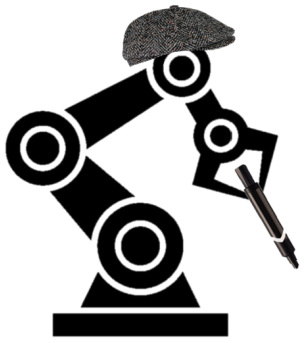


MADbot

Manipulation and Drawing Robot

Madhu Korada
Dhanesh Pamnani
Alec Trela

16-662 Robot Autonomy: Final Report

May 5, 2023

Carnegie Mellon University
The Robotics Institute

Table of Contents

Motivation

The motivation of this project is to pass a robot a string input, for which it could reproduce that text upon a canvas. The immediate system has scalability in terms of perception, manipulation, and learning. With more work this robotic system could replicate user handwriting and replicate their penmanship, which may have implications for assistance of users with degenerative neurological conditions such as Parkinson's Diseases.

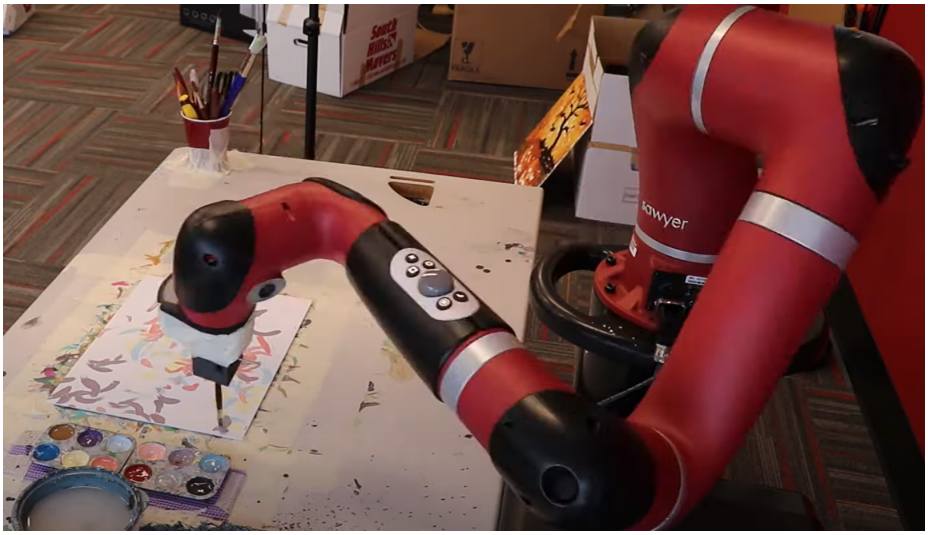


Fig 1. *FRIDA robot used to replicate learning generated art (link)*

Furthermore, this project can inform future work in drawing applications, which may have similar rehabilitation applications to both assist the aforementioned affected individuals, but also other groups of disabled individuals. We foresee a robotic system that could use deep learning models [2, 3] to generate art for users with physical limitations to allow them to express themselves creatively under a new avenue. This inspiration is drawn from FRIDA, a robot generated by the Bot Intelligence Group of CMU [1].

Key Challenges

In terms of challenges, we had three that persisted throughout the course of the project: dealing with transformations, working with a flexible brush, and waypoint generation. Another additional challenge was faced by our perception subsystem, however since this subsystem did not have time to get integrated it will only be touched on lightly.

Transformations

For a short time we faced difficulties in transforming our text generated waypoints onto the world-frame. This was due in part because we did not have an out-of-the-box visualizer for where our waypoints existed in the world frame. After generating a visualizer using matplotlib, (see overview of approach), we were able to circumvent these difficulties.

Brush

One of the persistent issues of working with this application was the use of the brush for writing. The inherent properties of the fibrous brush provided unique challenges that are not seen in a traditional pen or marker. Brushes are flexible, meaning that depending on the force and angle applied to them, they may produce a different deposition on a canvas. Additionally, their absorption properties may vary. This required a certain attention and credence be paid to the amount of time the brush is allowed to absorb water, how much the brush should be cleansed of residue liquid, as well as what height the brush would be placed relative to the board height.

