Multi-Directional Drawing using TM5-900 Manipulator

NTU 2021 Robotics Fall Semester

Chai-Cheng, Hsu
Electrical Engineering, NTU
B06502151
Taipei, Taiwan
hsudani@gmail.com

Kevin, Liao
Electrical Engineering NTU
A10921104
Taipei, Taiwan
kevin.liao@hotmail.de

Chun-Ting, Lee
Electrical Engineering NTU
B07901141
Taipei, Taiwan
lct841010tim@gmail.com

Pei-An, Hsieh
Electrical Engineering, NTU
B07901061
Taipei, Taiwan
peian8976@gmail.com

***Abstract*—This document has been written for the NTU Robotics course of the Fall 2021 Semester. In this report the challenges, solutions and implementations of our final project will be discussed. This lecture has been held by Prof. Li-Chen Fu and the TAs Vincent, Peggy, Tommy, Leo and Yu-Hsuan.**

***Index Terms*—TM5-900, ROS, Forward Kinematics, Trajectory planning, Pose estimation.**

I. Introduction

A. Related Work

The main idea of our final project is, to use an industrial arm - namely the provided TM5 900 manipulator - to perform a drawing on an arbitrary plane in a 3D space. This project has been chosen to combine several challenges in computer vision, trajectory planning and trajectory transformation. With all of these key components being quite important and relevant for real world applications.

Several similar drawing robots have been developed. Most notably the portrait drawing robot by Frederic Fol Laymarie et. al [1], which tries to mimic the drawing motion of an actual artist and has been shown in numerous art exhibitions. Other relevant works include [2], [3] these papers describe several drawing robots that create drawings based on normal input images. As basis of our image processing we choose a contour based approach, that extracts the contours of an image in the first step and then generates a trajectory based on the processed image.

Additionally, the aforementioned works only draw on fixed, well defined planes, which not only eliminates the need for an automated calibration method, but also simplifies the generation of the drawing trajectory. This impressive publication proposes a drawing robot on an arbitrary surface [4], which can even draw on non planar objects. We, however, limit our selves on an arbitrary plane.

B. Project Objective Formulation

Our projects main objective is to at least draw on an horizontal plane, with the additional objective of drawing on arbitrarily orientated planes. The objectives are listed here:

- Drawing on a horizontal plane (contours) [1]
- Drawing on arbitrary planes (contours) [2]
- Automatic, computer vision-based calibration

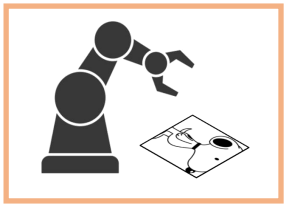

Fig. 1. Horizontal plane printing

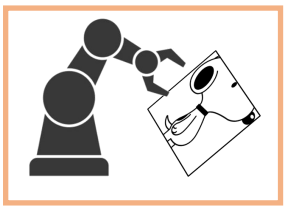

Fig. 2. Arbitrary plane printing

C. Proposed Workflow Diagram

This figure outlines our proposed workflow for the project (see fig. 3). From this workflow these main components of our project can be derived:

1) Image processing (contour generation + trajectory generation)
2) Camera Pose Estimation
3) Trajectory Transformation
4) Drawing of image

In the following chapters the implementation of each of these components will be described in detail.

