# Teaching Robots To Draw

**Atsunobu Kotani and Stefanie Tellex**
Department of Computer Science
Brown University
{akotani,stefie10}@cs.brown.edu

***Abstract*— In this paper, we introduce an approach which enables manipulator robots to write handwritten characters or line drawings. Given an image of just-drawn handwritten characters, the robot infers a plan to replicate the image with a writing utensil, and then reproduces the image. Our approach draws each target stroke in one continuous drawing motion and does not rely on handcrafted rules or on predefined paths of characters. Instead, it learns to write from a dataset of demonstrations. We evaluate our approach in both simulation and on two real robots. Our model can draw handwritten characters in a variety of languages which are disjoint from the training set, such as Greek, Tamil, or Hindi, and also reproduce any stroke-based drawing from an image of the drawing.**

## I. Introduction

The most recognized printing system today is an inkjet printer. By moving back and forth and spraying ink to the desired locations, printers replicate input images in a bit-map format. However, it is not the case that an inkjet printer draws with a writing utensil such as a pen or marker. In order to collaborate with humans, we would like for a manipulator robot to be able to draw on a white board, write a message with a pen on a post-it note, or draw a diagram. The ability to write would enable a robot to put up a sign directing people that a hallway was closed, to produce art using physical mediums such as a paint brush or a pen, or to address and mail a letter. Additionally, the robot could potentially engage in teaching activities at a white board, writing a math equation or drawing a diagram. These skills rely on the ability to produce a policy to draw with a writing utensil.

What differentiates handwriting from current printing technologies is its continuous drawing process. Shown an image of handwritten characters, robots should draw each target stroke in one consecutive drawing motion. Existing methods for robots that write with a utensil are unable to look at a bit-mapped image and directly produce a drawing policy. Instead, they require external information about the stroke order for character, such as human gestures [1, 2] or predefined paths for each letter [3]. This extra information makes it challenging for novice users to teach the robot how to draw new characters, because the stroke order information must be provided. A more recent reinforcement learning based approach [4] successfully learns to draw the target image, yet their model still struggles to draw each target stroke in one continuous drawing motion, and frequently draws the same parts over and over to replicate the image.

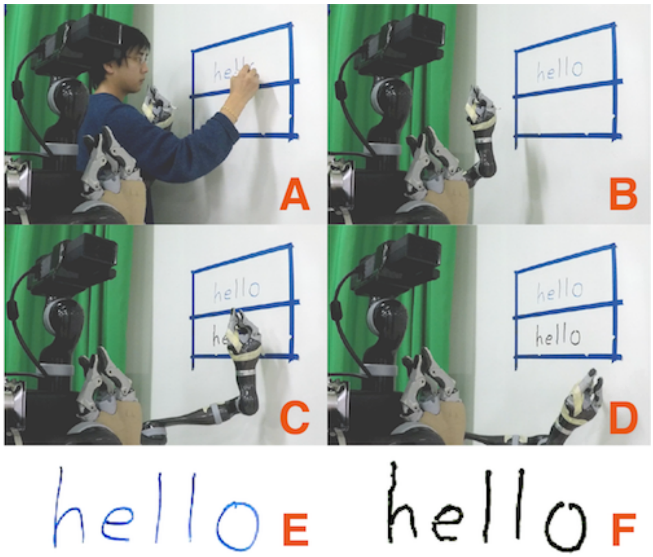


Fig. 1: Demonstration of our model reproducing the target image in a real robotic environment: A) a user drawing characters on a whiteboard, B) a robot taking a bitmapped image from its camera, C) a robot executing commands predicted by our proposed model in real-time D) finished process E) image of the user's drawing F) image of the robot's drawing

Our approach, in contrast, takes as input an image to draw, then generates commands for robots to replicate the image with a writing utensil. We divide the drawing problem into two scales: 1) the local scale, consisting of a $5 \times 5$ pixels window, and 2) the global scale, consisting of the whole image. We train two separate networks for the different scales. Unlike other approaches, our model does not require any predefined handcrafted rules, and learns drawing from stroke-order demonstrations provided only during its training phase.

We evaluate our system on a corpus of handwritten English, Japanese and Chinese characters [5] and show that it can look at a bitmap image of a character that it has not previously seen and accurately reproduce the character. In almost all instances it also predicts the correct stroke order and direction for the character. In addition we demonstrate that our approach enables two different robots to draw characters on paper and on a white board in 10 different languages as well as stroke-based drawings, including English, Chinese, French, Greek, Hindi, Japanese, Korean, Tamil, Urdu and Yiddish. Figure 1 shows an example of our robot reproducing the English word "hello."

