

# **Spline-FRIDA: Enhancing Robot Painting with Human Brushstroke Trajectories**

**Lawrence Chen**

CMU-CS-24-140

August 2024

Computer Science Department  
School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA 15213

**Thesis Committee:**

Jean Oh, Chair  
Jim McCann

*Submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Computer Science.*

**Keywords:** Robotics, Painting, Art, Generative AI, Machine Learning, Human-Computer Interaction

*To my parents*



## Abstract

A painting is more than just a picture on a wall; a painting is a *process* comprised of many intentional brush strokes, leading to a performance far richer than the final output. The shapes of individual strokes are an important component of a painting’s style. This is especially true for sparse sketches, where individual strokes are isolated and prominent. Prior work in modeling brush stroke trajectories either does not work with real-world robotics or is not flexible enough to capture the complexity of human-made brush strokes. In this work, we aim to develop a robotic drawing agent with controllable stroke-level style based on human trajectories.

To achieve this, we develop a framework to collect brush trajectories from human artists on a real canvas. We model these trajectories with an autoencoder. Finally, we incorporate the autoencoder into the planning pipeline in the FRIDA robotic painting system [22]. We find that, off-the-shelf, FRIDA’s brush stroke renderer struggles or fails to learn the complex trajectories from the human demonstration data, especially with narrow brushes or markers. We present a novel brush stroke renderer that is capable of generalizing to complex, human-made brush strokes while maintaining a small Sim2Real gap.

We conduct a survey on Amazon Mechanical Turk to evaluate our method quantitatively. The results indicate that Spline-FRIDA successfully captures the stroke styles in human drawings. They also suggest that Spline-FRIDA’s drawings are more human-like, higher quality, more true to the objective, and more artistic compared to FRIDA’s drawings.



## Acknowledgments

To begin, I would like to express my gratitude for my collaborators: Peter Schaldenbrand, Tanmay Shankar, Lia Coleman, and Jean Oh. This thesis would not have been possible without their contributions. Peter, thank you for your thoughtful feedback on all my ideas, for always being available to answer my questions on Slack, and for helping me get up to speed with FRIDA. Peter and Tanmay, I'm especially grateful for the time you dedicated to our weekly meetings. The advice and support you gave were invaluable in shaping the direction of this project. Lia, the pioneering work you did with the mocap system laid the groundwork for much of what followed, and is greatly appreciated. I also want to extend my thanks to my advisor, Jean, for all your support throughout the past year. I'm so grateful that you took a chance on me and gave me the opportunity to work with all of you. Additionally, a huge thank you to Jim McCann for agreeing to serve on my thesis committee and taking the time to review this work.

Next, I want to express my appreciation for my family—my parents, my sister, and even my cat. You were always there for me to provide care and guidance when I needed it most. Last but not least, I would like to thank my friends for their support, whether it be providing encouraging words, being there to listen, or simply helping me unwind when I needed a break. This includes, but is not limited to: Sam, Brian, Emily, Prem, Kelvin, Jason, Casey, Joyce, Claire, and Angel.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Related Work</b>	<b>3</b>
2.1	Stroke Primitives . . . . .	3
2.2	Differentiable Rendering . . . . .	3
<b>3</b>	<b>Background</b>	<b>5</b>
<b>4</b>	<b>Methods</b>	<b>7</b>
4.1	Overview . . . . .	7
4.2	Motion Capture Drawing Recording and Processing . . . . .	7
4.3	TrajVAE . . . . .	8
4.4	Traj2Stroke Model . . . . .	9
<b>5</b>	<b>Results</b>	<b>11</b>
5.1	Human Evaluations . . . . .	11
5.2	Trajectory Distributions . . . . .	13
5.3	Stroke Modeling Experiments . . . . .	15
<b>6</b>	<b>Limitations/Future Work</b>	<b>19</b>
6.1	RL-FRIDA . . . . .	20
<b>7</b>	<b>Conclusions</b>	<b>23</b>
	<b>Bibliography</b>	<b>25</b>



# List of Figures

1.1	<b>Spline-FRIDA drawings in different styles.</b> These are two drawings of a parrot made by our system. The left drawing uses longer, zig-zagging strokes, while the right drawing is composed of small circles. . . . .	1
3.1	<b>Weaknesses in FRIDA’s stroke model.</b> Left: FRIDA struggles with rendering thin strokes. Right: for a full drawing, errors in individual stroke predictions accumulate. The optimization process exploits blotchy predictions made by the renderer to create small dots, which appear quite different when actually executed. . . . .	5
4.1	<b>Rendering a stroke.</b> The inputs are a latent vector $z$ and an offset $\Delta$ . $z$ is fed through the decoder of a TrajVAE, generating a raw trajectory, which is then rotated and translated according to $\Delta$ . We then process the trajectory segments independently, obtaining darkness values for each. Finally, we take the max darkness over all segments. . . . .	8
4.2	<b>Mocap setup.</b> We use a motion capture system to track the position of the canvas and pen over time as an artist draws. Three mocap markers are placed along the corners of the canvas, and four are mounted at the end of the pen. The trajectories of each stroke are extracted, then rotated and translated such that the start of the trajectory is $(0, 0)$ and the end point is on the x-axis. . . . .	8
5.1	<b>Example drawings made by Spline-FRIDA.</b> Each column represents a distinct trajectory style and each row uses a different objective. For these examples, we maximize the CLIP similarity score between the canvas and the objective. The top row contains original drawings made by human artists on our mocap system. One VAE was fine-tuned on each human drawing and used to plan the drawings in each column. . . . .	12
5.2	<b>Confusion matrix for matching task.</b> The x-axis represents the index of the specific TrajVAE used to generate the drawing, and the y-axis represents the index of the human drawing participants thought was most similar. The five human drawings/styles the same ones as in the top row of Figure 5.1, with the same order. . . . .	13
5.3	<b>Mapping the latent space.</b> We visualize the TrajVAE latent space by drawing trajectories at their respective coordinates, projected down to 2 dimensions via t-SNE. To generate this plot, we use a TrajVAE that is trained on multiple sessions of human trajectory data. . . . .	14

5.4	<b>Visualizing the outputs of various stroke models.</b> The first three rows contain sharpie strokes, and the last three contain brush strokes. Traj2Stroke is near-perfect for the sharpie strokes, and reasonable for the brush strokes. The CNN-based models may fail to generalize to novel trajectories (rows 1, 3, 4). . . . .	17
5.5	<b>Spline-FRIDA’s low Sim2Real gap.</b> We compare a plan made by Spline-FRIDA with its execution (physically drawn with a robot). The two are nearly identical. .	18
6.1	<b>RL-FRIDA Preliminary Results.</b> This figure depicts some example plans made by RL-FRIDA. The left column shows drawings made by an agent trained to minimize the pixelwise L2 loss to the target, while the right column shows drawings made by an agent trained to minimize CLIP loss. . . . .	21

# List of Tables

5.1	<b>Opinions on FRIDA vs Spline-FRIDA.</b> Each cell shows the number of participants that chose the system for the given question. Overall, participants thought that compared to FRIDA, drawings made by Spline-FRIDA were more human-like, higher quality, more true to the objective, and more artistic. . . . .	15
5.2	<b>Quantitative comparison of stroke models.</b> This table shows the average L1 loss of each stroke model when predicting either sharpie or brush strokes (lower is better). Loss is calculated on dataset B (out-of-distribution) trajectories only. Traj2Stroke achieves the best results for sharpie strokes, and Traj2Stroke with U-Net is the best for brush strokes. . . . .	16



# Chapter 1

## Introduction

Paintings and drawings are used to convey messages of emotion, cultural values, and shared experiences. While these aspects can be conveyed by the objects or subjects within the painting, style is perhaps just as important to expressing those messages. “The ‘same’ content represented in a different form—in a different medium or mode or style or language—is *not* the same: what is the same through all variations of the form is only a tenuous abstraction, a *précis* of the full content” [18]. In the visual art space, there is evidence that people find the style of an image to be even more crucial than its content when interpreting the meaning of a generated image [21]. In particular, patterns in the shapes of individual strokes within a painting can contribute to the overall style and aesthetic of an artwork. Two examples of this can be seen in Figure 1.1. The drawing on the left uses a style with long, jagged strokes, giving the parrot a fluffy texture. On the other hand, the drawing on the right is composed of small circular strokes, resulting in a different texture. In both cases, the stroke shapes are crucial for defining the painting’s style and therefore the expression of the message that the artist intends to convey.

