A painting robot based on the Gen3-lite arm

Li Yukun, Liu Chengtao, Wang Bobin
Department of Electronic and Electrical Engineering
Southern University of Technology and Science
Shenzhen, China
{12011431, 12212708, 12212711}@mail.sustech.edu.cn

***Abstract*—This project successfully implemented a simple drawing robot system using the Kinova Gen3-lite robotic arm. The workflow of the entire system is mainly divided into three steps: First, capture an image using a camera and identify the line contours in the picture through computer vision technology; Second, perform path planning on the identified lines, find the drawing order and key points, and simplify the path; Third, control the robotic arm to move according to the planned path, simulate pen-up and pen-down actions, and draw the image on paper. This experiment proved that this system can effectively recognize simple images and drive the robotic arm to complete the drawing.**

***Index Terms*—Gen3-lite, Image Scanning, Recognition, Arm control.**

I. Introduction

This project aims to explore the implementation of a simple automated drawing robot system using the Kinova Gen3-lite robotic arm available in the laboratory. The core goal is clear and practical: to enable the robot to "understand" a simple image and draw the image it sees on real paper. To achieve this goal, we decomposed the complex task into three key and interconnected stages: First, the image scanning and recognition module; Second, the image recognition and trajectory planning module; Third, the robotic arm control module. By integrating the OpenCV image processing library and the control interface of the Kinova robotic arm, we built this basic drawing robot system.

II. Review

We referenced some fundamental computer vision and robotic control methods. In image recognition, edge detection(Canny algorithm) was primarily used to find lines, and dilation and erosion operations were applied to make the lines clearer and more complete. In path planning, lines were firstly simplified into skeletons, then the main path points were extracted, these points were connected in order, and locations requiring "pen-up" were marked. In robotic arm control, the arm's built-in interface was used to control its movement in three-dimensional space, change the height of the pen (Z-axis) simulating "pen-up" and "pen-down" actions, and adjust the height of the pen tip according to the force feedback.

III. Experimental Section

A. Image Scanning and Recognition

1) Preprocessing: Preprocessing is performed first. Three libraries are imported respectively: the OpenCV library for computer vision operations, a numerical computation library for processing image data, and a custom utility module(containing helper functions). Then parameters are configured: control to use the camera, set the image processing size to 480x640, and initialize the camera settings with brightness parameters.

2) Main Loop: Based on configuration, the image source (camera) is selected, and the image size is unified for subsequent processing. If an image is recognized, it is output; when no document is detected, a blank image is created as a placeholder. Subsequently, the recognized image is converted to grayscale to reduce computational load. Finally, Gaussian blur is applied to smooth the image and reduce noise interference.

3) Edge Detection and Morphological Operations: The currently set Canny threshold is obtained from the trackbar. Then the Canny algorithm is applied to detect image edges. Dilation operations connect disconnected edges, and erosion operations remove small noise points. These operations collectively enhance the continuity of document edges.

4) Contour Detection and Processing: First, the original image is copied for contour visualization. findContours is used to detect all contours in the image, where the RETR_EXTERNAL parameter indicates detecting only the outermost contours, and CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal line segments, retaining only the endpoints. Finally, all detected contours are drawn with thick green lines(10 pixels).

5) Enhancement Processing and Result Visualization: The corrected color image from the above steps is converted to grayscale. Adaptive thresholding is applied to achieve the goal of enhancing content contrast. Subsequently, Gaussian weighted mean is used to calculate the local threshold(7x7 neighborhood). The image is inverted to make text black and the background white. Median filtering is applied to remove salt-and-pepper noise. A 2x4 image array is organized based on whether a document is detected. Finally, the custom stackImages function is used to stitch and annotate all images, and the final result is displayed.

6) Save Function and Interactive Control: In this part, after scanning is completed, pressing the 's' key saves the current scan result (where the filename includes a counter to ensure uniqueness). A prompt box with a green background is drawn on the result image, displaying the red text "Scan Saved" to inform the user. It is briefly displayed for 300 milliseconds be-

fore restoring. Subsequently, the counter increments to prepare for the next save.