## II. Related Work

Traditional motion planning methods are insufficient for robot writing because of the inability to specify the highly non-Markovian goals and constraints needed to produce a character stroke. For example, RTT [6] or RRT* [7] will find a path on our image plane, yet is likely to result in a crooked path that does not consider the continuity of each stroke. In our work, we assume that the drawing happens in a locally connected part of the robot's workspace, so that local movements do not cause global reconfigurations. We can then express the drawing motion in terms of these local movements. In practice with our two different robots, Baxter and Movo, we found this constraint was easy to satisfy. Recently, Rakita et al. [8] introduced an approach that finds series of locally movable positions for robots that do not require a global reconfiguration, which could be used to satisfy this contstraint automatically. Our representation is analogous to Logo [9] or Postscript [10]. Just as they described various shapes in their simplified format, our approach translates the image to multi-dimensional local movements, and our contribution represents steps toward multi-joint printers which achieve better reproduction of not only images but also objects in 3D.

Most existing approaches for robotic drawing focus on predicting an agent's movement on a target image plane using conventional image processing techniques, and do not learn from demonstrations to gain universal knowledge of drawing characters. Many previous methods segment a character into a set of strokes from its geometric properties [11, 12]. Mueller et al. [13] proposed an iterative training procedure to fit a spline into a stroke. However, its performance largely depends on a manually provided initial spline position, in contrast to our method which needs only an image of the drawing.

Xie et al. [14] explored authentic brush drawing, Sumi-e, with reinforcement learning methods. They manually designed a cone-shape brush agent with a tip point and a center of a circle with its radius, and formulated the drawing problem as an MDP. They solve the MDP using policy gradient methods, and showed working examples in their simulated environment. However, their approach requires manual labels of the starting and ending location for each drawing region. Importantly, they formulated the drawing problem as an MDP, although the writing problem is essentially non-Markov. Our approach, in contrast, makes the simplifying assumption that the writing implement is either touching or not touching the writing surface, but requires only a bitmapped image to infer a policy for drawing a new character.

More recent deep-RL approaches [15, 4] learn a general drawing policy, yet frequently do not draw each target stroke in one continuous fluent motion; instead it frequently backtracks over a previously-drawn area to write over it again. Their approaches involve less supervision than previous ones, since they do not require strokes as input. Yet such redundant drawing actions can potentially cause loss in details. Due to its less supervision, they also require millions of frames of data during training. Collection of training data in robotic environments is often expensive, and our approach achieves better performance with less training data, although our data is augmented with strokes. However we only need these strokes during training and at test time can reproduce a drawing given only a bitmapped image.

## III. Technical Approach

Given the target image of a handwritten character, $X^{target}$, our goal is to generate a sequence of actions, $A = \{a_1, a_2, ..., a_L\}$, for a robot to reproduce $X^{target}$. In our experiment, we define $X^{target}$ as a $100 \times 100$ binary image, and a command at timestep $t$ as $a_t = (\Delta x, \Delta y, touch)$ where $\Delta x$ and $\Delta y$ are shifts in $x, y$ coordinates that range between $-100$ and $+100$. The variable *touch* is a boolean value which controls the touch / untouch status of the writing utensil to the canvas.