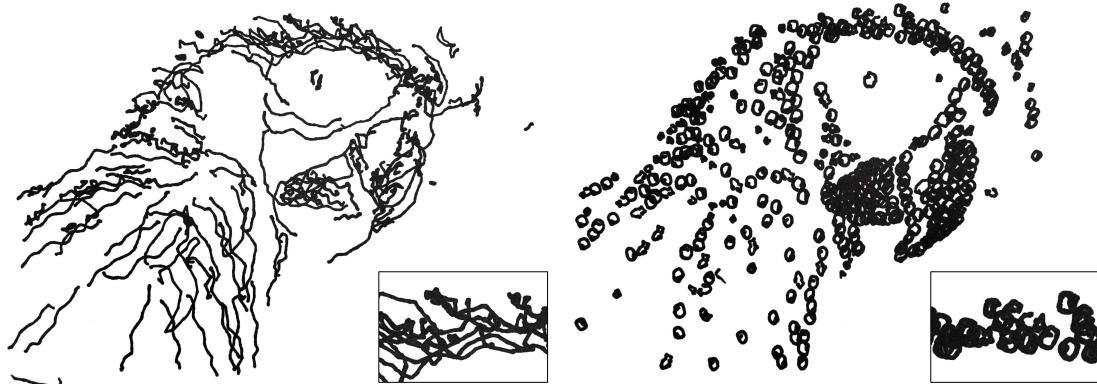


Figure 1.1: **Spline-FRIDA drawings in different styles.** These are two drawings of a parrot made by our system. The left drawing uses longer, zig-zagging strokes, while the right drawing is composed of small circles.

Furthermore, if robots are to support humans in the creation of artwork, it is important for

the robot to have flexible styles of strokes for the user to specify either through choice or demonstration. Many artists do not wish to automate the artistic process [6, 7], but some are open to co-creative assistants [1, 2, 9, 10]. The work operates on the assumption that giving more creative control to a user co-creating with a robot over the style of the image, allows them to feel more ownership over the artwork that they create with the robot.

In this thesis, we explore how stroke-level style control can be added to FRIDA [22], an open-source framework for robot painting. FRIDA uses simple Bézier curves as stroke primitives, but this representation lacks human style. To adapt FRIDA to generate more human-like brush strokes, we use motion capture to record human drawings with real-world brushes and markers on paper. We model these recorded trajectories with an autoencoder. We find that FRIDA’s brush stroke renderer is unable to model the complex strokes that humans create because it is a purely data-driven approach and lacks generalizability, leading to a large Sim2Real gap. Thus, we introduce a novel brush stroke renderer, Traj2Stroke, that is capable of modeling the stroke trajectories that we collected. This new renderer has a similarly small Sim2Real gap as the FRIDA renderer on simple strokes, but can also model much more complex strokes without breaking down. Traj2Stroke is fully differentiable, enabling it to be used in the FRIDA planning pipeline which can plan abstractly according to a robot’s capabilities, e.g., black-and-white sketches from input color photographs.

To summarize, our main contributions are as follows:

- We introduce Traj2Stroke, a method to differentiably render polylines with variable thickness. Compared to FRIDA’s renderer, Traj2Stroke has a significantly smaller Sim2Real gap between simulated and real sharpie strokes.
- We present a successful approach to modeling styles of brushstroke trajectories using variational autoencoders. This gives users control over the shapes of individual strokes used for a robot painting.
- We demonstrate the viability of using motion capture to track a human artist’s drawing in real time. We are able to accurately recover stroke trajectories from the raw position data. We open source a small dataset of human drawings collected with this method.

# Chapter 2

## Related Work

Stroke-Based Rendering (SBR) involves arranging primitive shapes to create an image, often with the goal of replicating some target image. Many recent works use forward prediction methods, in which a neural network learns to output the next stroke to add [4, 14, 20], as well as optimization-based methods, where stroke parameters are passed through a differentiable rendering pipeline and optimized via backpropagation [16, 22, 28].

### 2.1 Stroke Primitives

Most SBR research is focused on global planning and propose new algorithms to arrange stroke primitives. On the other hand, there has been little research into how the stroke primitives themselves should be defined. Some works use definitions that would be difficult to replicate on a physical robot. For instance, Learning to Paint defines strokes as translucent Bezier curves with arbitrary thicknesses [4]. Schaldenbrand et al. found that when Learning to Paint’s neural network was restricted to outputting realistic brush strokes, the quality of generated images suffered [20]. Paint Transformer uses a mask of a brush stroke that can be transformed, resized, and recolored [14]. However, the wide range of stroke sizes presents a problem, as some strokes can be half the size of the canvas, while others are thin enough to capture fine details. In practice, implementing a setup in real life that allows the robot arm to make arbitrarily sized brush strokes is infeasible.

Based on human art, many drawing tools, such as markers or brushes, can inherently be versatile and adaptable enough to produce a wide range of stroke styles. Specifically, altering the paths of individual strokes can result in diverse styles. This has been observed and researched extensively in the context of human handwriting replication [3, 15], but only to a lesser extent for drawings. We hope to further explore how to define stroke primitives by explicitly modeling the style of stroke trajectories used in a drawing.

### 2.2 Differentiable Rendering

In SBR, differentiable renderers are modules that take in stroke parameters and output a rendered image. They differ from traditional renderers in that gradients of the image with respect to the

parameters can be obtained. Having access to such a module is a crucial assumption of many modern SBR planners.

Learning to Paint [4] takes a reinforcement learning approach to SBR. Despite the fact that reinforcement learning does not inherently require a differentiable environment, they found that using a differentiable renderer greatly boosted the system’s performance and convergence rate compared to a model-free method. This is mainly because differentiable environment allows for end-to-end training of the DRL agent. Paint Transformer [14] also makes use of differentiable rendering so that a loss can be backpropagated from the output image all the way back to its stroke predictor. These examples show that differentiable rendering can be useful even in methods that are not optimization-based.

DiffVG [11] is a popular library for differentiable 2D rasterization that has been used in many optimization-based SBR methods [5, 25, 26]. It supports rendering arbitrary parametric curves, either open or closed, including polygons, ellipses, and polylines. Due to its popularity, we also considered using DiffVG to model Sharpie marker strokes for this work. However, we discovered that out of the box, the DiffVG library does not support rendering polylines that are differentiable with respect to stroke thickness. DiffVG lines are only differentiable with respect to the control points. Furthermore, DiffVG decouples strokes into a boundary shape and a fill color, which we found to be too restrictive because it does not allow us to model the gradual dropoff in darkness from the center of a stroke to the outside. Thus, we choose to implement our own differentiable renderer, Traj2Stroke, which is specialized for rendering polylines.

# Chapter 3

## Background

Our work is based on FRIDA [22], a robotic system that can paint what users describe via text prompts, images, or even audio recordings [17] in an interactive and collaborative manner [23].

FRIDA is an optimization-based SBR method. It works by using a fully differentiable rendering module that takes in a list of stroke parameters and outputs a rendered drawing. Initially, the planner assigns random values to the stroke parameters. It optimizes these values by passing them through the rendering module, computing a loss according to the user-specified objectives, and back-propagating it to the parameters. After many iterations, the system converges on a set of stroke parameters that can be executed on the robot.

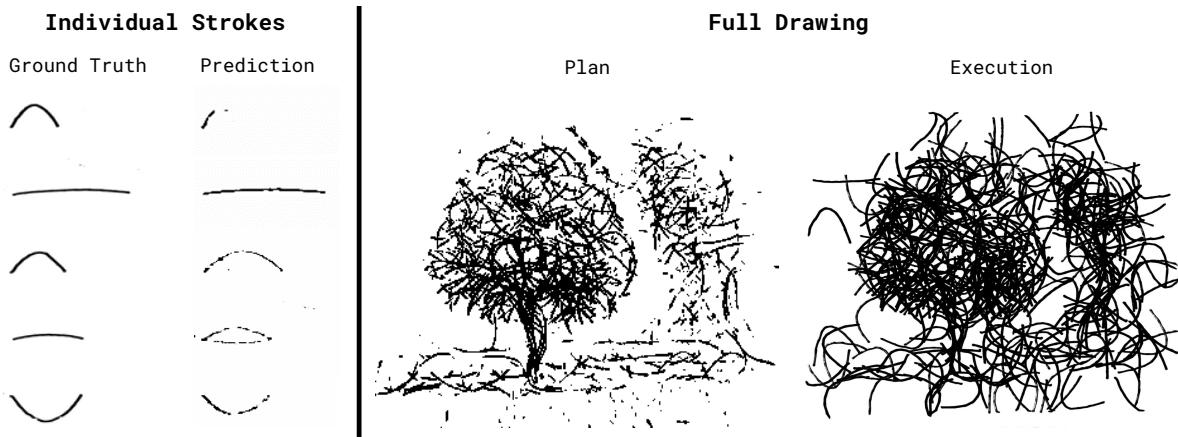


Figure 3.1: **Weaknesses in FRIDA’s stroke model.** Left: FRIDA struggles with rendering thin strokes. Right: for a full drawing, errors in individual stroke predictions accumulate. The optimization process exploits blotchy predictions made by the renderer to create small dots, which appear quite different when actually executed.

