Real-World AI Artist

1st Hao-Yu Liu
Department of Mechanical Engineering
National Taiwan University
Taipei, Taiwan
r12522839@ntu.edu.tw

2nd Shao-Yang Chang
Department of Biomechatronic Engineering
National Taiwan University
Taipei, Taiwan
r13631040@ntu.edu.tw

3rd Chi-Han Tsai
Department of CSIE
National Taiwan University
Taipei, Taiwan
r13922073@ntu.edu.tw

4th Ding-Shan Chen
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
b10901046@ntu.edu.tw

5th Sin-Ruei Yang
Department of CSIE
National Taiwan University
Taipei, Taiwan
b10902002@ntu.edu.tw

***Abstract*—The integration of artificial intelligence (AI) and robotics has unlocked transformative possibilities in diverse fields, including creative applications such as art and design. This study presents the development of an intelligent robotic system capable of interpreting human voice commands and autonomously producing visually interpretable artworks. A critical aspect of this work is the incorporation of novel methods for canvas localization, speech-to-text conversion, image generation and simplification, and trajectory optimization.**

Keywords—Robotic art, Robot Arm, Generative AI, Path Planning

I. Introduction

The rapid advancement of artificial intelligence (AI) and robotics has revolutionized industries ranging from manufacturing to healthcare, enabling applications such as automated assembly lines, medical surgery assistants, and education tools. Among these, the integration of AI-driven natural language processing (NLP) with robotic manipulation represents a significant leap forward, bridging the gap between human intent and machine action. This research focuses on developing a robotic arm system capable of interpreting human voice commands and autonomously creating artistic works.

A critical challenge in achieving this vision lies in optimizing the robotic arm's ability to perform artistic tasks with precision and efficiency. Traditional robotic drawing systems often rely on predefined patterns or require substantial human intervention to generate paths, limiting their adaptability to dynamic, user-specific inputs. Moreover, challenges such as accurate canvas localization, effective image simplification, and seamless trajectory optimization have hindered the widespread adoption of such systems. This project addresses these gaps by integrating advanced vision-based canvas detection, refined text-to-image generation models, and techniques for converting visual elements into smooth painting trajectories. These innovations aim to enhance both the efficiency of robotic drawing and the quality of the produced artwork.

As robots become increasingly integrated into daily life, the demand for intuitive and accessible human-robot interaction grows. Voice-controlled systems, in particular, offer a natural interface, enabling non-technical users to interact with complex robotic systems. When combined with AI-driven generative models, this interface opens up new opportunities in creative domains such as art and design, where human intuition and robotic precision converge. This study builds upon these advancements to develop an end-to-end system capable of understanding natural language commands and producing visually interpretable drawings in a fully autonomous manner.

Central to this system is a robust speech-to-text module that accurately transcribes spoken instructions, complemented by a large language model that extracts relevant objects or themes while filtering out irrelevant details. The extracted elements are then fed into a fine-tuned text-to-image generative model, which produces simplified visuals tailored for robotic drawing. To ensure precise execution, the system employs vision-based techniques for detecting and aligning the painting canvas within the workspace. This step minimizes spatial inaccuracies and ensures that the robotic arm's movements align perfectly with the intended drawing.

The generated visuals undergo further refinement through image simplification techniques that preserve artistic essence while eliminating extraneous details, reducing computational overhead and drawing time. These simplified images are then transformed into smooth, continuous trajectories using advanced path-planning algorithms, enabling the robotic arm to execute fluid and natural motions. By integrating these modules into a cohesive framework, the system demonstrates adaptability to diverse command styles and drawing complexities, bridging human creativity with robotic efficiency.

This research highlights the potential of integrating artificial intelligence and robotics in creative tasks. By addressing critical challenges in natural language understanding, visual processing, and robotic control, it lays the groundwork for future innovations in creative robotics. The project not only showcases the feasibility of voice-guided robotic art creation but also emphasizes the importance of system optimization for efficiency, reliability, and practical usability in dynamic, creative environments.

II. Related Work

The concept of robotic drawing has been explored extensively, with carious approaches aiming to blend precision and artistry. Reference [1] proposed a humanoid robot named Pica, equipped with a 7-DOF arm, to autonomously draw human portraits. This system employed sophisticated image processing techniques to transform captured facial images into simplified line drawings. Key innovations included the use of noise reduction algorithms and trajectory optimization to ensure smooth and efficient robotic drawing. The system highlighted the importance of balancing computational efficiency and artistic integrity particularly through methods like line thinning and feature refinement.

Similarly, [2] presented a robot capable of sketching human portraits in under five minutes. This research

emphasized mimicking human-like drawing behavior, integrating edge detection algorithms, and leveraging smooth motion trajectories for lifelike portrait creation. The system's unique ability to adjust its drawing style dynamically, coupled with its ergonomic hardware setup, demonstrated the potential of robotic systems in interactive and entertainment settings.