Waypoint Generation

Closely related to our previous challenge is waypoint generation. Generating a smooth trajectory that can be closely mirrored by the flexible brush required a high level of waypoints to be tracked. This also required close observation of the rate of waypoint tracking, since our setup leveraged a Buddha board (Figure 2).

Fig 2. *Buddha board used in our setup, which helped to satisfy our reset condition*

Without properly assigned timestamps, our drawings would begin to dry up, generating an illegible script. Noticeable time and attention was paid to the balance between speed of tracking and gross quantity of waypoints to ensure that all waypoints could reasonably be tracked without the script drying up and without stressing joint velocity limits of the arm.

Overview of Approach

During the ideation stage of this project, a cyber physical architecture was generated to create a structure that is both reliable and scalable (Figure 3). The large portions that were challenged during this project were: string manipulation, coordinate transformation, and waypoint generation/tracking.

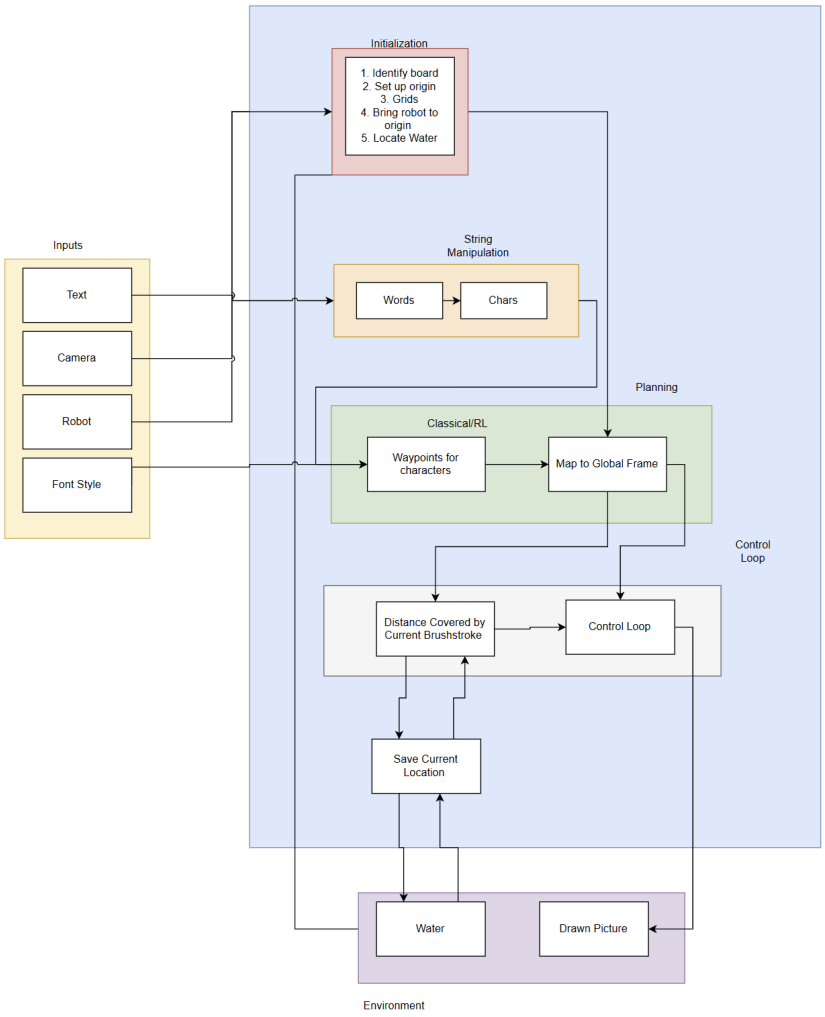


Fig 3. *Cyberphysical architecture generated as a scaffold for development*

Currently, our system takes in a string command, which is then parsed and processed to generate high-level information about the text related to major vertices and how they are connected (curves, lines on the board, or lines lifted off the board). This information is then transformed to the world frame before being transformed into waypoints. These waypoints are followed in a systematic cycle which includes wetting of the brush. The system is also currently ready to be integrated with a vision subsystem, which while developed, has not currently been integrated. Please refer to "Future Work" for more information.