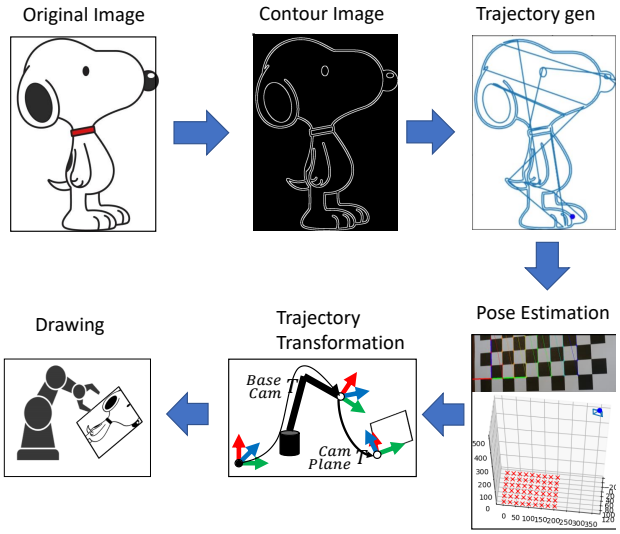

Fig. 3. Workflow for our implementation of multi-directional printing.

II. Hardware

Our project is based on the TM5-900 robot. The TM5-900 has been provided by the NTU and is a 6DOF industrial manipulator with high precision ($\pm 0.05mm$) and integrates a camera which will be used in the pose estimation part of our project (see fig. 4). Additionally, we build our own tool

Fig. 4. Image of the TM5 900 robot. [5]

head (see fig. 5). We created a spring loaded tool head in order to increase the robustness of the drawing procedure. In figure 6 a schematic drawing of the tool head and its main parts are shown. Furthermore a tool adapter (see fig. 7) and an adjustable table for our drawing plane has been build (see fig. 8). The tool adapter was designed by a 3D drawing software, Inventor, and was produced by plastic using a 3D Printer.

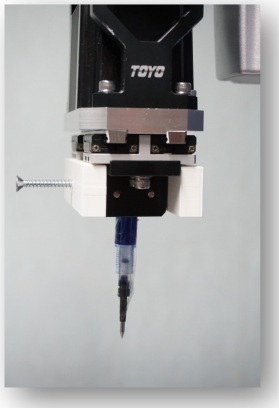

Fig. 5. Custom designed tool head

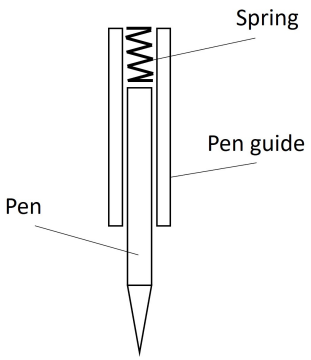

Fig. 6. Schematic drawing of the tool head

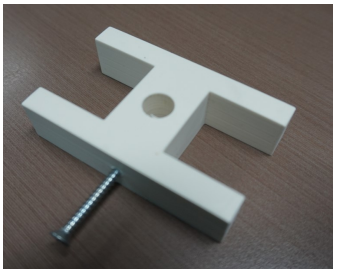

Fig. 7. Adapter between gripper and tool.

Fig. 8. Table with adjustable plane.

III. Image Processing

A. Obtaining Image Contours

In order to obtain the contours of arbitrary images, we first transform color images to binary images. Then, we apply Canny Edge Detection algorithm to detect the contours. This algorithm can be easily implemented using a function in OpenCV called Canny. We can observe that after this process, the original image (see figure 9) has been reduced to contours with one bit width (see figure 10).

B. Trajectory Planning for Robot Arm

Trajectory planning of this drawing task is accomplished based on an algorithm designed by ourselves. It requires the following steps:

1) Pixel Classification: First we classify pixels into two major point sets, start/end points and branch points. The start/end points are those pixels which have only one neighbor pixel that needs to be drawn. On the other hand, branch points are those pixels which have more than one neighbor pixel that needs to be drawn. We define neighbor pixels as the 8 pixels surrounding the pixel under classification. This classification criterion is shown in figures 11 and 12.