These studies underscore the importance of integrating visual processing, trajectory planning, and real-time execution to create robotic systems that can effectively human artistic skills. These works laid the foundation for enhancing robotic systems in terms of adaptability and efficiency. Building upon these concepts, our study aims to advance the field further by incorporating voice command interpretation, image generation, canvas localization, and optimized image-to-trajectory conversion, thereby expanding the scope of robotic creativity in dynamic environment.

III. Methodology

The process begins with a speech-to-text module, which transcribes verbal commands into text and translates them into English. The text is then processed by a language model to extract relevant artistic elements, which serve as inputs for a fine-tuned text-to-image generation model. The generated images are simplified and refined to create visually interpretable reference drawings optimized for robotic execution. The system employs vision-based techniques to accurately localize the painting canvas within the workspace, ensuring proper alignment with the robot arm. Path generation and trajectory optimization converts the simplified images into smooth, continuous paths, enabling precise and fluid drawing on the canvas.

A. Speech-to-Text

This section details the implementation process of a speech-to-text system integrated with noise reduction and translation capabilities. The system is designed to record audio using a selected microphone, process the recorded audio to reduce noise, and transcribe and translate the audio content into text.

1) Microphone Selection: This system uses the PyAudio [3] to enumerate available audio input devices. This is achived by iterating through the list of devices . For each device, the system retrieves and displays detailed device information, to identify and specify the index of the desired input device

2) Audio Recording: The audio recording process initializes a recording stream using the specified microphone index and captures audio data for a predefined duration. Key parameters for the recording include

a) Sample Rate: Number of samples per second.

b) Channels: Number of audio channels.

c) Sample Width: Number of bytes per sample.

d) Buffer Size: Number of frames read per iteration.

The captured audio frams are stored in memory and subsequently written to a Waveform Audio Format (WAV) file, ensuring compatibility with subsequent processing steps.

3) Noise Reduction: The system address the issue of ambient using noisereduce [4] to perform spectral noise reduction on the recorded audio. It suppresses background noise while preserving the clarity of the speech signal. The noise reduction step significantly enhances the transcription accuracy by improving the quality of the audio input.

4) Audio Transcription and Translation: The transcription and translation of audio content are handled by OpenAI's Whisper model [5] which is a state-of-art automatic speech recognition (ASR) and translation system. Feed the denoised audio file into the Whisper model and configure the task as translate to transcribe the audio and convert it into English text simultaneously.

B. Text-to-Image

To produce images that reflect the user's intent, we designed the following workflow:

- Understand what the user wants to draw.
- Use an AI model to generate an initial, random image.
- Ensure that the robotic arm can draw a simplified version of that image.

This approach can be divided into three parts, which we will now introduce in detail.

1) Identifying the User's Desired Content: In the previous stage, we capture the user's speech through a microphone and convert it to text using speech-to-text technology. Since natural language allows for a wide range of expressions, it would be impractical to confine users to a rigid format when describing what they want to draw. Therefore, users are free to describe their desired content in any manner they prefer. To process this input, we employ a Large Language Model (LLM) to identify the objects the user wishes to draw from the provided text. Given that our robotic arm is not designed to create intricate images, we focus solely on extracting objects, disregarding artistic styles or overly complex details at this stage. For this purpose, we selected OpenAI's GPT-4o-mini model [6], which offers both robust performance and high efficiency. We provide it with tailored system and user prompts to optimize the output:

- System prompt: "You are a helpful assistant that focuses on extracting the key objects or subjects from a given piece of text. The extracted list should be simple and limited to the main elements that would be illustrated."
- User prompt: "Here is some raw text: {raw_text} From this text, identify the main objects or subjects that should appear in a simple illustration. Keep the list short, direct, and easy to visualize. Only provide the objects, no additional context. Return them as a concise bullet point list."

After passing the transcribed text into this prompt, GPT returns the objects in a bullet-point list, for example:

- Flower
- Fish

We then transform this bullet-point list into a comma-separated string, e.g., "Flower, Fish" to represent the objects the user wants drawn.

2) Image Generation: Since we need simple images that the robotic arm can handle, we employ two main techniques:

a) Prompt Engineering: We carefully design the prompt for our text-to-image model so that it produces a simple line drawing. The final prompt looks like this: "A simple black and white line drawing of {objects_description}, minimalistic, thin, clean outlines, no shading, no background, pencil sketch style. And the object should be complete." Here, {objects_description} is replaced by the objects we extracted (e.g., "Flower, Fish").

b) Model Fine-Tuning: Because we are using a relatively small Stable Diffusion model [7], its ability to follow instructions strictly is limited. If we simply feed in our prompt, the resulting image may still be too complex. Therefore, we use a fine-tuned model whose style naturally leans toward simple line-art drawings. By combining our carefully designed prompt with a fine-tuned model, we can produce greatly simplified images with minimal extraneous elements. Details of the fine-tuned model will be discussed in the Experimental Setup section.