Methods Implemented

String Manipulation

Given a string containing letters and numbers, x/y coordinates are needed for a robot to write it on the Buddha board. To achieve this, a set of coordinates associated with all the letters and numbers is required. There are two major steps involved in the process of acquiring x/y values and plotting them on a matplotlib canvas, they are described below.

Part 1: Preprocessing - Parsing the x,y values of all the alphabets, numbers from the TTF file.

Font file processing: The **fontTools** library was utilized to read the TTF file and extract glyph information such as glyph ID, glyph name, and glyph commands. The library provides functionality to access the font's internal tables and data structures.

Glyph path extraction: The glyph path commands and points for each character in the input text were extracted using the fontTools library. These commands include *moveTo*, *lineTo*, *curveTo*, *qCurveTo*, and *closePath*, which define the shape of the character outlines.

Conversion to single-stroke coordinates: Generating single-stroke paths from traditional font files can indeed be challenging. An alternative approach is to use a font specifically designed for single-stroke rendering. These fonts are often called single-line fonts, stroke fonts, or engraving fonts. In our case, a single-line font TTF file was used to generate single-stroke coordinates that resemble human writing. The single-line font contains glyphs with minimal pen lifts and continuous paths, making it suitable for robotic writing applications.

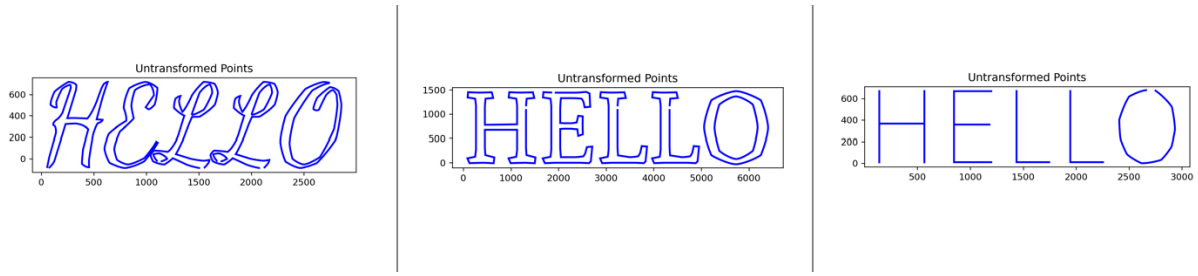


Fig 4. *Generated untransformed coordinated for dancing script, Relief Single Line Regular, and sura fonts - respectively*

Part 2: Inference Time - Querying the x,y values of the characters of the string

Path transformation: The extracted glyph paths were adjusted to account for character spacing and positioning. This involved updating the x_offset for each character based on the advance width obtained from the font's horizontal metrics table.

Coordinate extraction and visualization: The final XY coordinates were extracted from the transformed single-stroke glyph paths and visualized using the Matplotlib library. Custom functions such as extract_vertices_codes(), plot_all_points(), render_word(), and plot_single_glyph() were used to create a visual representation of the processed glyph paths.

Map String to World Frame

After obtaining relevant information regarding the letters that encapsulate the string to be written, as well as encoding its major vertices and waypoint commands (lift, place, line, curve), the vertices were mapped to the world frame. This first required scaling of the points, then shift of the points, followed by an inverse of x/y values and a negation of the x coordinate. This will be explained in detail moving forward.

Coordinate Scaling

In order to scale the points appropriately, the axes limits from the untransformed plots (Figure 4) were obtained and linearly scaled in relation to the board dimensions - with a tuned scaling parameter. The formula is described below:

$$(1)\ scale_x = 0.6 * height_{board} / (x_{max} - x_{min})$$

$$(2)\ scale_y = 0.6 * length_{board} / (y_{max} - y_{min})$$

Here, the height and length of the board were hand measured and the minimum and maximum x, y values were returned from the matplotlib graph shown in Figure 4. The 60% scaling parameter was influenced by visual inspection of outcomes, which when drawn at 100% either dried too quickly or had a visually unappealing aspect ratio for strings with large characters.

