted to `svg` format.

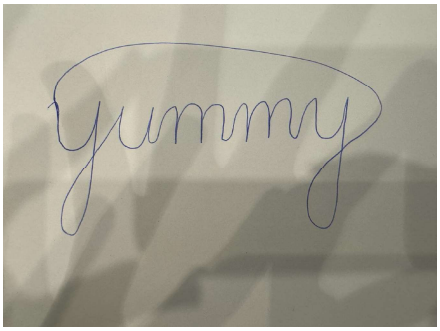
Wire Terminal Import:

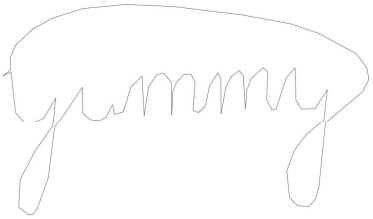




As we can see, the import of this shape yields the expected results, despite the angles.

Shape: yummy





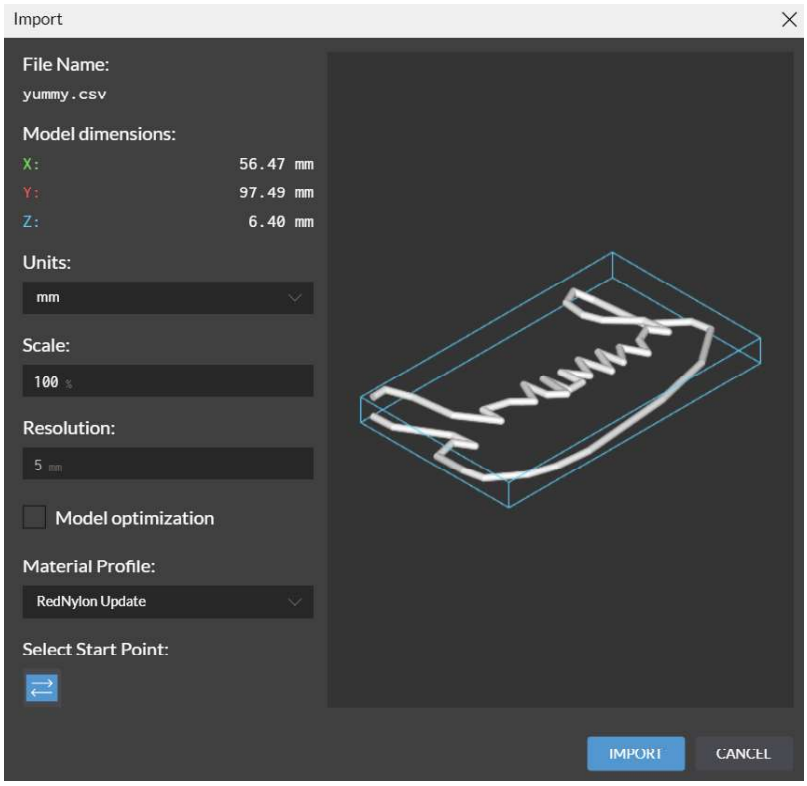
(J) (K) (L)

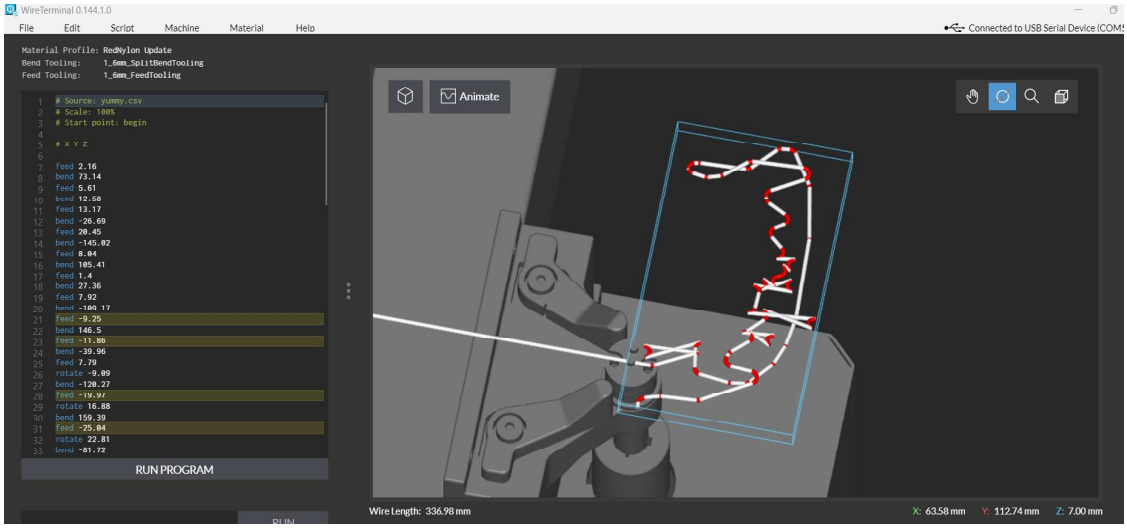
Key:

J. Hand-drawn artistic “yummy”.
K. Shape extracted using computer vision algorithms.
L. Shape squeezed to single pixel connected path and converted to `svg` format.

