Robot Calligraphy using Pseudospectral Optimal Control and a Simulated Brush Model

Jiaqi Chen
advised by Frank Dellaert

December 12, 2019

Abstract

Chinese calligraphy is unique and has great artistic value but is difficult to master. In this paper, we make robots write calligraphy. Learning methods could teach robots to write, but may not be able to generalize to new characters. As such, we formulate the calligraphy writing problem as a trajectory optimization problem, and propose a new virtual brush model for simulating the dynamic writing process. Our optimization approach is taken from pseudospectral optimal control, where the proposed dynamic virtual brush model plays a key role in formulating the objective function to be optimized. We also propose a stroke-level optimization to achieve better performance compared to the character-level optimization proposed in previous work. Our methodology shows good performance in drawing aesthetically pleasing characters.

Contents

1 Introduction

Chinese calligraphy is an art form that takes humans years of practice to master. Chinese characters are complex and a calligraphic brush is difficult to manipulate properly. In this paper we aim to make a robot write Chinese characters by using a simulated brush model and pseudospectral optimal control methods to optimize for a trajectory (more about pseudospectral optimal control in later sections).

Most research on making actual robots create art adopts either a learning-based or a trajectory optimization-based approach. The former often comes down to teaching by demonstration [1, 2], or self-correction [3]. By using learning methods, one can skip the difficulty of modeling the behavior of a real calligraphy brush. However, learning methods also have a large training cost and may not generalize well to unseen characters. On the other hand, trajectory optimization-based methods do not face these problems. In this research, we *simulate* the writing behavior of an actual brush, and then search for an optimal trajectory for the robot to execute [4, 5]). However, most simulated brush models [5, 6] do not account for the complex ways a brush deforms during the writing process. Being able to capture this complexity has an important influence on the final performance.

We propose a trajectory optimization-based method based on principles from pseudospectral optimal control [7,8], and also introduce a new dynamic virtual brush model to achieve fully automatic writing of Chinese characters. Pseudospectral methods are generally used for optimizing continuous trajectories and controls, but we assume the control is realized by the low-level inverse kinematics solvers on the robot. This essentially means that the robot will handle the specific motor control calculations while we only have to feed a trajectory of coordinates, (x, y, z) to execute. Different from previous work which optimizes the trajectory for the whole character at once [4, 5], we decompose the character into strokes and perform stroke-based optimization. Full character-based optimization can be computationally expensive and get stuck in local minima. We extract strokes and create initial trajectory estimates by leveraging the properties of vector-based character databases. The proposed virtual brush concentrates on the dynamic mechanisms of an actual calligraphic brush but has a simpler structure compared to previous work [9, 10]. As a baseline, we compare with a virtual brush model similar to Kwok et al. [4].

The primary contributions of this paper are:

1. We use pseudospectral methods to search for optimal writing trajectories to apply to calligraphy robots. Pseudospectral methods have natural modeling abilities for continuous trajectory optimization.

2. We design a new virtual brush model. Such a model is also able to simulate the real brush dynamics with higher accuracy, which leads to better optimized trajectories.

3. We exploit vector-based character libraries for easy stroke extraction and initialization for the stroke-level trajectory optimization.

2 Related Work and Literature Review

This section concentrates on calligraphy robots, even though there are many other art forms that incorporate robotics [11, 12, 13], such as sculpture [14], graffiti [15], etc. Most algorithms on calligraphy robots using a brush pen can be categorized as learning-based methods or trajectory optimization-based methods.

2.1 Calligraphy robots using learning-based methods

Many learning-based algorithms have been used for calligraphy robots in recent years. Sun et al. [1,16] propose to learn from demonstration. They invite calligraphers to write characters while holding the robot arm and record the robot joint positions to establish a mapping model for robot control. Mueller et al. [3] propose an iterative learning method by trial-and-learn. A camera is used to take pictures of what the robot writes down and then the stroke trajectories are optimized toward minimizing the difference between written pictures and reference images.

Some more advanced learning algorithms such as RNN [17], generative adversarial networks [18], deep reinforcement learning [19], local and global learning models [2], are also explored. These methods usually require many iterations of training to achieve good performance, which is inconvenient. Generalizing to new, not learned, and complicated characters is difficult.

2.2 Virtual brush models

Virtual brush models are mainly used in trajectory optimization-based algorithms. We can divide virtual brush models into two categories: physics-based models and data-driven models. With the help of this virtual brush, we are able to optimize for a trajectory for robots to execute.