FRIDA renders strokes using a convolutional neural network (CNN). Although the CNN approach works well for short, simple strokes made with a paintbrush, it has trouble converging when longer, thinner Sharpie marker strokes are used, as illustrated in Figure 3.1. We believe this issue arises because thin ground truth strokes require the renderer to achieve a higher level of precision, complicating its task. Consequently, the CNN often produces blotchy, incomplete

predictions. This is problematic since the errors compound during the planning stage (illustrated on the right side of Figure 3.1).

Another avenue for improvement is FRIDA’s stroke representation. FRIDA uses quadratic Bézier curves defined by three parameters—length, height above the canvas, and bend—to represent strokes. While this straightforward approach offers convenience, it falls short in capturing the complexity and variety inherent in human drawings. In this context, we propose a novel stroke representation and a new renderer to model long, complex trajectories that can enable diverse styles.

# Chapter 4

## Methods

### 4.1 Overview

Our approach to stroke modeling and rendering consists of (1) capturing and processing human demonstration data using motion capture technology, (2) modeling these trajectories by training an autoencoder, TrajVAE, and (3) using Real2Sim2Real methodology to fine-tune our novel rendering approach, Traj2Stroke. Importantly, we switch to a new stroke representation in which trajectories can be arbitrary polyline splines rather than quadratic Bézier curves.

A visualization of the rendering pipeline for a single stroke is shown in Fig. 4.1. Stroke parameters are expressed as  $(z, \Delta)$  pairs, where  $z$  is the latent vector describing the shape of the stroke, and  $\Delta$  represents its rotation and translation on the canvas.  $z$  is fed into the decoder of a variational autoencoder, which outputs a trajectory. This is then rotated and translated according to  $\Delta$ . The next part of the pipeline is the Traj2Stroke model, which differentiably renders the trajectory using a sequence of tensor operations. The output is an array of darkness values in the interval  $[0, 1]$ .

Finally, the individually rendered strokes are stamped onto the same canvas to create the final painting. Importantly, each step in this process is differentiable, which allows gradients to flow from the painting back to the stroke parameters.

### 4.2 Motion Capture Drawing Recording and Processing

A motion capture system is utilized instead of a drawing tablet to capture human brushstroke trajectories, as it better replicates the robot’s environment (both humans and robots use identical physical tools).

While the artist sketches, we continuously track the positions and orientations of the canvas and pen. For each frame, we calculate the position of the pen tip and determine its distance from the canvas. If this value is below a threshold, we consider the pen to be in the down position. Consecutive positions where the pen is down are merged into trajectories. Each trajectory is then standardized by translating it to the origin and rotating it to be horizontal (ending at  $y = 0$ ), as seen in Figure 4.2. We also resample each trajectory to have exactly 32 points.

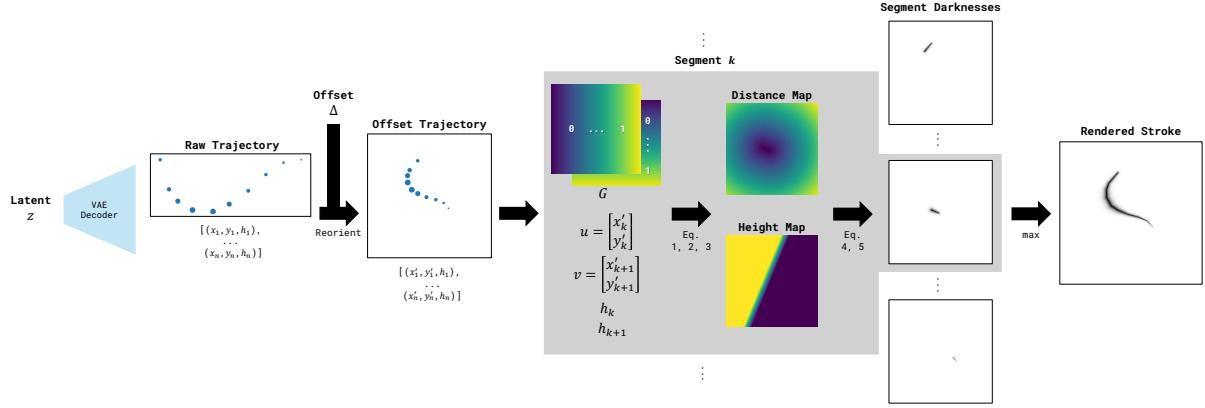


Figure 4.1: **Rendering a stroke.** The inputs are a latent vector  $z$  and an offset  $\Delta$ .  $z$  is fed through the decoder of a TrajVAE, generating a raw trajectory, which is then rotated and translated according to  $\Delta$ . We then process the trajectory segments independently, obtaining darkness values for each. Finally, we take the max darkness over all segments.

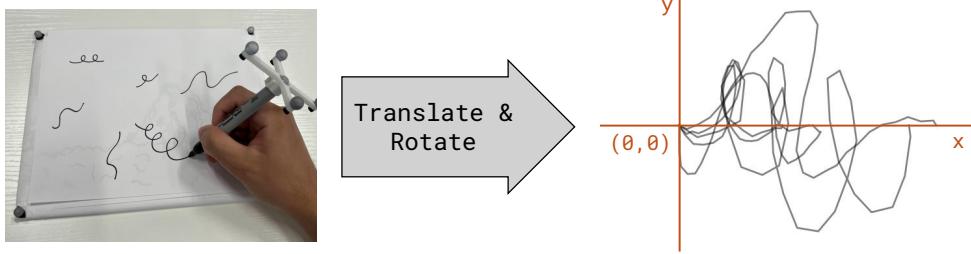


Figure 4.2: **Mocap setup.** We use a motion capture system to track the position of the canvas and pen over time as an artist draws. Three mocap markers are placed along the corners of the canvas, and four are mounted at the end of the pen. The trajectories of each stroke are extracted, then rotated and translated such that the start of the trajectory is  $(0, 0)$  and the end point is on the x-axis.

Thus, each human brushstroke trajectory is modeled as a polyline (piecewise linear) going through 32 control points. This polyline is encoded as a  $32 \times 3$  tensor. The coordinates  $(x, y, h)$  of each control point are defined by  $x$  and  $y$  as horizontal displacements (in the plane of the canvas) and  $h$  as vertical displacement (elevation above the canvas).

### 4.3 TrajVAE

After collecting and processing the motion capture data, we train variational autoencoders [8] to model these stroke trajectories. We name these TrajVAEs. During training, a TrajVAE takes a trajectory as input, passes it through an encoder that compresses it to a latent vector of size 64, and then sends it through a decoder to turn it back into a trajectory. We minimize the mean squared error between the input and output trajectories.

During a single motion capture recording, we record roughly 100 human-drawn trajectories. We found that this is not enough data for a TrajVAE to converge. Instead, we pretrain each TrajVAE on trajectories aggregated from multiple recording sessions, then fine-tune it on a single session to capture a more specific style. Each model converges very fast (less than a minute) and only requires a few (<20) trajectories in the fine-tuning dataset.

During the planning phase, only the VAE decoder is used. The design of the overall pipeline is modular so that different VAEs can be swapped in, allowing us to change the stroke style with no need for additional training.

## 4.4 Traj2Stroke Model

The TrajVAE model outputs a trajectory that the robot should draw, but to predict the appearance of the stroke given this trajectory, we developed a novel rendering approach that we call Traj2Stroke. Traj2Stroke takes a trajectory, as well as positional and rotational offsets, and renders it as an  $H \times W$  image. Importantly, this process is differentiable so that the planning process can backpropagate through it.