Fig. 9. Original image

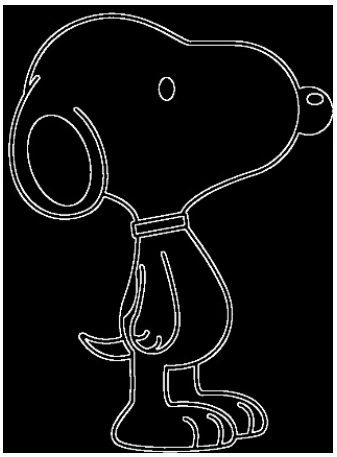

Fig. 10. Processed image

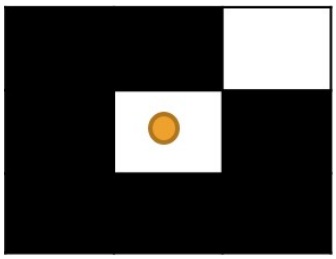

Fig. 11. Start/end points

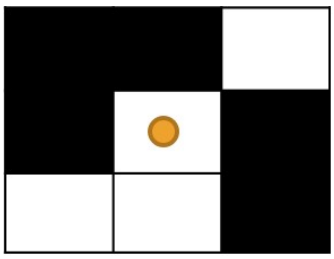

Fig. 12. Branch points

2) Connecting the Pixels: To form trajectories, we choose a point in the set of start/end points as the starting point. The main task is to search through its neighbor points for points that needs to be drawn and add them to the current trajectory. During this process, we need to assign another two attribute: “drawn” and ”visited” to the points. When the searching process goes through a point, we mark the point visited. We also mark it and its neighbors as drawn. The ”visited” attribute prevents the searching process to form loops in the trajectory. Hence we refresh the ”visited” attribute every time we start searching for a new trajectory. For the ”drawn” attribute, the reason why we also need to mark the neighbors as drawn is to avoid drawing similar trajectories multiple times. This scenario may happen under the circumstances similar to figure 13. The two red trajectories are both legit trajectories if we simply search through neighbors. However, the two trajectories are nearly the same when we draw them out in the real world. Therefore, there is no need to draw it twice considering the efficiency of the drawing process. Notice that the drawn attribute updates only when a trajectory is found, that is, it reflects on the searching process of the next trajectory.

With these two attributes, we can ameliorate the searching process as the following: We only search through the neighbor points that needs to be drawn and are yet visited. When we encounter a point which is a branch point and has not been drawn, add it to the current trajectory. If no neighbor points satisfy the first condition and we encounter a point which is a start/end point or a branch point which is drawn, we end the current trajectory. Under this process, trajectory 2 is not a legit trajectory if we have already found trajectory 1 (see figure 13).

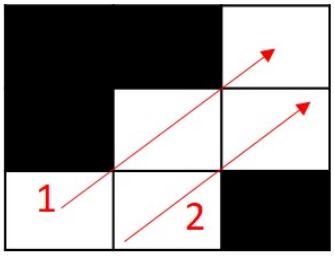

Fig. 13. An example of similar trajectories drawn.

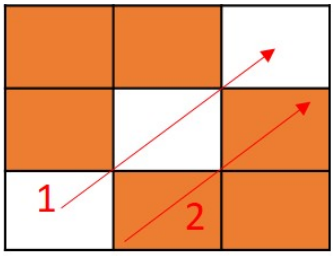

Fig. 14. An example of how using drawn attribute improves the drawing process. The orange tiles mark the drawn points.

3) Iterating Through the Whole Image: We now know how to form a trajectory, and the next step is to iterate this process until all the pixels that needs to be drawn are all drawn. In the step above, we chose a point in the set of start/end points as the starting point. It is reasonable for us to first iterate through the start/end set of points until all points in this set are drawn. However, the existence of closed curves in the image implies that only iterating through the set of start/end points as starting points will still leave out some points. This is because all points in this closed curve have more than one neighbor point that requires drawing and are all branch points. Hence we should set branch points that are not drawn as starting points and iterate through the set of branch points until all branch points are drawn. Finally, we will get the whole image drawn.