Physics-based virtual brush models strive to simulate the physical dynamics of a real brush from experimental observation [20] or physical laws [10]. Strassmann [21] proposes an initial design featuring four basic parameters of a hairy brush. Wong et al. [22] propose to use a cone to represent the bundle of the brush and use the cross-section of the cone, an ellipse, to represent the footprint. Xu et al. [9] propose a virtual brush model with much detail and complex mechanisms obtained from approximations and assumptions. However, obtaining and fitting good parameters to complicated virtual brush models mentioned above is difficult. As such, we propose a virtual brush that is very similar to Xu's model, but it has an easy structure to fit and implement, and even makes real-time trajectory optimization possible.

Data-driven virtual brush models are created from measuring and recording actual brush footprints. Kwok et al. propose a very simple virtual brush which draws droplet shapes with its size proportional to the writing height [4]. In their later work [6], they use a camera placed below the writing plane to collect footprints during the writing process. Lam et al. [5] define their writing mark as a polygon connected by eight points and fit their position parameters with the collected footprints. Considering the big calculation cost, Baxter et al. [23] build a deformation table and makes the calculation process much faster with complicated simulation effect.

2.3 Stroke extraction

Stroke extraction involves separating a character into its comprising strokes, and is difficult to do with good accuracy when analyzing just pixels of an image. We measure the "goodness" of stroke extraction by manually comparing images of extracted strokes to images of the correct strokes. There are three main categories of stroke extraction methods: skeleton-based [24, 25], region-based [26], and contour-based [27, 28, 29]. Most of these methods are complicated and cannot promise good accuracy between the extracted stroke and actual stroke. As such, we propose to use vector-based images as our character dataset, which, in contrast, provides a quick and accurate way to extract strokes.

2.4 Optimization methods

As mentioned above, Kwok et al. [4,6] propose to use Bezier curves (parametric curves that represent polynomials in a different way) to represent stroke trajectories and find the optimized trajectory by minimizing differences between simulated images and desired character images. But their algorithms have not yet been applied to actual robots. Furthermore, their algorithms perform character-level optimization, which is computationally expensive and cannot handle complicated characters without needing human intervention. Lam et al. [5] propose to minimize the width difference of the strokes between reference images and a simulated image written by the virtual brush as it moves along the middle axis of each stroke. However, their method is sensitive to small variations in stroke images, and so the results suffer from a loss of smoothness. The optimization method in Kwok's work is similar to our system, and so the differences should be clarified. First, their virtual brush is not able to predict the dynamic behavior of physical calligraphy brushes accurately while our dynamic virtual brush can. As such, their virtual brush could not guide the robots to write things in the correct position because they do not know what happens given one control command. You can see our brush's dynamic behavior as the "Inner State Sequence" in Fig. 3, where the virtual brush parameters get updated because the brush state changes due to deformations during the writing process. Second, their optimization method is inefficient and requires manual initialization, while ours is more efficient, performs better with complicated characters, optimizes at a stroke-level, and has an easier initialization. Third, using a genetic algorithm means a large calculation cost. Our optimization is computationally efficient and could potentially be used in a closed-loop control system for real-time optimization. Lastly, Kwok et al. only mention trying their system on actual robots as future work.

3 Progress and Evolution of Project

In the beginning of this project, after doing the literature review and understanding current existing work on robot calligraphy, we looked into datasets of Chinese characters. After searching for a while, we found that instead of just using JPG images, we could leverage stroke specific information about each character if we used SVG files. There

were existing databases online, such as MakeMeAHanzi, that could offer these SVG files. We first learned how to decompose each SVG into its respective Bezier Curves and then use those curves to draw a 'skeleton' of the desired Chinese character.

Figure 1: SVG images of characters YI, SHI, ER, NIAO, meaning 'one', 'stone', 'two', 'bird', with each stroke separated out as well as transformed into coordinate points that are easier for the robot to execute.

Because the 'skeleton' of each Chinese character was represented by Bezier Curves, we could then move onto making the robot execute a path that follows these curves. We started by attaching a marker to the robot's end effector and then calculating the exact control points that we want to robot to move along in order for it to write a specific character. Preliminary results of this are shown here.