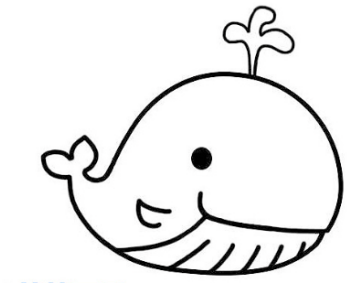

Fig. 1. Original Image

The Fig. 1 shows the original input image.

Fig. 2. Saved Image

The Fig. 2 shows the saved output image, which is the whale drawing that will be processed and traced by the robot.

B. Image Recognition and Trajectory Planning

1) Binarization: Binarization is the first step in the preprocessing phase. The grayscale image is converted into a binary image using a fixed threshold. In this step, the foreground (the path lines) is set to white, and the background is set to black. This transformation is essential because the task involves recognizing simple line drawings, which are typically represented using only black and white, without additional colors. Binarization simplifies the image, making it easier to extract the paths of the drawings while ensuring that only the relevant information (the lines) is kept.

2) Adding White Border and Skeletonization: To prevent the skeletonization process from being affected by the edges of the image, a white border is added around the image using the `copyMakeBorder` function. This ensures that the paths are not cut off at the borders and maintains the integrity of the path recognition process. After adding the border, the image undergoes skeletonization using the `skimage.morphology.skeletonize` function. The skeletonization process reduces the path to its central line, making it easier to extract and follow the path in subsequent steps. This step is particularly important because the image being processed is a simple line drawing, and skeletonization helps maintain the continuity of the path while simplifying it for further recognition and manipulation.

3) Path Extraction: After preprocessing, the next step is to extract the paths from the skeletonized image. Path extraction is crucial for defining the trajectory that the robot will follow when performing the drawing task. The extraction process begins by detecting the endpoints of the skeletonized paths. These endpoints, where the path begins and ends, are identified by analyzing the number of neighboring pixels. If a skeleton pixel has only one neighboring pixel, it is classified as an endpoint. Once the endpoints are identified, the path is traced starting from these points. The tracing algorithm follows the skeleton by exploring neighboring pixels that are part of the path, continuing until no more connected path pixels are found. This step ensures that the entire path is captured from start to finish. After all paths are extracted, they are stored as a series of points that represent the route for the robot to follow. This step is vital because it converts the skeletonized image into a continuous, traceable path that the robot can interpret and execute in the drawing task.

4) Path Merging and Simplification: After extracting the individual paths, the next step is to merge and simplify them to create a continuous trajectory for the robot to follow. The merging process involves joining paths that are close to each other, ensuring that the robot does not have unnecessary interruptions between segments. This step is essential for connecting fragmented lines, where shorter line segments are combined to form longer, continuous paths.

Path segments are merged if their endpoints are within a predefined distance threshold, which ensures that nearby paths are connected. This step is crucial for maintaining a smooth and uninterrupted drawing process.

Once the paths are merged, they are simplified to reduce the number of points and make the drawing process more efficient. Simplification is achieved using the `cv2.approxPolyDP` function, which approximates the path by retaining essential turning points and discarding unnecessary intermediate points. This is important because the robot moves based on individual points, and too many points can slow down its execution.

By reducing the number of points, the robot can follow the path more efficiently, thus improving performance and ensuring smooth operation during the drawing task. The simplification process ensures that while the number of points is reduced, the critical points required to maintain the quality of the drawing are preserved.

The merging and simplification of paths are essential steps in optimizing the robot's performance, ensuring that the draw-

ing process is efficient while maintaining the accuracy and quality of the path.

