Final Project Report
Team Name : Flex Forces
Submitted as part of : RSS Spring 2025 (CS5335)

Team:
Chainathan Santhanam Sudhakar
Yash Wakde
Varun Raghavendra
Shengjun Sun
Demo Link | Github Link

INTRODUCTION:

Our project focuses on robotic drawing of portrait images. Provided the system with an RGB portrait image, the robotic arm is expected to sketch essential features from the portrait image. In other words, the robotic sketch should retain essential features about the person presented in the portrait image. The evaluation metric of success will thus be, whether or not an audience can tell from the sketch who the person has been drawn.

During our initial discussions for choosing the project, we asked ourselves, why is this task important in the real world ? And we came up with some of these reasons below that convinced us to proceed with the project. By the end of the project, we see a lot of potential use of this drawing technology in real world scenarios :

- **Translate cognitive behavior of people with motor impairments / amputees :** This technology can be extended to translate words / sign languages / visual descriptions of what is there on their mind on canvas.
- **Humanoid Testing and Validation :** Drawing is a human-like task, making it a good benchmark for testing cognitive and motor skills in humanoid robots.
- **Other specialized based pattern based tasks :** The end effector can be replaced by other precise tools for performing other pattern based applications like welding, soldering, cooking, etching, carving, and the list is endless!

For completing this task, there is a pipeline which touches on different concepts about robotics learned within this semester. At the top of the pipeline is image processing: process the portrait image such that the essential features are obtained. Then, from the obtained essential features, do path planning for the robotic arm to execute upon. After the robotic arm executes all paths, then the drawing will be finished. A more detailed analysis of the pipeline is provided in the Implementation session.

The entire workflow of this project involves designing, implementing on the design, experimenting with the implementation, and evaluating the experiential result. From evaluating the experiential result, we identify drawbacks within the drawn image, and we analyze potential issues within the pipeline that results in the drawbacks. Then, we go back to our pipeline and pinpoint areas to modify, after which we re-experiment, and see whether the drawback still persists in the drawn image. From this project, we gained experiences of planning a general goal into a sequence of narrowed implementations, which we execute one after another and ultimately form a pipeline. The planning and troubleshooting experiences we've gained pave the way for our future success in the robotic area.

WHY IS IT CHALLENGING TO IMPLEMENT ?

Drawing with a robotic arm is challenging due to the need for high precision and fine motor control to replicate detailed lines and curves, requiring sub-millimeter accuracy. It involves solving complex inverse kinematics and optimizing smooth trajectories under joint constraints. Maintaining consistent contact force between the tool and surface adds further difficulty, as too much or too little pressure can compromise the drawing. Additionally, different drawing tools behave differently, making tool-surface interaction hard to model. Real-time perception and feedback are also crucial for adjusting movements dynamically and ensuring accuracy. The pose estimation of the pen/brush is another challenging effort, which could require advanced tactile sensing and accurate tracking.

IMPLEMENTATION

This implementation presents a complete pipeline for autonomous robotic sketching, combining image-based path extraction with robotic arm control. It consists of two main components: an edge detection and path extraction algorithm that converts an image into clean, drawable trajectories, and a robot control class that translates those trajectories into precise physical drawing motions on a canvas.

Edge Detection and Path Extraction Algorithm : The code implements a structured algorithmic pipeline to extract and process geometric path structures from an RGB image, transforming raw visual image into clean, resampled trajectories.

- The process begins with converting the input image to grayscale, followed by Gaussian blurring to reduce noise and applying the Canny edge detector to identify strong gradients and edges.
- These edges are binarized to create a black-and-white image where the foreground (edges) is represented by 1s.
- We use scikit-learn library and its components for performing operations on the image. Skeletonization is applied to thin the edges down to one-pixel-wide lines, preserving the topological structure of the image while simplifying the data. To further refine the structure, isolated pixels or small noisy objects are removed using morphological operations. From the skeletonized image, the algorithm now creates an undirected graph by linking nearby pixels as edges and considering each edge pixel as a graph node.
- This transformation allows for systematic analysis of the edge network using graph theory. The graph is then decomposed into its connected components, and within each component, the algorithm identifies branch linear paths between endpoints or junctions using a breadth-first search approach.
- These branches represent meaningful segments of the image's structure. Each branch is subsequently resampled using linear interpolation to enforce uniform spacing between points, ensuring geometric smoothness and consistency.
- The final output is a set of clean, uniformly spaced paths that closely follow the skeletonized edge structures of the original image.

Robot Class : The Robot Class code covers the setup, motion control, and path execution functionalities for the arm.

- Initialization : The robot's drawing workspace is defined as a square canvas with specific bounds and origin, all in real-world dimensions. The robot interface is then initialized, and startup routines are called to prepare the arm for motion. The robot's end-effector heights for "pen-up" and "pen-down" states are defined to simulate the drawing motion.
- The class provides utility functions to control the robot's posture (go_to_home_pose, go_to_sleep_pose), perform pen movements (pen_up, pen_down), and manipulate the gripper.
- We have also defined diagnostic functions to verify the workspace boundaries (check_canvas_bounds) and the vertical range of pen motion (check_pen_height_pos).