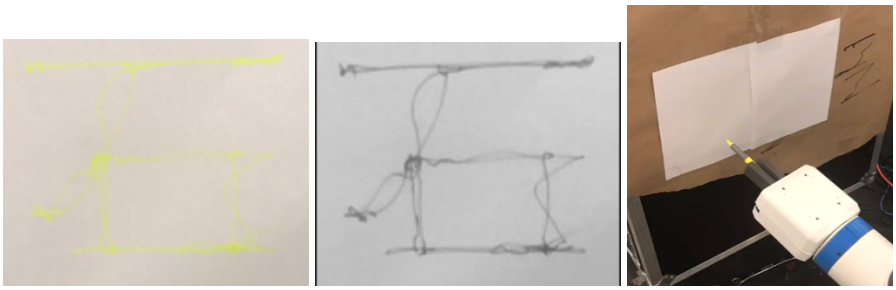


Figure 2: From left to right, original character drawn by robot, original with added contrast, robot drawing with marker

After we were able to make the robot execute a trajectory, we moved on to implementing a virtual brush that could simulate a real Chinese calligraphy brush. This was the next step because in order for us to optimize for a good trajectory for the robot to execute, we must first find a good formalization of the parameters we want to optimize. Having a virtual brush also means that we can simulate a robot writing Chinese calligraphy using ROS and Gazebo. As such, we aimed to implement a virtual brush modeled after the one by Xu et al. [20]. However, this brush was much too complex to build from scratch. Therefore we decided to started with a simple virtual brush, which captures the essence of a writing tool by having an x and y variable to represent its location above paper, and a z variable to represent its height above paper.

From the simple brush, we were able to evolve the virtual brush into a more dynamic virtual brush that captured more parameters of a real brush such as the length and width of contact of the brush tip with paper, the orientation of the brush tip, as

well as the direction of the brush movement. By using both the simple and dynamic virtual brush, we show that having a virtual brush that simulates a real brush as closely as possible would make the optimized trajectory much better. We also incorporated the use of Factor Graphs at this point by representing our Chinese character trajectory with Chebyshev polynomials (pseudospectral methods), making optimization possible.

4 Methodology

This following section details the methodology of our calligraphy robot. First we describe the vector-based image dataset of Chinese characters we are using, then the two virtual brushes that are used during trajectory optimization, and finally the optimization methodology itself.

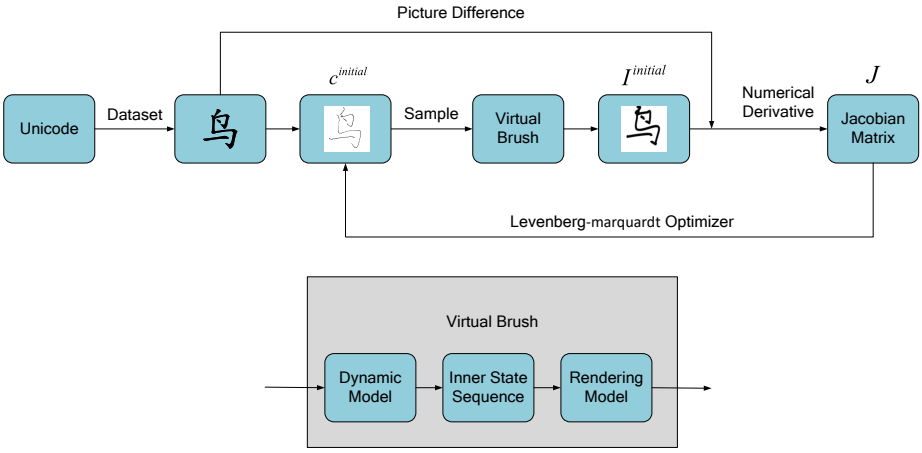


Figure 3: Flowchart of the Methodology

4.1 Dataset of Chinese characters

Vector-based images are a way to store images such that the details of the image are generated by mathematical formulations of curves. You can think of these mathematical formulations as describing different line segments for each stroke. As such, for a Chinese character, we can extract the parameterized lines that represent each stroke, instead of just looking at raw pixel data. This will be helpful later on because we want to optimize our trajectory stroke-by-stroke.

For this paper, we choose a Scalable Vector Graphics (SVG) dataset from MakeMeAHanzi. SVG files are the World Wide Web Consortium (W3C) standard for vector-based images, and are widely adopted by web browser applications. SVG files use Bezier curves to represent an image. As mentioned before, Bezier curves are parameterized and continuous curves defined by a set of control points. We can think of them as a different way to represent polynomials, where the line is "anchored" by the control

points. The dataset has a collection of SVG files, each depicting a different Chinese character with separable strokes.

4.2 Optimization based on a simple virtual brush

As described before, we use two virtual brushes for optimization, one of which is the simple virtual brush. The two brushes are used for comparison. Given a control trajectory defined by a set of coordinates x, y, z, the simple brush essentially draws a circle of radius $\hat{z}$ where $\hat{z}$ is dependent on the height of the brush from the paper.