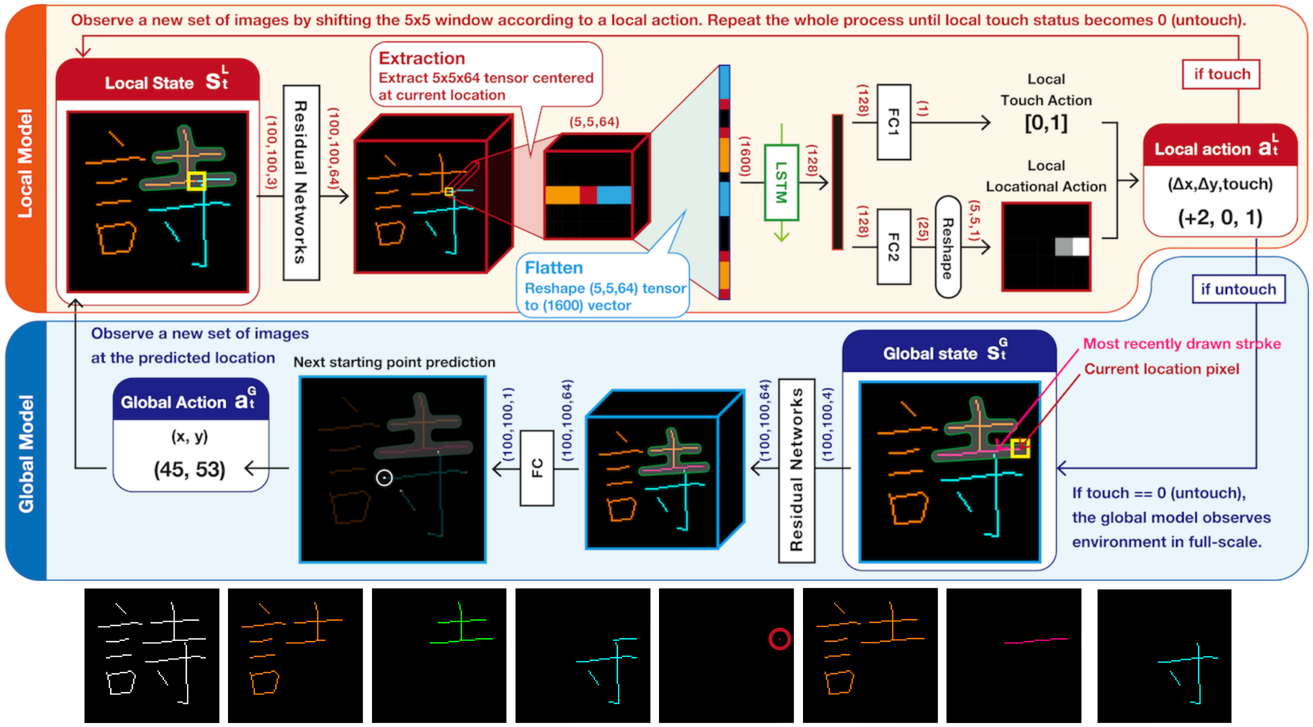
We aim to train a parametrized function approximator $f_\theta$ such that $A = f_\theta(X^{target})$. While it is possible to directly estimate $\theta$, we discovered that dividing the problem into two sub-problems and separately training two specialized distinct models achieves better performance. The first sub-problem is to make the drawing agent follow each stroke from its start to end. We design the *Local Model* with parametrized weights $\theta_L$ for this task. The second sub-problem is to predict the starting location of the next stroke at the end of current stroke. We designed the *Global Model* with weights $\theta_G$. The local model predicts where to move the pen-tip next in its $5 \times 5$ pixel environment. Once it reaches to an end, the global model predicts the next starting point of the new stroke. We repeat this process iteratively until the entire target image is visited by our network, and obtain the full action sequence $A = \{a_1^G, a_1^L, a_2^L, ..., a_n^L, a_m^G, a_{n+1}^L, ...\}$. We chose the size of canvas and numbers for tensor dimensions empirically, balancing performance and computation speed as well as memory usage. The overview of our network is shown in Figure 2.

### A. Local Model

Given the starting point, the goal of our local model is to follow the stroke until it reaches the end. A local state at timestep $t$, $s_t^L$, is a set of three images;

1) $X_t^{L_{env}}$ : already visited region by our local model,
2) $X_t^{L_{con}}$ : target region continuously connected with current location of our local model,
3) $X_t^{L_{dif}}$ : difference image between target image $X^{target}$ and $X_t^{L_{env}}$, which is indeed the unvisited region of the target image.

Figures 2b to 2d are the example of these three images.


Observe a new set of images by shifting the 5x5 window according to a local action. Repeat the whole process until local touch status becomes 0 (untouch).
Local Model
Local State $s_t^L$
(100,100,3)
Residual Networks
(100,100,64)
Extraction
Extract 5x5x64 tensor centered at current location
(5,5,64)
Flatten
Reshape (5,5,64) tensor to (1600) vector
(1600)
LSTM
(128)
(128)
FC1
(1)
Local Touch Action
[0,1]
(128)
FC2
(25)
Reshape
(5,5,1)
Local Locational Action
if touch
Local action $a_t^L$
(Δx,Δy,touch)
(+2, 0, 1)
if untouch
Global Model
Observe a new set of images at the predicted location
Next starting point prediction
Global Action $a_t^G$
(x, y)
(45, 53)
(100,100,1)
FC
(100,100,64)
Residual Networks
(100,100,64)
(100,100,4)
Global state $s_t^G$
Most recently drawn stroke
Current location pixel
If touch == 0 (untouch), the global model observes environment in full-scale.

