**2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan**

Robotic calligraphy - Learning how to write single strokes of Chinese and Japanese characters

Samuel Mueller, Nico Huebel, Markus Waibel, and Raffaello D'Andrea

***Abstract*— A robot testbed for writing Chinese and Japanese calligraphy characters is presented. Single strokes of the calligraphy characters are represented in a database and initialized with a scanned reference image and a manually chosen initial drawing spline. A learning procedure uses visual feedback to analyze each new iteration of the drawn stroke and updates the drawing spline such that every subsequent drawn stroke becomes more similar to the reference image. The learning procedure can be performed either in simulation, using a simple brush model to create simulated images of the strokes, or with a real robot arm equipped with a calligraphy brush and a camera that captures images of the drawn strokes. Results from both simulations and experiments with the robot arm are presented.**

I. Introduction

Chinese and Japanese calligraphy (the art of beautifully writing with a brush) requires complex motions. Human calligraphy masters need years of practice. This makes it a challenging problem for robot learning. In addition, robotic calligraphy requires the trajectory for creating an accurate brush stroke to be determined beforehand because the soft tip of a calligraphy brush (which bends easily) does not allow for any force feedback and makes it difficult to accurately observe the drawing process with a camera.

In the past other robotic platforms for Chinese and Japanese calligraphy have been presented. In [1], [2] robots with three degrees of freedom (DOF) were used, in [3] the rotation around the z-axis (4 DOF) was additionally considered, and in [4] the pitch and roll (5 DOF) were also considered.

Most algorithms for reproducing Chinese characters fall into one of three groups. Algorithms from the first group extract the drawing trajectories from an image of a Chinese character [1], [5], [6]. Algorithms from the second group obtain a brush model and its parameters from experiments [7], [8] and then use these models to find the trajectories for drawing the Chinese character [7], [9]. The properties of brushes were studied in detail in [3]. Algorithms from the third group parametrize the strokes and tune the parameters manually [10].

Human calligraphers, on the other hand, learn and hone their skills over many years of training, during which they repeat the strokes over and over again. Little research has been done on improving the calligraphy skills of robots using experience from previous iterations. One notable exception is [11], where the researchers have used visual feedback to correct the xy-coordinates of the strokes, specifically the connection points of strokes.

The authors are with the Institute for Dynamic Systems and Control, ETH Zurich, Switzerland. The contact author is N. Huebel, e-mail: nhuebel@ethz.ch

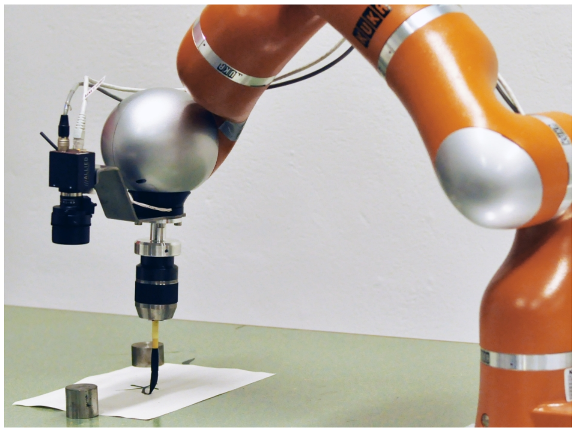


Fig. 1. The experimental setup consisting of a KUKA Light Weight Robot, a Prosilica GC 655C camera, and a brush.

Here we present a robotic drawing system that uses visual feedback from an attached camera and a novel iterative learning process to improve its drawing performance by using its experience from previous iterations. This process is also shown in the attached video. In section II the components for the experimental setup and their implementation will be explained, and some experimental results are shown and discussed in section III. Finally, section IV will summarize the results and give an outlook on the future direction of the project.

II. System Setup

A. Hardware Platform

Fig. 1 depicts the drawing setup, which consists of a KUKA Light-Weight Robot [12], a camera and a calligraphy brush. The robot is mounted onto a table, which operates as the drawing surface. The Prosilica GC655C gigabit ethernet camera has a resolution of 659x493 pixels. It is attached to a bent metal plate that is screwed between a collet chuck and the robot's tool flange. The collet chuck holds the calligraphy brush and provides a secure grip while allowing the brush to be mounted and exchanged quickly. Currently, the brush must be removed and dipped into the ink manually after each drawing iteration. The characters are drawn on standard A4 recycling paper that is secured to the surface of the table using two weights.