Despite the word complexity, the yummy shape was converted successfully.

Wire Terminal Import:

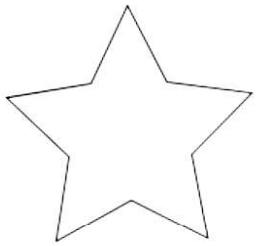




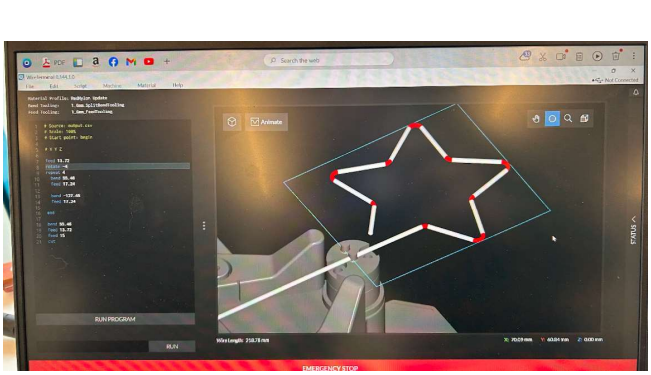
When initially importing the yummy image, we can see there are some modifications made within WireTerminal, which alter the basic shape. However, once imported, the drawing changes once again. This is mainly due to the small angles which are close together. To circumvent this, more preprocessing should be added to our program.

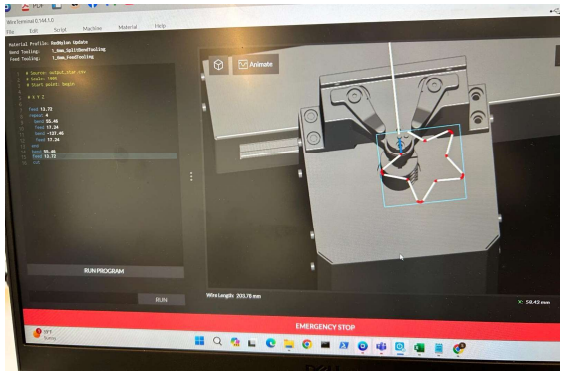
Images that are digitally sketched (and are filled-in):

Shape: star (`svg` file imported)



Wire Terminal Software Import:



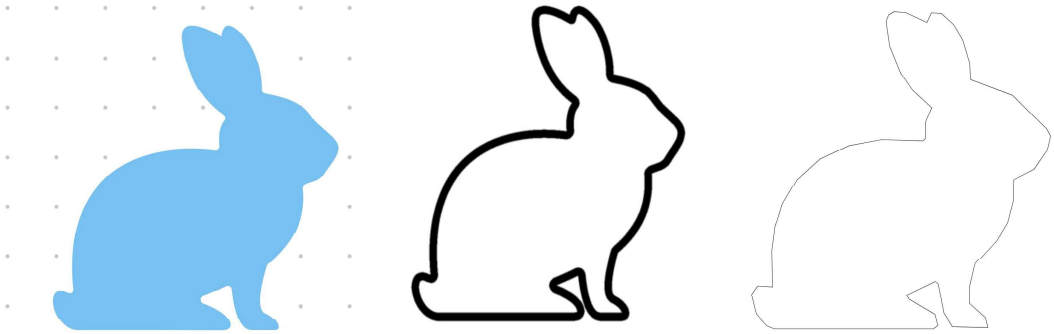


Machine Output:

We can see that the program has no problem interpreting a star, and that the output in the WireTerminal is correct. However, when it comes to bending the star, it seems like there are issues with the machine. Indeed, it is not configured correctly for the wire, preventing us from properly printing a star. Furthermore, it seems like the wire tends to be stuck in the machine when it is reorienting itself, further preventing us from printing it. The last image above shows several attempts where the machine tried to bend the wire into the star (all using the same code the WireTerminal software generated from the csv file containing the coordinates).

This video https://drive.google.com/file/d/1FIZSyjYGJEw82nPpjcTn3eql-URxDRZZ/view?usp=sharing shows approximately the result. Here's another example of the star wire bend: https://drive.google.com/file/d/1iOVtSN7Zmcgpe2cCJW6d2-NMnrtbiATI/view?usp=sharing
Here's another attempt at the star wirebend: https://drive.google.com/file/d/1po8TIsuWuixrHqbHy6cFcJKvirCsUFH-/view?usp=sharing