(a) $X^{target}$ (b) $X_t^{L_{env}}$ (c) $X_t^{L_{con}}$ (d) $X_t^{L_{dif}}$ (e) $X_t^{G_{loc}}$ (f) $X_t^{G_{env}}$ (g) $X_t^{G_{last}}$ (h) $X_t^{G_{dif}}$

Fig. 2: Our proposed network architecture: comprised of two sub-models, local and global models. Numbers in color show shapes of tensors, and FC stands for a fully-connected layer. Our local state $s_t^L$ is a combination of $(X_t^{L_{env}}, X_t^{L_{con}}, X_t^{L_{dif}})$, and our global state $s_t^G$ is of $(X_t^{G_{loc}}, X_t^{G_{env}}, X_t^{G_{last}}, X_t^{G_{dif}})$. The FC layer of the global model is applied to the last dimension of the encoded tensor, as $1x1$ convolution of stride 1.

The unique characteristic of our local model design is that we apply an extraction procedure to the encoded tensor of shape $(100, 100, 64)$ to extract the $(5, 5, 64)$ tensor centered at the current location of the agent. The reasons why we extract local information are:

1) Generalization of knowledge: Every image of handwritten characters is different, and in order to gain general knowledge of drawing, it is crucial to work in smaller scale where an agent will encounter similar situations more frequently.
2) Computational expensiveness: Feeding large images directly into RNN (recurrent neural networks) to predict action sequence is computationally expensive. By extracting a small region, we can reduce the size of input tensors to RNN cells and achieve less computational expense and faster training.
3) Information selection: While the agent draws a stroke, the most important region to focus on is the one around the current location. In a broad view, our local network can be seen as a *structural* attention mechanism where we force our model to attend the $5 \times 5$ local region around the current location.

In order to preserve continuity in drawing actions, we use the Long Short-Term Memory (LSTM) [16] in our local network. As a simple example, when the agent reaches the intersection of two lines, it has choices of going either North, South, East, or West. If we know that the agent came from the North, we can make a reasonable guess that it should go South in order not to disrupt the continuity of the drawing motion.

Now, we define how our local network predicts the next action $a_t^L$. Given a local state at timestep $t$ as $s_t^L$ and current location as $(x_t, y_t)$, our local model first encodes the input tensor, $s_t^L$ using residual networks [17]:

$$e_t^L = f_{\theta_L^{Residual}}(s_t^L) \tag{1}$$

Our residual networks consist of four residual blocks, each of which contains two sub-blocks of 1) batch normalization layer [18], 2) rectified linear unit [19], and 3) two-dimensional convolutional layer. Convolution layers in these four blocks have channels of $[[16, 16], [16, 16], [16, 32], [32, 64]]$, stride of 1, width of 3 and height of 3. After the residual networks, we have an encoded tensor $e_t^L$ of shape $(100, 100, 64)$, and we then apply the extraction procedure to $e_t^L$ centered at $(x_t, y_t)$ and receive a new tensor, $e_t^{L^*}$ of shape $(5, 5, 64)$. To feed $e_t^{L^*}$ into the LSTM, we reshape it into a vector $v_t^L$ of length $5 \times 5 \times 64 = 1600$:

$$v_t^L = reshape(e_t^{L^*}) \tag{2}$$

We feed $v_t^L$ to the LSTM and receive context vector $c_t^L$ and hidden state representation $h_t^L$ as:

$$c_t^L, h_t^L = f_{\theta_L^{LSTM}}([v_t^L; h_{t-1}^L]) \tag{3}$$

Two components of local action $a_t^L$, the local touch action $a_t^{L_{touch}}$ and the locational action $a_t^{L_{loc}}$ are calculated from context vector $c_t^L$:

$$\begin{aligned} a_t^{L_{touch}} &= \sigma(f_{\theta_L^{FC1}}(c_t^L)) \\ a_t^{L_{loc}} &= \text{argmax } f_{\theta_L^{FC2}}(c_t^L) \end{aligned} \tag{4}$$

where $\sigma$ is a sigmoid activation function. Finally, the loss function for our local model at timestep $t$ is given as two cross-entropy values for locational and touch action:

$$L_t^{Local} = L_t^{Local^{Loc}} + L_t^{Local^{Touch}} \tag{5}$$

The locational loss is a cross-entropy for predicted action $a_t^L$ and ground truth action $a_t^{L^*}$. Because locational actions have shape of $(5, 5)$, to compute a scalar cross-entropy value, we reshaped $a_t^{L^*_{loc}}$ to a one-hot vector of shape $(1, 25)$ and $a_t^{L_{loc}}$ to $(25, 1)$.

$$L_t^{Local^{Loc}} = -a_t^{L^*_{loc}} \log(a_t^{L_{loc}}) \tag{6}$$

Note that this log function is element-wise. Similarly, we have a predicted touch action value $a_t^{L_{touch}}$ as a scalar and $a_t^{L^*_{touch}}$ as a binary scalar label.