4.2.1 Simple virtual brush model

Before we get into too much detail, it is worth mentioning that x, y, z represents the control command given to the robot whereas $\hat{x}, \hat{y}, \hat{z}$ represents the control command given to the virtual brush, where $\hat{z}$ is actually the radius of the circle drawn and the coordinates are in the reference frame of an image.

Our simple virtual brush is similar to the one in [4] except that our brush draws a circle while their brush draw a droplet shaped mark. Given a control sequence of x_i, y_i, z_i, where i denotes the ith control command, the simple virtual brush draws a circle with a radius $\hat{z}$ at the location x_i, y_i of the image.

The circle that we draw is not a solid circle, but instead has a radial distribution from the center that resembles a Sigmoid function:

$$r = \frac{\sqrt{(\hat{x}_i - m)^2 + (\hat{y}_i - n)^2}}{\hat{z}_i} \tag{1}$$

$$p_{m,n} = 255 \frac{1}{1 + e^{-10r}} \tag{2}$$

where r is the ratio of the distance between the coordinates m, n of the image frame and the center of the current circle, and the radius of the circle $\hat{z}$ that should be drawn. Therefore the $p_{m,n}$, the pixel value at coordinates m, n of the image is a scaled Sigmoid of r.

A key feature of our virtual brush is that it creates images that are differentiable. This means that instead of drawing image matrices where pixels are represented by integer values between $[0, 255]$, our image pixels are represented by floats, or decimal point numbers, defined by a Sigmoid function, refer to (2).

The motivation behind a continuous function describing our pixel values is because our optimization method will eventually be performing gradient descent on our pixel values, meaning they have to be differentiable.

4.2.2 Simple virtual brush initial stroke trajectory generation

As mentioned before, we use control points x, y, z to represent the robot trajectories, but for optimization, we use $\hat{x}, \hat{y}, \hat{z}$ so as to not confuse the real world trajectory and simulation trajectory. These $\hat{x}, \hat{y}, \hat{z}$ are initialized as Chebyshev-Gauss-Lobotto

points that we sample from the Bezier curve representations of the vector-based images. Chebyshev-Gauss-Lobotto points [30], also known as Chebyshev nodes [31], are a way of selecting a set of points from a curve, but instead of selecting points uniformly, we sample more densely at the beginning and end of the curve.

Having a good initial estimation has a big influence on the performance of our trajectory-optimization algorithm, because it lessens the work that the optimization algorithm has to do to get to an ideal solution. We initialize our trajectory by using the "skeleton" of the character extracted from our vector-based images. This is because the skeleton already offers the general shape, direction, and substructures of each character, and we just want to fine-tune and perfect this trajectory.

Fig. 4 shows an example of the Chinese character 'bird' with the path, $\hat{x_i}, \hat{y_i}$ that we use for initialization show in the middle of each stroke.

Going more into detail about Chebyshev-Gauss-Lobatto (CGL) points [30], they represent roots of the Chebyshev polynomials of the first kind. Sampling based on these points rather than sampling uniformly, we can avoid numerical instability or wild behaviors in the resulting trajectories [32]. Using CGL points also minimizes the oscillations of the fitted curve, which is important because with oscillations, small perturbations or errors in the beginning of optimization will be magnified as optimization continues.

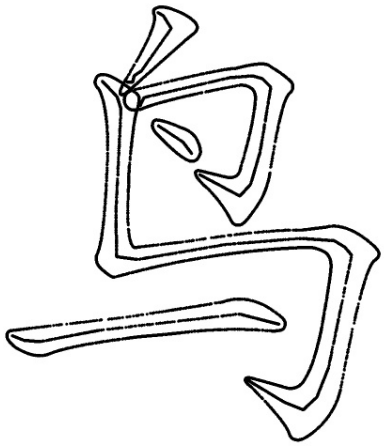


Figure 4: An example character of 'Bird' in the dataset, the 'skeleton' is shown in the middle of each stroke

4.2.3 Optimization for stroke trajectories using simple virtual brush

Using the simple virtual brush, we optimize the stroke trajectory by minimizing the pixel differences between the image drawn by the virtual brush and the desired image.

$$\min_X C = \{\| V(X) - I_i \|\} \tag{3}$$

$$X = [x_0, ..., x_N; y_0, ..., y_N; s(z_0, ..., z_N)] \tag{4}$$

This objective function is described as finding the set of control points X that minimizes the L2-norm between the image drawn by the virtual brush $V(X)$ and the desired image I_i. Values of $V(X)$ and I_d are an image matrix represented by a matrix of pixels. The points in X represent the CGL points mentioned earlier. The points z_i are also multiplied by a scale s that converts the values of z_i to the same magnitude as x_i and y_i, for ease of optimization. The subscript i in this case represents the ith control command.