Shape: bunny

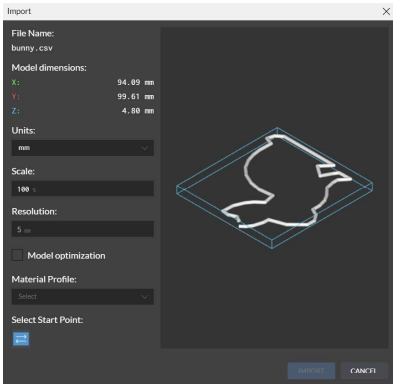


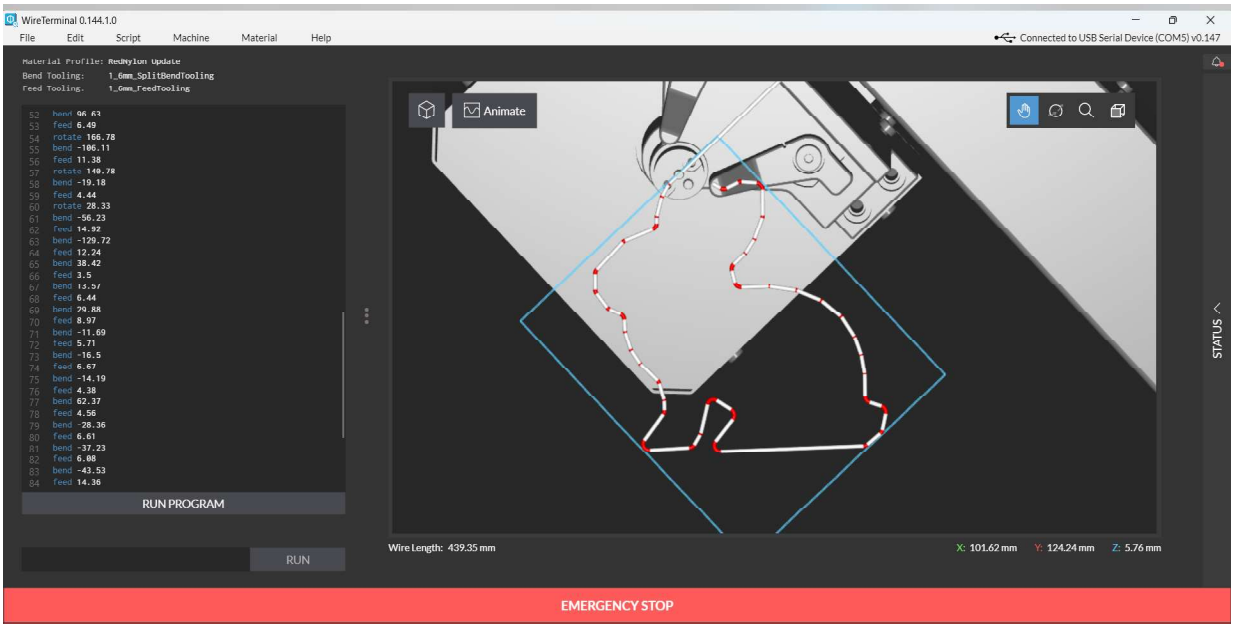
(M) (N) (O)

Key:

M. Image of a bunny.
N. Shape extracted using computer vision algorithms.
O. Shape squeezed to single pixel and converted to `svg` format.

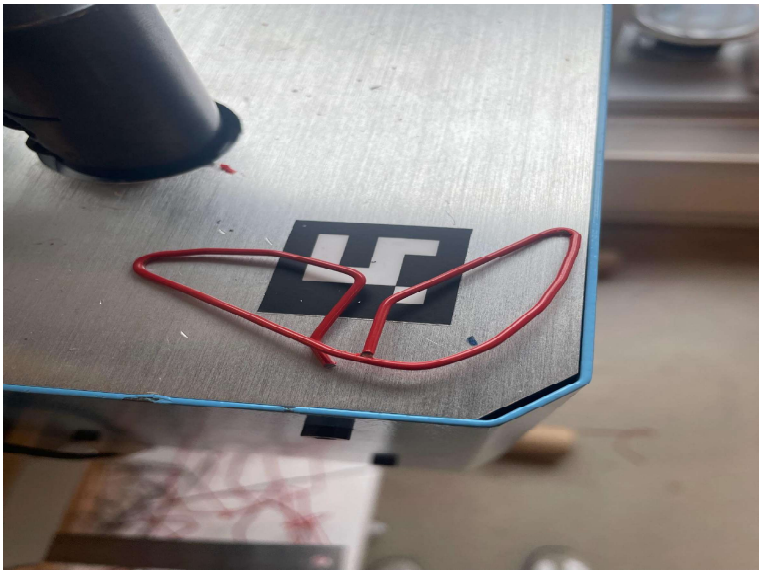
Wire terminal output:



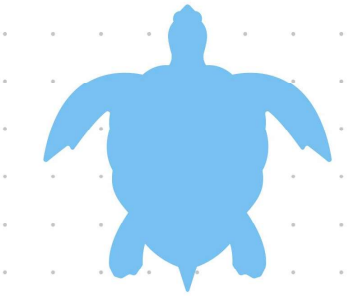


From the results of the image above, we can see the result is an accurate representation of the input, with all its details.

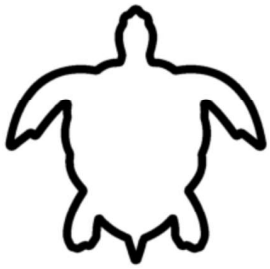
In the following video https://drive.google.com/file/d/1fJMP8O1HGWbwLLMNBm99IoBBgTx-78I-/view?usp=sharing , we can see one of the tries we did for the bunny. Here's another attempt of the bunny wirebend: https://drive.google.com/file/d/1mrVftZpzIcFjozfIluXpDdEr0bjpI1WQ/view?usp=sharing. However, as mentioned before, the machine is imperfect and it actually shows it missing the very first wire bend for the shape.