To train this model, we randomly sample trajectories from TrajVAE, execute them on the robot, and take before/after pictures of the canvas for each stroke. Next, we input the sampled trajectories into the Traj2Stroke model to get predicted stroke masks. These masks are stamped onto the before-stroke pictures, and the resulting prediction is compared to the after-stroke pictures. We minimize a weighted L1 loss that places higher weight on pixels covered by the new stroke. A separate Traj2Stroke model must be trained for each drawing medium (marker/brush).

After receiving a trajectory  $[(x_1, y_1, h_1), \dots, (x_n, y_n, h_n)]$  and offsets  $\Delta = (\Delta_x, \Delta_y, \Delta_\theta)$ , the Traj2Stroke model begins by reorienting the trajectory to be in the reference frame of the canvas (see Figure 4.1). To do this, it rotates the  $x$  and  $y$  components by  $\Delta_\theta$ . It then scales and translates them so that each point  $(x_i, y_i, h_i)$  becomes

$$(m_x x_i + b_x + \Delta_x, m_y y_i + b_y + \Delta_y, h).$$

$m_x, m_y, b_x$ , and  $b_y$  are learnable parameters used to model any small affine error that may occur during camera calibration. We expect that  $m_x, m_y \approx 1$  and  $b_x, b_y \approx 0$ .

The trajectory has now been converted to canvas coordinates, and we denote it as

$$[(x'_1, y'_1, h_1), \dots, (x'_n, y'_n, h_n)].$$

We proceed by rendering each its  $n - 1$  segments separately. Fix an arbitrary  $k$ , and note that segment  $k$  goes from  $(x'_k, y'_k, h_k)$  to  $(x'_{k+1}, y'_{k+1}, h_{k+1})$ .

Our approach to rendering the segment is to first define a constant  $H \times W \times 2$  tensor  $G$  of canvas coordinates, where  $H$  and  $W$  are the dimensions of the canvas. One channel of this tensor contains the  $x$  coordinates, and the other contains the  $y$  coordinates, as seen in Figure 4.1. For convenience, we also define  $u = [x'_k \ y'_k]^T$  and  $v = [x'_{k+1} \ y'_{k+1}]^T$ .

We compute a *Distance Map* that stores the distance of each coordinate in  $G$  to the segment. This is computed with the following equation (note that the vector operations involving  $G$  are

done element-wise):

$$\text{Distance Map} = \min(\| (G - u) - \text{proj}_{v-u}(G - u) \|, \|G - u\|, \|G - v\|). \quad (4.1)$$

That the first term computes the distance from each point in  $G$  to the line through  $u$  and  $v$ , and the last two terms calculate the distance to the endpoints. Thus, taking the min of the three yields the desired result.

We also compute a *Height Map*, which represents the height of the brush tip as it moves over the segment. For each coordinate, we project it onto the segment and compute the height by linear interpolation between  $h_k$  and  $h_{k+1}$ :

$$T = \text{clamp}_{[0,1]} \left( \frac{\|\text{proj}_{v-u}(G - u)\|}{\|v - u\|} \right) \quad (4.2)$$

$$\text{Height Map} = (1 - T) \cdot h_k + T \cdot h_{k+1}. \quad (4.3)$$

We make the assumption that there is an affine relationship between the height of the brush tip and the thickness of the stroke. Thus, we introduce two learnable parameters  $\alpha$  and  $\beta$ , and obtain a *Thickness Map* like so:

$$\text{Thickness Map} = \alpha \cdot \text{Height Map} + \beta. \quad (4.4)$$

If the distance between a coordinate and the segment is less than the stroke thickness, then that coordinate should be affected by the stroke. We assume there is a gradual dropoff in darkness as we get further from the center of the segment. This reasoning motivates the following calculation for the darkness values:

$$\text{Darkness} = \left[ \text{clamp}_{[0,1]} \left( 1 - \frac{\text{Distance Map}}{\text{Thickness Map}} \right) \right]^c. \quad (4.5)$$

Thus, coordinates directly on the segment get a darkness value of 1, and coordinates that are a stroke thickness away get a darkness value of 0. This also introduces another learnable parameter  $c$  which determines how quickly the darkness values drop off as we get further from the segment.

Finally, we take the max darkness values over all segments to obtain the rendered stroke.

In total, the Traj2Stroke model has only 7 learnable parameters:

$$(x_m, y_m, x_b, y_b, \alpha, \beta, c).$$

Essentially, the output is constrained to be a polyline that follows the trajectory of the brush. During training, the model only learns the small affine transformation, the thickness parameters, and the darkness dropoff. As we will show in Section 5.3, this design is more precise and more robust to overfitting compared to CNN renderers.

# Chapter 5

## Results

Figure 5.1 shows an array of drawings produced by Spline-FRIDA. We capture five human drawings using the mocap system, each with distinct stroke styles, which are presented in the top row. Each human drawing is used to fine-tune a separate TrajVAE, resulting in five unique TrajVAEs. Each TrajVAE is then used to plan a series of drawings with various objectives. These objectives are displayed in the left column.

The individual styles of the drawing trajectories are preserved by the TrajVAEs. For instance, the fourth human drawing exhibits tiny, curly lines, which are reproduced in the drawings produced by its corresponding TrajVAE. Similarly, the fifth human drawing is composed of small circles, which is also true for the robot drawings in its column.

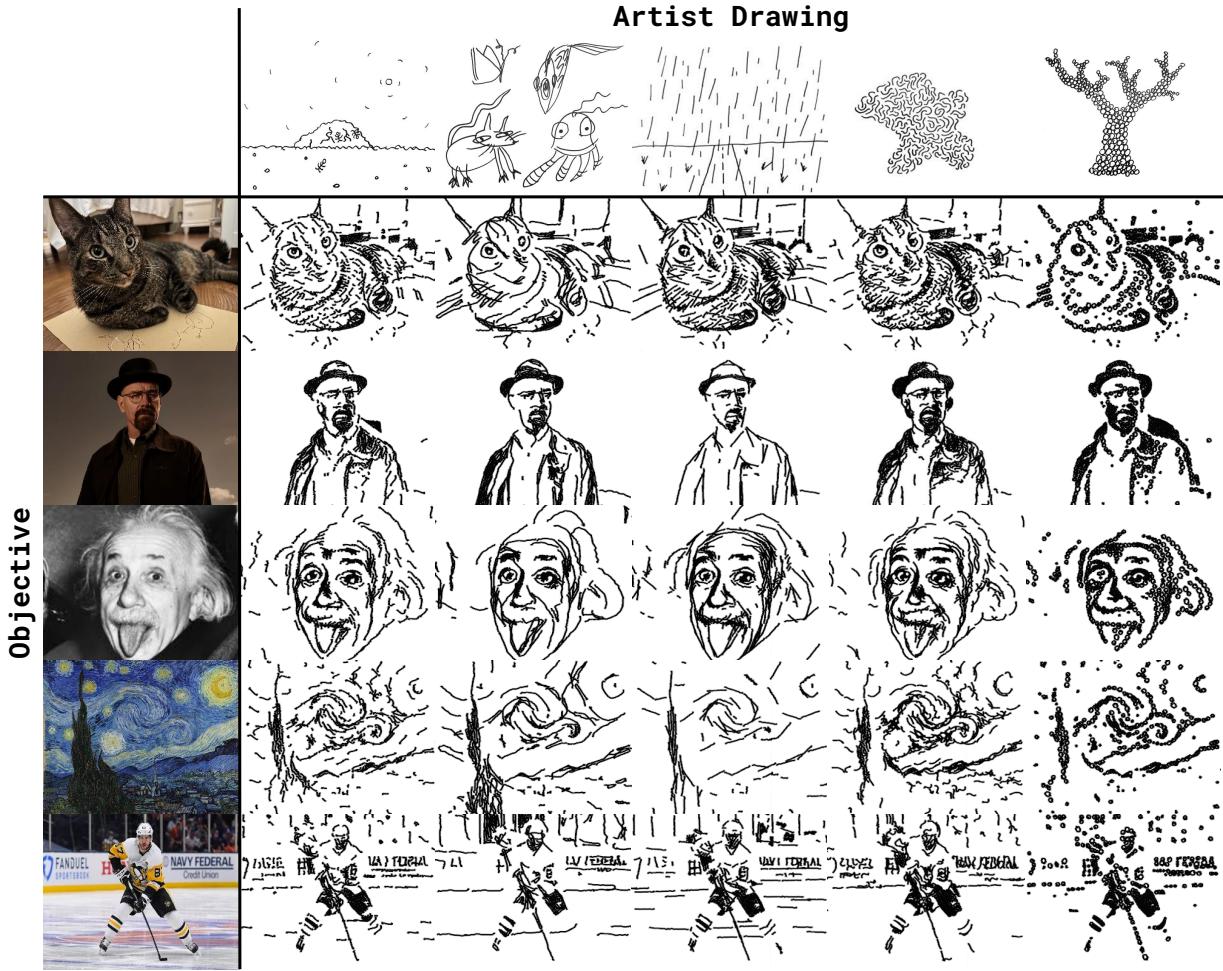
### 5.1 Human Evaluations

To what extent is Spline-FRIDA able to capture the stroke style of a drawing? And, in general, are Spline-FRIDA’s drawings better than those made by FRIDA? To obtain quantitative answers to these questions, we conducted a survey and released it to 100 participants on Amazon Mechanical Turk.