We solve this objective function with the Levenberg-Marquardt (LM) algorithm which is a combination of the Gauss-Newton algorithm and gradient descent:

$$X_{i+1} = X_i + \delta$$

$$[\mathbf{J}^T\mathbf{J} + \lambda \mathbf{diag}(\mathbf{J}^T\mathbf{J})]\delta = \mathbf{J}^T[V(X_i) - I_i]$$

where δ is the update of X at each iteration. The Jacobian matrix J is the numerical derivative of $V(X_i)$. By controlling the damping factor λ, the LM algorithm appears more similar to gradient descent when we are far from an optimal value and more like Gauss-Newton, which updates X more precisely when we are close to an optimal solution.

4.3 Optimization based on a dynamic virtual brush

In addition to a simple virtual brush, we also propose a new dynamic virtual brush that has higher accuracy and performance. The basic concept of using this brush to perform optimizations is the same, but the brush is parameterized differently to provide a more accurate simulation of a real calligraphy brush.

4.3.1 Dynamic virtual brush model parameterization

The dynamic virtual brush uses four state parameters to describe the writing mark that it will leave on paper: width w, drag d, offset o, and orientation α, as shown by Fig. 5. Width w describes the width of the writing mark, drag d describes the length of the writing mark, offset o describes the distance from the writing mark to the center of the brush handle, and orientation α describes the the orientation of the writing mark. An orientation α of 0 means that the mark is pointing towards the right. These parameters all depend on the height of the virtual brush $\hat{z}$. The curve that we use to draw the mark is a quadratic curve defined by drag d and width w. We also do not allow the brush hairs to "split" during writing, as it sometimes do when the brush is pressed too hard on paper. We do not account for this feature at it is unconventional for brush hairs to split during the calligraphy process. When splits do happen during the writing process, we reshape the brush by dipping it in ink, and rewrite the character.

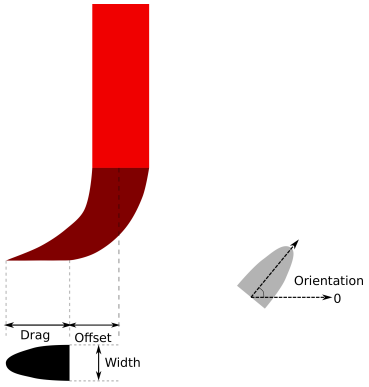


Figure 5: Dynamic virtual brush model and its main components: width, drag, offset, and orientation

4.3.2 Dynamic virtual brush parameter updating

During the writing process, according to the control trajectory (x, y, z), the virtual brush updates its state parameters. This is because in real calligraphy, the virtual brush gets deformed after writing a stroke and we account for this as a part of our model.

Parameter updating process. The parameters depend on the change in brush height z. The relation between z and the each parameter is found through experimentation and measurement. We use a brush to draw many brush footprints/marks and then measure the width, drag, and offset as well as record the brush height z. Then we fit a linear relationship to each of the parameters. Fig. 6 shows an example of how we fit the drag d with respect to brush height z. The influence of writing speed is ignored based on experiments in Kwok's work [33], because it does not significantly change the resulting character, assuming that the brush writes at a reasonable speed.

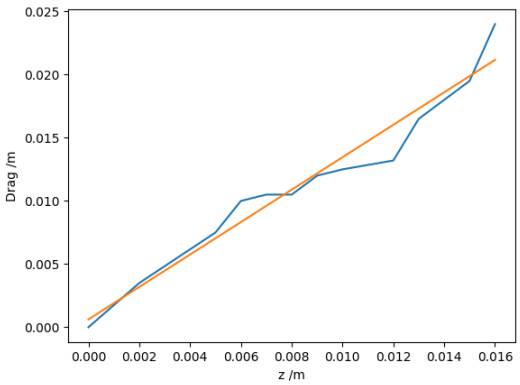


Figure 6: Actual measurement (blue) and linear fitting result (red) between drag and z

