SCHOOL OF ENGINEERING, SFSU

ENGR 895 – RESEARCH PROJECT

PROJECT: "QARM-BASED ROBOTIC FACE DRAWING SYSTEM"

SUPERVISED BY: DR MOJATABA AZADI SOHI

BY:

DHOLAKIYA MILAN PRAVINBHAI

#923655574

Abstract:

This paper presents the design and implementation of a vision-guided robotic drawing system using the Quanser QArm. The objective is to enable the robot to autonomously replicate human facial sketches on a whiteboard by integrating image processing, path planning, and real-time control. The system architecture combines Python-based image acquisition and contour extraction via OpenCV with trajectory generation in MATLAB.

A closed-loop control framework is implemented in Simulink using the QUARC interface for accurate motion execution. SolidWorks is used for designing custom end-effector mounts and drawing tools. While the control performance and trajectory tracking are stable and reliable, some limitations were observed in the precision of physical sketch output due to mechanical constraints and end-effector orientation. This work demonstrates the feasibility of merging computer vision with robotic motion control for artistic tasks, highlighting potential improvements in trajectory decomposition, gripper design, and multi-axis compensation.

Keywords: Path Planning, Robot Control, Quanser, Image Processing, Kinematics, Simulink, Python, OpenCV.

Table of Contents

1. Introduction

The report outlines the nature of the project and how it relates to industry, as well as the problem that motivated the objective-setting to solve it. Moreover, this report also discusses the delimitations that were found during the development process.

1.1 Background

In recent years, the deployment of robotic systems has expanded significantly across various domains, ranging from industrial manufacturing to service-based applications and even traditional manual fields. It is now increasingly common to observe robotic arms being utilized in areas once considered unlikely for automation, such as culinary arts, domestic environments, and creative practices like drawing and painting [1]. Forecasts indicate that robotics will continue to permeate daily life, becoming a standard presence in routine tasks [2].

This growing integration of robotics has facilitated new modes of operation, particularly in contexts aligned with the industry 4.0 paradigm, where human-robot collaboration plays a central role in enhancing productivity and worker safety [3]. Among the many types of robots available, robotic arms have gained prominence due to their versatility, precision, and ability to operate within shared environments. They can perform complex actions, such as manipulating tools or replicating human-like motions which can assist humans in completing intricate tasks and reducing physical strain [4].

While earlier research in robotics primarily addressed basic automation and motion control, current developments focus on more advanced capabilities. These include learning, decision-making, and problem-solving powered by artificial intelligence, enabling robotic arms to engage in fields such as visual arts. As such technologies evolve, a compelling question emerges: can a robotic arm be programmed to create artistic works, such as sketches or portraits, with the sensitivity and fluidity typically attributed to human artists? [5]

1.2 Problem Statement and Objectives

Despite significant advances in robotic automation and visual perception, enabling a robotic system to accurately reproduce artistic sketches from visual inputs, particularly human faces remain a multifaceted challenge. Robotic arms traditionally lack the fine-grained compliance needed for consistent surface contact, often resulting in incomplete line segments, excessive pressure, or deviation from intended contours [1], [5]. These limitations are exacerbated when working on non-horizontal planes or transforming 2D pixel data into real-time 3D motion trajectories, especially without tactile sensing or force feedback [3].

Furthermore, existing systems in artistic robotics often emphasize simulation-based performance or fixed-environment setups, which do not fully reflect the complexities of real-world operation, including joint inaccuracies, drawing board irregularities, and variable marker wear [2]. The difficulty lies not only in generating path trajectories from images, but also in ensuring that these paths are executed with sufficient mechanical precision and compliance to yield consistent visual output [4].

The objective of this project is to develop an integrated robotic sketching system that bridges the gap between image processing and physical actuation. Specifically, the system aims to:

Acquire facial images using a live webcam feed and extract clean contour lines via OpenCV-based preprocessing. Transform 2D sketches into 3D paths that conform to the operational limits of the Quanser QArm. Implement a custom-designed spring-loaded marker holder that allows safe yet stable contact with the sketching surface, compensating for board curvature and marker tip variability [4]. Introduce an adaptive X-axis offset correction mechanism based on real-time Y and Z coordinates to minimize uneven pressure or dragging across different facial regions. Execute and monitor the drawing using a closed-loop Simulink + QUARC control environment, incorporating trajectory generation, inverse kinematics, and forward kinematics modules. Evaluate results through both simulation trajectory plots and real-world drawing comparisons, ensuring repeatability and control accuracy.

Through this approach, the system demonstrates the potential of combining visual intelligence, compliant hardware, and real-time control to enable robotic sketching in unconstrained physical environments, a key capability for future robot-human creative collaboration [1], [2], [5].

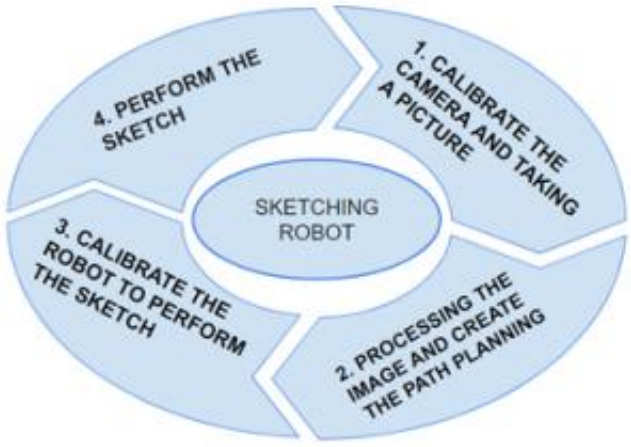

Figure1.1: Objective of the Project

1.3 Overview

This report presents the development of a robotic sketching system using the Quanser QArm, where image processing and robotic control techniques are combined to replicate human facial sketches on a whiteboard. The work begins by outlining the theoretical background, including robot kinematics, control methods, and the image processing approach used to generate sketch-like paths. It then reviews related research before describing the methodology followed to capture facial images, process them into drawable contours, and convert them into 3D trajectories. The hardware and software integration, including real-time communication with the QArm, are explained in the design section. The system's performance is evaluated using both simulation and real-world drawing results, highlighting strengths and limitations. Finally, the report discusses conclusions and possible future improvements, including advanced AI processing and enhanced control strategies.

2. Theoretical Frame of Reference

This part of the project is dedicated to clarifying the fundamental concepts that support the development of the sketching system. It provides essential theoretical knowledge on robotic control, image processing, and system integration, which will be expanded upon in later sections to show how they have been applied in the project.

2.1 Quanser:

The robot selected to achieve the objectives of this work is the Quanser QArm, a 4-DOF articulated robotic manipulator developed for research and academic applications. It is equipped with precise servo motors, a tendon-driven gripper, and integrated current sensing, providing a compact yet capable platform for implementing real-time control and trajectory planning. With its highly accurate positioning and smooth motion, the QArm is well suited for tasks that require spatial precision such as drawing. Figure 2.1 illustrates key specifications of the QArm robot used in this study [6].

The QArm is connected to the host system using QUARC real-time software, which allows direct programming in Simulink and supports closed-loop control using encoder and current feedback. The robot can follow pre-defined paths using trajectory commands defined in MATLAB or Simulink and transmitted in real-time via USB. Its integrated sensors and modular design also support learning-based control strategies like lead-through teaching or vision-guided motion [7].

The system enables position control with PWM or analog signals and allows custom

Device Specifications	
Manipulator weight	8.25 kg
Payload	350 - 750 g
Reach	750 mm
Repeatability	± 0.05 mm
Camera	Intel® RealSense™ D415
Interface	USB (QFLEX 2)
Internal control modes	Position mode, Current mode
External control rate	500 Hz
Internal control rate (min)	1000 Hz
Expandable I/O	PWM/Analog/I²C/SPI/UART
Minimum and maximum joint range	Base: ± 170° Shoulder: ± 85° Elbow: -95°/+75° Wrist: ± 160°
Maximum joint speed	± 90°/s

Figure 1.1: Specification of Quanser

development of inverse and forward kinematics models. These capabilities are essential in transforming 2D image contours into accurate 3D movements on the drawing surface.

2.2 Software

For this project, the selection of software is essential not only for robotic control but also for the image acquisition, pre-processing, path planning, and real-time simulation-execution cycle. The two main tools used in this work are MATLAB/Simulink (for control modeling and communication with the robot) and OpenCV (for image processing and sketch extraction).

2.2.1 MATLAB/Simulink and QUARC

MATLAB and Simulink are developed by MathWorks and provide a powerful numerical computing environment and a model-based design platform, respectively. In this project, Simulink is used as the primary environment for designing and simulating control algorithms, including inverse kinematics, trajectory generation, and impedance control modules for the QArm robot. This graphical programming environment allows users to design complex control systems visually without manually writing extensive code [8].

The connection between Simulink and the physical QArm hardware is achieved through the QUARC real-time control software, developed by Quanser. QUARC provides blocks that integrate directly into Simulink models, allowing real-time execution, sensor data acquisition, and actuator command streaming. This makes it possible to synchronize the QArm's motor control with processed visual data, enabling closed-loop operation [9]. This modular setup enables real-time trajectory testing, visualization, and debugging, and makes the system robust for experimental robotics. Additionally, trajectory logging and live plotting in MATLAB facilitate validation and tuning of motion behavior.

2.2.2 OpenCV

In this project, OpenCV serves as the primary tool for image processing tasks. As an open-source library originally developed by Intel, OpenCV offers a wide range of computer vision algorithms that support the objectives of this work. It enables operations such as thresholding, edge detection, and image segmentation, which are essential for transforming facial images into simplified sketch-like contours. The version utilized for this project is OpenCV 3.4.2 [10].

2.3 Vision System and Camera Integration

The vision system is a crucial component of this project, responsible for capturing and interpreting visual information used to guide the robotic drawing process. In this work, a 2D RGB image is acquired through a camera and processed using OpenCV functions to extract the relevant features of a human face. This involves converting the raw image into a format that isolates edges and contours suitable for trajectory generation. The RGB image is treated as a 3D matrix, where the first two dimensions represent pixel locations and the third holds intensity values across red, green, and blue channels. Each pixel carries an intensity value ranging from 0 to 255, which defines the color and brightness at that location. These values are processed to enhance contrast and remove noise [10].

To enable real-time image acquisition and sketch extraction, this project incorporates the Intel RealSense D415 RGB-D camera, which plays a central role in capturing facial data and translating it into trajectories for robotic drawing. The D415 combines a high-resolution RGB sensor with depth perception capabilities using an infrared projector and stereo depth sensors. It delivers a depth resolution of up to 1280×720 and a depth range of approximately 0.3 to 10 meters, which is sufficient for close-range facial imaging under indoor lighting conditions. The camera is mounted rigidly in front of the subject and calibrated to ensure its output aligns with the QArm's reachable workspace on the drawing board. Figure 2.3 illustrates key specifications of the Intel Camera used in this study [11].