For the first part of the survey, we asked participants to match Spline-FRIDA drawings with human drawings that have the same stroke style. More specifically, for each participant, we pick a random row of Figure 5.1 (technically, an extended version with 11 different objectives), shuffle it, and for each drawing, ask them to pick the closest human drawing in stroke style. We told participants to “focus on the characteristics of individual strokes, such as their trajectories, shapes, and curves.” The results of this experiment are seen in Figure 5.2.

The high values along the diagonal suggest that, in general, participants were able to choose the correct human drawing used to style each robot drawing. Style 5, which corresponds to the drawings made with small circles, seemed to be particularly distinguishable. Meanwhile, style 1 (which uses very short strokes) was often confused with style 4 (which also uses short strokes but is more curly), and style 2 (which uses long strokes) was confused with style 3 (which also can have long strokes, but tend to be completely straight). Nevertheless, all five encoded styles are most strongly associated with the correct human drawings.

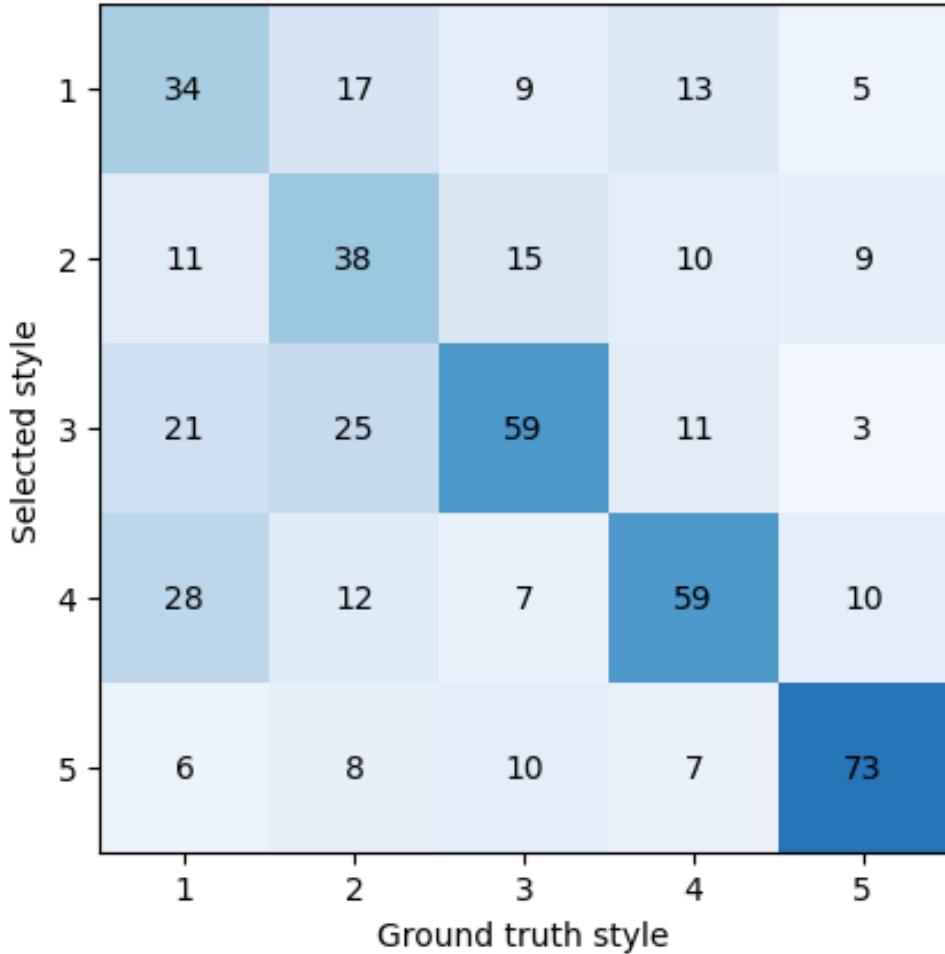
The second part of the survey asked participants’ subjective opinions on Sharpie drawings



**Figure 5.1: Example drawings made by Spline-FRIDA.** Each column represents a distinct trajectory style and each row uses a different objective. For these examples, we maximize the CLIP similarity score between the canvas and the objective. The top row contains original drawings made by human artists on our mocap system. One VAE was fine-tuned on each human drawing and used to plan the drawings in each column.

made by FRIDA vs. Spline-FRIDA. Each participant was shown an objective image and two robot drawings of it, one from FRIDA and one from Spline-FRIDA. Both robot drawings were executed on the physical robot so that any Sim2Real gap comes into play. The questions and tallied responses are shown in Table 5.1.

Respondents believed that Spline-FRIDA’s drawings, in comparison to FRIDA’s, appeared more human-made, had higher overall quality, better matched the reference image, and were more artistic. Respondents also perceived the Spline-FRIDA drawings as more “abstract”, although opinions on this were somewhat split.



**Figure 5.2: Confusion matrix for matching task.** The x-axis represents the index of the specific TrajVAE used to generate the drawing, and the y-axis represents the index of the human drawing participants thought was most similar. The five human drawings/styles the same ones as in the top row of Figure 5.1, with the same order.

## 5.2 Trajectory Distributions

In figure 5.3, we visualize the latent space for a VAE trained on multiple drawing sessions. We encode all of the human trajectories into latent vectors, then project them down to 2 dimensions using t-SNE [24]. We then draw each human trajectory at its corresponding 2d coordinates.

We observe human trajectories spread throughout the space, forming several homogeneous clusters. This structured organization indicates that the VAE effectively learns a correlation between trajectories and latent vectors. Consequently, an optimization-based planning algorithm is likely to be effective.