Parameters are updated gradually and smoothly with inertia. In the actual writing process, most strokes are written continuously and smoothly. Therefore, inertia is introduced to the virtual brush to create such an effect. For example, given two adjacent control commands (x_i, y_i, z_i) and $(x_{i+1}, y_{i+1}, z_{i+1})$, we update the width parameter by adding a ratio of the previous width to the newly calculated width:

$$w_{i+1} = w_i(k_{inertia}) + Width(\hat{z}_i)(1 - k_{inertia})$$

where w_i is the width parameter of the ith point in one stroke, and $Width(\hat{z}_i)$ calculates a new width given control command $\hat{z}_i$. It is observed that the position of the tip of the brush seldom changes shortly after the brush is pushed against the paper. In other words, the points at the beginning of the stroke have a bigger inertia ratio than those in the middle of one stroke, and so we design two inertia ratios to describe such an effect.

Updating the orientation parameter.

The orientation parameter is updated differently as it depends on the orientation of the brush at the end of a previous stroke. From the control commands, we know that the orientation of the brush will tend towards the opposite of the movement of the brush. However, at the end of a stroke, the brush tip will stay deformed and be oriented the same way until the next stroke begins. Therefore, the beginning orientation of a new stroke is the ending orientation of the previous. The orientation of the first stroke is determined by the direction of the trajectory coordinates x, y.

4.4 Similarities with the simple virtual brush

We represent the strokes in the same way as the simple virtual brush. The initial stroke trajectory generation is also the same.

4.5 Optimization for stroke trajectories using a dynamic virtual brush

Because the orientation parameter of one stroke is dependent on the orientation of the last stroke, we optimize each stroke trajectory in order. This is different in the simple virtual brush where we can optimize the strokes in parallel.

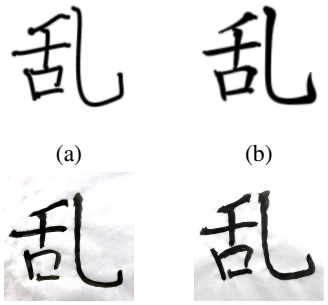
The new objective function is the same as the simple virtual brush except that it includes an orientation parameter.

$$\min_X \{\| V(X) - I_i \|; \alpha_{i-1}\}$$

where all the variables are the same as the cost function for the simple virtual brush, except we also include α_i which is the orientation parameter.

5 Results

The dynamic virtual brush model yields better written results than the simple virtual brush model, although the latter can produce very high-quality simulated images. Fig.



(a) (b)

(c) (d)

Figure 7: Character ‘messy’, pronounced ‘luan’ (a) Initial trajectory; (b) Trajectory after optimization with simple virtual brush; (c) Written result following simple brush optimization; (d) Written result following *dynamic* brush optimization

7 shows a comparison between the dynamic and the simple virtual brush model. In panel (b), the simulated image produced by the simple model shows a discernible character. However, because the deformation of the real brush is not modeled, the results on the real robot are of lower quality.

Hence, although minimizing cost between the simulation images and the original character images is used as the objective function, generating too small of a simulation error will lead to over-fitting, and this is true for both models. In other words, the capacity of the virtual brush model to model the actual brush sets the performance limit of the project, and using optimization methods to surpass the limit may lead to bad results. From our experience, an average pixel error between 8% and 16% of all pixels is usually enough to generate good written results.

Another possible cause of over-fitting is the degree of the polynomial parameterization used to represent the character strokes. Currently we choose a degree for every character once the simulation error of Eq. 3 falls into the range $[8\% - 16\%]$. But such a method does not promise to find the global optimum, but could provide a more practical and robust solution for the robot to execute. In the future, we can tune the degree of our Chebyshev polynomials to better fit each stroke.

In Fig. 8, we present the results of our approach, including photographs of characters that have been drawn by a Fetch robot in our lab. We show results for four different Chinese characters. Both the simulation and written images before and after optimization are shown for easy comparison.

6 Conclusion and Future Work

All in all, we have shown that it is possible to use pseudospectral methods to get a robot to write Chinese characters. Our results are good in that they show real physical writing, as opposed to just simulations.