- The core functionality lies in the draw_image() method, where an input image is first processed using generate_sketch_paths() a utility function that extracts and resamples contour paths from the image to match a target spacing and fit within the robot's canvas.
- Each path is converted from pixel to canvas coordinates and sequentially executed using draw_path(). This method carefully moves the end-effector to trace each path by calculating and applying relative (delta) displacements, mimicking the motion of a pen drawing on paper. The robot lifts and drops the pen at the start and end of each path to simulate proper sketching behavior.
- Finally, once all paths are drawn, the robot is returned to its home position and waits for the next commands.

DESCRIPTION OF THE DEMONSTRATION OF THE EXPERIMENT

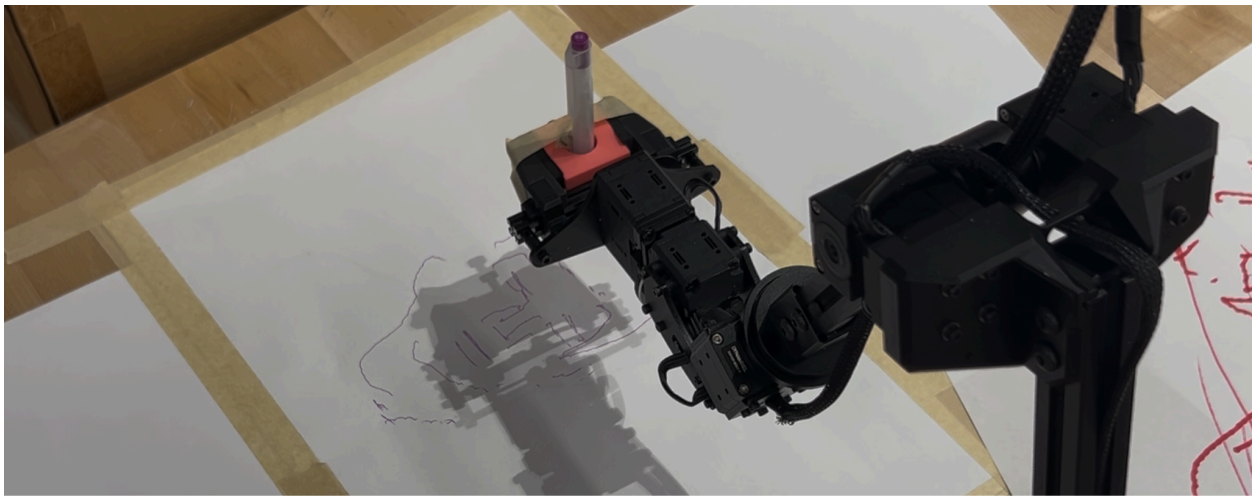

Fig 1. A snapshot of the Trossen Widowx 250s Robotic Arm completing a portrait

Brief Specifications of Trossen Widowx 250s Robot :

The **Widowx 250s** robotic arm is a 6-DOF arm, which has a maximum payload of 250 grams. It has a wide reach and span of 650mm and 1300mm respectively. Span and Reachability determine the maximum area the robot can access, which is crucial for deciding the size and placement of the canvas, thereby providing freedom to draw sketches on large canvas sizes such as ours. If the canvas is placed outside the robot's effective working bounds, it won't be able to reach all poses and points, resulting in incomplete sketches or collisions. The robotic arm is rated for a repeatability of 1mm, also suggesting that, for the same drawing to be repeated, there is negligible margin of error and can prove consistency and accuracy in achieving desired poses.

Environment and Setup:

- The drawing surface is fixed and flat, taped firmly to the table and we made sure that it is a writable surface with as little friction as possible! We tested our drawing bot on surfaces like cardboard pizza box, glossy chart and A3/A4 White sheets.

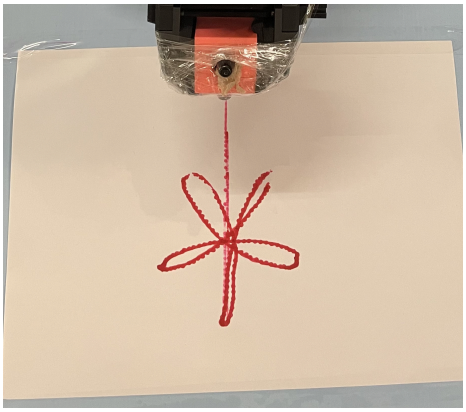

Fig 2. A snapshot of a flower drawn using our algorithm on an A4 Sheet

- We 3D Printed a custom adjustable pen holder (with screw to tighten or loosen) using Northeastern's Makerspace, which really helped us hold the pen in a rigid position and draw with less jitter and obtain smooth paths. We further applied an adhesive tape to ensure better fixation. We also made sure the gripper was fully closed to the size of the pen holder.