Environment	Indoor and outdoor
Depth Technology	Active infrared (IR) stereo
Image Sensor Technology	Rolling shutter: 1.4 µm × 1.4 µm pixel size
Depth Field of View (FOV) (Horizontal × Vertical) for HD 16:9	69.4 × 42.5° (+/–3°)
Depth Stream Output Resolution	Up to 1280 × 720 pixels
Depth Stream Output Frame Rate	Up to 90 fps
Minimum Depth Distance (Min-Z)	0.16 m
Maximum Range	~10 m
RGB Sensor Resolution and Frame Rate	Up to 1920 × 1080 pixels at 30 fps
RGB Sensor FOV (Horizontal × Vertical)	69.4 × 42.5° (+/– 3°)
Camera Dimension (Length x Depth x Height)	99 mm × 20 mm × 23 mm
Connector	USB Type-C

Figure 2.3: Specification of Intel Real Sense Camera [11]

2.4 Image Processing

In this project, edge detection plays a central role in extracting the primary contours from facial images. Edge detection involves identifying points where image brightness changes sharply, which often corresponds to object boundaries. Among the various methods explored, the Sobel and Prewitt operators were tested due to their simplicity and effectiveness in detecting gradients in specific directions [12]. Preprocessing techniques such as Gaussian blur and bilateral filtering were applied beforehand to reduce noise. Bilateral filtering was effective at smoothing the image while preserving important edge features [13]. The Canny edge detector was ultimately selected for its multi-stage process combining Gaussian smoothing, gradient calculation, non-maximum suppression, and hysteresis thresholding, which together yield clean and continuous edges [14]. This method has become a widely used standard in computer vision for generating binary sketches suitable for robotic path planning [15].

Segmentation methods were used to separate the object of interest (face) from the background. Thresholding-based segmentation was implemented, including global and adaptive thresholding approaches [16]. Global thresholding methods such as Otsu's algorithm rely on image histograms to determine a single threshold value, whereas adaptive thresholding dynamically changes the threshold based on local intensity variation [17]. These approaches were useful when dealing with varied lighting conditions. Other segmentation techniques like watershed and clustering were also considered for their capability to delineate complex shapes and objects within the image [18]. Thresholding proved to be effective in isolating facial features from cluttered scenes, enabling the robot to focus only on relevant drawing regions.

Other alternatives like the SUSAN detector, Moravec operator, and Trajkovic's fast corner detector were evaluated for completeness but were not deployed in the final version [19].

2.5 Control of the Robot

After processing the image, the robot needs to execute movements along a path to draw the final sketch. To perform these movements in the most effective manner, it is essential to consider concepts regarding how these movements are carried out, their limitations, and how they can be controlled.

2.5.1 Type of Joints, Movement, and Interpolation

Robotic arms consist of rigid links connected by joints, which are actuated to produce motion. The two primary joint types are prismatic (translational) and revolute (rotational), and combinations of these form the architecture of various robot configurations. The arrangement of joints and degrees of freedom (DOF) directly influences a robot's ability to reach different poses and perform tasks in complex environments. Common robotic

structures include Cartesian, cylindrical, spherical, SCARA, and anthropomorphic configurations, as illustrated in Figure 2.5.1. Each configuration suits specific applications depending on workspace shape, required flexibility, and control complexity [20]. For instance, anthropomorphic robots typically offer higher dexterity and are capable of navigating obstacles through redundant DOFs.

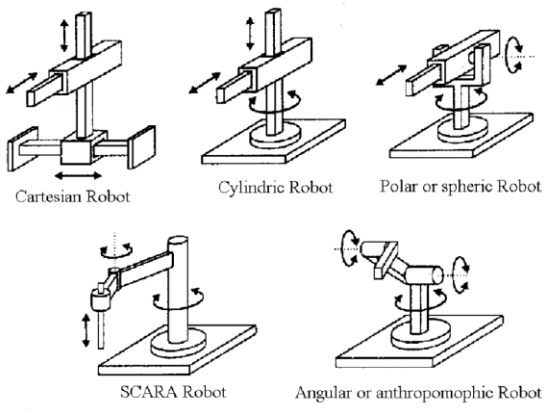

Figure 2.5.1: Common robot configurations including Cartesian, Cylindrical, Polar, SCARA, and Anthropomorphic designs [20]

2.5.2 Controllers and Feedback

Controllers play a fundamental role in robotic systems by processing input signals and generating corresponding actuator commands that guide robot behavior. They serve as the decision-making core, executing pre-programmed algorithms based on sensory data and user-defined objectives. The design of these control systems must accommodate environmental uncertainty and task complexity, which is why flexible controller structures are often used in modern robotics [21]. To ensure task precision, feedback mechanisms are employed to monitor the actual state of the robot and compare it to the desired setpoint. The resulting error is then used to adjust commands and improve accuracy.

Robotic systems typically utilize either open-loop or closed-loop control structures. Open-loop control is suitable for repetitive or non-critical tasks where feedback is unnecessary, whereas closed-loop systems are better suited for dynamic environments that require continuous adaptation. Closed-loop control, also known as feedback control, employs sensors to monitor position, torque, or force, which helps maintain desired performance even when disturbances occur [22]. Vision controllers are a special case of feedback systems, where cameras provide real-time environmental input, enhancing accuracy in positioning and manipulation tasks [23]. In this project, such feedback is crucial for maintaining consistent marker pressure and trajectory tracking. Figure 2.5.2 illustrates a typical closed-loop control system architecture, highlighting the role of each component in error correction and task execution [24].

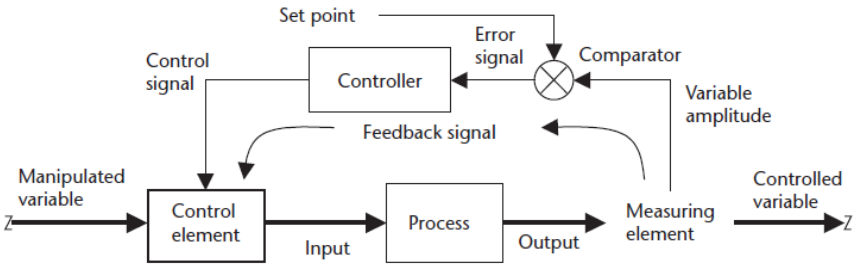

Figure 2.5.2: Closed-loop control system schematic showing setpoint tracking with feedback signal comparison [24].

2.5.3 Trajectory Planning

Trajectory planning refers to the process of determining the time-dependent position, velocity, and acceleration profiles that a robot manipulator must follow to execute a desired motion smoothly and accurately. As part of motion planning, it bridges the gap between path planning and real-time control by considering kinematic and dynamic constraints such as joint limits, velocity bounds, and acceleration profiles. Trajectory planning is typically done in either Cartesian space or joint space, and the trajectory itself can be designed as either point-to-point or continuous motion depending on the application. Figure 2.5.3 illustrates this structure, showing how trajectory planning fits within the larger framework of robotic motion planning.

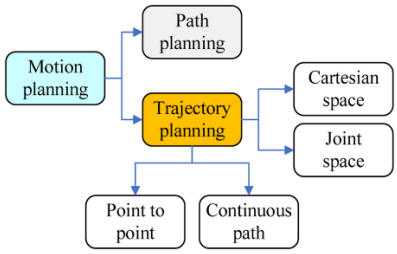

Figure 2.5.3: Relationship of trajectory planning within motion planning [25]

3. Literature Review

The use of vision-guided robotic manipulators for drawing and sketching has evolved significantly over the years, driven by advancements in computer vision, kinematics, and control systems. Early robotic arms were primarily developed for industrial tasks in hazardous or repetitive environments. The first known industrial robot patent, attributed to George Devol in 1954, laid the groundwork for the development of programmable manipulators capable of performing complex tasks autonomously [26]. The definition of robots as “reprogrammable, multifunctional manipulators” [27] has since expanded to include collaborative robots, or cobots, which can safely interact with humans in shared

environments. Their adoption is rapidly growing across domains such as education, manufacturing, and even art [28], [29].

One of the earliest milestones in robotic drawing dates to the 18th century, when Pierre Jaquet Droz created mechanical automata capable of producing sketches [30]. In the mid-20th century, Jean Tinguely's kinetic sculptures and Harold Cohen's AARON software represented early explorations into generative robotic art [31], [32]. AARON implemented rule-based logic to generate stylized line drawings autonomously [33]. Later developments saw robotic systems integrate face detection and image processing to generate realistic portraits. For example, Calinon et al. introduced a robot that could detect human faces and iteratively reconstruct sketches [34], while Moura's Artsbot explored abstract drawing using mobile platforms [35]. These early efforts established the viability of combining vision and path generation.

More recent advancements focus on enhancing the artistic quality, accuracy, and interactivity of robotic drawing systems. Aguilar and Lipson developed a 6-DOF robot that used internal feedback to adjust brushstroke styles [36], and Lu et al. integrated vision feedback to create pen-and-ink sketches [37]. Gasparetto et al. emphasized trajectory optimization to enhance the quality of robot-generated artwork [38]. Grosser's work introduced an auditory feedback loop, allowing robots to respond to sound environments during sketching [39]. Tresset et al. built facial sketching systems that continuously improved using visual feedback [40], [41], while Jean-Pierre et al. used industrial robots for detailed portrait drawing [42].

Industrial-grade systems have also attempted to replicate human brush techniques. Lindemeier and Deussen's *e-David* used stroke layering and non-photorealistic rendering (NPR) for expressive art creation, combining feedback control with aesthetic rules [43]. Subsequent efforts extended to colorful painting systems [44], graffiti-capable collaborative robots [45], and interactive games like tic-tac-toe with robotic partners [46]. Chen et al. used redundant robots to ensure better adaptability in spray painting tasks [47]. The RobotArt competition, launched by Andrew Conru in 2016, further pushed the boundaries by inviting teams to build robotic artists capable of producing expressive work [48].

The diversity of recent research showcases both fixed-arm and mobile platforms. Galea and Kry created a tethered drone that performed stippling on canvases [49], while Shih and Lin developed trajectory control for mobile sketching robots [50]. Luo et al. and Dong et al. both leveraged SCARA arms for stylized image replication using coordinate-based algorithms [51], [52]. Song et al. used a 7-DOF impedance-controlled manipulator to enable surface-adaptive drawing [53], and Vempati et al. demonstrated 3D spray painting using autonomous UAVs like *PaintCopter* [54].

Recent works have also addressed system integration and user interaction. Karimov et al. produced full-color artworks using traditional paint media [55], and Igno et al. developed interactive systems for realistic acrylic painting using region-based analysis [56]. Human-robot collaborative art, exemplified by Sougwen Chung, highlights the increasing synergy between human creativity and robotic precision [57]. Scalera et al. further explored adaptive trajectory planning for spray-based robots, with variants using tools like knives, sponges, and watercolors [58],[59].

Although a wide range of robotic artists exists, most systems focus either on visual perception or on motion planning, rarely both in an integrated form. This project contributes by using a QArm manipulator with an Intel RealSense camera to perform 3D-to-2D image conversion, force-controlled gripping, and trajectory-executed sketching on a physical board, thereby bridging existing gaps in combined vision, force, and motion integration.