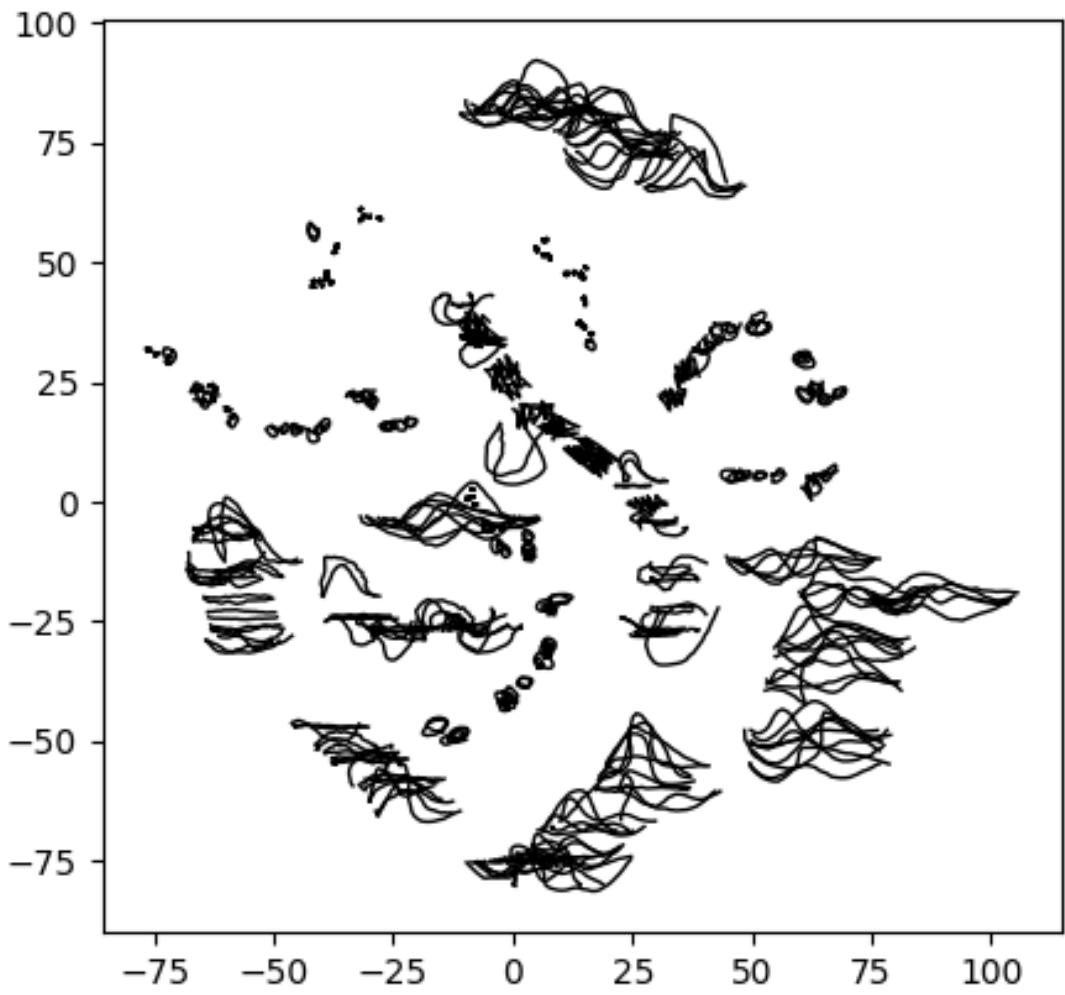


Figure 5.3: **Mapping the latent space.** We visualize the TrajVAE latent space by drawing trajectories at their respective coordinates, projected down to 2 dimensions via t-SNE. To generate this plot, we use a TrajVAE that is trained on multiple sessions of human trajectory data.

	FRIDA	Spline-FRIDA
Which drawing looks more like it was drawn by a human (rather than a robot)?	27	73
Which drawing looks better overall?	16	84
Which drawing better matches the reference image?	16	84
Which drawing is more artistic?	18	82
Which drawing is more abstract?	40	60

Table 5.1: **Opinions on FRIDA vs Spline-FRIDA.** Each cell shows the number of participants that chose the system for the given question. Overall, participants thought that compared to FRIDA, drawings made by Spline-FRIDA were more human-like, higher quality, more true to the objective, and more artistic.

## 5.3 Stroke Modeling Experiments

The purpose of a stroke model is to differentiably render trajectories. We experiment with a variety of methods to do this and evaluate them both quantitatively and qualitatively in a controlled experiment. In total, we tried four methods:

### CNN

Our baseline, a convolutional neural network. This is analogous to FRIDA’s renderer architecture, except it takes in full trajectory rather than the (length, bend, height) that FRIDA uses.

### CNN with CoordConv

To render a trajectory, one subproblem the renderer must solve is mapping Cartesian coordinates to one-hot pixel space. Liu et. al. [13] showed that traditional CNNs can have difficulty with this, so we implement their suggestion of using CoordConv layers instead of traditional convolutions. This means adding two additional channels to the input of each convolution: one containing the x-coordinates of each pixel, and the other containing the y-coordinates.

### Traj2Stroke

Our new method described in Section 4.4.

### Traj2Stroke with U-Net

Our new method, but with an additional Pix2Pix network appended after the output layer. The goal of this additional layer is to refine the Traj2Stroke output by learning subtle effects such as

texture and bristle drag. The architecture of the new layer closely follows that of U-Net [19]. We freeze the U-Net weights during the first half of the training and unfreeze them for the second half. The purpose of this is to train the Traj2Stroke portion first and get it as close as possible to the ground truth, before using the U-Net to refine it. Inspired by the success of ControlNet [27], we utilize a zero-convolution so that the U-Net has no effect on the output while its weights are frozen.

An important metric we wish to evaluate is generalizability. We define a stroke model as generalizable if it can generate reliable stroke predictions, even when given trajectories that are dissimilar to the ones used to train it. This is important because stroke models are time-consuming to train; it requires physically executing trajectories on a robot. A generalizable stroke model only needs to be trained once per drawing medium. It can then be used out-of-the-box for planning with any TrajVAE, thus saving time and effort.

In order to evaluate generalizability, we train and test the stroke model on trajectories from different distributions. More precisely, we create two datasets, A and B. Both datasets contain (trajectory, stroke image) pairs. For dataset A, the trajectories are sampled from a generic TrajVAE, trained on a session that we judge to have good stroke diversity. For dataset B, we use trajectories from more specialized TrajVAEs, trained on sessions with very unique styles. We train the model using dataset A, and we evaluate generalizability by checking its performance on dataset B.

We run the experiment twice, once for each of two drawing mediums: a sharpie and a thin paintbrush. The experiment results can be seen in Table 5.2. The Traj2Stroke architecture without U-Net achieves the lowest loss on sharpie strokes. Adding the U-Net hurts performance on sharpie strokes, though it achieves the best results on brush strokes. We do not get much benefit from using CoordConv over traditional convolutions.

Medium	CNN	w/ CoordConv	Traj2Stroke	w/ U-Net
Sharpie	.00107	.00095	<b>.00055</b>	.00098
Brush	.00162	.00163	.00158	<b>.00153</b>

Table 5.2: **Quantitative comparison of stroke models.** This table shows the average L1 loss of each stroke model when predicting either sharpie or brush strokes (lower is better). Loss is calculated on dataset B (out-of-distribution) trajectories only. Traj2Stroke achieves the best results for sharpie strokes, and Traj2Stroke with U-Net is the best for brush strokes.

Visually, example predictions generated by each model can be seen in Figure 5.4. All examples are from dataset B, meaning that these trajectories are from a TrajVAE different from the one used during training. The vanilla CNN fails to generalize in certain cases. Changing standard convolution layers into CoordConv layers does not seem to help much. The Traj2Stroke model is near-perfect for the sharpie strokes. It also generates reasonable output for the brush strokes, although it is unable to capture the texture. The U-Net layer is an attempt to mitigate this; its design aims to combine the precision of Traj2Stroke with the flexibility of convolutions. While the outputs indeed seem to capture the texture of paint, they also tend to be more blotchy, which is concerning for an optimization-based planner.

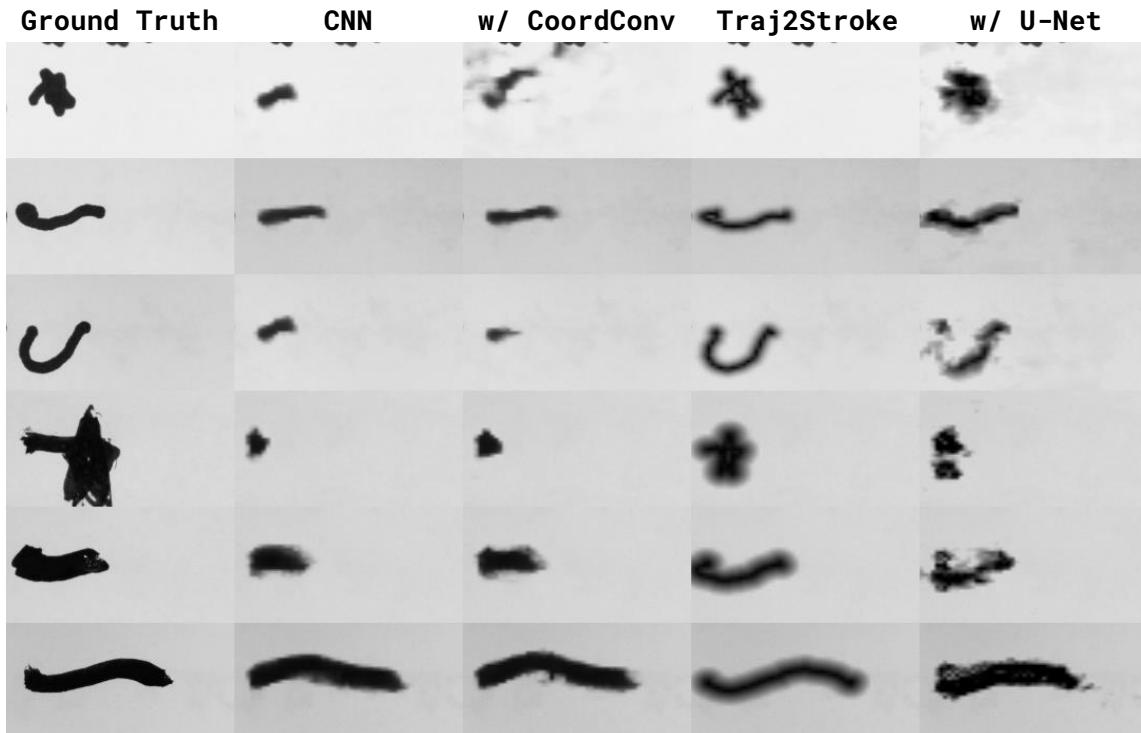


Figure 5.4: **Visualizing the outputs of various stroke models.** The first three rows contain sharpie strokes, and the last three contain brush strokes. Traj2Stroke is near-perfect for the sharpie strokes, and reasonable for the brush strokes. The CNN-based models may fail to generalize to novel trajectories (rows 1, 3, 4).