IV. Visual Pose Estimation

In order for the robot to calibrate on its own, some kind of pose estimation relative to the plane must be found. For that we decided to use a 2D-to-3D pose estimation algorithm, also known as Perspective-n-points problem. In figure 15 the wanted pose transformation is illustrated.

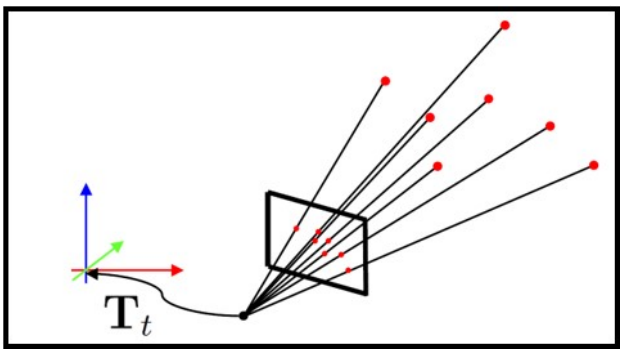

Fig. 15. Transformation from plane origin to camera coordinate.

A numeric solver is already implemented in OpenCV called solvePnP. We simple provide this function with corresponding

3D and 2D points and are able estimate our required transformation ${}^{Plane}_{Cam}T$. Since we can simply choose a known pattern, the 3D points can be trivially calculated. However, the method on how to derive our 3D points, defines the origin of our plane. The 2D points on the other hand have to be detected using a corner detection algorithm.

To verify the correctness of the estimated transformations following experiments have been conducted. As seen in figure 16 we fed the algorithm with three different input images of different perspectives. When compared to our estimated poses, we get plausible and expected results.

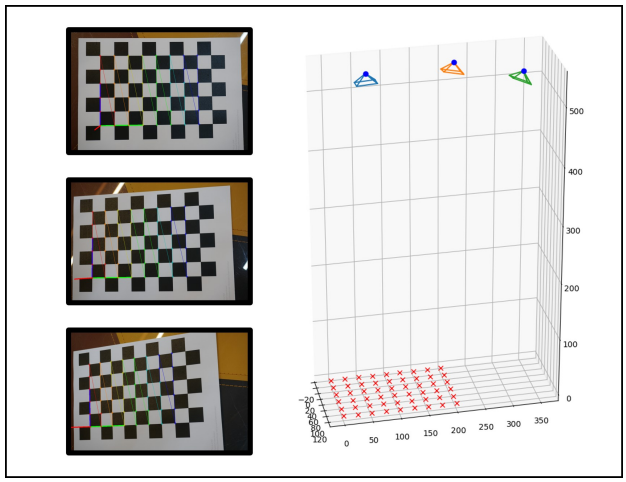

Fig. 16. (l) Three input images with obvious perspective changes. (r) Estimated poses.

V. Trajectory Transformation

A. Base-Gripper Transformation

When operating the TM5-900, we send a pose $(x, y, z, \alpha, \beta, \gamma)$, consisting of Cartesian position and Euler angles in the robot base coordinate, to control the gripper pose. The transformation matrix ${}^{Base}_{Grip}T$ between the gripper and the base can thus be obtained. The original pose of the gripper is set as (400, 300, 700, -180, 0 135), therefore

$$
\begin{aligned}
{}^{Base}_{Grip}R &= R_x R_y R_z \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -\sqrt{\frac{1}{2}} & -\sqrt{\frac{1}{2}} & 0 \\ \sqrt{\frac{1}{2}} & -\sqrt{\frac{1}{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} -\sqrt{\frac{1}{2}} & -\sqrt{\frac{1}{2}} & 0 \\ -\sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 0 \\ 0 & 0 & -1 \end{bmatrix}.
\end{aligned} \tag{1}
$$

$$
\begin{aligned}
{}^{Base}_{Grip}T &= \begin{bmatrix} & & & x \\ & {}^{Base}_{Grip}R & & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} -\sqrt{\frac{1}{2}} & -\sqrt{\frac{1}{2}} & 0 & 400 \\ -\sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & 0 & 300 \\ 0 & 0 & -1 & 700 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{2}
$$