Coordinate Shift

The coordinate shift was taken by obtaining the end-effector position, when grasping the Buddha board brush, as the brush contacted the bottom left hand corner of the board. This value was simply added to the scaled coordinates described above.

Inverting and Negation

After this point, the visualization of coordinates was complete. However, the axes of the visualizer and the robot's frame are not aligned. The x-axis and y-axis were flipped. After flipping the axes, the positive y-axis was in the opposite orientation of our negated terms. As such, we proceeded to negate this term to ensure that the points plotted by the visualizer corresponded to the correct world frame coordinates.

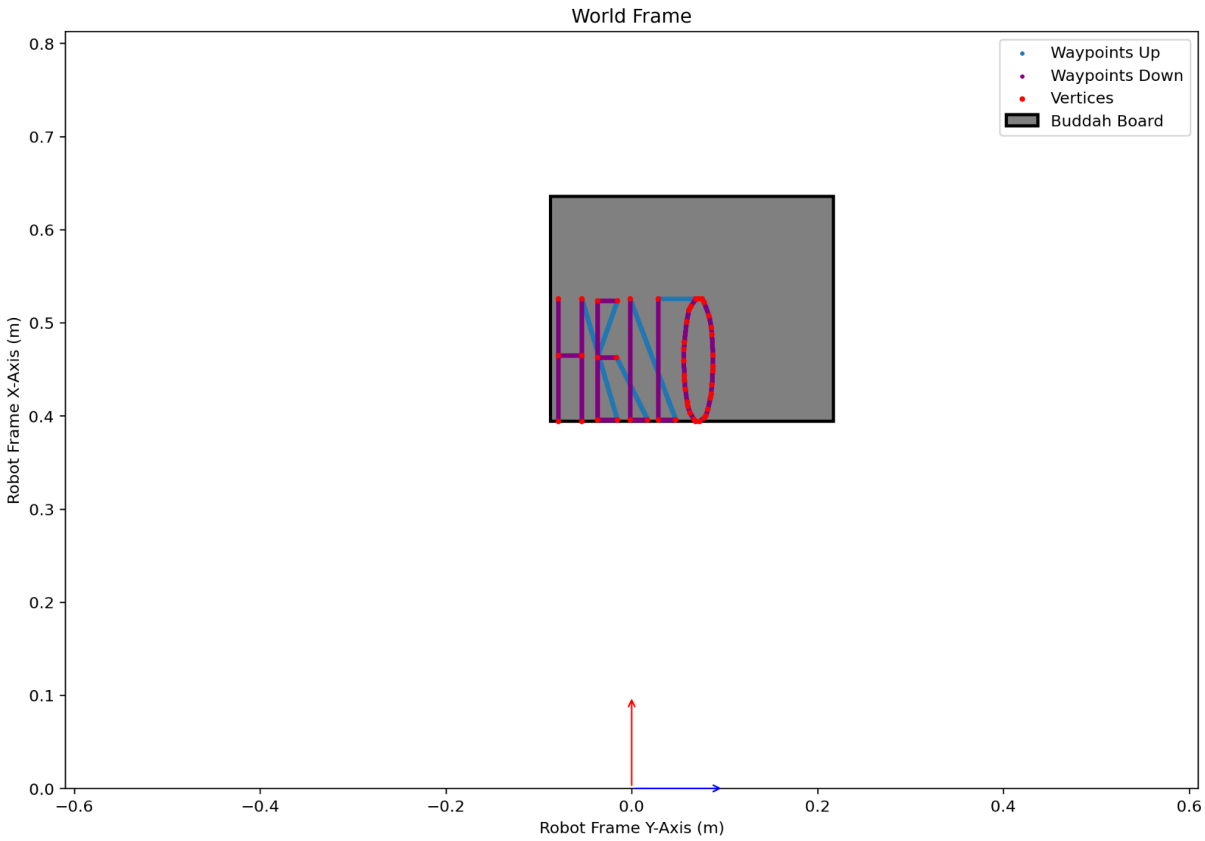


Fig 5. *Generated transformed coordinates for the Relief Single Line Regular font using our custom visualizer*