$$\begin{aligned} L_t^{Local^{Touch}} =& - a_t^{L^*_{touch}} \log(a_t^{L_{touch}}) \\ & - (1 - a_t^{L^*_{touch}}) \log(1 - a_t^{L_{touch}}) \end{aligned} \tag{7}$$

### *B. Global Model*

The goal of our global model is to predict the starting point of the next stroke in a full-scale image plane. When $a_t^{L_{touch}} = 0$, the global model observes the current state $s_t^G$, which is a set of four images in Figures 2e to 2g,

1) $X_t^{G_{loc}}$ : current location of our local model,
2) $X_t^{G_{env}}$ : already visited region by our local model,
3) $X_t^{G_{last}}$ : recently visited region by our local model since the last global prediction,
4) $X_t^{G_{dif}}$) : difference image between target image $X^{target}$ and $X_t^{G_{dif}}$.

The global network also has the residual network to encode state images, and it shares all weights with the one in our local model, except for the very first initialization layer. To adjust the channel size of input tensors, the initialization layer in our residual network maps a tensor of shape $(N, N, M)$ to $(N, N, 16)$. Due to the discrepancy in shapes between local and global states, the size of the layer is different. We obtain the global action $a_t^G$ as:

$$\begin{aligned} e_t^G &= f_{\theta_G^{Residual}}(s_t^G) \\ c_t^G &= f_{\theta_G^{FC}}(e_t^G) \\ a_t^G &= \underset{(x,y)}{\text{argmax}}\ c_t^G \end{aligned} \tag{8}$$

and the loss function for the global model at timestep $t$ is cross-entropy for predicted action $a_t^G$ and ground truth action $a_t^{G^*}$. Both $a_t^G$ and $a_t^{G^*}$ have shape of $(N, N)$ but converted to shapes of $(N^2, 1)$ and $(1, N^2)$ respectively to calculate the scalar cross entropy value by taking the dot product of the two vectors:

$$L_t^{Global} = -\, a_t^{G^*_{loc}} \log(a_t^{G_{loc}}) \tag{9}$$

## IV. Evaluation

The aim of our evaluation was to assess the trained model's performance at generating policies for writing characters in both simulation as well as on a real robot.

### *A. Training Procedure*

To generate sufficient amount of training data for our model in simulation, we use KanjiVG [5], a database of Japanese Kanji characters. This database consists of both images of the character as well as an ordered list of strokes in the SVG format. To retrieve a sample, we randomly selected points on each defined curve, and for the $i$th stroke of the character, we have an ordered list of actions; $a_i^{RAW} = \{a_{i,0}^{RAW}, a_{i,1}^{RAW}, ..., a_{i,N}^{RAW}\}$. We then decompose raw actions $a_i^*$ into local actions which are in a strict range of $-2$ and $+2$. This leads us to obtain $a_i^{L^*} = \{a_{i,0}^*, a_{i,1}^*, ..., a_{i,M}^*\}$, where the size of the new action sequence $M$ tends to be much larger than the size of the raw action sequence $N$. For the global model, we collected the starting point of each stroke. We used 10,000 unique characters for training and left 2,000 for validation. To aid in generalization, we also applied shifting, scaling and shearing effects to images and associated strokes.

We initially planned to train our model on a character basis, which consists of multiple strokes. A character sample in our modified Kanji dataset has about 12.5 strokes, each of which involves of 15.8 local actions, leading to 200 local actions per character on average. Due to the relatively large size of full-scale images ($100 \times 100$), training a whole character at once in a recurrent manner is computationally expensive. To prevent our local model from carrying information about hundreds of past actions, we decided to train our network on a stroke basis to shorten the length of training sequences from a few hundred to 1~20. This decision means that we disregard all dependency between a current stroke and its past strokes. We train each stroke as if it just started drawing, by resetting the LSTM states to zero for every beginning of a new stroke. For training neural networks, we use Adam [20] with a learning rate of $1e^{-4}$.