The robot is connected to a KUKA Robot Controller, which is responsible for low-level control. The KUKA Robot

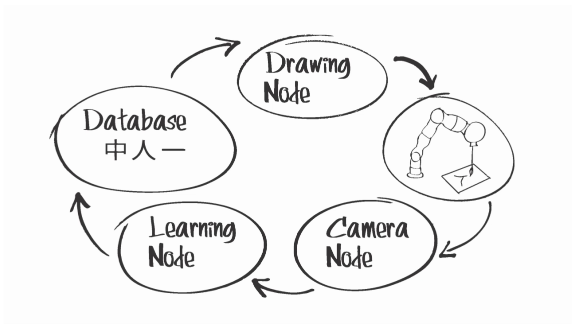


Fig. 2. Drawing procedure for the robotic calligraphy system.

Controller is connected to a remote computer via a Fast Research Interface [13]. All algorithms described hereafter run on this remote computer and use the Robot Operating System [14]. The algorithms that take care of the real-time interaction with the robot and the trajectory execution use the Orocos framework [15].

B. Drawing procedure

The drawing procedure is depicted in Fig. 2. The database contains a reference image and a description of the brush trajectory for each stroke. The brush trajectories are represented as splines. The drawing node takes these descriptions and transforms them into a trajectory that is then executed by the robot. Before drawing the stroke, the pose (position and orientation) of the paper on the table is detected automatically using the camera. Four dots representing the corners of the 12x12cm drawing area mark the paper's surface. By using the camera image as feedback, the camera is rotated and positioned directly above the center of the drawing area on the paper. The position of the drawing area is now calculated using the robot's inverse kinematics, the constraints for the paper to lie flat on the table, and the camera calibration. Each time a character is drawn, the robot positions the camera above the paper, and the camera node detects the drawing area, processes the image, and sends it to the learning node. Then the learning node calculates the error between the reference character and the drawn character and uses this error to update the trajectory information, which is then stored in the database for the next iteration.

C. Trajectory representation

B-splines were chosen to describe the drawing trajectories of the strokes, because they provide advantages such as guaranteed smoothness, free choice of interpolated data and local control. Including the speed as an attribute of the spline's control points enables the smooth interpolation of the drawing speed. However, the presented implementation keeps the speed constant.

$$C(t) = \sum_{i=0}^{n} Q_i N_{i,p}(t) \tag{1}$$

$$N_{i,j}(t) = \frac{t-t_i}{t_{i+j}-t_i} N_{i,j-1}(t) + \frac{t_{i+j+1}-t}{t_{i+j+1}-t_{i+1}} N_{i+1,j-1}(t)$$
$$N_{i,0}(t) = \begin{cases} 1 & \text{if } t_i \le t < t_{i+1} \text{ and } t_i < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Equation (1) is the general equation of a B-spline describing a curve $C(t)$ with the running parameter $t \in [0,1]$. The control points Q_i are weighted by a recursive basis function $N_{i,p}$ described in equation (2). B-splines have local control, i.e. a point on the curve represented by a B-spline of degree p is influenced only by the $p+1$ surrounding control points. This property is important for the learning procedure since it allows details of the stroke to be improved locally without affecting the rest of the stroke. The control points are not limited to geometric information, but can contain any number of entities that should be interpolated, such as position, orientation, velocity, or acceleration. In the implementation of the presented robotic drawing platform the control points contain Cartesian coordinates and the speed along the curve. The orientation of the brush is assumed to be strictly vertical at all times. Cubic, clamped, uniform B-splines were chosen to represent the strokes. Below is a brief characterization of the chosen spline representation. A general discussion of B-splines can be found in [16].

cubic:
: The degree $p = 3$ was chosen for the splines because cubic splines are sufficiently smooth and intuitive, and they keep the changes local.

clamped:
: A clamped B-spline starts and ends exactly at the first and last control point. This property facilitates the connection of multiple strokes when forming a character.

uniform:
: The distance between the knots in the spline's knot vector indicates the speed with which the running parameter t progresses along the spline. A uniform knot vector causes a uniform speed along the curve and simplifies the stroke description, as the knot vector is only determined by the number of control points. In the case of a non-uniform knot vector, all knot values must be part of the stroke description.