The visualizer shown in Figure 5 plots all the waypoints that are tracked by the Franka arm, the generation of these waypoints is explained below. It is also worth noting that the axes limits chosen for the visualizer are based on the limits defined by the Franka's workspace, which was obtained by measuring the length of whiteboard in front of the Franka, as well as the distance from the base of the Franka to the end of the whiteboard.

Generate Waypoints for the String

With the vertices successfully transformed to the world frame, we next needed to generate waypoints to be tracked. At this point we have properly transformed vertices as well as encoded information about the tasks to be performed between them (lift or draw).

Between two vertices, the waypoints to be tracked were formed via the following method:

1. Calculate the Euclidean distance between two vertices
2. Convert the distance to centimeters
3. Multiply the number of waypoints to generate between the two vertices relative to the distance between them
    a. By default, there were 30 waypoints generated per 1 cm of distance.
4. With the number of waypoints specified, the waypoints were generated through linear interpolation
    a. If the two points were aligned horizontally, simple linear interpolation was not possible since the slope between them is undefined. As such, a simple ranged step was generated with a constant x-coordinate.

However, the above interpolation method did not include any z-axis coordinate. The current state of our system only considers two states: either drawing or lifting. From the string manipulation work, we already obtained information relating to whether or not the points between vertices should be lifted or drawn. As such, we used the encoding between waypoints to inform us whether the third axis in the task space should be a lift value or a draw value. The draw value was obtained from the coordinate described in the “Coordinate Shift” subsection: by placing the end-effector on the corner of the Buddha board while holding the brush. The lift height was simply this height plus 1 cm. The generated waypoints gave us the following outputs in the visualizer (Figure 6).