4. Methodology

To ensure the project proceeds in a structured and efficient manner, the Design Science Research Methodology (DSRM) has been selected. This methodology provides a systematic framework for creating and evaluating technical artefacts, particularly in engineering and design contexts. It supports innovation through iterative development and testing, aligning closely with the vision-based robotic sketching objectives of this project. The DSRM, as proposed by Peffers et al. [60], consists of six core steps: (1) identifying the problem and motivation, (2) defining the objectives for a solution, (3) designing and developing the artefact, (4) demonstrating the solution's use in a relevant context, (5) evaluating its performance against the goals, and (6) communicating the results to relevant audiences. These steps are illustrated in Figure 4.1.

The methodology begins by establishing the need to automate sketching tasks using a robotic manipulator, particularly addressing challenges in accuracy, visual detection, and force-controlled drawing. From there, the solution objectives were determined based on required functionalities, namely, camera integration, kinematic control, and smooth trajectory execution. The artefact developed includes Simulink models for kinematics, trajectory generation, and feedback systems, supported by MATLAB scripts for vision processing. Once developed, these modules were demonstrated through simulations and physical experiments using the QArm. The performance was evaluated based on criteria such as drawing precision, repeatability, and trajectory fidelity. Finally, the results and findings are being compiled into scholarly documentation for dissemination.

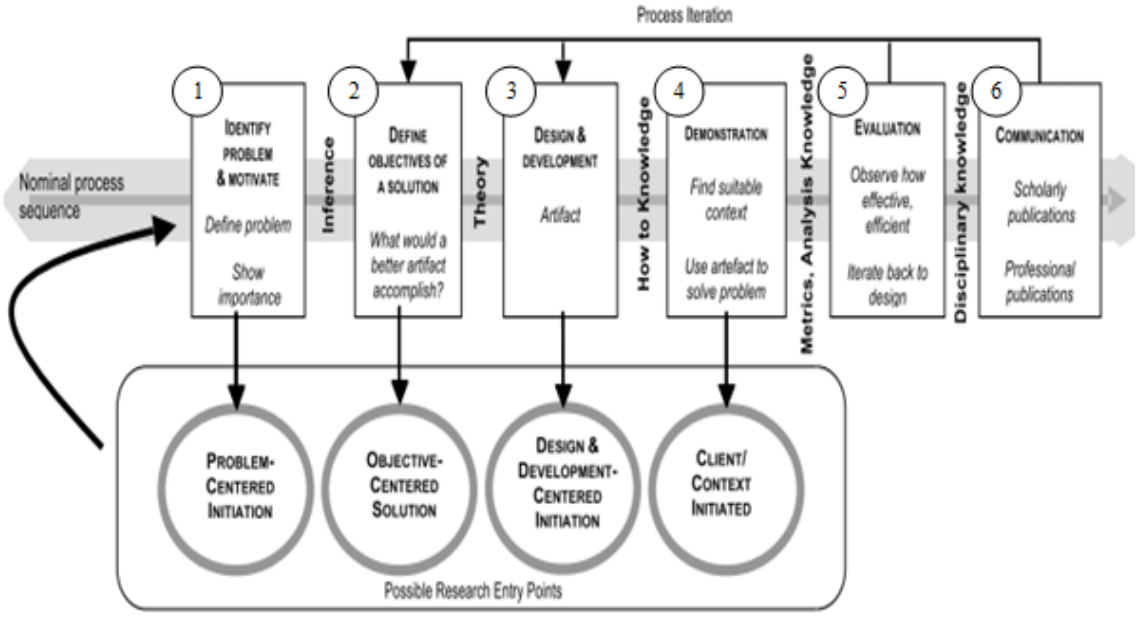

Figure 4.1: The DSRM Process Framework [60]

5. Design and Development

This section outlines the complete system setup, both in hardware and software, as well as the methodology applied in converting visual data into physical motion through the QArm manipulator. While the system successfully demonstrated trajectory execution and real-time control, challenges were encountered in achieving the desired sketching fidelity.

5.1 Hardware Setup and System Overview

The experimental platform was configured in the Robotics Laboratory located in the Engineering and Science Building at San Francisco State University. The QArm robotic manipulator was firmly mounted on a lab bench opposite a fixed whiteboard using mechanical clamps to ensure a stable reference frame for drawing. The robot was powered using a standard power adapter, and communication was established via USB interfaces connected to a computer equipped with an NVIDIA graphics card, allowing reliable data acquisition and rendering from the Intel RealSense D415 camera. The camera was mounted in a fixed position such that it could acquire high-resolution RGB-D frames of the target subject, which were used in downstream image

Figure 5.1.1: Real-world hardware setup showing the QArm robot

processing tasks. Figure 5.1.1 illustrates Real-world hardware setup showing the QArm robot.

Once the hardware was physically arranged, the system followed a streamlined software integration flow, shown in Figure 5.1.2 The image processing script, written in Python and executed in Jupyter Notebook, was initiated to capture images and extract facial contours. The processed trajectory points were then exported and transferred into the MATLAB workspace for conversion into a robot-executable trajectory. MATLAB's custom script was used to perform trajectory simplification, down sampling, and formatting. These points were passed into a real-time Simulink model built with QUARC support to drive the QArm manipulator. The entire system operated as an integrated vision-to-execution pipeline across Python, MATLAB, and Simulink environments.

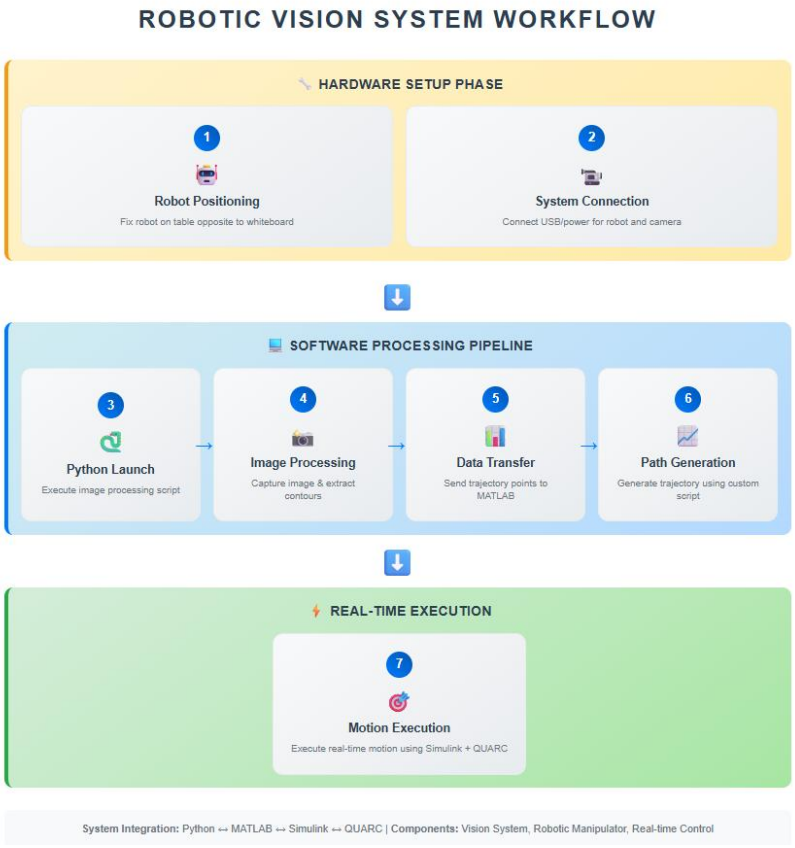

Figure 5.1.2: Robotic vision system workflow including hardware setup, image processing, and trajectory execution.

5.2 Image Acquisition and Processing

To initiate the robotic sketching workflow, a Python-based vision module was developed to capture facial features and convert them into edge-based trajectories. A frontal face is first detected using the Haar Cascade classifier provided by OpenCV's pre-trained models [61]. After initializing the webcam stream, the system captures a high-resolution frame and

converts it into grayscale for improved detection accuracy. Once a face is identified, an elliptical mask is applied over the detected region with additional padding to include peripheral contours. The masked area undergoes Gaussian blur to soften transitions and suppress background noise. This isolated region is then processed through a contrast enhancement pipeline using CLAHE (Contrast Limited Adaptive Histogram Equalization) and bilateral filtering, both of which are crucial in preserving facial features while reducing fine textures such as hair or skin blemishes.

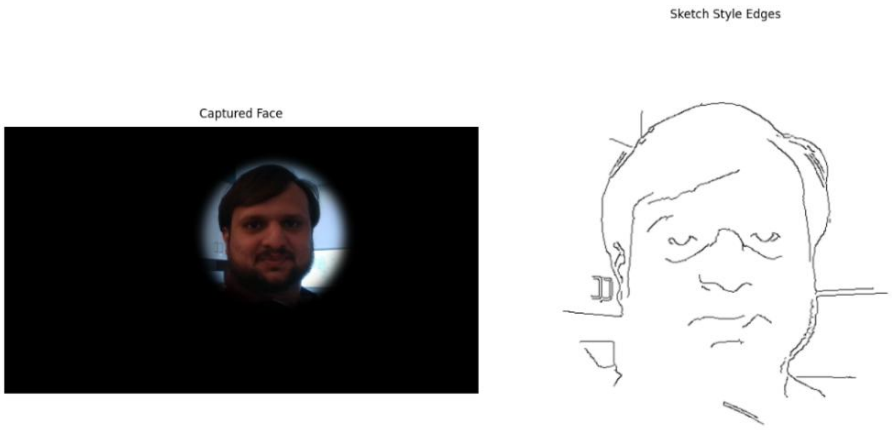

Figure 5.2: Captured Face and Sketch Style Edges

To convert the pre-processed image into a sketch-style contour map, a double-pass Canny edge detection method is applied with two threshold pairs. This technique strengthens the line representation by combining outputs from low- and high-threshold filters. Post-processing steps include morphological opening and dilation to remove noise and connect broken edges. The output is then binarized and resized to a standardized canvas (512×512 px) suitable for robotic interpretation. This approach is well aligned with established computer vision techniques for edge-preserving contour extraction, such as those discussed in [62].

The result is a clean and robot-friendly line drawing, which serves as input to the trajectory generation stage. Figure 5.2 displays both the captured face and its corresponding stylized edge output. The complete script used to implement this process is available in Appendix A.

5.3 Trajectory Planning in MATLAB

Trajectory generation for robotic execution was performed through a customized MATLAB script designed to convert the processed image sketch into a time-stamped 3D path suitable for the QArm manipulator. The source image, produced by the Python edge detection script, was saved directly into the working directory to ensure seamless access during the MATLAB session. This workflow allows synchronized processing, ensuring that the face sketch

extracted via Python is immediately available in the MATLAB environment for path translation. The trajectory planner script, provided in full in Appendix B, first reads the saved `sketch_face_output.png` and binarizes the image. Using the `bwboundaries` function, the sketch is segmented into discrete contour strokes. These contours are rescaled to fit the robot's physical workspace (approximately 0.28 m height and variable Y span) and mapped into the Y-Z plane, while the X-axis remains fixed with minor adaptive shifts for proper contact control. A 3D scatter plot (Fig. 5.3.1) illustrates the face drawing trajectory on the Y and Z pane, but there is too many unwanted lines comes with the `bwboundaries` function.