D. Drawing Node

Two different ways for generating trajectories are used in the system. Most motions are simple transition to a new pose (position and orientation). Time optimal trajectories are generated for these motions using the trajectory generator described in [17]. However, this method is not suitable for following a given trajectory. Therefore, the drawing trajectories are directly created from the B-spline that represents the given stroke according to the pseudocode shown in Pseudocode 1.

E. Camera Node

After a drawing cycle the camera is positioned at the previously determined location above the drawing area and

```
// Initialization
C = loadSpline(selected_stroke_ID);
t = 0;
step_size = 0.00001;
command_rate_of_robot = 0.02;
reached_stroke_end = false;
current_trajectory_point = C.evaluate(t); //
    Calculate point on spline at t
while (not reached_stroke_end) {
  distance_total = 0;
  // In our implementation the control points
      contain speed information
  distance_max = current_trajectory_point.
      desired_speed * command_rate_of_robot;
  while (distance_total < distance_max && t < 1)
      {
    t += step_size;
    t = min(u, 1.0); // Keep t in interval [0,1]
    next_point = C.evaluate(t);
    distance_total = EuclideanDistance(
        current_trajectory_point, next_point);
  }
  if (t >= 1) {
    reached_stroke_end = true;
    continue
  }
  next_point = TransformToRobotCoordinateFrame(
      next_point);
  robot_joint_angles = InverseKinematics(
      next_point); // 1Calculate joint angles
      from Cartesian pose
  trajectory.push_back(robot_joint_angles);
  current_trajectory_point = next_point;
}
return trajectory
```

Pseudocode 1. Pseudocode for generating the drawing trajectory from the stroke description.

an image is acquired. It is converted to grayscale, rectified using the camera calibration, and filtered by an adaptive thresholding algorithm. The adaptive threshold technique eliminates blurriness induced by the fixed-focus camera and copes with the shadows cast by the robot as well as different lighting conditions (see Fig. 3(a)). The four markings at the corners of the drawing area are detected and its content is rectified using a perspective transformation. The result is saved at a normalized dimension of 300x300 pixels, resulting in a resolution of 2.5 px/mm (see Fig. 3(b)). The resolution is currently limited by the hardware setup, as the fixed-focus camera has a resolution of 659x493 pixels and it cannot be positioned closer to the drawing surface due to the mounted brush.

F. Learning Node

The Learning Node contains the following iterative learning procedure:

1) Generate error data from the difference between the reference image and the image of the current drawing.
2) Fit an error spline to the error data.
3) Update the stroke description of the next drawing spline based on the previous drawing spline and the error spline.

To generate the error data, the reference image is overlayed with the image of the drawing from the current iteration and each pixel is classified according to one of the following categories: background, reference-only, overlapping and current-iteration-only. This classification is visualized in Fig. 5(a). The drawing spline is evaluated and its x-y position is projected onto the image (dotted blue line in Fig. 5(b)). The pixels are analyzed along the projected trajectory's normal line at each point $\mathbf{p}_k$. The pixels of the normal line are determined by Bresenham's line algorithm [18]. For each pixel m on the normal line, the vector $\mathbf{v}_m$ pointing to it from the point $\mathbf{p}_k$ on the projected trajectory is weighted depending on the category of the pixel. Background and overlapping types have a weight of $w_m = 0$, reference-only pixels have a weight of $w_m = 1$, and current-iteration-only regions have a weight of $w_m = -1$. The resulting error point on the xy-plane $\mathbf{P}_{k,xy}$ that corresponds to the point $\mathbf{p}_k$ on the projected trajectory is the calculated as follows:

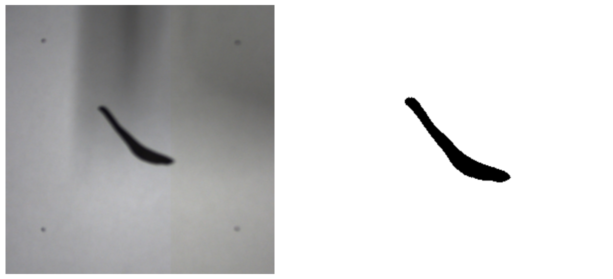
$$\mathbf{P}_{k,xy} = \mathbf{p}_k + \sum_{m \in \text{normal line}} w_m \cdot \mathbf{v_m}.$$

This causes reference-only pixels to have an attractive behavior pulling the trajectory towards them because in the current iteration they have not been covered with ink, whereas current-iteration-only pixels have a repellent behavior because they have been covered by the ink but should not have been covered.