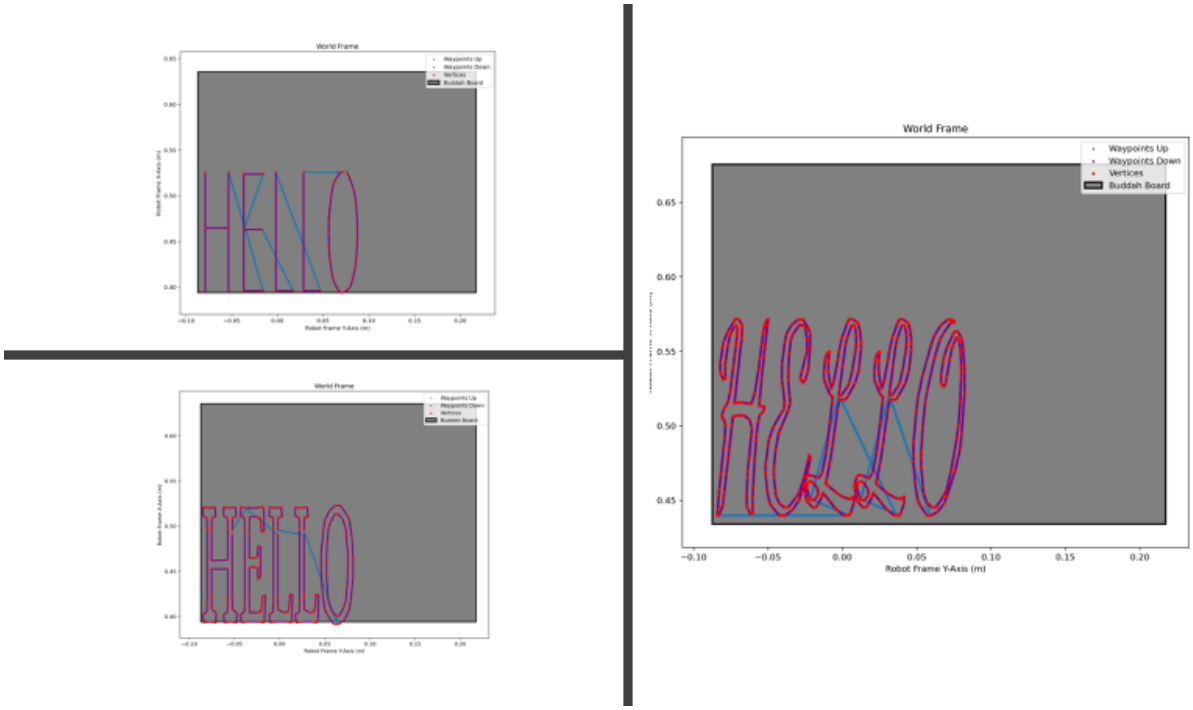


***Fig 6**. Generated (and zoomed) transformed coordinates for dancing script (right), Relief Single Line Regular (top left), and sura fonts (bottom left)*

After preliminary testing using this method, we also noticed that the brush began to dry up, even though the brush seemed suitably saturated with water. To resolve this issue, we included a decrement for drawn points of 0.001 mm. This was a tuned parameter, informed by the total number of waypoints generated for most paths - which was around 5000. This equated to roughly a decrease of 5 mm across the course of the drawing procedure. We found this to be suitable as the brush has a length of approximately 20 mm.

Track Waypoints

After generating the waypoints, the system was prepared to write letters. It did so following the following routine:

1. Go above the water bowl
    a. This was a predefined location some suitable height over the centroid of the bowl using the go_topose() routine as defined by the frankapa library
2. Dip the brush
    a. The same centroid location, with a reduction of height such that the brush was completely submerged using the go_topose() routine as defined by the frankapa library
3. Brush over the side

    a. Some predefined location that allowed the brush to be run lightly over the side of the bowl to remove excess water from the brush tip. Without such a maneuver, the first brush stroke would result in a sizable blob leading to unrecognizable scripts. This was done, again, using the go_topose() routine as defined by the frankapa library
4. Maneuver to the first waypoint
    a. This was obtained from the first waypoint in an array of waypoints, and again leveraged go_topose()
5. Track the waypoints
    a. Using the waypoints that were previously described, using a mildly altered version of run_recorded_trajectory.py
    b. The relative impedance gains were tuned to command more rigid tracking of the arm: the final values used were 1600 along each axis
    c. The trajectory tracking functionality requires a translational component and quaternion. The translational component has already been described, and the quaternion was a constant value that described the end-effector as it was positioned during the reset pose.
    d. Here, we also tuned the task duration and time-step to ensure that the arm was moving fast enough to finish writing all characters.
6. Reset the joints to await next instructions

Demonstration and Evaluation

Characters

During the onset of this work, a preliminary step we took was to track waypoints using simply the goto_pose(). However, this reported jerking motions of the arm for curved motions between the vertices (Figure 7).

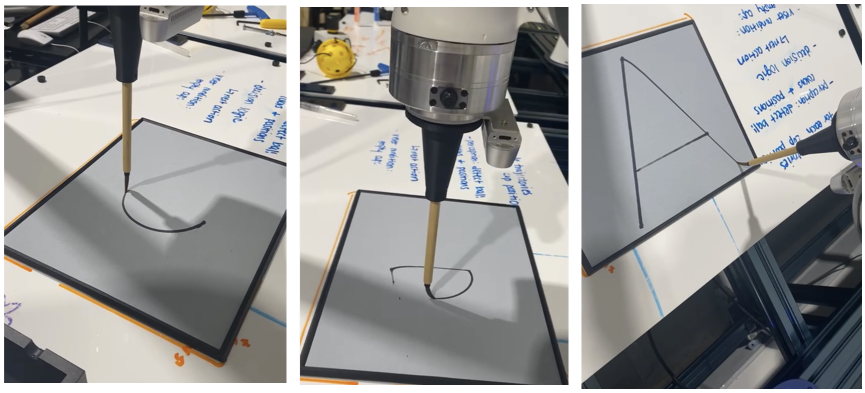


***Fig 7**. Character primitives: C, D, A*

Strings

At this point, the waypoints described in “Generate Waypoints for the String” were generated in order to resolve the jerky movements and to make the tracking problem scalable to larger strings (Figure 8).