To measure the performance of our approach, we introduce two metrics; pixel accuracy and stroke accuracy. Pixel accuracy measures how similar the target image and the drawn image are, by calculating $TP/(T + P - TP)$, where $T$ stands for the number of true pixels (nonzero pixels in the target image), $P$ for positive pixels (nonzero pixels in the drawn image), and $TP$ for true-positive pixels (nonzero pixels in both the target and the drawn image). We also measure the performance of our models by stroke accuracy, which checks if the model drew one stroke in one continuous action. This metric is calculated by 1) comparing each drawn stroke with every

| ID | Description | Pixel Accuracy | Stroke Accuracy |
|---|---|---|---|
| 1 | Our Proposed Model in Figure 2 | 0.9988 (0.9990) | **0.9630** (**0.9646**) |
| 2 | Model 1 + (a) Replacement of a LSTM cell with a Fully-Connected layer | 0.9985 (0.9973) | 0.9564 (0.9532) |
| 3 | Model 1 + (b) Replacement of Extraction Procedure with Initial Extraction | 0.9964 (0.9957) | 0.8746 (0.8822) |
| 4 | Model 3 + (a) Replacement of a LSTM cell with a Fully-Connected layer | 0.9995 (0.9999) | 0.8666 (0.8809) |
| 5 | (c) Global Model without Local Model | 0.2693 (0.2773) | 0.1565 (0.1680) |

TABLE I: Pixel and Stroke Accuracy for Training and Testing Data in Various Models (Testing results in parentheses)

target stroke, 2) collecting the best $TP/(T + P - TP)$ score between stroke images, and 3) taking an average for all drawn strokes to obtain a score for a character sample. We randomly chose 100 unique samples from our training and testing dataset, and calculated these measure.

To evaluate our model performance, we made three types of modifications to our model.

(a) First, we replaced the LSTM cell in our local model with a fully-connected layer to discard sequential aspects from our network. This ablation tests the effectiveness of our recurrent approach to preserve continuity of drawing actions.

(b) Second, we considered when to apply our extraction procedure. Instead of waiting to extract a $(5, 5, 64)$ tensor from the $(100, 100, 64)$ encoded tensor, we extract a $(5, 5)$ patch from each image in our local state to obtain a $(5, 5, 3)$ tensor, feed it into the residual network and receive another $(5, 5, 64)$. We name this procedure as *Initial Extraction* to differentiate it from our original extraction procedure.

(c) Finally, we removed the local model and modify our global model to form a loop to itself to generate a sequence of actions. This ablation measures the effects of our local model. The modified global model needs to predict the next locational shift as well as touch/untouch status, and for touch status detection, the global network adds an extra fully-connected layer. The locational action space is not bounded to $-2$ and $+2$, so we trained this model with $a^{RAW}$ instead of $a^{L}$ as target actions.

We now have 4 distinct models as our baseline, and Table I displays the results for each model for the training and testing datasets. To compare accuracy scores for different models, we terminated the training process after feeding a randomly sampled stroke data from the training set to the model for 100,000 times.

### B. Results

All models except for Model 5 achieved the near-perfect pixel accuracy for both training and testing data. This demonstrates the effectiveness of our approach to locally solve the drawing problem for each stroke. The main contribution of our local model to the entire network is that it restricts the action space to a local $5 \times 5$ pixel region around the current location. While Model 5 needs to select one pixel to move next from $10,000$ choices, the local model has only 25 choices.

The best stroke accuracy was achieved by our proposed model (Model 1), and with the Model 2 results, it is apparent that extraction mechanism in the local model plays an essential role in improving the performance.

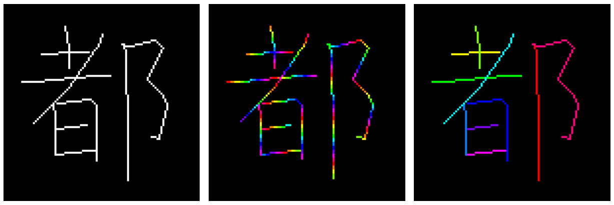
However, it is important to spotlight that LSTM did not contribute much for improving accuracy. Comparing Model 1 with 2 and Model 3 with 4, the performance gain is very small. Still, the recurrent approach did not harm our performance, and in our future work, we will investigate these architectures in a more dynamic writing domain, such as a paint brush, where the state of the drawing agent is largely dependent on past executed sequence of actions.

Finally, an example of our model successfully replicating the target image in its simulated environment is shown in Figure 3. Different colors in Figure 3c indicate different strokes, and how our model naturally segmented the static image into a set of strokes.

### C. Robotic Demonstration

To illustrate our system works in various robotic environments, we tested our model with two robots, Baxter and Movo. We directly apply our trained model to the real robotic environment, which creates a need to preprocess the original target image to match the image format of our training data, such that the line width has to be 1, and the image has to be size of $100 \times 100$, and so on. If our model sees a vertically-long one-stroke drawing, it is likely to divide the stroke into regions, individually solve the drawing problems, and combine the results together once all is completed. To adjust the line width, we used technique of skeletonization [21] which extracts the center line of a stroke-based drawing.