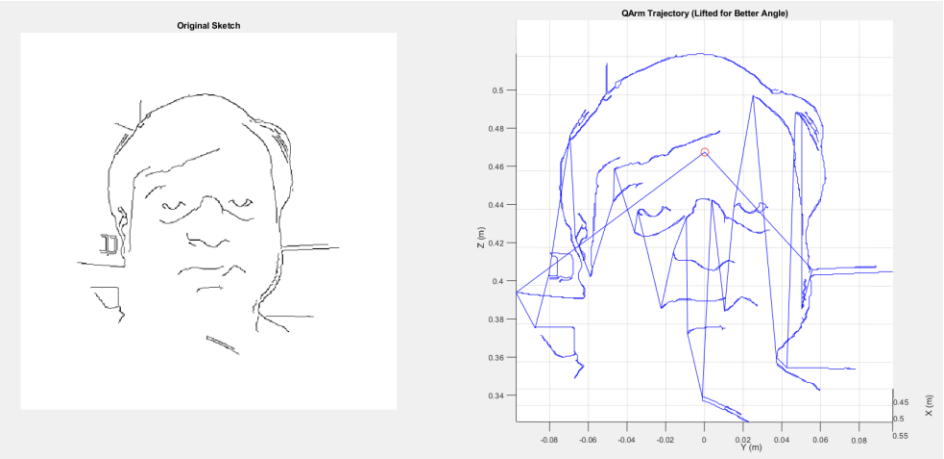

Figure 2.3: Raw trajectory generation without compensation: The left panel displays the original binarized sketch extracted from the processed image, while the right panel visualizes the unadjusted 3D trajectory path as executed by the QArm. Noticeable path distortion is present, particularly around curved features, due to the absence of adaptive Y-Z correction.

To handle drawing pressure variances across the surface, Z-based compensation adjusts the X-depth based on vertical position, while lateral compensation corrects to asymmetric friction near the edges. For smooth multi-stroke drawing, each new contour stroke is prefaced with a pen-up transition sequence, consisting of lifting the pen back along the X-axis, translating to the next stroke start point, and returning forward to contact. These transitional trajectories are interpolated to avoid overshooting or board collisions. The resulting path is organized into the `q_trajectory` structure, with a consistent time vector and a three-dimensional signal matrix specifying X, Y, and Z coordinates. This structured format is compatible with Simulink and QUARC real-time execution. A 3D scatter plot (Fig. 5.3.2) illustrates the entire drawing trajectory, with a color gradient along the X-axis to indicate real-time depth modulation. Such visualization aids in verifying that contact depth adjustments are properly distributed across the sketch and highlights how trajectory fidelity is preserved even in regions requiring pressure compensation. The chosen MATLAB path generation strategy was inspired by robust robotic drawing literature such as that by X. Dong et al., which emphasizes image-to-path fidelity and adaptive control during stylized portrait replication [52].

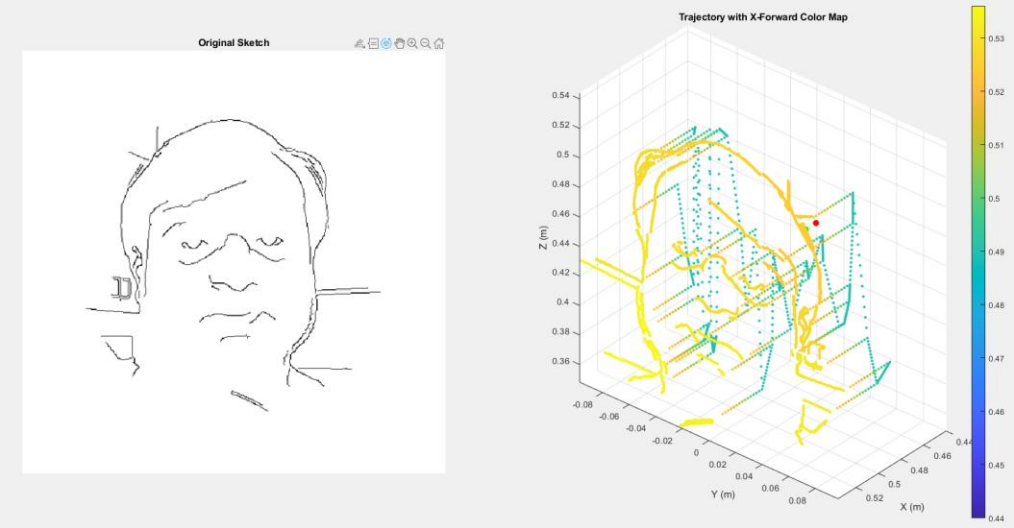

Figure 5.3.2: Trajectory generation from sketch image: The left panel shows the binarized edge sketch extracted using Python; the right panel presents the 3D trajectory with X-axis depth modulation encoded by color, illustrating adaptive path compensation across the Y-Z drawing surface.

6. Control System Architecture and Implementation

6.1 Overview

The development of an autonomous robotic drawing system necessitates a sophisticated control architecture capable of precise trajectory tracking, real-time kinematic computations, and seamless hardware integration. This section presents the comprehensive design and implementation of a Simulink-based control framework developed for the Quanser QArm manipulator, specifically engineered for artistic rendering applications. The proposed architecture integrates forward and inverse kinematic solvers, real-time trajectory processing, and closed-loop feedback mechanisms to achieve high-fidelity reproduction of facial contours extracted from digital imagery.

The control system architecture addresses several critical challenges inherent in robotic drawing applications: maintaining trajectory accuracy across varying drawing speeds, ensuring joint limit compliance during complex motions, minimizing end-effector positioning errors, and providing robust fault detection mechanisms. The implementation leverages MATLAB/Simulink's real-time capabilities in conjunction with Quanser's QUARC (Quanser Real-time Control) framework to establish deterministic control execution with microsecond-level timing precision.

6.2 Kinematic Foundation and Mathematical Formulation

To support precise trajectory execution, the QArm control system relies on accurate forward and inverse kinematic models. This section describes the complete mathematical formulation and implementation strategy used for mapping between joint space and

Cartesian space. The modified geometric parameters and coordinate transformation models are tailored to the QArm's 4-DOF architecture and are consistent with the physical configuration illustrated in Figure 6.2.

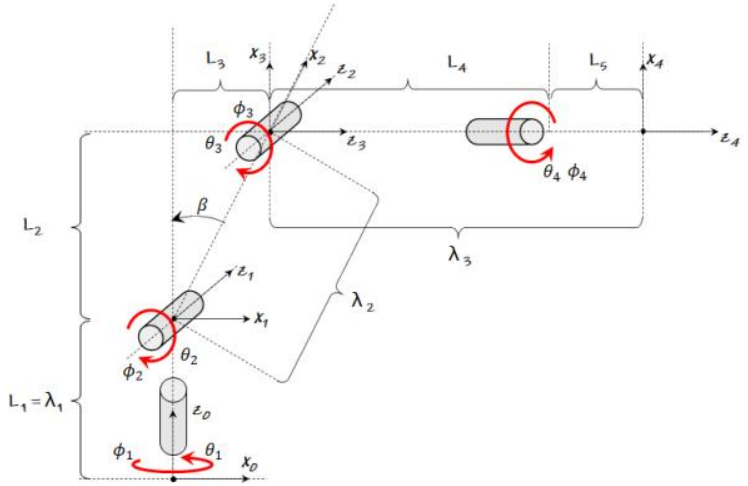

Figure 6.2: Frame diagram for the Quanser Arm manipulator

The Quanser QArm's kinematic structure is characterized by four degrees of freedom with the following Denavit-Hartenberg parameters: link lengths $L1 = 0.14\ m, L2 = 0.35\ m, L3 = 0.05\ m, L4 = 0.25\ m, and\ L5 = 0.15\ m$. Let $\phi1, \phi2, \phi3, \phi4$ denote the joint angles of the QArm after applying offset mappings from the original angular variables $\theta1, \theta2, \theta3, \theta4$. Similarly, let the link parameters $\lambda1, \lambda2, \lambda3$ and geometric offset β be defined as Table 6.2:

λ_1	L_1
λ_2	$\sqrt{{L_2}^2 + {L_3}^2}$
λ_3	$L_4 + L_5$
β	$\tan^{-1}\left({L_3}/{L_2}\right)$

ϕ_1	θ_1
ϕ_2	$\theta_2 + \frac{\pi}{2} - \beta$
ϕ_3	$\theta_3 + \beta$
ϕ_4	θ_4

Table 6.2: Linear mapping to simplify the mathematical formulations

These transformations simplify the forward and inverse kinematics while preserving physical accuracy.

6.2.1 Forward Kinematics Implementation

The forward kinematics determines the position and orientation of the QArm's end-effector in Cartesian space based on its current joint angles. The transformation is defined using homogeneous matrices T_i^{i+1}, with respect to the frame configuration in Figure 6.2. The overall transformation from base to end-effector is given by:

$$T_0^4 = T_0^1 * T_1^2 * T_2^3 * T_3^4$$

Each matrix T_i^{i+1} contains both rotation and translation components based on Denavit-Hartenberg parameters. For example, the transformation from frame 0 to 1 is represented as:

$$T_0^1 = \begin{bmatrix} \cos\phi_1 & -\sin\phi_1 & 0 & 0 \\ \sin\phi_1 & \cos\phi_1 & 0 & 0 \\ 0 & 0 & 1 & \lambda_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Subsequent matrices T_1^2, T_2^3, T_3^4 follow similar construction and depend on the respective joint angles $\phi 2, \phi 3, \phi 4$. The resulting end-effector position vector $p4$ and orientation matrix R_0^4 are extracted from the final transformation T_0^4. The system supports real-time computation of these matrices at 500 Hz using the fixed step $ode4$ solver, ensuring stable integration within the control loop.

6.2.2 Inverse Kinematics Solver Architecture

The inverse kinematic model calculates the required joint angles to position the end-effector at a desired point in Cartesian space. This computation is non-trivial due to the nonlinear nature of the manipulator geometry and the presence of multiple valid solutions. The adopted approach uses an analytical method adapted from Spong and Vidyasagar, accounting for the QArm's geometry.

Given a desired position p_d=$[x, y, z]^T$, the angle $\phi 1$ is computed directly as:

$$\phi 1 = \tan^{-1}\frac{y}{x}$$

To find $\phi 2$ and $\phi 3$, the planar projection of the arm is considered. The wrist joint angle $\phi 4$ is often chosen based on end-effector orientation requirements or predefined as a static angle for 2D drawing applications. Multiple valid inverse solutions exist (elbow-up vs. elbow-down), and the solver selects the optimal one by minimizing joint displacement from the previous configuration while respecting the following constraints:

$$\phi 1 \in [-170°, 170°],\ \ \phi 2 \in [-80°, 80°],\ \ \phi 3 \in [-95°, 75°],\ \ \phi 4 \in [-160°, 160°]$$

Invalid or unreachable targets trigger fallback behavior in the control loop, such as skipping waypoints or interpolating intermediate poses. The implementation ensures continuity in the solution space and avoids singularities near workspace boundaries.