Based on these findings, we choose to implement the base Traj2Stroke model (without U-Net) for Spline-FRIDA. As illustrated in Figure 5.5, the resulting Sim2Real gap for Sharpie drawings is very low. This is a huge improvement compared to the original FRIDA results depicted in Figure 3.1.

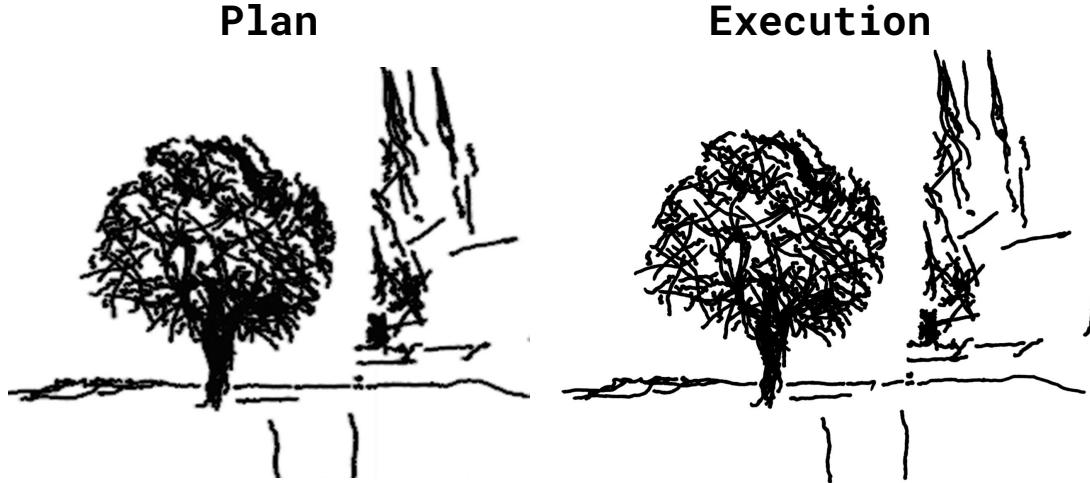


Figure 5.5: **Spline-FRIDA’s low Sim2Real gap.** We compare a plan made by Spline-FRIDA with its execution (physically drawn with a robot). The two are nearly identical.

# Chapter 6

## Limitations/Future Work

Artistic style is nuanced and subjective. Style can be broken down into roughly 7 elements: Lines, shapes, space, texture, value, and color [21]. In this thesis, we focus on capturing style by modeling brush stroke trajectories. These are intra-stroke attributes which only cover the style elements of lines and shapes. In future work, we hope to adapt the method to capture the other, inter-stroke style elements.

HelpingHand [15] offers an alternative approach to stroke stylization by running a *postprocessing* algorithm that refines existing drawn trajectories to better align with a large dataset of human-made strokes. By applying HelpingHand’s algorithm to Spline-FRIDA’s painting plans, it would be possible to enhance the stylistic quality of the strokes further. Moreover, HelpingHand has the capability to transform low-dimensional trajectories, like the 3-DOF paths used in Spline-FRIDA, into fully articulated 6-DOF trajectories, encompassing 2D position, pressure, 2D tilt, and rotation. This integration could serve as an efficient method to incorporate these additional degrees of freedom into Spline-FRIDA, resulting in drawings that more closely emulate human styles.

Optimization-based planning can be computationally expensive. In particular, our pipeline takes about an hour to plan a 400-stroke painting on a NVIDIA GeForce 4090. Additionally, it also requires a significant amount of memory. This is because during each optimization iteration, all the gradients need to be stored for every stroke’s forward pass simultaneously, resulting in a linear relationship between the number of strokes and the memory requirements. To address this issue, we modify our algorithm so that each iteration, we only compute gradients for a random subset of  $n$  strokes, where  $n$  is the maximum number that does not exceed GPU memory (we use  $n = 80$  in practice). The result is that only a random subset of strokes are updated by each backward pass.

Optimization methods for stroke-based rendering can also be susceptible to local minima, especially when using a pixel-wise loss function with an image objective. Fundamentally, this is due to the fact that strokes in the plan can take only small steps during optimization. As discussed by Zou et al. [28], when using an L2 objective, there can be a “zero-gradient” problem where strokes that are initialized in bad locations do not move because the objective function is locally flat. We also encounter this issue when using an L2 objective. We find that using CLIP loss generally works much better.

We find that optimization-based methods for paint planning are inherently limited by these

issues. Thus, in future work, we would like to explore alternatives that are more cost-effective for painting planning. One promising direction is reinforcement learning methods like Learning To Paint [4], which train policy networks to output subsequent strokes given the current canvas and objective. Paint Transformer [14] also has an interesting approach, in which they treat paint planning as a stroke prediction problem. While these papers both focus on simulated painting environments, our work introduces a robust method for bridging the gap between simulated and real-world brushstrokes. Consequently, we believe that integrating Traj2Stroke with approaches like LearningToPaint or PaintTransformer holds significant potential.

## 6.1 RL-FRIDA

As of the time of this writing, development has started but not finished for a new painting system called RL-FRIDA. RL-FRIDA trains a policy network via reinforcement learning to predict stroke parameters given images of the current canvas and the goal.

Currently, our RL agent utilizes the model-based DDPG algorithm introduced by Learning To Paint [4]. This algorithm is similar to traditional DDPG [12], but makes use of a differentiable renderer to allow gradients to flow through the environment. This property can be used to update the actor and critic networks much more effectively, resulting in faster convergence.

In order to train the agent to draw using Sharpie strokes, we replace Learning To Paint’s differentiable renderer with Traj2Stroke. Preliminary results of this are shown in Figure 6.1. We experiment with both L2 loss and CLIP loss as the source of reward for the agent. In both cases, the model is able to generate reasonable drawings after training for roughly 15 hours, though the quality is still lacking compared to drawings made by Spline-FRIDA. On the other hand, RL-FRIDA is order of magnitudes faster at inference time than Spline-FRIDA and FRIDA. It needs less than a second to plan out batches of 64 drawings at a time.