To adjust the z-coordinate, the width d_{ref} of the stroke in the reference image and the width d_i of the current iteration i is measured along the normal line at each point $\mathbf{p}_k$ and its difference is used to correct the z-coordinate of the brush:

$$z_{k,i+1} = z_{k,i} + c(d_i - d_{ref}),$$

with c being a constant that represents the relation between a change in the z-coordinate and the corresponding change in the thickness of the stroke.

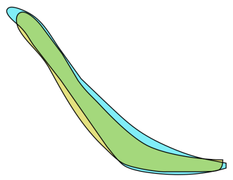


(a) A raw camera image, including the dots that mark the drawing area.

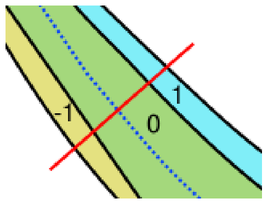
(b) Normalized 300x300 pixel image stored in the character database.

Fig. 3. Image acquisition

Fig. 4. The learning algorithm adapts the drawing trajectory iteratively to the bending of the calligraphy brush.



(a) Overlay of reference stroke and current iteration. The four categories of the pixel classification are colored accordingly: background (white); reference stroke only (blue); current iteration only (yellow); overlapping area (green).



(b) To generate error data, a normal line (red) perpendicular to the trajectory (blue dotted line) is analyzed. The vectors from each point on the trajectory to the pixels on its associated normal line are weighted by the indicated numbers, resulting in a repellent behavior of the yellow area while making the blue area attractive to the error data.

Fig. 5. Generating error data.

Next, data points for fitting the error spline are created by combining this information $\mathbf{P_k} = [\mathbf{P}_{k,xy}^T, z_{k,i+1}]^T$.

Because the algorithm tends to contract the spline, an addition was made to expand the trajectory: After the spline is completely evaluated, its beginning and end are tangentially extended and the same strategy is applied to create the error data points for the tangents. This ensures that the spline can grow along the reference, as can be seen in Fig. 7.

Next, an error spline is fitted through the $(m+1)$ data points in the error point cloud $\hat{P}$. Let $\hat{Q}$ be a column vector that contains $(n+1)$ control points.

$$\hat{Q} = \begin{bmatrix} \mathbf{Q_0} \\ \vdots \\ \mathbf{Q_n} \end{bmatrix}, \quad \hat{P} = \begin{bmatrix} \mathbf{P_0} \\ \vdots \\ \mathbf{P_m} \end{bmatrix}$$

The control points in $\hat{Q}$ should be determined such that the sum of the squared errors $E(\hat{Q})$ is minimal. The least squares problem is formulated as follows:

$$E(\hat{Q}) = \frac{1}{2} \sum_{k=0}^{m} \left| \sum_{j=0}^{n} N_{j,p}(t_k)\mathbf{Q_j} - \mathbf{P_k} \right|^2 .$$

The term $\sum_{j=0}^{n} N_{j,p}(t_k)\mathbf{Q_j}$ is a point on the yet unknown B-spline at parameter position t_k. $N_{j,p}(t_k)$ denotes the basis function of the B-spline as defined in equation (2).

The minimum of the error is found by taking the derivative with respect to the control points $\mathbf{Q_i}$ and setting it to zero:

$$\frac{\partial E}{\partial \mathbf{Q_i}} \overset{!}{=} 0 = \sum_{k=0}^{m} \left(\sum_{j=0}^{n} N_{j,p}(t_k)\mathbf{Q_j} - \mathbf{P_k} \right) N_{i,p}(t_k)$$
$$= \sum_{k=0}^{m} \sum_{j=0}^{n} N_{i,p}(t_k) N_{j,p}(t_k)\mathbf{Q_j} - \sum_{k=0}^{m} N_{i,p}(t_k)\mathbf{P_k}.$$

Using the matrix

$$A = \begin{bmatrix} N_{0,p}(t_0) & N_{1,p}(t_0) & \dots & N_{n,p}(t_0) \\ N_{0,p}(t_1) & N_{1,p}(t_1) & \dots & N_{n,p}(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ N_{0,p}(t_m) & N_{1,p}(t_m) & \dots & N_{n,p}(t_m) \end{bmatrix}$$

this can be written as

$$A^T A\hat{Q} - A^T \hat{P} = 0$$

and the control points can be calculated by solving the matrix equation

$$\hat{Q} = \left(A^T A\right)^{-1} A^T \hat{P}.$$

The control points C_{i+1} of the trajectory for the next iteration $i+1$ are obtained by linearly combining the control points from the error spline with the ones from the previous trajectory:

$$C_{i+1} = k_p \cdot \hat{Q} + (1 - k_p) \cdot C_i,$$

where $k_p \in [0, 1]$ is the learning rate.