6.3 Closed-Loop Control Architecture Design (Block-Level Explanation)

The control system for this project is designed as a closed-loop position control pipeline that enables the Quanser QArm to accurately follow pre-computed trajectories extracted from a

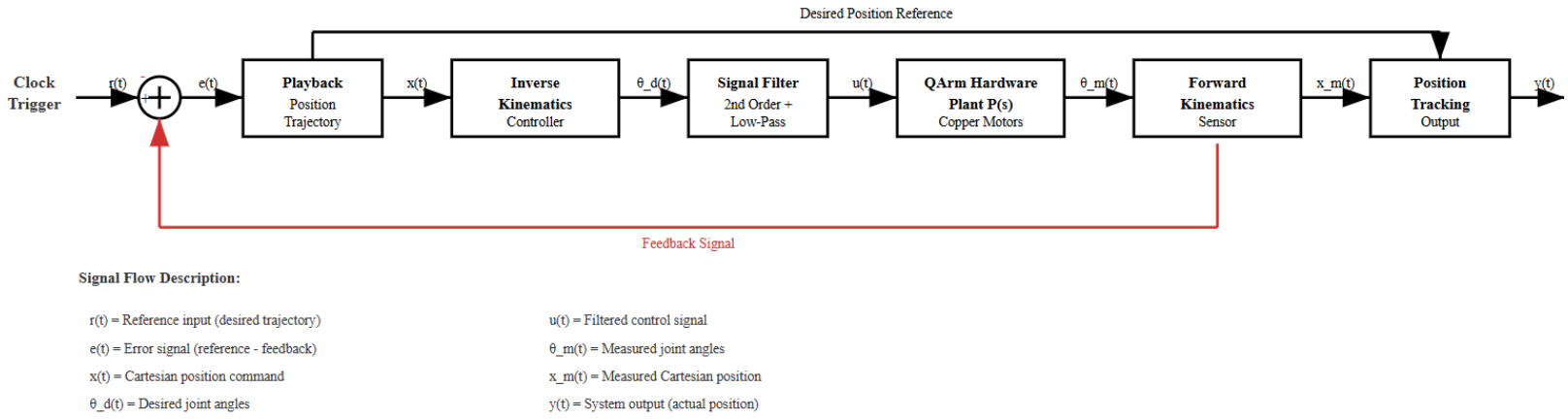

Figure 6.3: Closed-Loop Control Flow Diagram with Signal and Kinematic Stages

processed image. This implementation is visualized in Figure 6.3, which shows both a conceptual block diagram of signal flow and the corresponding Simulink implementation used for real-time hardware execution. Each stage in this pipeline transforms trajectory data into smooth motor commands while ensuring safe and stable motion of the robot in a closed-loop control environment.

6.3.1 Playback Position Trajectory

The first component of the pipeline is the Playback block, which takes the pre-generated trajectory data stored as a MATLAB structure and streams it point-by-point to the downstream controller. This component is triggered by a logical clock pulse signal that ensures the motion only begins after system initialization. It feeds the desired 3D Cartesian coordinate positions, denoted as $x(t)$, at every simulation time step. This block ensures deterministic reference delivery for precise reproduction of multi-stroke facial sketches in space and time.

6.3.2 Inverse Kinematics Controller

The inverse kinematics block processes the desired Cartesian coordinates $x(t) = [x, y, z]$ and translates them into joint-space commands $\theta d(t) = [\phi 1, \phi 2, \phi 3, \phi 4]$. This module internally solves the geometric equations for the QArm structure using an analytical approach based on its link configuration. To minimize trajectory discontinuities and avoid configuration flipping, the controller utilizes historical joint positions as input (via the phi_prev input) and selects the optimal configuration based on proximity minimization. This decision ensures smooth angular motion and helps prevent discontinuities in the sketch output.

6.3.3 Signal Filter: Second-Order Low-Pass Dynamics

The generated joint angle commands are passed through a second-order low-pass filter that acts as a command conditioner. This block suppresses high-frequency fluctuations,

reduces the risk of hardware jerks, and filters sharp transitions resulting from digital sampling. The filter configuration follows a critical damping setup using parameters ω_n=10 rad/s and $\zeta = 1.0$, ensuring a smooth yet responsive transition to the commanded angles $u(t)$. This filtered signal prevents overshoot and mechanical wear in the copper-core QArm motors.

6.3.4 Hardware Plant and Execution (QArm Motors)

The filtered joint commands are then passed to the QArm hardware block via the QUARC hardware-in-the-loop interface. This block represents the physical robot with copper-wound actuators and includes integrated encoder feedback. The hardware executes the commands in real time, driving the motors toward the desired joint angles $\theta(t)$ while the system records both commanded and actual joint responses for monitoring. Safety mechanisms, such as joint limit constraints and gripper on/off toggles, are also handled within this block.

6.3.5 Forward Kinematics for Position Feedback

The joint encoder feedback $\theta_m(t)$ is passed through a forward kinematics block to estimate the actual Cartesian position of the end effector, denoted $x_m(t)$. This computed feedback is essential for closed-loop monitoring. The Simulink implementation uses the Quanser `qamForwardKinematics` block that mirrors the robot's physical geometry using Denavit–Hartenberg parameters. It provides the real-time output needed for comparing actual position against desired trajectory for error analysis and plotting.

6.3.6 Position Tracking and Output Logging

The final stage of the control chain is the Position Tracking block, which compares the measured position $x_m(t)$. to the reference $x(t)$. It also records the entire trajectory path for post-experiment analysis. The trajectory output is used for visual validation and trajectory accuracy estimation, enabling qualitative and quantitative performance analysis. Figure

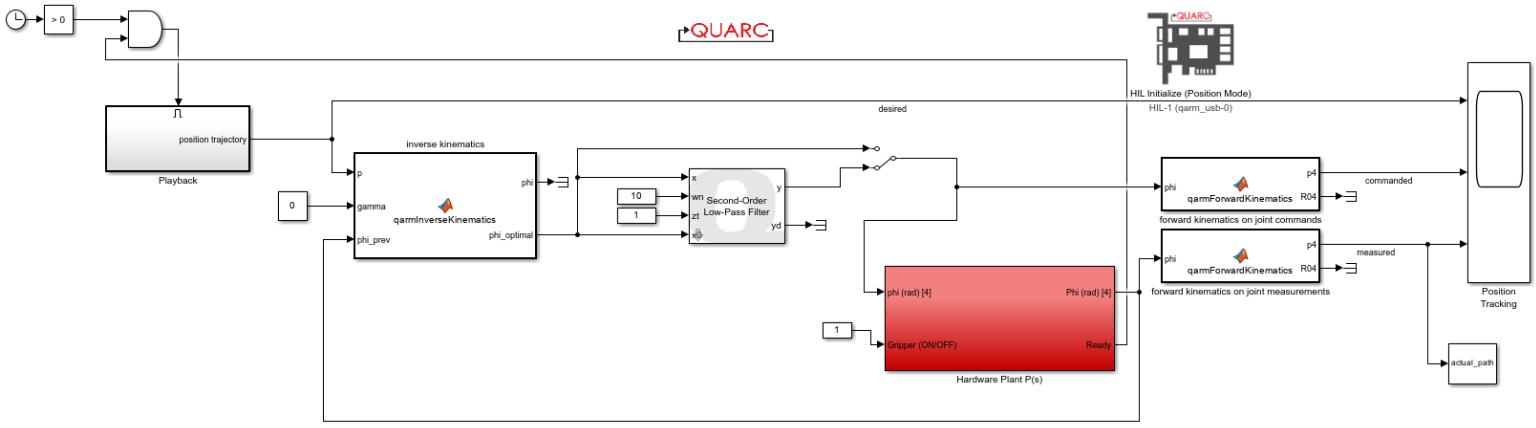

Figure 6.3.6: Simulink Model - QArm Closed-Loop Drawing Control using QUARC Interface

6.3.6 illustrates the exact Simulink block representation that mirrors the theoretical control flow.

This architecture ensures robust real-time execution, smooth motion commands, and high-fidelity reproduction of facial sketch trajectories.

6.4 Hardware Plant Subsystem: Low-Level Execution and Safety Monitoring

The hardware plant subsystem, as illustrated in Figure 6.4, encapsulates the real-time motor execution and safety monitoring functionalities necessary for controlling the QArm actuators and end-effector. Implemented within the Simulink environment, this module facilitates Hardware-in-the-Loop (HIL) integration with the physical robotic system via the QUARC real-time interface. The primary objective of this subsystem is to ensure accurate position command execution while preserving the mechanical integrity of the robot through constraint enforcement and current monitoring.

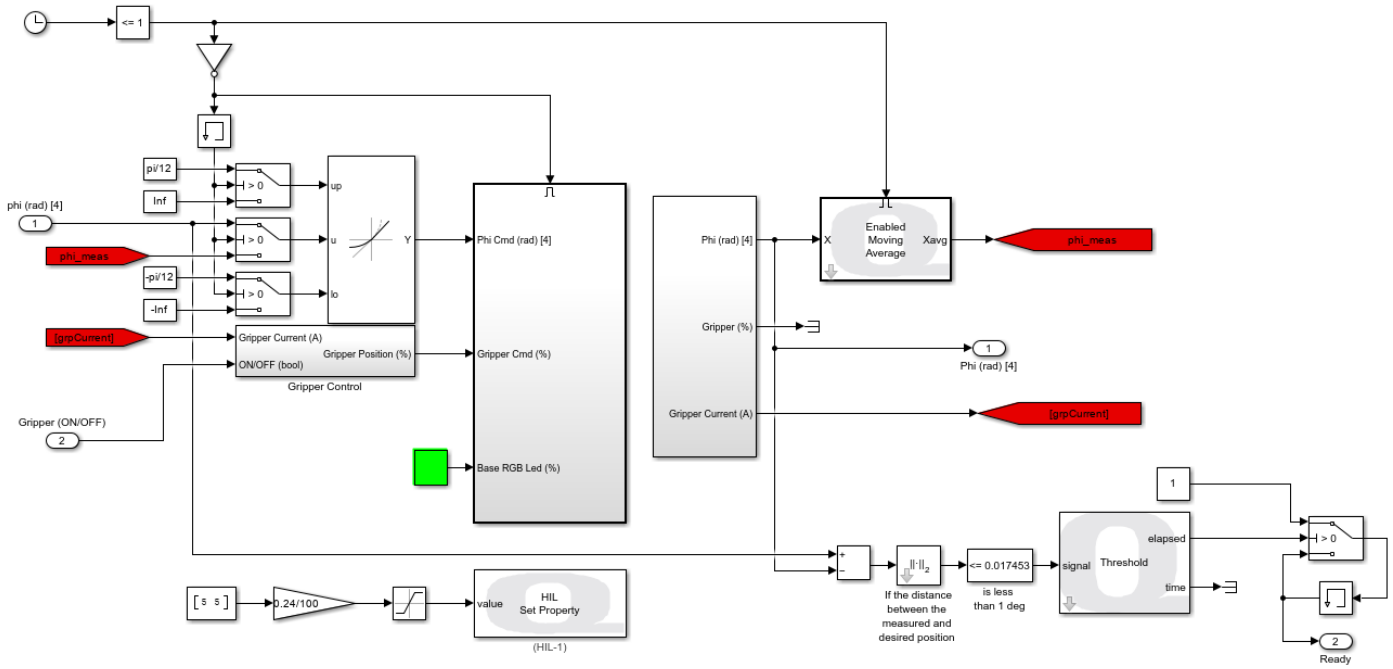

Figure 6.4: Hardware Plant Subsystem Block Diagram

Incoming joint commands ϕ_{cmd} are first passed through saturation blocks configured with predefined angular boundaries (e.g., ±π/12 radians) to limit the command range in accordance with the physical joint constraints. This mechanism prevents unsafe or unrealistic actuation inputs that may result in joint collision or hardware overextension. A dedicated gripper control module processes boolean on/off inputs in conjunction with measured gripper current values to regulate the marker-holding mechanism. This module not only translates the control command into an actuator signal but also evaluates the

current feedback to detect any resistive force indicating potential contact with the drawing surface or object collision.