B. Gripper-Camera Transformation

Now we know the transformation between the gripper and the base. The transformation between the robot and the plane is obtained, however, by taking a picture of the plane, which is the transformation between the camera and the plane. The transformation matrix ${}^{Grip}_{Cam}T$ can be obtained considering the position offset $(0, 80, 0, 0, 0, 0)$ between the gripper and the camera.

$$
{}^{Grip}_{Cam}T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 80 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}
$$

C. Camera-Plane Transformation

In section IV, the transformation ${}^{Cam}_{Plane}T$ is obtained by taking picture of a given pattern on the plane.

$$
{}^{Cam}_{Plane}T = \begin{bmatrix} & {}^{Cam}_{Plane}R & & {}^{Cam}_{Plane}Q \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}
$$

D. Plane-Base Transformation

The transformation between the robot base and the plane is obtained as

$$
{}^{Base}_{Plane}T = {}^{Base}_{Grip}T\,{}^{Grip}_{Cam}T\,{}^{Cam}_{Plane}T \tag{5}
$$

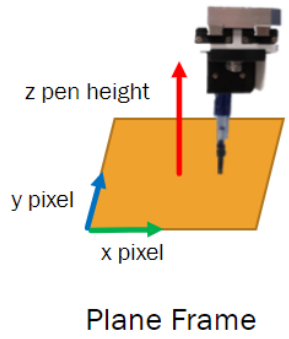

Fig. 17. The drawn picture and the pen in the plane frame

Fig. 18. Snoopy drawn by the robot

E. Trajectory Transformation

In the plane frame, the xy pixel of the picture correspond to the xy axis of the frame. To make it simple, the robot draws perpendicularly to the plane, which means the pen is always along the z axis. That is, the xyz axis correspond to the xy pixel of the picture and the pen height respectively. Applying equation 5 to the position in the plane frame, the gripper position in the base frame is obtained as

$$^{Base}P = {}^{Base}_{Plane}T\,{}^{Plane}P \tag{6}$$

The gripper pose, always perpendicular to the plane, can be obtained by transforming the rotational matrix ${}^{Base}_{Plane}R$ inside ${}^{Base}_{Plane}T$ to Euler angles.

$$\begin{aligned} {}^{Base}_{Plane}R &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \\ \alpha &= atan2(r_{32}, r_{33}) \\ \beta &= atan2(-r_{31}, \sqrt{{r_{32}}^2 + {r_{33}}^2}) \\ \gamma &= atan2(r_{21}, r_{11}) \end{aligned} \tag{7}$$

VI. Experiment Setup

A. Software

The image processing and trajectory planning code is written using Python 3 in trajectory.py. It requires packages numpy, cv2, matplotlib, and copy. We saved the generated trajectories to a text file so that the python scripts of the ROS nodes could read it. To control the TM5-900 robot arm, we ran the RosNode in TM-Flow on the computer of the robot arm and communicated with it running ROS2 on another computer. We modified image_sub.py and sendscript.py from assignment 4 to calculate the transformation from camera to plane and from pixel to camera respectively. Notice that between two trajectories, we move the robot arm up 1 cm to avoid drawing unwanted curves while the robot moves to the next starting point. This is also written in sendscript.py.

B. Hardware

Set our adjustable plane to an arbitrary angle and make the gripper grab the drawing tool. Tape the drawing paper on the plane, and place the given pattern (e.g. checkerboard) that is used to calculate the transformations on it. After the robot arm takes a picture for transformation calculation and starts moving toward the drawing plane, move the pattern away.