From the figure we can see that the optimization achieves good performance for

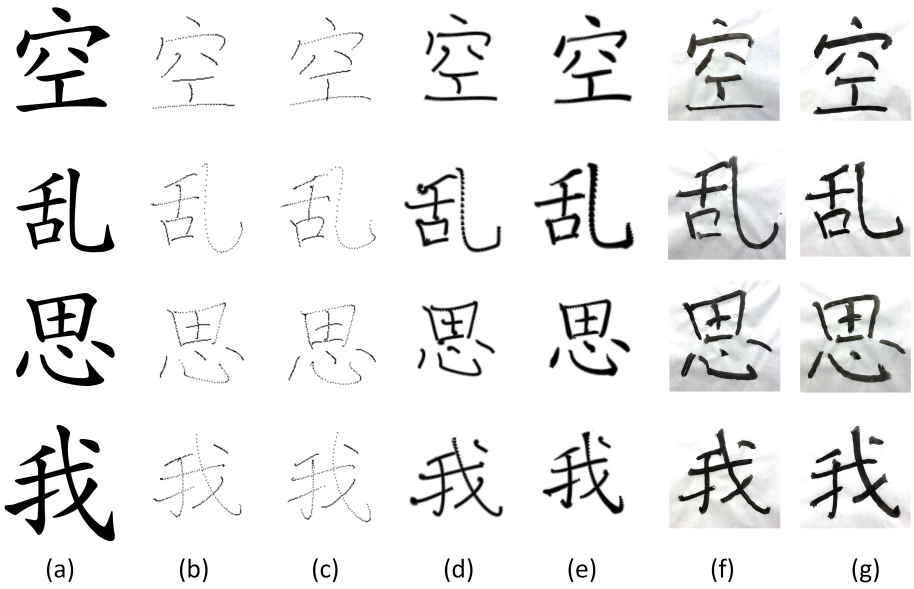


Figure 8: The optimization of different characters (From top down, ‘kong’, ‘luan’, ‘si’, ‘wo’ meaning ‘empty’, ‘messy’, ‘think’, ‘me’). (a) The original character pictures from the dataset; (b) Initial trajectory estimation; (c) The trajectory obtained from optimization; (d) Simulated image drawn by the virtual brush using the initial trajectory; (e) Simulated image drawn by the virtual brush using an *optimized* trajectory; (f) Written image following initial trajectories; (g) Written image following the *optimized* trajectories

simulated images. However, because of a “sim2real” gap in the virtual brush model, the proposed method still has a ways to go in terms of approaching the smoothness and definition of detail displayed in the reference images.

One immediate future improvement is the potential use of a different virtual brush, specifically one described in [10]. This brush could help us close the “sim2real” gap as it better captures the dynamics behind the brush by using a physics as an inspiration. Other improvements include using some sort of visual feedback system to allow the robot to self correct during the writing process. Another area for future work is in the actual mechanics and motion planning of the robot itself. Instead of giving the robot control points to execute, we could potentially give it joint angles in order to make the writing smoother. Lastly, we can also use style transfer to incorporate stylistic modifications to the calligraphy characters such that the robot can write with a more artistic style.

References

[1] Y. Sun, H. Qian, and Y. Xu, "Robot learns chinese calligraphy from demonstrations," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4408–4413, 2014.

[2] A. Kotani and S. Tellex, "Teaching robots to draw," *2019 International Conference on Robotics and Automation (ICRA)*, pp. 4797–4803, 2019.

[3] S. Mueller, N. Huebel, M. Waibel, and R. D'Andrea, "Robotic calligraphy — learning how to write single strokes of chinese and japanese characters," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1734–1739, 2013.

[4] K.-W. Kwok, S. M. Wong, K. W. Lo, and Y. Yam, "Genetic algorithm-based brush stroke generation for replication of chinese calligraphic character," *2006 IEEE International Conference on Evolutionary Computation*, pp. 1057–1064, 2006.

[5] J. H. M. Lam and Y. Yam, "Stroke trajectory generation experiment for a robotic chinese calligrapher using a geometric brush footprint model," *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2315–2320, 2009.

[6] K.-W. Kwok, K. W. Lo, S. M. Wong, and Y. Yam, "Evolutionary replication of calligraphic characters by a robot drawing platform using experimentally acquired brush footprint," *2006 IEEE International Conference on Automation Science and Engineering*, pp. 466–471, 2006.

[7] G. N. Elnagar and M. A. Kazemi, "Pseudospectral chebyshev optimal control of constrained nonlinear dynamical systems," *Computational Optimization and Applications*, vol. 11, pp. 195–217, 1998.

[8] F. Fahroo and I. Ross, "Direct trajectory optimization by a chebyshev pseudospectral method," *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, vol. 6, pp. 3860–3864 vol.6, 2000.

[9] S. Xu, F. C. M. Lau, and Y. Pan, "A computational approach to digital chinese painting and calligraphy," in *A Computational Approach to Digital Chinese Painting and Calligraphy*, 2009.