5) Demonstration of the Robot Drawing: In this section, we showcase the process of the robot drawing a whale, which demonstrates the effectiveness of the image recognition and path generation system. The following images illustrate the stages of the process:

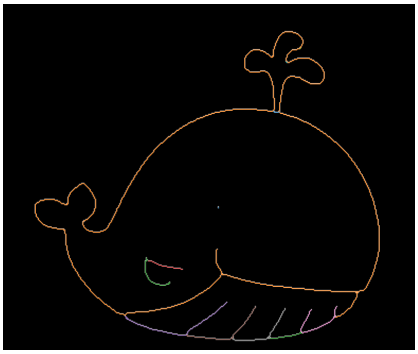

Fig. 3. Recognized Path

The third image depicts the path recognized from the saved image. The robot uses this path to understand where to move and draw.

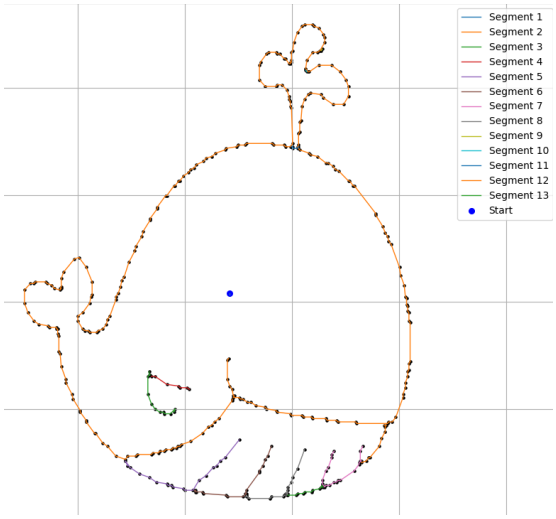

Fig. 4. Robot Path

The fourth image shows the path that the robot will follow to reproduce the drawing. It is a simplified and continuous trajectory, which ensures that the robot can execute the drawing smoothly and accurately.

These images together illustrate how the robot takes the original image, processes it to recognize the path, and then follows that path to draw the whale.

6) Path Normalization and Output: Once the paths are simplified and merged, the next step is to normalize the coordinates of the path points. Normalization is necessary because it converts the coordinates from the pixel space of the image to a normalized coordinate system ranging from 0 to 1. This allows the robot to interpret the coordinates in a standardized way, regardless of the size or resolution of the original image.

The normalization process is performed by dividing the x and y coordinates of each path point by the width and height of the image, respectively. This maps the coordinates to the [0,1] range, which is ideal for robotic motion control.

After normalization, the path points are saved to a text file, with each point written in the format `x,y`, representing the normalized coordinates. Additionally, the paths are separated by a `BREAK` keyword, which indicates where the robot should lift its pen (i.e., a break between paths).

The following is the basic structure of how the paths are saved:

```
x1,y1
x2,y2
...
BREAK
x1',y1'
x2',y2'
...
```

This saved path data serves as input for the robot, which uses the coordinates to move along the correct trajectory to reproduce the drawing. The normalization and output steps ensure that the robot receives accurate and efficient path data, allowing it to perform the drawing task smoothly.

C. Robotic Arm Control

1) Force Estimation via Jacobian Transpose: To enable force-controlled painting without dedicated force sensors, we implement a model-based force estimation approach using the manipulator's joint torque measurements. The end-effector forces are computed through the Jacobian transpose relationship:

$$\mathbf{F}_{ee} = (\mathbf{J}^T)^{-1}\boldsymbol{\tau} \tag{1}$$

where $\mathbf{F}_{ee} \in \mathbb{R}^6$ represents the estimated Cartesian forces and torques at the end-effector, $\mathbf{J} \in \mathbb{R}^{6\times 6}$ is the manipulator Jacobian matrix, and $\boldsymbol{\tau} \in \mathbb{R}^6$ denotes the measured joint torques.