*a) Baxter:* Baxter first takes an image of a hand-written characters from its wrist camera, and then generates relevant commands and starts drawing with its marker, which we rigidly attached to the gripper. This



(a) target image (b) local moves (c) segmentation

Fig. 3: Example of our model reproducing the target image in a simulated environment. Left : target image, Center : trace of our local network scanning the whole target image, Right : segmented strokes by local and global model

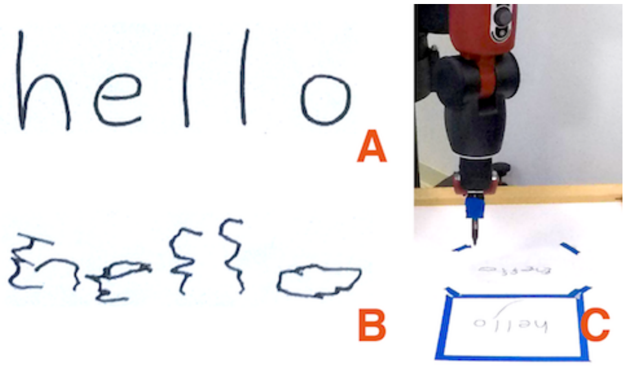


Fig. 4: Demonstration of Baxter reproducing the target image : A) the target image the Baxter tried to replicate B) the drawn image by the Baxter C) the Baxter in motion

process is shown in Figure 4, and the drawn characters reproduce the input characters and general appearance. However there is significant error in the drawn characters, due to position errors in the robot's end effector. In the future we plan to explore an end-to-end closed loop learning of policies for writing to mitigate this problem.

*b) Movo:* We tested our model on our MOVO robot, using the Kinova Jaco arm and the Kinect 2 as a sensor. With its precise movement capabilities MOVO reproduces the target image very accurately. Overall, the robot demonstration produces a policy for drawing recognizable characters, including languages such as Greek, Hindi and Tamil, which were not previously seen during training. Figure 1 shows scenes from a robot reproducing English text written on a whiteboard. Photographs of drawn and handwritten examples appear in Figure 5.

Our model's ability to reproduce English cursive, as shown in Figure 5, raises the question of the ability of this framework to reproduce handwritten signatures. For example, given an image of a signature, our approach could infer a policy for reproducing that signature with a pen held by a robot. This capability has ramifications in areas where signatures written with a writing utensil are used to verify agreement, such as legal documents.

## V. Conclusion

Overall we have presented an approach for inferring a sequence of commands for drawing a character given a bit-mapped image of the character. We demonstrated that our approach gains general knowledge of handwriting and replicates the target image in both simulation and two different robotic environments. Most significantly, our trained model accurately predicts drawing procedures for foreign characters, such as Greek, Hindi and Tamil, which are not in the training dataset, as well as arbitrary line drawings.

In the future, we will explore more advanced types of drawing such as drawing with a paint brush, where the speed of the stroke, the height/altitude/azimuth of the brush relative to the canvas plane all affect the drawing result. In this vein we are interested in closing the learning loop by enabling the robot to inspect what it has drawn and improve its drawing policy based on observation of what it actually drew. We would also explore the connections between writing and motion planning, since we should plan motions to draw a character that satisfy constraints such as being in the robot's work space and avoid collisions with other objects.

In the longer term, we see analogs between drawing and other tasks such as cutting with a saw, frosting a cake, or inferring a policy for a laser cutter. We would like to apply our techniques and algorithms for enabling a robot to creatively infer policies for these sorts of tasks.