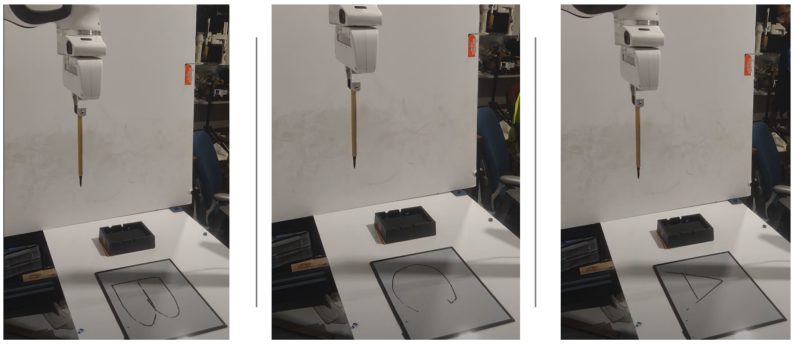


***Fig 8**. Letter primitives using waypoint tracking with an impedance controller*

To assess the performance of our drawing robot, we conducted several tests using a Buddha board to write with water. We evaluated the robot's ability to write accurately and legibly, as well as its speed and efficiency in producing long strings of text. We also tested its ability to switch between different fonts and adjust the size of the text.

Overall, we were satisfied with the results of our tests, as the robot was able to produce the intended text with a high degree of accuracy and legibility. While we observed some variability in its speed and efficiency, we were able to fine-tune the robot's hyperparameters to optimize its performance.

In addition to these technical metrics, we also assessed the user experience of using the robot. We found that the robot was easy to operate and required minimal setup and calibration. We also noted that it generated very little noise and did not create any mess or waste.

Although our current evaluations were limited to using a Buddha board with water, we believe that our results provide a solid foundation for further testing with other writing instruments and surfaces. We also recognize the need for future evaluations to be conducted in more diverse and realistic settings to fully gauge the robot's potential for practical applications.

Future Work

At present, the current system is able to write an arbitrarily lengthed script on the Buddha board with ample time before drying. While the infrastructure is adaptable and modular, there are many areas that could be improved to generate a more autonomous system.

Perception

Moving forward, there is much to gain by integrating a perception subsystem. The board dimensions, offset, and rotation are currently being hardcoded. A preliminary study into perception yielded results that allowed us to locate the corners of our canvas, as well as its offset in the world frame (Figure 7).