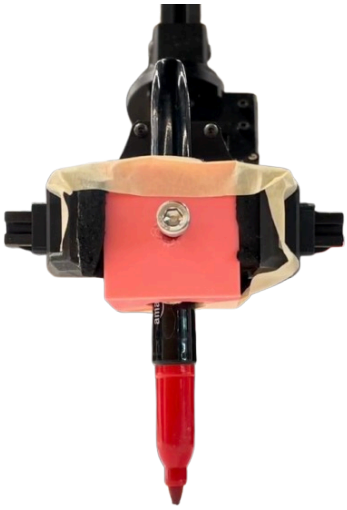

Fig 3. A closer look of the Pen Holder (pink block) mounted on the manipulator arm

- We also tried with different pens with different thickness. An observation was that when we used a marker with a thicker tip, we noticed that the facial sketch had a lot of overlapping paths drawn and hence we moved to a pen with a smaller tip to make the facial features distinguishable.
- Pen Calibration Sequence : We start the robot in its home position, and in the first step, it reaches a "pen_up" height and then we place the pen and calibrate the "pen_down" distance in such a way that the pen tip is convincingly touching the surface of the drawing sheet. We made sure once the pen is placed and calibrated there's no human intervention during the demo, and everything is smooth and complete till the bot achieves a full portrait.

Task Execution:

- The robot begins tracing the first path from the sketch. The movement is linear and composed of multiple short, deliberate strokes that form part of a larger contour.
- After completing each stroke, the robot lifts the pen slightly ("pen up") before repositioning itself to the next starting point; this prevents unintended markings during travel.
- These pen-up and pen-down transitions appear to be dynamically controlled, ensuring smooth, clear lines and well-separated strokes.
- The robot follows a set of predefined trajectories, which were likely derived from an image and processed into a series of paths using a contour extraction and simplification algorithm (as suggested by the call to generate_sketch_paths() in the code).
- Each path drawn seems to correspond to a particular contour or shape from the original image. The robot draws multiple such paths in succession, carefully repositioning between each.

Shapes that we have achieved using our algorithm so far : Square (Each Waypoint is the corner of the square) which is extendable to other polygons, Circle, Flower Petals, facial contours, face.

CHALLENGES WE FACED DURING THE PROJECT AND HOW WE OVERCAME THEM

1. **Excessive Waypoints from Sketch Conversion:**

- *Challenge:* The initial sketch extraction process generated a dense set of points, resulting in unnecessarily long and complex motion paths.
- *Solution:* We implemented a resampling technique to reduce the number of waypoints while preserving the shape of the sketch. This significantly improved the efficiency and smoothness of the robot's movements.

2. **Mechanical Instability of the Pen:**

- *Challenge:* The robot gripper was too wide and lacked the precision to firmly hold the pen, causing frictional issues and bending during motion.
- *Solution:* We designed and 3D printed a custom pen mount to securely attach the pen to the robotic arm, ensuring stability during sketching.

3. **Erratic Robot Behavior During Consecutive Movements:**

- *Challenge:* When executing consecutive movement instructions using the Interbotix package, the robot arm often glitched skipping commands, jerking unpredictably, or terminating execution prematurely.
- *Solution:* We addressed this by tuning the motion hyperparameters such as movement duration and number of interpolation waypoints. Additionally, we introduced short idle delays between successive instructions, which helped stabilize execution.

4. **Speed vs Accuracy Tradeoff:**

- *Challenge:* Higher motion speeds caused the robot to overshoot curves or generate vibrations, affecting sketch quality, while slower speeds made drawing unnecessarily time-consuming.
- *Solution:* We tuned the robot's velocity and acceleration parameters to strike a balance between speed and precision, resulting in cleaner sketches without sacrificing efficiency.

CURRENT LIMITATIONS & THINGS WE WOULD CHANGE GIVEN ANOTHER SEMESTER

1. **Lack of Visual Feedback and Real-Time Correction:**

    - The robot currently lacks a camera or vision system, so it cannot detect or correct drawing errors during execution. Any deviation due to surface friction or joint inaccuracies goes uncorrected.

2. **No Real-Time Error Recovery:**

    - There is no feedback loop to detect failed motions or recover from unexpected behavior mid-execution. This limits the system's robustness for long or complex sketches.

3. **IK Solver Instability:**

    - The current system uses the default inverse kinematics solver from the Interbotix package. At times, it causes the robot to freeze or slow down unexpectedly between movements, affecting sketch continuity.

4. **No Collision Detection:**

    - The robot operates without awareness of obstacles in its workspace. While it works for our current setup, introducing collision detection would be essential for more complex or dynamic environments.

5. **Basic Sketch Generation Pipeline:**

    - The sketch generation process relies on classical edge detection and thresholding, which often require manual tuning. A more advanced pipeline using generative models (e.g., sketch simplification using diffusion or GAN-based models) could improve quality and reduce preprocessing effort.

6. **From sampled points into clusters:**

    - From a collection of sampled points into a set of clusters. In this process, the parameters like 'minimum number of points for forming a cluster' and 'distance threshold within which two points would be considered as in one cluster' need to be tuned through trial-and-error.