III. Experiments

A. Learning the initial stroke in simulation

It takes roughly two minutes to complete one drawing iteration, which consists of placing and localizing the paper, drawing, letting the ink dry, acquiring and processing the image. A simulation was developed to speed up this process, where the trajectory is learned in simulation until convergence before the first iteration is executed on the robot. The required number of iterations depends on the learning rate k_p, the number of control points, and the initial spline. After simulating various combinations the results presented in this section use a learning rate $k_p = 0.2$ and twelve control points because these parameters achieved a good error score after few iterations.

The simulation uses the simplest possible brush model: a horizontal cross section of a cone, i.e. a circle with a radius proportional to the brush's negative z-coordinate. This approach yields a good initial approximation of the trajectory in simulation, which can then be adapted to the real test bed in only a few iterations.

Fig. 6 depicts the decrease in the global error over several iterations in simulation, which is shown in Fig. 7. The error does not converge to zero because the brush model is too simple and the number of control points is too low to accurately represent the reference stroke.

B. Robotic calligraphy

Fig. 8 shows the results when executing the trajectory that was learned in simulation (Fig. 8(b)) on the robot using a real brush (Fig. 8(c)). The real drawing is then improved using the same learning process (Fig. 8(d)).

C. Drawing repeatability

The brush itself introduces a surprisingly high amount of noise into the drawing. It is reasonable to expect that subsequent drawings made immediately and without removing the brush and dipping it into the ink would be nearly identical. However that is not the case, as can be observed in Fig. 9.

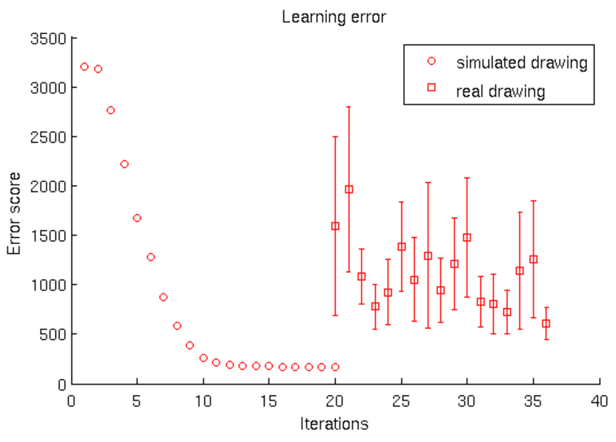


Fig. 6. Number of pixels that were not classified as background or overlapping when learning the 'na' stroke shown in Fig. 7. The iterations in simulation converge to an optimum that depends on the number of used control points. The learned trajectory (iteration 20) is then executed on the robot (iteration 20 to 36).

IV. Conclusions and Future Work

This paper presented a robotic calligraphy system that was used to draw single strokes of Chinese and Japanese calligraphy characters. The main contribution of this paper is the described learning procedure that iteratively improves the drawing quality by using visual feedback after each iteration. B-splines were chosen to describe the drawing trajectories of the strokes, because they provide advantages such as guaranteed smoothness, free choice of interpolated data and local control.