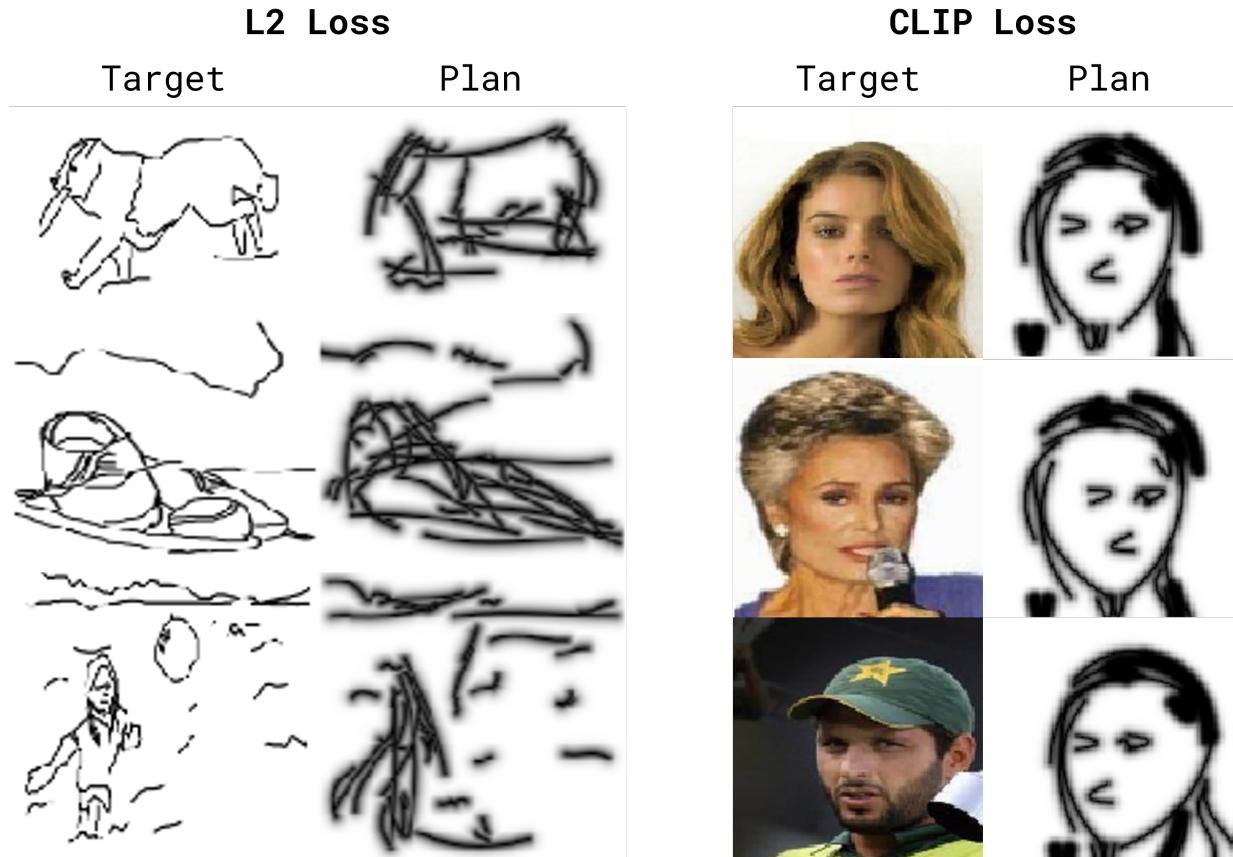


Figure 6.1: **RL-FRIDA Preliminary Results.** This figure depicts some example plans made by RL-FRIDA. The left column shows drawings made by an agent trained to minimize the pixel-wise L2 loss to the target, while the right column shows drawings made by an agent trained to minimize CLIP loss.



# **Chapter 7**

## **Conclusions**

In this thesis, we introduce a method to plan and execute robot sketches while allowing the user to control the style of strokes used. Using motion capture, we collect datasets of human stroke trajectories in multiple styles. These datasets are used train variational autoencoders that generate stroke trajectories with specific styles. We then incorporate these into a differentiable rendering pipeline, which allows us to plan drawings via an iterative optimization process. However, we discover that FRIDA’s neural renderer fails on complex, thin brush strokes. Thus, we experiment with different ways of rendering the trajectories. Our best performing architecture, Traj2Stroke, is robust and requires only a small dataset of ground truth strokes.



# Bibliography

- [1] Cole Bateman. *Creating for Creatives: A Humanistic Approach to Designing AI Tools Targeted at Professional Animators*. PhD thesis, Harvard University, 2021. 1
- [2] Nicholas Davis, Chih-PIn Hsiao, Kunwar Yashraj Singh, Lisa Li, and Brian Magerko. Empirically studying participatory sense-making in abstract drawing with a co-creative cognitive agent. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, pages 196–207, 2016. 1
- [3] Tom S. F. Haines, Oisin Mac Aodha, and Gabriel J. Brostow. My text in your handwriting. *ACM Trans. Graph.*, 35(3), may 2016. ISSN 0730-0301. doi: 10.1145/2886099. URL <https://doi.org/10.1145/2886099>. 2.1
- [4] Zhewei Huang, Wen Heng, and Shuchang Zhou. Learning to paint with model-based deep reinforcement learning, 2019. 2, 2.1, 2.2, 6, 6.1
- [5] Ajay Jain, Amber Xie, and Pieter Abbeel. Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models, 2022. 2.2
- [6] Chipp Jansen and Elizabeth Sklar. Exploring co-creative drawing workflows. *Frontiers in Robotics and AI*, 8:577770, 2021. 1
- [7] Harry H. Jiang, Lauren Brown, Jessica Cheng, Mehtab Khan, Abhishek Gupta, Deja Workman, Alex Hanna, Johnathan Flowers, and Timnit Gebru. Ai art and its impact on artists. In *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society*, AIES ’23, page 363–374, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702310. doi: 10.1145/3600211.3604681. URL <https://doi.org/10.1145/3600211.3604681>. 1
- [8] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. 4.3
- [9] Tomas Lawton, Francisco J Ibarrola, Dan Ventura, and Kazjon Grace. Drawing with re-framer: Emergence and control in co-creative ai. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, pages 264–277, 2023. 1
- [10] Shayla Lee and Wendy Ju. Adversarial robots as creative collaborators. *arXiv preprint arXiv:2402.03691*, 2024. 1
- [11] Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 39(6):193:1–193:15, 2020. 2.2
- [12] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yu-

- val Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019. URL <https://arxiv.org/abs/1509.02971>. 6.1
- [13] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coord-conv solution, 2018. URL <https://arxiv.org/abs/1807.03247>. 5.3
- [14] Songhua Liu, Tianwei Lin, Dongliang He, Fu Li, Ruifeng Deng, Xin Li, Errui Ding, and Hao Wang. Paint transformer: Feed forward neural painting with stroke prediction, 2021. 2, 2.1, 2.2, 6
- [15] Jingwan Lu, Fisher Yu, Adam Finkelstein, and Stephen DiVerdi. Helpinghand: example-based stroke stylization. *ACM Trans. Graph.*, 31(4), jul 2012. ISSN 0730-0301. doi: 10.1145/2185520.2185542. URL <https://doi.org/10.1145/2185520.2185542>. 2.1, 6
- [16] Xu Ma, Yuqian Zhou, Xingqian Xu, Bin Sun, Valerii Filev, Nikita Orlov, Yun Fu, and Humphrey Shi. Towards layer-wise image vectorization, 2022. 2
- [17] Vihaan Misra, Peter Schaldenbrand, and Jean Oh. Robot synesthesia: A sound and emotion guided ai painter, 2023. 3
- [18] Duncan Robertson. The dichotomy of form and content. *College English*, 28(4):273–279, 1967. 1
- [19] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. URL <https://arxiv.org/abs/1505.04597>. 5.3
- [20] Peter Schaldenbrand and Jean Oh. Content masked loss: Human-like brush stroke planning in a reinforcement learning painting agent, 2021. 2, 2.1
- [21] Peter Schaldenbrand, Zhixuan Liu, and Jean Oh. Styleclipdraw: Coupling content and style in text-to-drawing translation. *arXiv preprint arXiv:2202.12362*, 2022. 1, 6
- [22] Peter Schaldenbrand, James McCann, and Jean Oh. Frida: A collaborative robot painter with a differentiable, real2sim2real planning environment, 2022. (document), 1, 2, 3
- [23] Peter Schaldenbrand, Gaurav Parmar, Jun-Yan Zhu, James McCann, and Jean Oh. Cofrida: Self-supervised fine-tuning for human-robot co-painting. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024. 3
- [24] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL <http://www.jmlr.org/papers/v9/vandermaaten08a.html>. 5.2
- [25] XiMing Xing, Chuang Wang, Haitao Zhou, Jing Zhang, Qian Yu, and Dong Xu. Diffsketcher: Text guided vector sketch synthesis through latent diffusion models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 15869–15889. Curran Associates, Inc., 2023. URL [https://proceedings.neurips.cc/paper\\_files/paper/2023/file/333e67fc4728f147d31608db3ca78e09-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2023/file/333e67fc4728f147d31608db3ca78e09-Paper-Conference.pdf). 2.2

- [26] Ximing Xing, Haitao Zhou, Chuang Wang, Jing Zhang, Dong Xu, and Qian Yu. Svdreamer: Text guided svg generation with diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4546–4555, June 2024. 2.2
- [27] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models, 2023. URL <https://arxiv.org/abs/2302.05543>. 5.3
- [28] Zhengxia Zou, Tianyang Shi, Shuang Qiu, Yi Yuan, and Zhenwei Shi. Stylized neural painting, 2020. 2, 6