VII. Results

The way we accomplished the robot arm drawing task is by following these steps:

1 Pick an image, and generate its trajectory by running trajectory.py.
2 Place the given pattern and drawing paper on an arbitrary plane to be drawn on.
3 Start the robot and follow the steps of assignment 4 to run image_sub.py and sendscript.py. It will first take a picture of the pattern, and obtain the transformations.
4 Then the robot will move toward the plane and start drawing.
5 Wait until the robot arm finishes drawing.
6 Move the robot arm up by running sendscript.py again.

The result of drawing the sample snoopy image is in Fig.18. This snoopy image is drawn on a plane with an arbitrary plane angle. We can observe that the robot drew the image with high precision and covered all the details. The whole drawing process took about 10 minutes and used 21 trajectories. Still, there are some distortions in the drawing which is due to the fact that the tool we designed wobbles slightly in the horizontal direction during the drawing process. This means that the point of the drawing pen sometimes lags behind the gripper. When the robot is not only drawing a straight line, the lag causes distortions.

VIII. Future Works

Since it is just a prototype, there are plenty of room for improvement. The four main aspects that we can improve in the future are listed below:

A. Trajectory Planning

The trajectory planning algorithm we used is just an algorithm efficient enough for us to prove our concept. To enhance efficiency, one can start from searching for starting points nearer to the ending point of the last trajectory. On the other hand, adding velocity information to each step will enable the drawing robot to complete more challenging drawing tasks, e.g., calligraphy and watercolor.

B. Pose Estimation

Although we can get the correct desired pose, the robot arm may collide with the drawing plane while moving to the drawing position. This means that environmental information and collision avoidance mechanism is needed when the robot detects the drawing plane and decides whether it is possible to draw on the plane at its current position.

C. Drawing Tool

The most crucial issue for us to improve the quality of our drawing is the stiffness of our drawing tool. The wobbling in the horizontal direction should be fixed to increase precision.

D. Multi-color & Multi-tools

In this case, we only use one color to paint our picture. In the future, we can design a table placed in a certain place to contain other pens that have different color. The robot arm can reach out to the table to change colors and paint the picture more vividly. Furthermore, we can try to use different tools, such as paint brush or calligraphy brush to try different style of painting.

IX. Conclusion

Most drawing robots draw on a horizontal plane. With a six degree of freedom robot arm, we can take advantage of its dexterity to perform drawing on arbitrary angled planes. This is an area that has not been thoroughly researched, and we designed a prototype for accomplishing this task. The robot arm executed the task successfully and provided a proof of concept. Using this experiment as a foundation, one can develop robot arms and control methods for more ambitious drawing tasks in the future.

X. Work Distribution

Work Distribution Chart

Hsu	Liao	Hsieh	Lee
25%	25%	25%	25%
3D-printed adapter/ Drawing Tool/ Trajectory Transfor-mation	Camera Pose Estimation/ Drawing Tool	Image Processing/ Trajectory Planning	Camera Pose Estimation/ Trajectory Transfor-mation

XI. Resources

Source code and demo video links are listed below:
This is the GitHub link for the python codes.
This is the link for the demo video: Team8_Multi-Directional Drawing.mp4.

References

[1] Tresset, Patrick, and Frederic Fol Leymarie. "Portrait drawing by Paul the robot." Computers & Graphics 37.5 (2013): 348-363.

[2] Calinon, Sylvain, Julien Epiney, and Aude Billard. "A humanoid robot drawing human portraits." 5th IEEE-RAS International Conference on Humanoid Robots, 2005.. IEEE, 2005.

[3] Deussen, Oliver, et al. Feedback-guided stroke placement for a painting machine. 2012.

[4] Song, Daeun, Taekhee Lee, and Young J. Kim. "Artistic pen drawing on an arbitrary surface using an impedance-controlled robot." 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018.

[5] TM5-900, TM 6-axis robot incl. controller, reach 900mm, pay-load 4kg. Weidinger.eu. (n.d.). Retrieved January 13, 2022, from https://www.weidinger.eu/en/p/wl48027