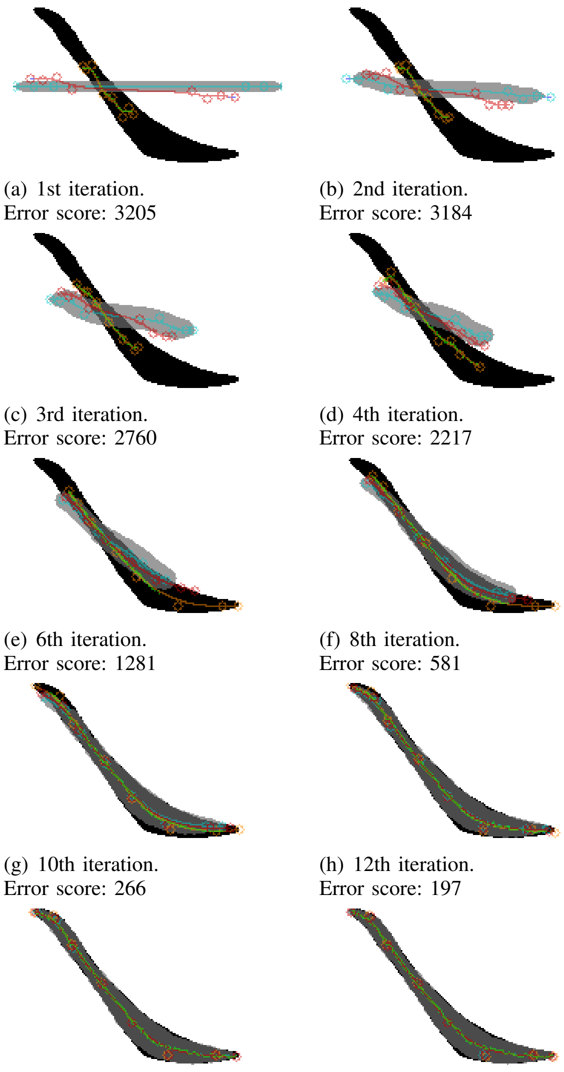
With the current implementation convergence is not guaranteed (see Fig. 10) and depends on the initial condition, such as the position or shape of the initial spline. This could be solved by upgrading the system with image processing capable of generating an initial spline that is already close to the shape of the reference as presented in e.g. [5].

Some strokes in Chinese and Japanese calligraphy have very complex motions at their beginnings and ends. The drawing result is approximated by our system, but the learned trajectories are very different from the ones used by humans.

Several extensions to the presented procedure are planned. One of these is to implement an automated procedure for dipping the ink that also standardizes the initial shape of the brush. Another is to implement an automated procedure that determines the optimal number of control points necessary to represent a stroke and to investigate other strategies for adapting the drawing spline, such as using an ellipse instead of the normal line. These strategies may eliminate the need to extend the trajectory's beginning and end to expand the spline and may address the problems shown in Fig. 10. Another more ambitious plan is to automatically combine the learned strokes into characters.

Acknowledgments

This research was funded by the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no. 248942 RoboEarth.



(a) 1st iteration. Error score: 3205

(b) 2nd iteration. Error score: 3184

(c) 3rd iteration. Error score: 2760

(d) 4th iteration. Error score: 2217

(e) 6th iteration. Error score: 1281

(f) 8th iteration. Error score: 581

(g) 10th iteration. Error score: 266

(h) 12th iteration. Error score: 197

(i) 14th iteration. Error score: 181

(j) 16th iteration. Error score: 172

Fig. 7. Learning series of the slanted down stroke 'na'. Black: reference shape; gray: simulated drawing of current iteration; green dots: generated error data; light blue line: trajectory of current iteration; orange line: B-spline fitted to error data; red line: trajectory of next iteration; circles: control points of B-splines.

References

[1] F. Yao, G. Shao, and J. Yi, "Extracting the trajectory of writing brush in chinese character calligraphy," *Engineering Applications of Artificial Intelligence*, vol. 17, no. 6, pp. 631–644, 2004.

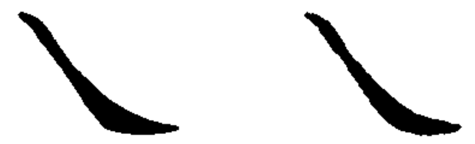
[2] Y. Man, C. Bian, H. Zhao, C. Xu, and S. Ren, "A kind of calligraphy robot," in *Information Sciences and Interaction Sciences (ICIS), 2010 3rd International Conference on*, 2010, pp. 635–638.

[3] 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," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2006, pp. 5183–5188.

[4] D. Wang, Y. Zhang, and C. Yao, "Stroke-based modeling and haptic skill display for chinese calligraphy simulation system," *Virtual Reality*, vol. 9, no. 2-3, pp. 118–132, 2006.

[5] J. Lam and Y. Yam, "A skeletonization technique based on delaunay triangulation and piecewise bezier interpolation," in *Information, Communications Signal Processing, 2007 6th International Conference on*, 2007, pp. 1–5.