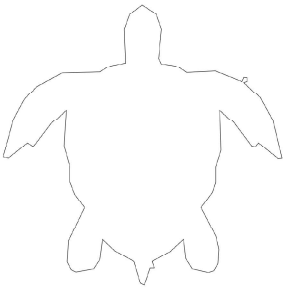
Turtle:



(P)



(Q)

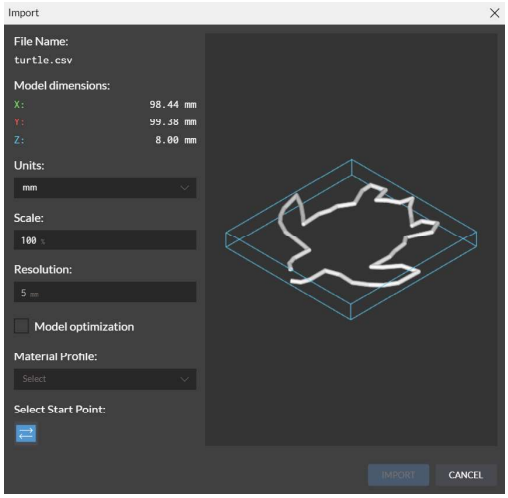


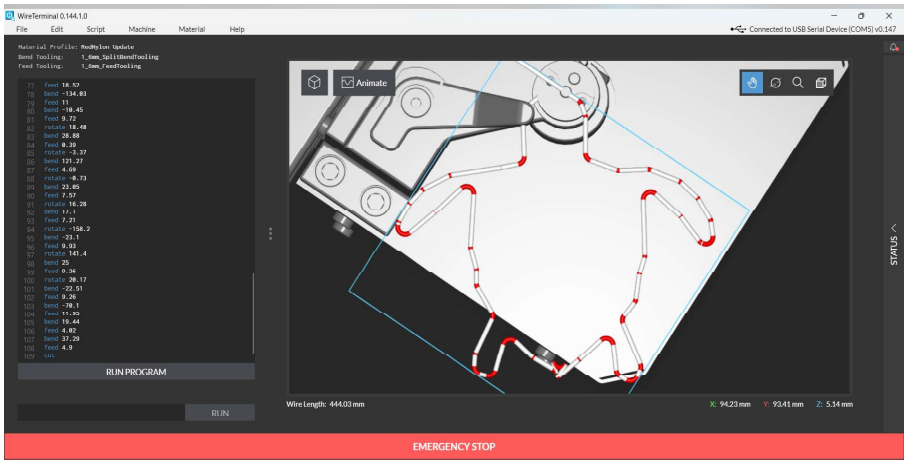
(R)

Key:

P. Image of a turtle.
Q. Shape extracted using computer vision algorithms.
R. Shape squeezed to single pixel and converted to `svg` format.

Wire terminal output:

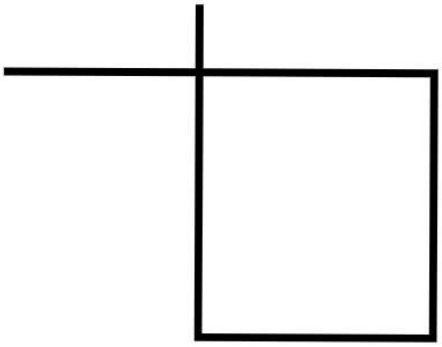


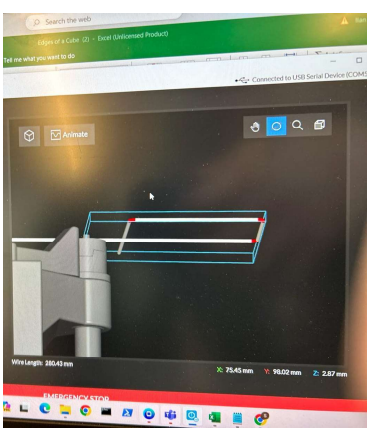


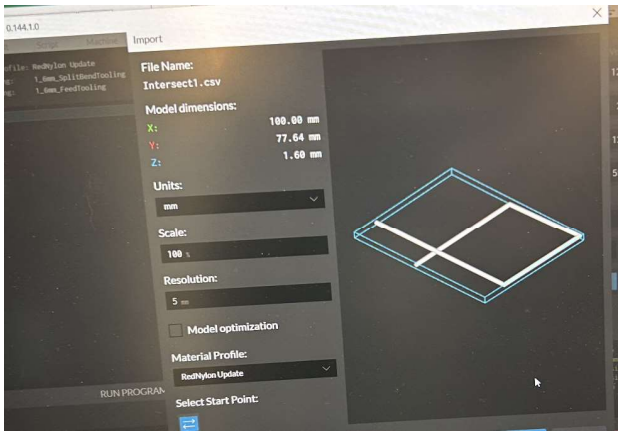
From the results of the image above, we can see the result is an accurate representation of the input. However, it seems like the tail of the turtle is missing. In other words, although the details are there, the points that are too close to one another are causing issues.