[10] N. S.-H. Chu and C.-L. Tai, "Real-time painting with an expressive virtual chinese brush," *IEEE Computer Graphics and Applications*, vol. 24, pp. 76–85, 2004.

[11] S. Kudoh and K. Ogawara, "Painter robot : Manipulation of paintbrush by force and visual feedback," in *Painter Robot : Manipulation of Paintbrush by Force and Visual Feedback*, 2007.

[12] Y. Lu, J. H. M. Lam, and Y. Yam, "Preliminary study on vision-based pen-and-ink drawing by a robotic manipulator," *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 578–583, 2009.

[13] L. Scalera, S. Seriani, A. Gasparetto, and P. Gallina, "Watercolour robotic painting: a novel automatic system for artistic rendering," *Journal of Intelligent and Robotic Systems*, pp. 1–16, 2018.

[14] X. Niu, J. Liu, L. Sun, Z. Liu, and X. Chen, "Robot 3d sculpturing based on extracted nurbs," *2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1936–1941, 2007.

[15] Y. Jun, G. Jang, B.-K. Cho, J. Trubatch, I. Kim, S.-D. Seo, and P. Y. Oh, "A humanoid doing an artistic work - graffiti on the wall," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1538–1543, 2016.

[16] Y. Sun and Y. Xu, "A calligraphy robot — callibot: Design, analysis and applications," *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 185–190, 2013.

[17] K. Sasaki, K. Noda, and T. Ogata, "Visual motor integration of robot's drawing behavior using recurrent neural network," *Robotics and Autonomous Systems*, vol. 86, pp. 184–195, 2016.

[18] F. Chao, J. Lv, D. Zhou, L. Yang, C.-M. Lin, C. Shang, and C. Zhou, "Generative adversarial nets in robotic chinese calligraphy," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1104–1110, 2018.

[19] R. Wu, W. Fang, F. Chao, X. Gao, C. Zhou, L. Yang, C.-M. Lin, and C. Shang, "Towards deep reinforcement learning based chinese calligraphy robot," *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 507–512, 2018.

[20] S. Xu, M. Tang, F. C.-M. Lau, and Y. Pan, "A solid model based virtual hairy brush," *Comput. Graph. Forum*, vol. 21, pp. 299–308, 2002.

[21] S. Strassmann, "Hairy brushes," in *SIGGRAPH*, 1986.

[22] H. T. F. Wong and H. H.-S. Ip, "Virtual brush: a model-based synthesis of chinese calligraphy," *Computers and Graphics*, vol. 24, pp. 99–113, 2000.

[23]

[24] C.-L. Liu, I.-J. Kim, and J. H. Kim, "Model-based stroke extraction and matching for handwritten chinese character recognition," *Pattern Recognition*, vol. 34, pp. 2339–2352, 2001.

[25] J. Zeng, W. Feng, L. Xie, and Z.-Q. Liu, "Cascade markov random fields for stroke extraction of chinese characters," *Inf. Sci.*, vol. 180, pp. 301–311, 2010.

[26] R. Cao and C. L. Tan, "A model of stroke extraction from chinese character images," *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, vol. 4, pp. 368–371 vol.4, 2000.

[27] C. Lee and B. Wu, "A chinese-character-stroke-extraction algorithm based on contour information," *Pattern Recognition*, vol. 31, pp. 651–663, 1998.

[28] Y. Sun, H. Qian, and Y. Xu, "A geometric approach to stroke extraction for the chinese calligraphy robot," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3207–3212, 2014.

[29] F. Chao, Y. Huang, C.-M. Lin, L. Yang, H. Hu, and C. Zhou, "Use of automatic chinese character decomposition and human gestures for chinese calligraphy robots," *IEEE Transactions on Human-Machine Systems*, vol. 49, pp. 47–58, 2019.

[30] P. T. Furgale, T. D. Barfoot, and G. Sibley, "Continuous-time batch estimation using temporal basis functions," *2012 IEEE International Conference on Robotics and Automation*, pp. 2088–2095, 2012.

[31] L. N. Trefethen., "Approximation theory and approximation practice," in *Approximation theory and approximation practice*, 2013.

[32] J.-P. Berrut and L. N. Trefethen, "Barycentric lagrange interpolation," *SIAM Review*, vol. 46, pp. 501–517, 2004.

[33] K. W. Lo, K.-W. Kwok, S. M. Wong, and Y. Yam, "Brush footprint acquisition and preliminary analysis for chinese calligraphy using a robot drawing platform," *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5183–5188, 2006.