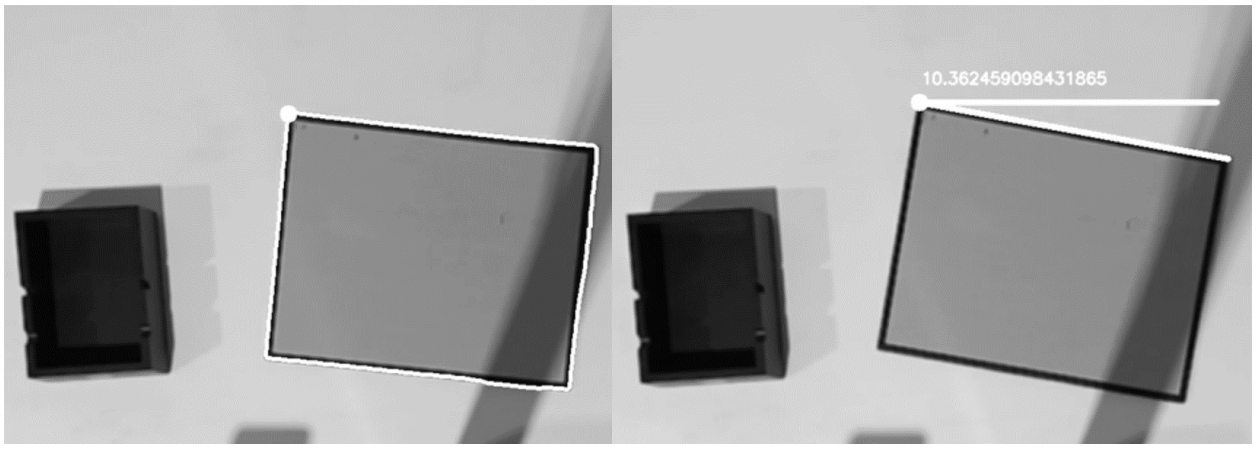


***Fig** 7. Online recognition of board corners and rotation*

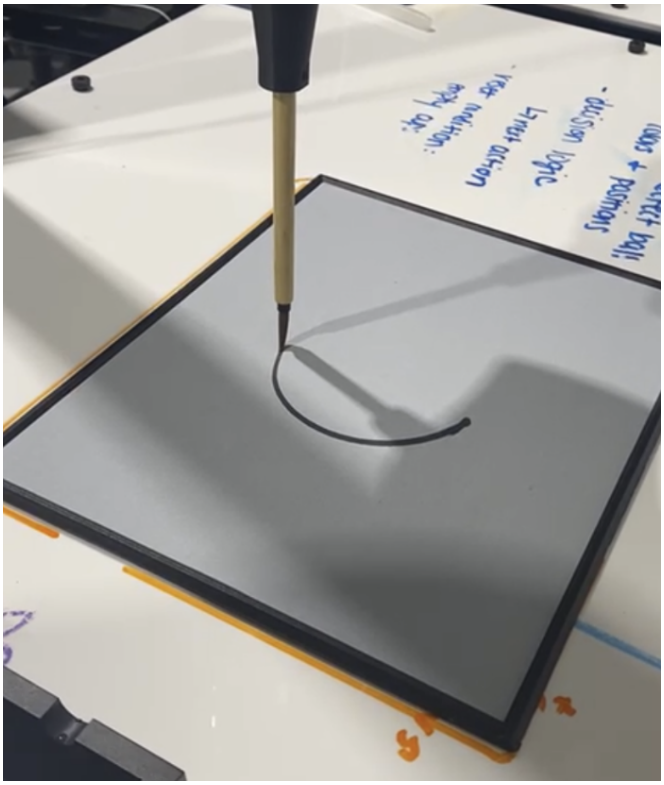
This architecture could reasonably and rapidly be expanded to find the location of the bowl centroid. This, when combined with depth information from the Azure Kinetic, could allow us to flexibly expand our abilities to any board configuration and bowl position.

Learning

We also recognize the various applications for our system to include learning. A reasonable extension would be to add visual learning methods by training a text recognition system to monitor the condition of the board. This would allow us to monitor when letters are beginning to dry, and when to wet the brush again for most consistent deposition of water.

Another point to extend is to learn some form of reinforcement to attempt to harness more of the capabilities of our Franka. In its current state, the manipulator is being used like a plotter. However with 6 DOF, there is an opportunity to explore more complex maneuvers. Especially for complex calligraphy-style scripts, which could benefit from articulation of the wrist joints.

Demonstration Video



https://drive.google.com/drive/folders/1T-k-EnOMEhMdzheVCFhHp8ecP-eA_X9I?usp=share_link

References

[1] "Carnegie Mellon's AI-Powered Frida Robot Collaborates with Humans to Create Art." *News*, 17 Mar. 2023, https://www.cmu.edu/news/stories/archives/2023/february/carnegie-mellons-ai-powered-frida-robot-collaborates-with-humans-to-create-art.

[2] Rashad, Fathy. "How I Built an AI Text-to-Art Generator." *Medium*, Towards Data Science, 8 Oct. 2021, https://towardsdatascience.com/how-i-built-an-ai-text-to-art-generator-a0c0f6d6f59f.

[3] Tiernan, Jonny. "The 9 Best AI Art Generators Available in 2023." *Vectornator Blog*, Vectornator Blog, 21 Mar. 2023, https://www.vectornator.io/blog/best-ai-art-generators/.