Images with Intersections

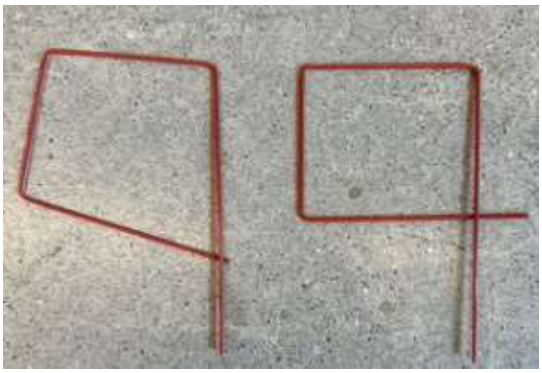
Intersecting square (`svg` input file):

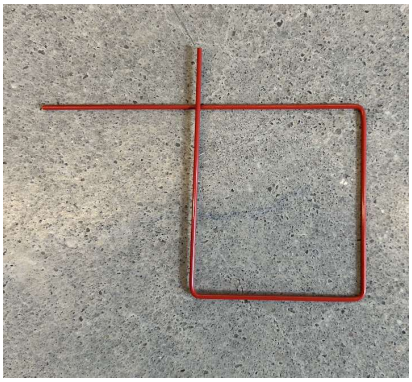






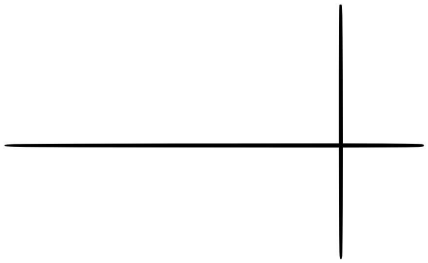
In the images above, we can see that the square, despite the intersection it has, is well handled by our software. Indeed, intersecting points are layered on top of each other. The result is an accurate representation of the shape in the WireTerminal.





Above are the result of the wire bending for the intersecting square.
In this video: https://drive.google.com/file/d/13DkupjRY7desWVImf3EcA-m55__ptXLJ/view?usp=sharing , we can see the results of wire bending for the shape on the left. As one can see, the angles at which the shape is bent are incorrect. From our analysis, it seems like this was caused by the configuration of the wire in the machine itself. Here's an example of a successful square build:
https://drive.google.com/file/d/13bImkGJukGqEHLAXM-9fu8mJBgnMuS-I/view?usp=sharing

Intersecting lines:



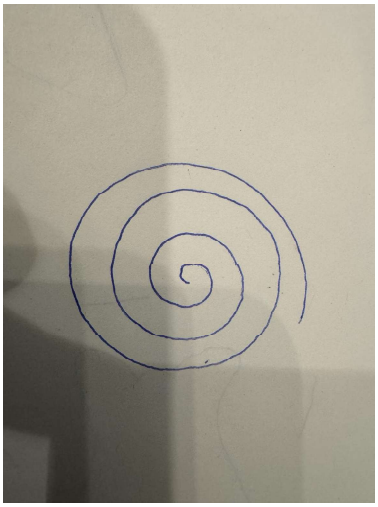
This shape is more complex (for our algorithm) than the previous ones, as it does not form a closed loop or contour and has intersections. The following video illustrates how the program imports it:

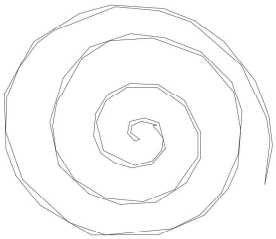
- https://drive.google.com/file/d/1a3zaYkgn8v0heIW6_2ffhwXCQRA1EWKN/view?usp=sharing

Although it seems correct at first glance when importing it, as soon as the program loads it and generates the bendscript code, the shape seems to change. In other words, even though the program seems to take care of the points, there is more preprocessing that should be done to allow for such shapes to be rendered properly by the WireTerminal software.

Additional limitations of our algorithm are detailed next:

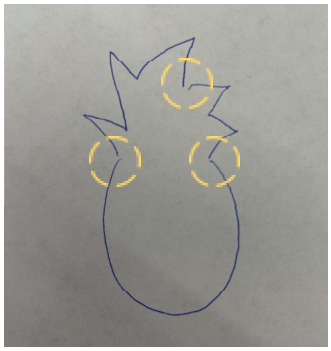
Open geometries:



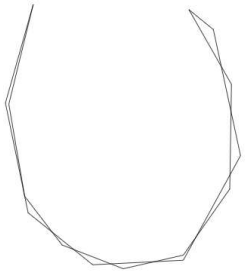


Here, the spiral is an open geometry. However, when generating the resulting `svg`, we can see two different lines are zig-zagging on top of one another. The reason behind this is that the algorithm tries to reach the center of the geometry from an arbitrary point, and then tries to backtrack from it to complete the shape.