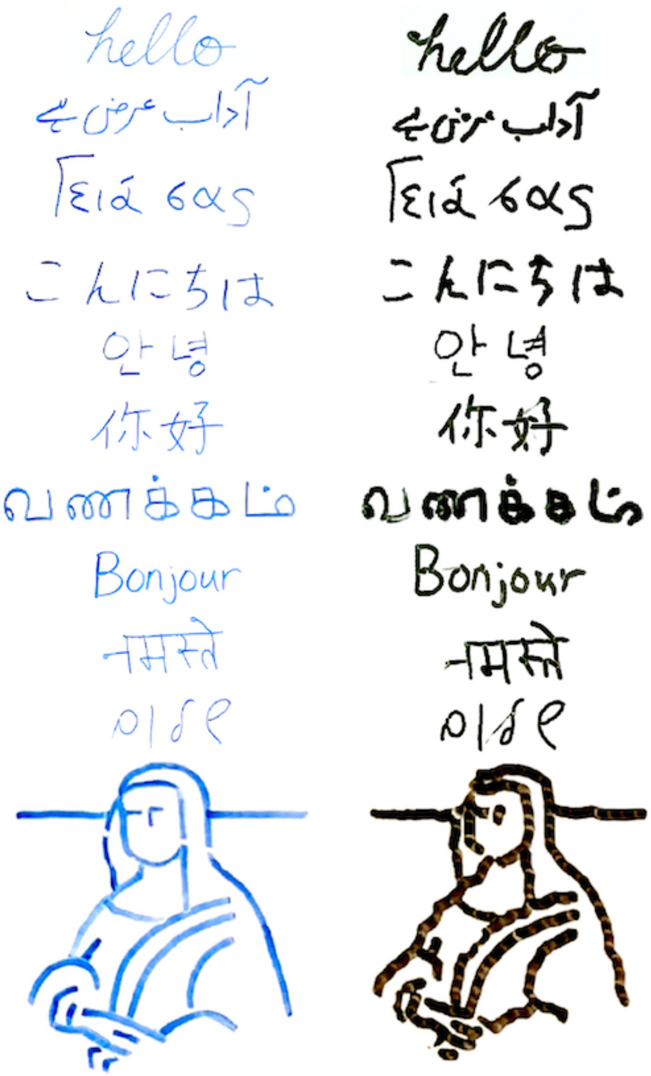


Fig. 5: "Hello" in different languages: from the top - English cursive, Urdu, Greek, Japanese, Korean, Chinese, Tamil, French, Hindi and Yiddish, and a sketch of the Mona Lisa. Blue strokes on the left are hand-drawn on a white board; black strokes on the right are drawn by the robot on the same white board after viewing the input image on the left.

## VI. Acknowledgments

This work is supported by the National Science Foundation under grant number IIS-1652561.

## References

[1] Sylvain Calinon and Aude Billard. Learning of gestures by imitation in a humanoid robot. Technical report, Cambridge University Press, 2007.

[2] Fei Chao, Yuxuan Huang, Xin Zhang, Changjing Shang, Longzhi Yang, Changle Zhou, Huosheng Hu, and Chih-Min Lin. A robot calligraphy system: From simple to complex writing by human gestures. *Engineering Applications of Artificial Intelligence*, 59:1–14, 2017.

[3] Deanna Hood, Séverin Lemaignan, and Pierre Dillenbourg. When children teach a robot to write: An autonomous teachable humanoid which uses simulated handwriting. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, pages 83–90. ACM, 2015.

[4] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. *arXiv preprint arXiv:1804.01118*, 2018.

[5] Ulrich Apel. KanjiVG. `http://kanjivg.tagaini.net/`, 2009. [Online; accessed 25-August-2018].

[6] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[7] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[8] Daniel Rakita, Bilge Mutlu, and Michael Gleicher. Relaxedik: Real-time synthesis of accurate and feasible robot arm motion. In *Proceedings of Robotics: Science and Systems*, jul 2018. doi: 10.15607/RSS.2018.XIV.043. URL `http://graphics.cs.wisc.edu/Papers/2018/RMG18a`.

[9] Roy D Pea. Logo programming and problem solving.[technical report no. 12.]. 1983.

[10] Adobe Press. *PostScript language reference manual*. Addison-Wesley Longman Publishing Co., Inc., 1985.

[11] Fenghui Yao, Guifeng Shao, and Jianqiang Yi. Extracting the trajectory of writing brush in chinese character calligraphy. *Engineering Applications of Artificial Intelligence*, 17(6):631–644, 2004.

[12] Yuandong Sun, Huihuan Qian, and Yangsheng Xu. A geometric approach to stroke extraction for the chinese calligraphy robot. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3207–3212. IEEE, 2014.

[13] Samuel Mueller, Nico Huebel, Markus Waibel, and Raffaello D'Andrea. Robotic calligraphy—learning how to write single strokes of chinese and japanese characters. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1734–1739. IEEE, 2013.

[14] Ning Xie, Hirotaka Hachiya, and Masashi Sugiyama. Artist agent: A reinforcement learning approach to automatic stroke generation in oriental ink painting. *IEICE TRANSACTIONS on Information and Systems*, 96(5):1134–1144, 2013.

[15] Kazuma Sasaki, Hadi Tjandra, Kuniaki Noda, Kuniyuki Takahashi, and Tetsuya Ogata. Neural network based model for visual-motor integration learning of robot's drawing behavior: Association of a drawing motion from a drawn image. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 2736–2741. IEEE, 2015.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[19] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[20] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] Ta-Chih Lee, Rangasami L Kashyap, and Chong-Nam Chu. Building skeleton models via 3-d medial surface axis thinning algorithms. *CVGIP: Graphical Models and Image Processing*, 56(6):462–478, 1994.