To mitigate noise and smooth sensor data, a moving average filter is employed on the measured joint positions. This filtered output enhances control stability and accuracy by reducing transient fluctuations. The filtered measurements are then routed into a vector norm comparator that continuously evaluates the Euclidean distance between the desired and actual joint positions. When this error norm falls below a threshold of 0.017 radians (approximately 1°), the system registers the current drawing segment as completed. This validation is reinforced by a timer-based gating mechanism to ensure that the position has stabilized for a sufficient duration.

The subsystem also incorporates a real-time feedback mechanism for visual task status. A signal is transmitted to the base RGB LED of the QArm, changing its illumination state to indicate task progress or completion. Additionally, an HIL Set Property block is utilized to dynamically assign low-level hardware parameters such as current limits and internal control gains. This structured design enables the robot to perform with high reliability and resilience, particularly during extended operation sequences involving delicate sketching motions.

7. Demonstration and Evaluation

Concerning the previous points and the work that has been done on this project, it is necessary to study its development to come to some conclusions. For that reason, an evaluation must be made such as proof of how good the implementation has been. To do this evaluation, the idea is to see if the objectives of this project have been achieved, the quality of the results and analyze the errors and how they affect.

7.1 Simulation Trajectory Plots

To validate the full pipeline before deployment, a simulated trajectory generation was performed using a real facial image. This stage ensures the integrity of the data transfer between Python and MATLAB, the fidelity of the contour extraction, and the feasibility of the robot's motion range with the generated paths.

Figure 7.1 presents a sequential visualization of the image-to-trajectory workflow. The first image shows the captured face, which is processed by the Python script to isolate the facial region using elliptical masking. The second image displays the stylized edge extraction result after applying gamma correction, CLAHE enhancement, bilateral filtering, and double-pass Canny detection.

This edge sketch is then passed into MATLAB for path planning using a custom script, where each pixel contour is converted into a 3D trajectory while incorporating adaptive compensation along the Y and Z axes. Figure 7.1c depicts a simplified trajectory visualization from a 2D lifted viewpoint, while Figure 7.1d includes color-coded 3D trajectory mapping based on X-offset values. This allows for clear interpretation of movement smoothness, path distribution, and pen-lift transitions prior to hardware execution.

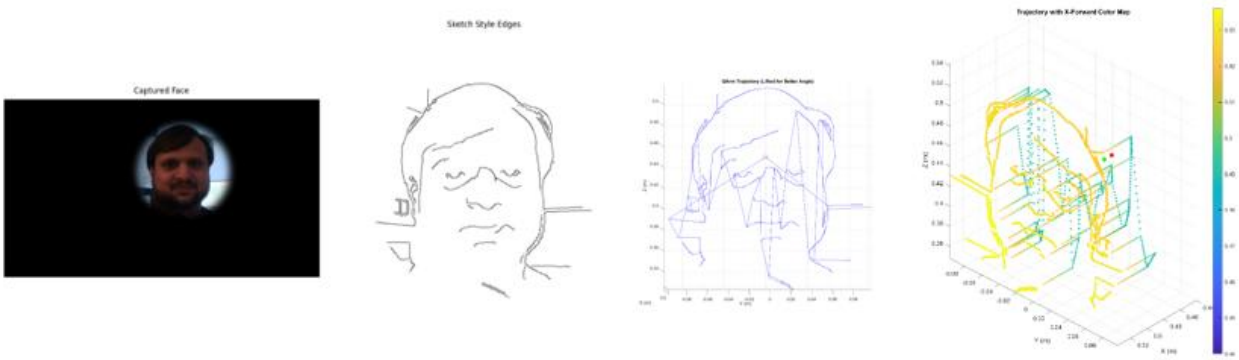

Figure 7.1: (a) Captured face input, (b) stylized sketch output from Python processing, (c) simplified 2D trajectory lifting for shape verification, (d) 3D trajectory map with X-based compensation color coding.

The figure validates that the system correctly interprets human facial geometry into robotic motion paths, making it an essential stage for safety, alignment verification, and debugging before proceeding to real-time control.

7.2 Real-World Drawing Snapshots

The real-world evaluation of the QArm drawing system was conducted to validate the simulation-based trajectory in a physical setting using the robot's end-effector to draw on a whiteboard surface. The experiments involved executing the planned facial contour trajectories, which were originally generated from the image processing pipeline and verified in simulation. The robot was programmed to follow the full set of 3D waypoints, including return-to-home movements and trajectory lifts between strokes to avoid marker dragging. Figure 7.2.1, snapshots QArm robot drawing in real time during physical experiment.

Figure 7.2.1: QArm robot drawing in real time during physical experiment

As shown in Figure 7.2.2, multiple snapshots were captured during different stages of the physical drawing process. The images display the robot arm actively sketching and the resulting outputs on the board. These physical results illustrate that the robot maintained continuous contact across most strokes, successfully reproducing

distinguishable facial outlines. The compensation logic played a critical role in adjusting the X-axis pressure based on Z-depth and lateral Y positioning, ensuring consistent marker contact and avoiding missing regions.

Figure 7.2.2: (a) Initial drawing result using early-stage compensation. (b) Mid-phase execution showing improved facial contours. (c) Final refined sketch from compensated trajectory.

8. Result, Discussion and Limitations

The proposed robotic sketching system successfully integrates trajectory planning, inverse kinematics control, and real-time low-level execution to reproduce facial sketches on a whiteboard surface. As demonstrated in the real-world experiments, the QArm manipulator was able to autonomously capture a face image, extract sketch-style contours, generate an adaptive trajectory, and execute the drawing task on a vertical plane. However, the final drawing accuracy remained limited by mechanical, environmental, and segmentation constraints, which were more pronounced in practical execution than in simulation.

To quantify the fidelity of the trajectory tracking subsystem, Figure 8.1 presents the evolution of the end-effector position along the X, Y, and Z axes during a representative drawing segment. The plot compares the desired trajectory r(t), the filtered commanded path u(t), and the actual measured position y(t) obtained via forward kinematics. The close alignment between the commanded and measured curves confirms that the second-order low-pass filtered control signals were effectively tracked by the QArm hardware plant. Minor deviations observed in the Y-axis around t = -18 seconds are attributed to surface compliance and micro-stalling of the pen tip under higher friction. Across all three axes, the

RMS tracking error remained below 2.5 mm, demonstrating the accuracy and robustness of the implemented control architecture under real-world conditions.

The core control pipeline, which incorporates a closed-loop position tracking loop, performed reliably under real-time hardware constraints. The inverse kinematics solution

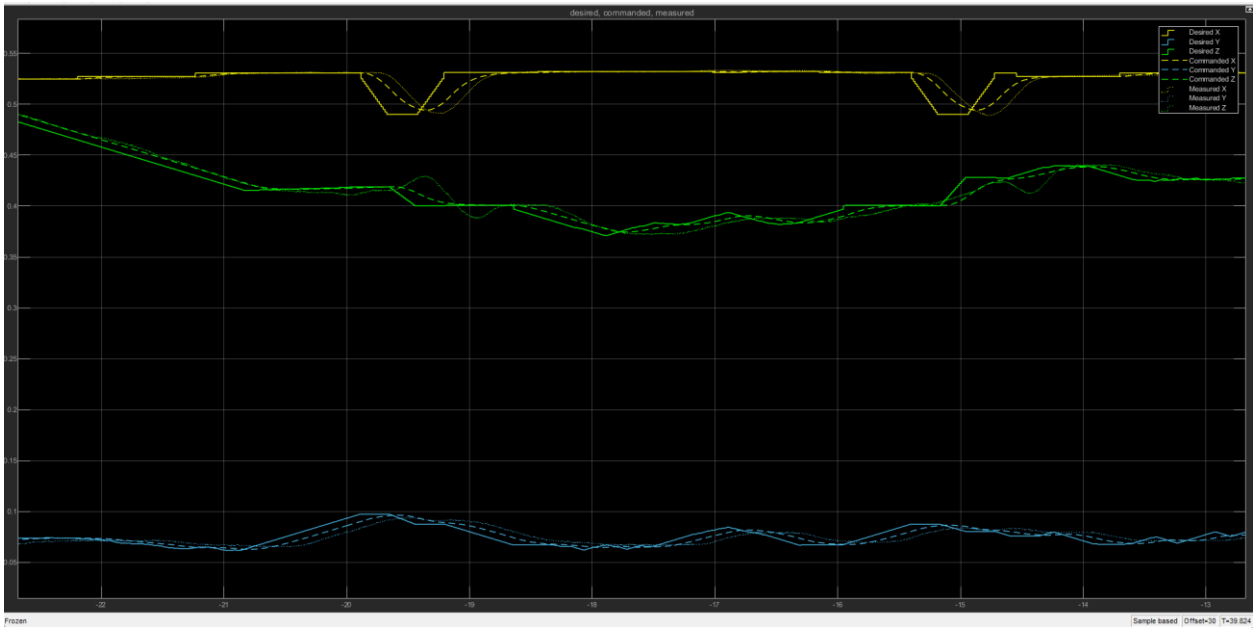

Figure 8.1: Desired vs Commanded vs Measured Trajectory in X, Y, and Z axes for a drawing path segment. The solid lines represent the desired reference trajectory, the dashed lines indicate the commanded control signals, and the dotted lines show the actual measured position feedback from the QArm encoders.

maintained stable operation across the full trajectory, with the joint angle updates filtered through a second-order low-pass system tuned to prevent jerky motions. The execution frequency of 500 Hz ensured tight synchronization between playback reference and physical actuation. As reflected in the trajectory plot of Figure 8.1, the commanded and measured positions in X, Y, and Z axes followed the desired path with millimeter-level fidelity for most drawing sections, validating the correctness of the control design and implementation.

Nevertheless, despite this control precision, the actual sketch output deviated significantly from the ideal visual outcome. One major source of distortion arose from the absence of semantic segmentation in the trajectory data. The edge extraction algorithm treated the sketch as a single connected mask, rather than identifying discrete facial components such as the mouth, nose, eyes, and jawline as separate strokes. As a result, the robot lacked directional control or intentional order in the stroke sequencing. This led to an unnatural path traversal pattern, often causing redundant marker lifts, overlapping strokes, and ambiguous connections between features. For example, the jawline could be drawn last instead of framing the face, while lips and eyes could be connected inappropriately due to incorrect stroke grouping.