The forward kinematics and Jacobian computation utilize the Denavit-Hartenberg (DH) parameters specific to the Gen3 Lite configuration. The transformation matrix for each link i is calculated as:

$$^{i-1}\mathbf{T}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_i \\ \sin\theta_i\cos\alpha_i & \cos\theta_i\cos\alpha_i & -\sin\alpha_i & -d_i\sin\alpha_i \\ \sin\theta_i\sin\alpha_i & \cos\theta_i\sin\alpha_i & \cos\alpha_i & d_i\cos\alpha_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

The basic Jacobian for each joint is computed using the cross product formulation:

$$\mathbf{J}_i = \begin{bmatrix} \mathbf{z}_{i-1} \times (\mathbf{p}_{ee} - \mathbf{p}_{i-1}) \\ \mathbf{z}_{i-1} \end{bmatrix} \tag{3}$$

where $\mathbf{z}_{i-1}$ is the rotation axis of joint i and $\mathbf{p}_{ee}$, $\mathbf{p}_{i-1}$ represent the end-effector and joint positions, respectively.

2) Force-Controlled Trajectory Tracking: The painting task requires maintaining consistent contact force between the brush and canvas while following predefined artistic paths. We implement a proportional-derivative (PD) force controller that modulates the vertical position (z-axis) based on force feedback:

$$\Delta z = K_p e_f + K_d \dot{e}_f \tag{4}$$

where $e_f = F_{desired} - F_z$ represents the force error, $F_{desired} = 13$ N is the target contact force, and $K_p = 0.005$, $K_d = 0.001$ are the proportional and derivative gains, respectively.

The controller incorporates several key features for robust operation:

- **Contact Detection**: The controller activates only when $|F_z| > F_{min}$, where $F_{min} = 1$ N, preventing unstable behavior in free space.
- **Saturation Limits**: Position adjustments are constrained to $|\Delta z| \leq 0.01$ m per control cycle to ensure smooth motion.
- **Cumulative Tracking**: The controller maintains a cumulative position offset to prevent steady-state errors.

3) Path Generation and Execution: The artistic trajectory is defined by a sequence of 2D waypoints stored in a text file, with special "BREAK" markers indicating brush lift-off points. The path execution algorithm implements the following coordinate transformations:

$$\begin{bmatrix} x_{robot} \\ y_{robot} \end{bmatrix} = \mathbf{R}(\phi) \begin{bmatrix} x_{path} \\ y_{path} \end{bmatrix} \tag{5}$$

where $\mathbf{R}(\phi)$ is a 2D rotation matrix with $\phi = -50$ to align the drawing coordinate frame with the robot base frame. The path coordinates are additionally scaled and offset to map the artistic design to the physical workspace:

$$x_{scaled} = \frac{x_{raw} + 0.1}{2.5}, \quad y_{scaled} = \frac{y_{raw} + 0.1}{2.5} \tag{6}$$

4) Implementation Architecture: The control system is implemented using the Robot Operating System (ROS) framework with the following node structure:

1) **Jacobian Node** (`jacobian.py`): Subscribes to joint states from the Kortex driver and publishes:
   - `/tool_pose_cartesian`: Current end-effector position
   - `/tool_velocity_cartesian`: Cartesian velocities
   - `/tool_force_cartesian`: Estimated contact forces
2) **Painting Control Node** (`painting.cpp`): Implements the main control logic:
   - Subscribes to force feedback for real-time control
   - Manages gripper commands for brush manipulation
   - Executes Cartesian motion commands with constrained velocities
   - Implements asynchronous action monitoring for motion completion

The Cartesian motion commands utilize the Kortex API's `ExecuteAction` service with velocity constraints:

- Translation velocity: 0.2 m/s
- Orientation velocity: 10°/s

5) Experimental Protocol: The painting sequence follows a structured protocol to ensure repeatability and safety:

1) **Initialization**: Set Cartesian reference frame to robot base and activate action notifications
2) **Homing**: Execute predefined home action (ID: 1) to reach a safe starting configuration
3) **Tool Pickup**:
   - Move to brush location at $(x = 0.162, y = -0.201, z = 0.15)$ m
   - Close gripper to 95% position
4) **Painting Execution**:
   - For each waypoint in the path file:
     - If "BREAK" marker: Lift to $z = 0.15$ m
     - Otherwise: Maintain $z = 0.028$ m nominal height with force feedback adjustment
     - Apply coordinate transformation and execute motion
5) **Completion**: Return to home position and open gripper