[6] F. Yao and G. Shao, "Modeling of ancient-style chinese character and its application to CCC robot," in *Networking, Sensing and Control,*


(a) Reference image. (b) 17th simulation iteration. Error score: 170

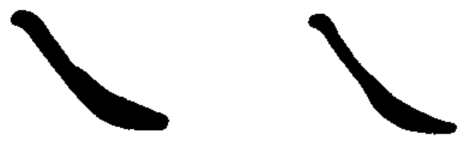
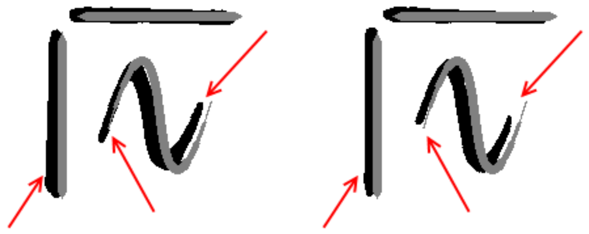

(c) 1st drawing iteration using the trajectory generated by the 20th simulation iteration. Error score: 1794 (d) 4th drawing iteration. Error score: 872

Fig. 8. The learning of a stroke by simulation, showing the refinement and adaptation of the robot to the drawing dynamics of the brush.


(a) (b)

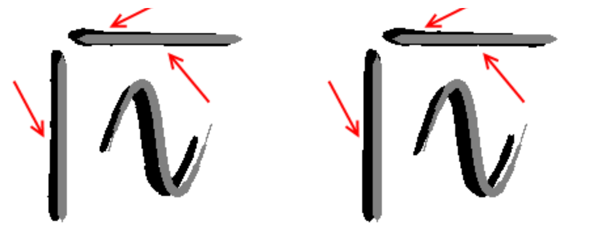

(c) (d)

Fig. 9. Repeated drawing of the same test figure without removing the brush and without dipping it into the ink. The drawings differ without this external disturbance. The first drawing of the series has been discarded because the brush did not have an initial bending. This bending is the reason why all four drawings (black) are slightly left of the reference shapes (gray). In (a) and (b) the difference in the thickness of the drawing of the left stroke and the start and end point of the wave shape are indicated. In (c) and (d) differences in thickness and position of the drawings are indicated.

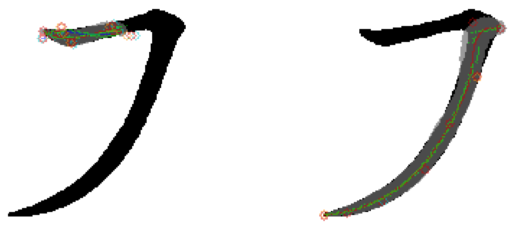

(a) Both ends of the spline converge to the same end of the stroke. (b) Convergence into a local optimum.

Fig. 10. The procedure is sensitive to the initial conditions. Both examples started with a horizontal line at different positions as initial spline.

2006. ICNSC '06. Proceedings of the 2006 IEEE International Conference on, 2006, pp. 72–77.

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

[8] H. Leung, S. Wong, and H.-S. Ip, "In the name of art," *Signal Processing Magazine, IEEE*, vol. 25, no. 4, pp. 49–54, 2008.

[9] 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," in *Automation Science and Engineering, 2006. CASE '06. IEEE International Conference on*, 2006, pp. 466–471.

[10] F. Yao, G. Shao, and J. Yi, "Trajectory generation of the writingbrush for a robot arm to inherit blockstyle chinese character calligraphy techniques," *Advanced Robotics*, vol. 18, no. 3, pp. 331–356, 2004.

[11] K. W. Kwok, Y. Yam, and K. W. Lo, "Ga-based homography transformation for vision rectification in robot drawing system," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, 2005, pp. 2047–2052.

[12] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schaeffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald, and G. Hirzinger, "The KUKA-DLR Lightweight Robot arm - a new reference platform for robotics research and manufacturing," in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, 2010, pp. 1–8.

[13] G. Schreiber, A. Stemmer, and R. Bischoff, "The fast research interface for the KUKA lightweight robot," in *IEEE Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications - How to Modify and Enhance Commercial Controllers (ICRA 2010)*, 2010.

[14] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, 2009.

[15] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the Orocos project," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 2, 2003, pp. 2766–2771.

[16] G. E. Farin, *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Code*, 4th ed. Orlando, FL, USA: Academic Press, Inc., 1996.

[17] Francisco Ramos, Mohanarajah Gajamohan, Nico Huebel, and Raffaello D'Andrea, "Time-optimal online trajectory generator for robotic manipulators," ETH Zurich, Zurich, Tech. Rep., 2013.

[18] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965.