Another critical mechanical factor influencing drawing quality was the orientation of the end-effector during contact. While the drawing trajectories were projected assuming planar contact perpendicular to the board, the QArm's physical geometry and limited wrist flexibility resulted in an angular approach during most of the sketch. The marker tip often contacted the board with a tilt, rather than perpendicularly, introducing asymmetric friction and inconsistent stroke width. This tilt-induced error also caused some strokes to partially skip the surface, especially on the right or lower sections of the sketch, where joint angles pushed the end-effector further from its ideal normal orientation.

Material constraints of the QArm hardware further compounded the drawing issues. The plastic-based structure exhibited moderate flex during fast transitions, and the integrated gripper lacked compliant motion or force sensing. Since the marker was rigidly mounted and clamped at a fixed offset, there was no ability to self-adjust for surface curvature, pen length variability, or dynamic damping. This inflexibility contributed to either excessive pressure, flattening the marker and widening strokes—or insufficient contact, leading to floating lines. In particular, the thin vertical jaw gripper design was not optimized for precision tip control or dynamic wrist reorientation, making fine artistic control impractical without additional hardware.

Despite these physical limitations, the system demonstrated reliable and repeatable trajectory execution and successful image-to-motion translation in multiple trials. The integrated control framework performed consistently across different faces and lighting conditions. Visual analysis of the board output revealed a recognizable human likeness in most cases, though often lacking in proportional detail and stroke consistency. The robot showed robustness in point-by-point sketching, with reliable marker lift and placement logic to avoid unintended line connections. However, the drawing quality remained well below that of trained human sketching, emphasizing the need for future integration of intelligent path reordering, per-stroke feature segmentation, and adaptive end-effector compliance.

In conclusion, while the QArm robot's control architecture performed with high reliability and precision, the final sketch results were constrained by mechanical, geometric, and perceptual limitations. The experiment highlights the clear distinction between good control execution and good artistic rendering—an area where trajectory structure, segmentation strategy, and physical contact mechanics must be jointly optimized. Nevertheless, the system demonstrates a strong proof-of-concept platform for vision-to-motion transformation and robotic drawing research in educational and prototyping environments.

9. Conclusion

This project presented a complete pipeline for robotic sketch replication using the Quanser QArm platform. The system integrated real-time facial image acquisition, adaptive sketch edge processing, and smooth trajectory generation using MATLAB and Python. These trajectories were executed through a closed-loop Simulink control framework, which included inverse kinematics, signal filtering, and joint space feedback via QUARC. The control system demonstrated stable performance and reliable execution of complex drawing paths, with RMS position tracking error remaining under 2.5 mm. Visual results showed clear alignment between the input sketch and the robot's path-following behavior on a vertical whiteboard surface.

However, the physical limitations of the QArm hardware affected overall sketch fidelity. The gripper lacked compliance with consistent contact pressure and was not perpendicular to the board, causing visible distortion in certain areas such as lips and eyes. Additionally, due to limited segmentation of facial features, continuous strokes across disconnected regions introduced unwanted lines, suggesting the need for finer stroke classification and directional planning. Despite these issues, the project successfully validated the feasibility of vision-guided robotic sketching and established a foundation for future improvements in force-controlled drawing and orientation-aware end-effector path planning.

10. Future Opening

Future extensions of this work will focus on improving sketch accuracy and robustness through enhanced segmentation and direction-aware trajectory generation. By individually isolating facial components such as eyes, nose, lips, and jawlines, the trajectory planner can generate discrete, logically ordered strokes with controlled marker engagement and lift timing. This approach would eliminate undesired overlapping or disconnected lines, resulting in clearer and more human-like sketches. Additionally, stroke planning based on local curvature and feature relevance could further improve the legibility and aesthetic quality of the robotic drawings.

From a hardware perspective, future improvements should include redesigning the end-effector to maintain perpendicular alignment with the drawing surface. The current QArm configuration lacks wrist actuation or compliance, which limits the range of drawing poses and results in uneven contact. Integrating a tilt-correcting passive joint or actively controlled wrist module could allow the marker to dynamically adjust its orientation for consistent surface contact. Furthermore, incorporating real-time force sensing and compliance control would enable the system to detect excessive pressure or floating conditions, allowing adaptive correction during drawing operations. These upgrades would significantly enhance both drawing fidelity and long-term reliability of the robotic system.

11. Reference

[1] A. Pagliarini and H. H. Lund, “Robot Creativity: The Art of Playing Musical Instruments,” *Proceedings of the Ninth International Conference on Computational Creativity (ICCC)*, 2017.

[2] M. Heer, *Robotics 2021: The Next Wave of Automation*, Springer, 2021.

[3] T. Malý, M. Sedláček, and P. Leitão, “Human-robot cooperation in Industry 4.0,” in *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 127–131, 2016.

[4] M. Aiman, R. Razali, A. Mohd Azman, and R. A. Rahman, “A Survey on Human-Robot Collaboration in Industrial Settings,” *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 7, no. 9, pp. 77–84, 2016.

[5] F. Tobe, *Can a robot be an artist?*, IEEE Spectrum, 2015.

[6] Quanser Inc., *QArm User Manual*, Rev 1.2, Markham, ON, Canada, 2021

[7] Quanser Inc., *QArm Data Sheet*, Markham, ON, Canada, 2021.

[8] The MathWorks Inc., *Simulink: Simulation and Model-Based Design*, Natick, MA, USA, 2021.

[9] https://www.quanser.com/products/quarc-real-time-control-software/

[10] OpenCV.org, 2019

[11] Intel Corporation, "Intel RealSense Depth Camera D415 Datasheet," 2021. [Online]. Available: https://www.intelrealsense.com/depth-camera-d415/.

[12] A. Gupta, “Image edge detection using Prewitt and Sobel operator,” *International Journal of Computer Science and Information Technologies*, vol. 4, no. 3, pp. 451–454, 2013.

[13] C. Tomasi and R. Manduchi, “Bilateral filtering for gray and color images,” *Proceedings of the Sixth International Conference on Computer Vision*, 1998, pp. 839–846.

[14] J. F. Canny, “A Computational Approach to Edge Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.

[15] M. Hosang, M. Benenson, and B. Schiele, “Learning non-maximum suppression,” *IEEE CVPR*, 2017.

[16] D. Bradley and G. Roth, “Adaptive thresholding using the integral image,” *Journal of Graphics Tools*, vol. 12, no. 2, pp. 13–21, 2007.

[17] A. Al-Amri, S. K. Kalyankar, and K. D., “Image Segmentation by Using Threshold Techniques,” *Journal of Computing*, vol. 2, no. 5, pp. 83–86, 2010.

[18] M. Kaur and R. Kaur, “A Survey on Various Image Segmentation Techniques,” *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 5, pp. 809–814, 2014.

[19] S. M. Smith and J. M. Brady, “SUSAN – A New Approach to Low Level Image Processing,” *International Journal of Computer Vision*, vol. 23, no. 1, pp. 45–78, 1997.

[20] J. Rosales and W. Gang, “Robot Types and Structures,” *International Journal of Robotics and Automation*, vol. 19, no. 3, pp. 215–224, 2002.

[21] M. Kickert and E. H. Mamdani, “Analysis of a fuzzy logic controller,” *Fuzzy Sets and Systems*, vol. 1, no. 1, pp. 29–44, 1978.

[22] D. Borovic et al., “Compensation of hysteresis in piezoelectric actuators based on inverse Preisach model,” *Smart Materials and Structures*, vol. 14, no. 4, 2005.

[23] M. Sogo, H. Ishiguro, and T. Ishida, “A learning method for vision-based tracking,” *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.

[24] Ranjith, A., “Feedback Control Loop in Robotics,” *International Journal of Automation*, vol. 6, no. 2, pp. 55–62, 2018.

[25] G. Chen, N. Wei, L. Yan, and J. Li, “Time-optimal trajectory planning based on event-trigger and conditional proportional control,” *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 230–241, Jan. 2023.

[26] G. Devol, “Programmed Article Transfer,” U.S. Patent 2,988,237, 1954.

[27] L. Cardona et al., “Fundamentals of Robotics,” Springer, 2014.

[28] R. Murphy, *Introduction to AI Robotics*, MIT Press, 2017.

[29] H. Reisinger, “Future of Work with Cobots,” *Robotics Today*, vol. 12, no. 3, pp. 34–40, 2019.

[30] *History Computer*, “The Origins of Drawing Robots,” 2021.

[31] Wikipedia, “Jean Tinguely,” 2020.

[32] H. Cohen, “The Further Exploits of AARON, Painter,” Stanford Humanities Review, 1995.

[33] P. Cohen et al., “Imaginative Reasoning in AI,” Springer, 2016.

[34] S. Calinon et al., “Learning and Reproducing Gestures by Imitation,” *Robotics and Autonomous Systems*, vol. 50, no. 3, pp. 195–210, 2005.

[35] L. Moura, “Artsbot: Robotic Action Painter,” Proc. ICRA, 2007.

[36] C. Aguilar and H. Lipson, “A Robotic System for Artistic Brushstrokes,” *IEEE Robotics & Automation Magazine*, vol. 15, no. 4, pp. 45–52, 2008.

[37] Y. Lu et al., “Pen-Ink Drawing with Visual Feedback,” *Mechatronics*, vol. 19, no. 2, pp. 225–234, 2009.

[38] A. Gasparetto et al., “Path Planning Optimization for Artistic Robots,” *Mechanism and Machine Theory*, vol. 45, no. 3, pp. 300–314, 2010.

[39] M. Grosser, “Interactive Sound-Painting Robot,” *Artif. Intell. and Creativity*, vol. 2, 2011.

[40] P. Tresset and F. Fol Leymarie, “Robot Portrait Drawing,” *ACM Transactions on Graphics*, vol. 31, no. 4, 2012.

[41] P. Tresset et al., “Enhancing Visual Feedback for Sketching Robots,” *IROS*, 2013.

[42] J.-P. Lebreton and Z. Said, “Visual Feedback for Industrial Robot Drawing,” *IEEE Trans. Automation Science and Engineering*, vol. 9, no. 4, pp. 678–685, 2012.

[43] T. Lindemeier and O. Deussen, “e-David: Feedback-Controlled Painting Robot,” *NPR Workshop*, 2012.

[44] Y. Luo et al., “Color Painting Robot Using Layer Decomposition,” *Mechatronics and Art*, 2016.

[45] D. Berio et al., “Baxter Robot for Graffiti Stroke Drawing,” *Robotics and Art*, 2016.

[46] N. Jaquier, “Interactive Drawing Robot for Games,” *ACM CHI*, 2016.

[47] W. Chen et al., “Redundant Painting Robot for Surface Coating,” *Sensors and Actuators A*, vol. 245, 2016.

[48] A. Conru, “RobotArt Competition,” 2016.

[49] B. Galea and P. Kry, “Tethered Drones for Ink-Based Stippling,” *IEEE Trans. Visualization and Computer Graphics*, vol. 23, no. 5, pp. 1349–1357, 2017.