The force feedback control loop operates asynchronously, updating the target position based on the latest force measurements. The controller's discrete-time implementation includes adaptive time-step calculation to handle variable update rates:

$$\dot{e}_f^{(k)} = \frac{e_f^{(k)} - e_f^{(k-1)}}{\Delta t^{(k)}} \tag{7}$$

where $\Delta t^{(k)}$ is dynamically computed from system timestamps, with a minimum threshold of 1 ms to prevent numerical instability.

D. System Performance

The implemented force-controlled painting system successfully maintains consistent brush-paper contact throughout complex artistic trajectories. The PD controller's response characteristics ensure stable force regulation while accommodating surface irregularities and mechanical compliance in the brush bristles. The modular ROS architecture facilitates real-time monitoring and debugging, with all critical signals accessible through standard ROS topics for visualization and analysis. The following is the final picture drawn:

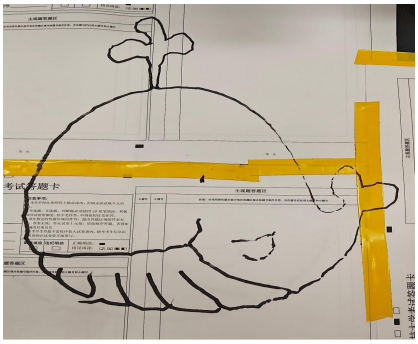

Fig. 5. Painted Picture

For the specific image recognition and painting process, please refer to `Show.mp4`

IV. Conclusion

A. Contributions

This paper presented a comprehensive robotic painting system that successfully integrates computer vision, path planning, and force-controlled manipulation to enable autonomous artistic creation. The developed system demonstrates the feasibility of transforming visual input into physical artwork through the seamless coordination of multiple technological components.

The key contributions of this work include:

1) **Robust Image Processing Pipeline**: The implementation of adaptive edge detection and morphological operations ensures reliable extraction of line drawings under varying lighting conditions. The skeletonization and path simplification algorithms effectively reduce computational complexity while preserving essential artistic features.
2) **Model-Based Force Estimation**: By leveraging the Jacobian transpose relationship, we achieved real-time force feedback without requiring dedicated force sensors, significantly reducing system cost and complexity while maintaining painting quality.
3) **Adaptive Force Control**: The PD controller with contact detection and saturation limits successfully maintains consistent brush-paper contact force throughout the painting process, compensating for surface irregularities and mechanical compliance.
4) **Integrated System Architecture**: The ROS-based framework facilitates modular development and real-time monitoring, enabling efficient debugging and system optimization.

Experimental results demonstrate that the system can successfully reproduce simple line drawings with high fidelity. The painted whale image validates the effectiveness of our approach, showing clear lines with consistent thickness and proper handling of discontinuous paths through the BREAK marker mechanism.

B. Limitations and Future Work

While the current system performs well for simple line drawings, several areas warrant further investigation:

- **Complex Artwork**: Extension to handle shading, color mixing, and multiple brush types would enable more sophisticated artistic expression.
- **Adaptive Control**: Implementation of learning-based controllers could improve force regulation across different paper textures and brush types.
- **Path Optimization**: Development of algorithms to minimize pen lifts and optimize drawing sequences could reduce execution time and improve efficiency.
- **Real-time Feedback**: Integration of visual feedback during painting could enable closed-loop control for enhanced accuracy and error correction.

The successful implementation of this painting robot system demonstrates the potential for robotic systems in creative applications. By combining precise mechanical control with intelligent perception and planning, we have shown that robots can effectively bridge the gap between digital art and physical creation. This work lays the foundation for more advanced robotic art systems and contributes to the growing field of creative robotics.