Unconnected shapes:



(S)



(T)

Key:

S. Hand-drawn pineapple.
T. Shape squeezed to single pixel and converted to `svg` format (from shape extracted using computer vision algorithms).

Despite the appearance, the crown of the pineapple does not intersect with the base shape (as shown by the yellow circles), creating an output that took the largest contour (the base) and attempted to close it (removing the smaller contour, being the crown of the pineapple).

Reflecting surfaces and “streaky” writing utensils:

(U)

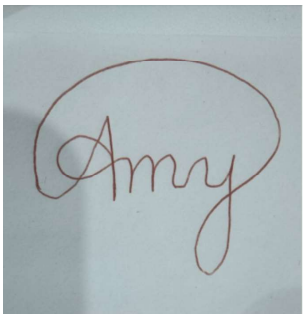
(V)

Key:

U. Hand-drawn Mickey Mouse.
V. Shape extracted using computer vision algorithms.

Drawings on reflective surfaces, like whiteboards, and those made with markers or tools with varying color intensity, posed challenges in generating accurate outputs. The reflective nature of whiteboards and differences in color intensity affected edge detection, resulting in inconsistencies in the generated paths.

Complex Names:



(W)

(X)

(Y)

Key:

W. Hand-drawn artistic “Amy”.
X. Shape extracted using computer vision algorithms.
Y. Shape squeezed to single pixel connected path and converted to `svg` format.

Note: Though the algorithm can handle the intersecting “y” loop well, it struggled with the more complex “A” drawn, removing the top portion of the “A” in “Amy” when converting from the extracted pixel path to `svg`.

The examples above show the limitations of our algorithm, or in other words, the necessary criteria of the input image for successful `svg` conversion and coordinate generation. The current requirements are: a closed shape drawn on a non-reflective surface without streaky lines (i.e.g, as those resulting from a marker on a white board).

[16 points] Relevance: describe in a paragraph (10 lines max) what material from the lectures/activities you used in your homework assignment and how.

Our homework assignment was heavily influenced by insights gained from Marco Perry's guest lecture on wire-bending machines at PensaLabs. Through this lecture, we deepened our understanding of the wire-bending processes and became aware of the limitations of the machine he contributed to developing. Practical experimentation during the Valentine's Day activity, where we utilized the wire bender Fusion 360 plugin, further illuminated the challenges faced by novice users in achieving precise wire bending results, especially with intricate designs. Combining Marco Perry's lecture with the insights from the Personal Fabrication lecture, which highlighted the importance of abstracting domain and machine knowledge in system adoption, provided us with a holistic perspective on wire bending challenges. One key insight from these experiences is that wire-bending models must follow a single, continuous path, which can be challenging to identify in complex drawings. To address this, we opted to convert drawings into the SVG format, leveraging its ability to define a clear path for wire bending. Additionally, we learned that wire-bending models cannot intersect with themselves during the bending process, necessitating compromises in the representation of its design. To mitigate this issue, we implemented a solution to elevate (e.g., bend) the design when intersections occur, ensuring a consistent top view and preventing collisions with the machine. Overall, these insights have guided our approach to designing wire-bending models that are both accurate and machine-compatible, enhancing our understanding of wire-bending challenges and solutions.

[16 points] Process: describe in a paragraph (10 lines max) how you approached the challenge, how your design/code changed over time, where you may have iterated, and how you would do it differently if you were to do this again.

We began our project by delving into the intricacies of `SVGs`, aiming to grasp their structural composition and how these graphical representations can be translated into coordinates. Early on, we encountered the challenge of interpreting displacement in the Z-axis, which led us to streamline our approach by confining the coordinate system to the XY-plane, effectively setting the Z-coordinate to zero. This decision was made to align with the constraints of representing 2D drawings in wire bending processes. To convert hand-drawn shapes into digital format, we opted to process simple images using Gaussian blur and Canny edge detection techniques. These methods accurately delineated the contours of the depicted shapes, albeit without filling them. The transformation to grayscale, a crucial step in this process, inadvertently amplified image noise. To mitigate this issue, we strategically employed dilation and erosion techniques to refine the contours, focusing on isolating the largest detected contour while disregarding smaller, noise-induced ones.

In refining our approach, we experimented with various smoothing algorithms and thresholds to address drawing imperfections and achieve faithful contour representation. We explored additional techniques such as mixed blur using Median Blur alongside Gaussian Blur and Thresholding to clean out noise, particularly in images with vivid background colors and complex bends.

The challenge of scaling, inherent in computer vision applications, prompted us to explore OCR technology for direct dimension interpretation from the drawings. While this venture initially yielded disproportionate images, it provided insights for potential improvement and refinement.

Recognizing the limitations of our scaling approach through OCR, we established a standardized scale of 10 cm by 10 cm for our models. This decision provided a consistent frame of reference, simplifying the scaling process amidst broader challenges.

As we progressed, we aimed to work with more complex shapes, including those featuring intersections. This prompted significant modifications to our code to accurately process these more intricate shapes.