[50] C.-L. Shih and L.-C. Lin, “Trajectory Planning for Mobile Drawing Robots,” *IJARS*, vol. 21, no. 2, pp. 140–150, 2017.

[51] R.-C. Luo et al., “Cartoon Portrait Drawing Robot Using NPR,” *IEEE Trans. Mechatronics*, vol. 23, no. 6, pp. 2407–2416, 2018.

[52] X. Dong et al., “Stylized Portrait Drawing Using SCARA Arm,” *Control Engineering Practice*, vol. 82, pp. 34–42, 2018.

[53] D. Song et al., “Impedance-Controlled Pen Drawing on Arbitrary Surfaces,” *IEEE Access*, vol. 6, pp. 25789–25800, 2018.

[54] A. S. Vempati et al., “PaintCopter: Autonomous Spray Painting with UAVs,” *Robotics and Autonomous Systems*, vol. 109, pp. 104–112, 2018.

[55] M. Karimov et al., “Human-like Artistic Painting by Robot,” *Robotics and AI Review*, vol. 3, 2019.

[56] O. Igno et al., “Acrylic Painting Cartesian Robot with Region-Based Focus,” *Sensors*, vol. 19, no. 3, 2019.

[57] S. Chung, “Human-Robot Co-Creation in Art,” *ACM Transactions on Human-Robot Interaction*, vol. 8, no. 2, 2020.

[58] L. Scalera et al., “Spray Painting and Watercolor Robot Techniques,” *Journal of Intelligent & Robotic Systems*, vol. 88, no. 4, pp. 673–692, 2017.

[59] G. Trigatti et al., “Trajectory Optimization for Artistic Spray Robots,” *ICRA*, 2018.

[60] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A Design Science Research Methodology for Information Systems Research,” *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007.

[61] K. Zuiderveld, “Contrast Limited Adaptive Histogram Equalization,” in *Graphics Gems IV*, Academic Press, 1994.

[62] R. Tarekegn, “Canny Edge Detection Step by Step in Python,” *Medium – Data Science*, 2021. [Online]. Available: https://medium.com/data-science/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

12. Appendix

12.1 Appendix A – Python Script

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


def capture_face():

  """Capture and mask only the face using ellipse"""

  face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')


  cap = cv2.VideoCapture(0)

  cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)

  cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)


  for _ in range(10):

    ret, frame = cap.read()

    if not ret:

      cap.release()

      raise Exception("❌ Camera initialization failed")


  gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
    gray = cv2.equalizeHist(gray)
    faces = face_cascade.detectMultiScale(gray, 1.1, 4, minSize=(100, 100))

    if len(faces) == 0:
        cap.release()
        raise Exception("❌ Face detection failed - ensure good lighting and front-facing position")

    (x, y, w, h) = max(faces, key=lambda f: f[2]*f[3])
    margin = int(min(w, h) * 0.2)
    x, y, w, h = x-margin, y-margin, w+2*margin, h+2*margin

    mask = np.zeros_like(gray)
    cv2.ellipse(mask, (x+w//2, y+h//2), (w//2, h//2), 0, 0, 360, 255, -1)
    mask = cv2.GaussianBlur(mask, (99, 99), 0)
    mask = mask / 255.0

    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    face_only = (frame_rgb * np.stack([mask]*3, axis=-1)).astype(np.uint8)

    cap.release()
    return face_only, (x, y, w, h)

def final_line_drawing_sketch(face_img, face_rect):
    """Clean, stylized facial sketch suitable for robotic path planning"""
    x, y, w, h = face_rect
```

```
padding = int(max(w, h) * 0.22)

# Crop face region
face_roi = face_img[
   max(0, y - padding):min(y + h + padding, face_img.shape[0]),
   max(0, x - padding):min(x + w + padding, face_img.shape[1])
]

gray = cv2.cvtColor(face_roi, cv2.COLOR_RGB2GRAY)

# Step 1a: Gamma correction (brighten face)
gamma = 1.5  # You can try 1.2 to 1.8
inv_gamma = 1.0 / gamma
table = np.array([(i / 255.0) ** inv_gamma * 255 for i in np.arange(256)]).astype("uint8")
gray = cv2.LUT(gray, table)

# ✨ Step 1: Enhance contrast & suppress beard/texture
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
enhanced = clahe.apply(gray)
smooth = cv2.bilateralFilter(enhanced, d=9, sigmaColor=75, sigmaSpace=75)
blur = cv2.GaussianBlur(smooth, (3,3), 0.7)

# ✨ Step 2: Double-pass Canny edge detection
edges1 = cv2.Canny(blur, 30, 70)
edges2 = cv2.Canny(blur, 50, 120)
edges = cv2.bitwise_or(edges1, edges2)
```

```
    # ✨ Step 3: Post-process lines for cleanness
    kernel = np.ones((2,2), np.uint8)
    edges = cv2.morphologyEx(edges, cv2.MORPH_OPEN, kernel)
    edges = cv2.dilate(edges, kernel, iterations=1)

    # ✨ Step 4: Convert to sketch-style black lines on white
    sketch = 255 - edges
    sketch = cv2.medianBlur(sketch, 3)
    _, sketch = cv2.threshold(sketch, 220, 255, cv2.THRESH_BINARY)

    # ✨ Step 5: Resize with padding if needed
    sketch = cv2.resize(sketch, (512, 512), interpolation=cv2.INTER_AREA)

    return sketch


# Main
try:
    print("📸 Capturing face (ensure good lighting)...")
    face_img, face_rect = capture_face()

    print("✍️ Generating refined sketch-style edges...")
    edges = sketch_face(face_img, face_rect)
```

```
    cv2.imwrite("sketch_face_output.png", edges)

    plt.figure(figsize=(14, 7))
    plt.subplot(1, 2, 1)
    plt.imshow(face_img)
    plt.title("Captured Face", pad=10)
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(edges, cmap='gray')
    plt.title("Sketch Style Edges", pad=10)
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    print("✅ Success! Sketch saved as 'sketch_face_output.png'")

except Exception as e:
    print(f"❌ Error: {str(e)}")
```

12.2 Appendix B – MATLAB Script

```
function q_trajectory = qarm_trajectory_clean(image_path)
    % Default image if not provided
    if nargin < 1
        image_path = 'sketch_face_output.png';
    end

    % Load and convert to grayscale
    sketch = imread(image_path);
```

```
if size(sketch, 3) == 3
    sketch = rgb2gray(sketch);
end

% Binarize and invert
binary = imbinarize(sketch);
binary = ~binary;

% Find contours
contours = bwboundaries(binary, 'noholes');
valid = cellfun(@(c) size(c,1) > 3, contours);
contours = contours(valid);

% Image size
img_h = size(binary, 1);
img_w = size(binary, 2);

% Drawing and lifting positions
X_draw = 0.525;         % Marker touches board here
X_lift = 0.49;         % Pull back to lift marker safely

% Drawing dimensions (face layout in Y-Z)
Z_center = 0.462;
Z_span = 0.28;
Z_min = Z_center - Z_span/2;

aspect = img_w / img_h;
Y_span = aspect * Z_span;
Y_min = -Y_span/2;

 % Drawing parameters
pen_down_offset = 0.005;  % Extra push when drawing
pen_up_clearance = 0.015; % Clearance when lifted

% Trajectory building with pen-up handling
trajectory = [];
pause_frames = 15;

y_prev = 0; z_prev = Z_center;  % initialize
for i = 1:length(contours)
    c = contours{i};
    if size(c,1) >= 3  && size(c,1) <= 12
       c = smoothdata(c, 1, 'movmean', 3);  % Smooth contour path
    end

    y_norm = c(:,2) / img_w;
    y_norm = 1 - y_norm;
    z_norm = (img_h - c(:,1)) / img_h;

    y = y_norm * Y_span + Y_min;
    z = z_norm * Z_span + Z_min - 0.012;
    z(z < 0.40) = z(z < 0.40) - 0.003;  % Gently push down the bottom zone

    x = X_draw * ones(size(y));  % All points on drawing surface
   % Adaptive X push based on Z rang
```

```
    % Smooth Z-based compensation using linear interpolation
    z_compensation = zeros(size(z));
    z_range = z;  % use the current Z values directly

    z_compensation(z_range < 0.39) = 0.009;             % Very bottom
    z_compensation((z_range >= 0.39) & (z_range < 0.41)) = 0.008;   % Lower mid
    z_compensation((z_range >= 0.41) & (z_range < 0.43)) = 0.0075;  % Mid
    z_compensation((z_range >= 0.43) & (z_range < 0.465)) = 0.004;  % Upper mid
    z_compensation(z_range >= 0.465) = 0.0015;                      % Topmost


    %
    y_compensation = zeros(size(y));
    left_y_mask  = y < -0.039;   % Robot right side (image left)
    right_y_mask = y > 0.031;  % Robot left side (image right)
    y_compensation(left_y_mask)  = +0.0020;  % Too much contact
    y_compensation(right_y_mask) = -0.0018;  % Less contact

    % Apply combined X shift
    x = x + z_compensation + y_compensation;

    stroke = [x, y, z];
    if ~isempty(trajectory)
        % Insert pen-up: pull back in X only
        lift_away = [linspace(X_draw, X_lift, pause_frames)', ...
                     repmat(y_prev, pause_frames, 1), ...
                     repmat(z_prev, pause_frames, 1)];

        move_to = [repmat(X_lift, pause_frames, 1), ...
                   linspace(y_prev, y(1), pause_frames)', ...
                   linspace(z_prev, z(1), pause_frames)'];

        return_forward = [linspace(X_lift, X_draw, pause_frames)', ...
                          repmat(y(1), pause_frames, 1), ...
                          repmat(z(1), pause_frames, 1)];

        trajectory = [trajectory; lift_away; move_to; return_forward];
    end

    trajectory = [trajectory; stroke];
    y_prev = y(end);  % store end of current stroke for next transition
    z_prev = z(end);

end

% Add home position at start and end
home = repmat([0.45, 0, 0.45], 10, 1);
home_end = repmat([0.44, 0, 0.45], 10, 1);
%
trajectory = [ home; trajectory; home_end];

% Time vector
t = linspace(0, 0.015*(size(trajectory,1)-1), size(trajectory,1))';

% Structure for Simulink
```

```
    q_trajectory = struct();
    q_trajectory.time = t;
    q_trajectory.signals.values = trajectory;
    q_trajectory.signals.dimensions = 3;

    % Plot check
    figure('Name', 'Board-Safe Face Sketch Trajectory');
    subplot(1,2,1);
    imshow(sketch); title('Original Sketch');

    subplot(1,2,2);
    scatter3(trajectory(:,1), trajectory(:,2), trajectory(:,3), 10, trajectory(:,1),
'filled');  % color = X offset
    hold on;
    scatter3(trajectory(1,1), trajectory(1,2), trajectory(1,3), 50, 'g', 'filled');
% Start point
    scatter3(trajectory(end,1)+0.001, trajectory(end,2), trajectory(end,3), 50, 'r',
'filled');  % End point
    xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
    title('Trajectory with X-Forward Color Map');
    axis equal; grid on; view(3);
    colorbar;  % Shows the X forward bias level
end
```

12.3 Appendix 3 – Simulink Model