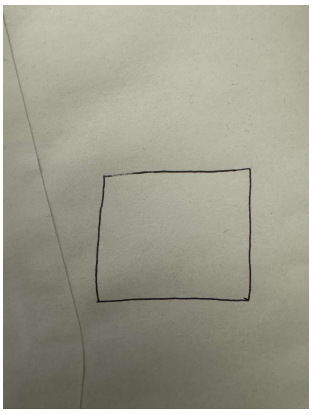
Throughout our project, we encountered significant issues with the wire bender machine. Attempting to wire bend our generated SVGs revealed challenges of the machine's execution, particularly when shapes had bends too close to each other. This resulted in the machine losing hold of the wire, leading to missed turns and eventual collisions, compromising the output shape.

In hindsight, we acknowledge the need for a better understanding of the terminal software and its limitations in converting coordinates to bendscript commands and sharp corners to bends all while ensuring that the coordinates follow a specific path that the machine can execute without

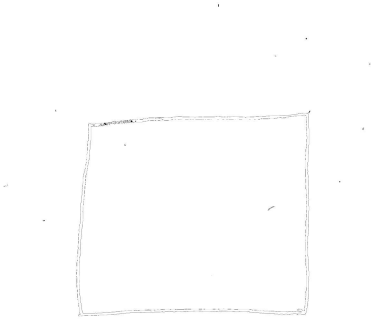
error (e.g., collisions). Additionally, understanding the mechanisms that would help address the hardware issues of the machine such as missing the wire in a simple turn, that is, one that it shouldn't have a problem with (encountered a lot throughout this project), is essential for future iterations of this project. Otherwise, no matter how well we can convert shapes into machine-readable coordinates, if the machine is not capable of wire-bending the shape correctly, then our contribution will not be adopted nor used by the community of users. Additionally, exploring the possibility of taking coordinates in the other two planes (XZ and YZ planes rather than XY plane) may offer solutions to some of the issues when certain shapes caused collisions with the machine during wire bending processes. In other words, the machine limitations could be incorporated into the algorithm such that a close approximation to the shape desired can be properly wire bentby the machine.

Below is a work in progress in the initial stages of of project:

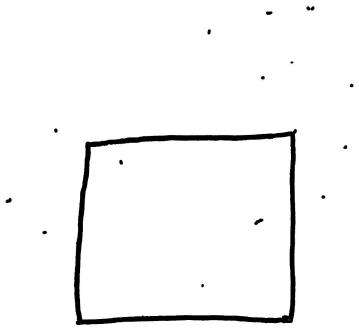
Hand-drawn shape of a square:



Processed shape (initial stages):

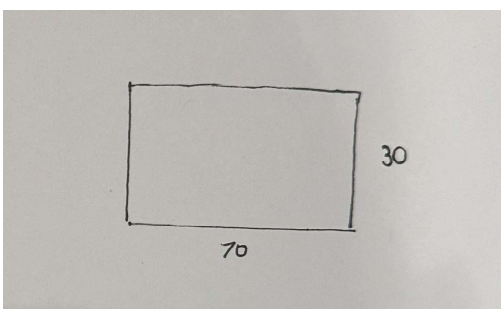


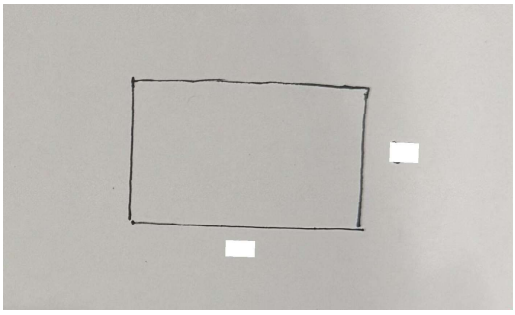
Initial Detected Shape before setting thickness



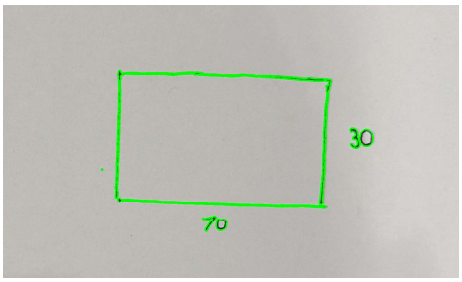
Shape after setting the thickness with noise

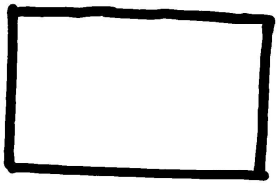
OCR Detection for scaling:



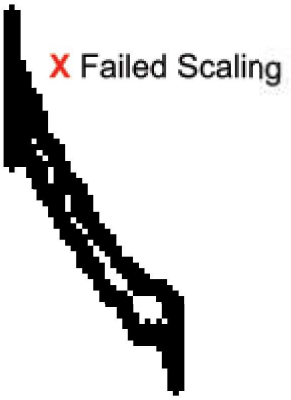


OCR Detection Output: `Detected dimensions {'30', '70'}`

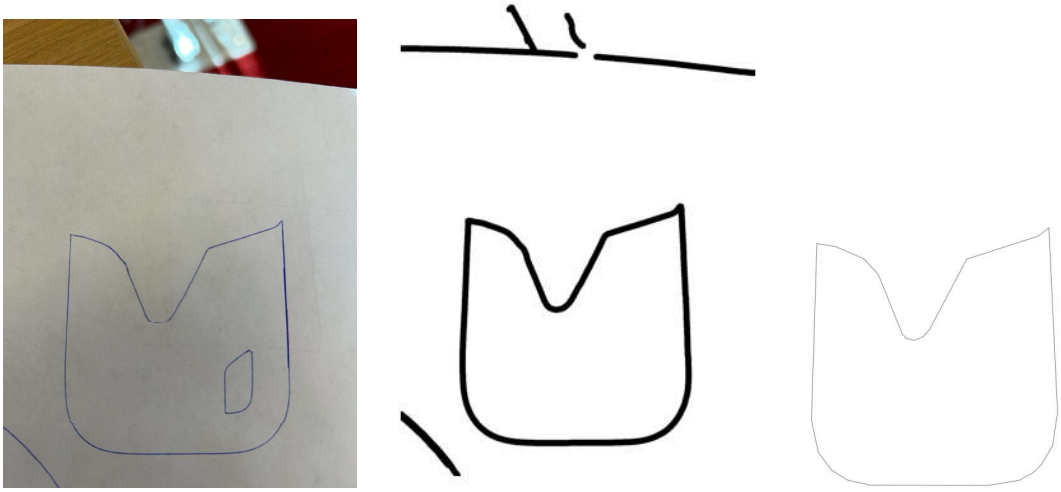




Initial scaling attempts would deform the shape (as shown below with the rectangle depicted above):



Background subtraction and largest object contour extraction:



(Z) (AA) (BB)

Key:

Z. Hand-drawn shape.
AA. Shapes extracted using computer vision algorithms.
BB. Shape squeezed to single pixel connected path and converted to `svg` format.

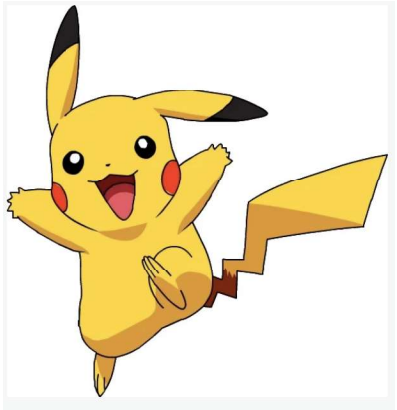
SVG's of complex pictures with fills and vivid colors :

(CC) (DD) (EE)

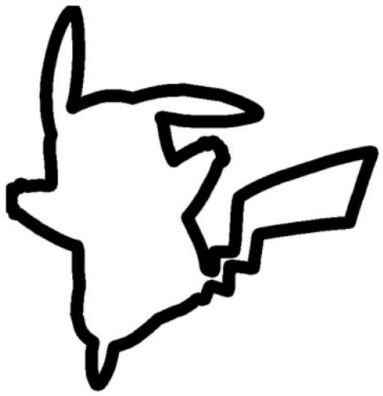
Key:

CC. A computer-generated image of a butterfly.
DD. Shapes extracted using computer vision algorithms.

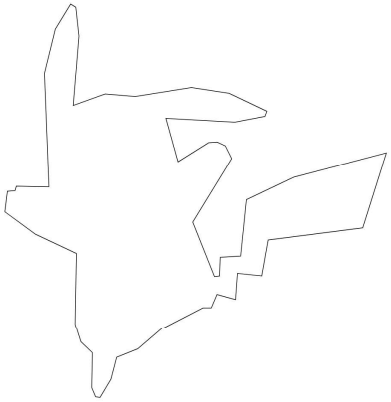
EE. Shape squeezed to single pixel connected path and converted to `svg` format.



(FF)



(GG)



(HH)

Key:

FF. A computer-generated image of a Pikachu.
GG. Shapes extracted using computer vision algorithms.
HH. Shape squeezed to single pixel connected path and converted to `svg` format.

[16 points] Teamwork: describe in a paragraph (10 lines max) how you decided to form the team you work with, what unique qualities each of you have, and how you used these together to make this project a success. “We did everything together” is not good enough.

We decided to form the team considering William and Harshini’s strong background in research for healthcare, technology, and interactive devices, Paul’s interest in 3D design software and physical engineering of objects, and Khushi’s interest in designing interactive devices. We were interested in making this project happen because of how wide-scale the applications are: dentistry, baking/cooking, art/sculpture, automotive, jewelry, etc. To complete the project each divided this assignment into smaller functions where Paul wrote a function that handles overlapping lines and scales coordinates to produce a shape of X size on the wire bender. Harshini and William wrote code that converts `png`/`jpg` of hand drawings to `svg` and then `svg` to coordinates. They also considered edge cases where drawings had unconnected objects, causing the `svg` coordinates to have gaps/conflicts. They continued furthering their contributions by devising a solution that would tell the machine to cut the wire before proceeding with bending the next shape. This task was put on pause in favor of producing a csv file for each of the unconnected shapes identified. Khushi worked a function that converts heic images to `png` and converts digitally drawn images to `svg` coordinates by removing background color.