3) Post-Processing: Finally, we apply post-processing steps to further simplify the generated image into clean line drawings that the robotic arm can handle. We use traditional image processing and morphological operations to achieve skeletonization/thinning. The goal is to extract single-pixel-wide lines from the generated image's edges. The steps are as follows:

a) Reading and Preprocessing: Read the image using `cv2.imread()` [8] and convert it to grayscale with `cv2.cvtColor()` [9] to simplify processing. Finally, apply bilateral filtering using `cv2.bilateralFilter()` [10] to reduce noise while preserving edges.

b) Edge Detection: Use Canny edge detection (`cv2.Canny()`) [11] to obtain a binary edge map from the grayscale image.

c) Skeletonization: First, convert the binary edge map into a 0/1 binary format to prepare it for further processing. Then, create a morphological structuring element using `cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))`[12] [13]. By iteratively applying erosion and dilation, and subtracting the results, the outer layers of the edges are gradually peeled off. This process continues until only a skeleton of single-pixel-wide lines remains, effectively simplifying the structure while retaining its essential shape.

d) Output: Save the skeletonized image for subsequent path generation by the robotic arm.

C. Canvas Localization

This section details how to obtain 3D coordinates and 2D image plane coordinates of the four corners of the canvas. The camera we use is the built-in camera on the TM5-900 [14] robotic arm, mounted above the end-effector gripper. Before capturing an image for canvas localization, we position the robotic arm's end-effector at a fixed location, ensuring that the gripper and camera face directly downward, perpendicular to the table surface. This setup allows the camera to capture the entire table in the frame.

1) Camera Calibration: We perform camera calibration using chessboard images to obtain intrinsic parameters, including the camera matrix and distortion coefficients, through a three-step process. First, corner detection is conducted by reading each chessboard image and identifying intersection points of the chessboard pattern using the `cv2.findChessboardCorners()` [15] function. Next, in the point accumulation step, object points (3D coordinates in real-world space) and image points (2D coordinates in the image plane) are gathered. Finally, the calibration step leverages these accumulated points in the `cv2.calibrateCamera()` [16] function to compute the camera matrix and distortion coefficients.

2) Canvas Definition: We define the canvas space as the inner rectangle of four ArUco markers [17] as the Fig. 1 shows. However, to minimize detection error of ArUco markers, we can actually obtain the fourth corner's position from the positions of three corners by the parallelogram rule of vector addition [18].

3) Pose Estimation: To determine the canvas corners in the camera frame, we load the ArUco marker dictionary and detection parameters using OpenCV's `DICT_4X4_50` [19] and `cv2.aruco.DetectorParameters()` [20]. With `cv2.aruco.ArucoDetector()` [21], we detect marker corners and IDs in the image. We then compute the position and orientation of each marker with `cv2.solvePnP()` [22], using object points, detected corner coordinates, the camera matrix, and distortion coefficients. The marker's corners are transformed to the camera frame via the equation: `corner_camera_frame = R_marker @ corner_3D + tvec_marker`, where `R_marker` and `tvec_marker` are from `cv2.solvePnP()`. The fourth corner is determined by the canvas definition.

4) Projection to Image Plane: To project 3D coordinates in the camera frame onto the 2D image plane, we use the `cv2.projectPoints()` [23] function. This function computes the corresponding image plane coordinates based on the given 3D points in the camera frame. The inputs required for this process include the 3D coordinates in the camera frame, the camera matrix, and distortion coefficients. Using these parameters, `cv2.projectPoints()` accurately maps the 3D points onto the 2D image plane.

D. Path Generation

To enable the robotic arm to draw patterns, we need to convert the skeletonized image generated in the Text-to-Image step into a sequence of robotic arm coordinates. This requires transforming the image into a path represented in world coordinates. The process consists of three steps: converting the image to a path using image coordinates, mapping image coordinates to camera coordinates, and transforming camera coordinates to world coordinates.

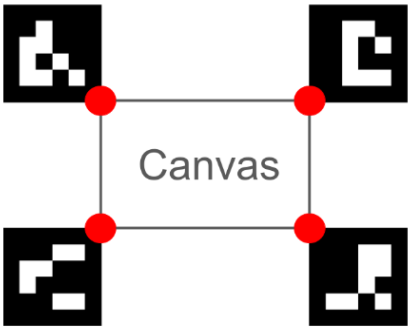

Fig. 1. Canvas definition

1) Image to Path with Image Coordinate: In order to generate the continuous paths from the image, Algorithm 1 is designed. The changeable argument *NextMask* and *Neighbor* allow user to modify the movement distance between points, which can adjust the number of instruction of robot arm movement for striking the balance between fineness and time consumption. The function defined in 3rd line is to find path from given start point. The variable, *momentum*, is to make to generated path tend to go smoothly by forcing the movement to next point is most similar to previous movement. It is observed that the generated number of paths is reduced when introduced the momentum mechanism. The if statement in 35th line is to filter out small paths with length less than given threshold value *MinPathLength* in order to reduce the number of paths with minimum number of deleted points.

2) Image Coordinate to Camera Coordinate: To perform coordinate transformation, the homography matrix H is calculated using OpenCV's `findHomography()` [24] function. The four corners of the canvas, obtained from the Canvas Localization step, are used as the target points, while the four corners of the image—(0, 0), (m, 0), (m, n), and (0, n)—serve as the source points. The paths list in camera coordinates is then computed by multiplying each element in the paths list with H. The resulting camera coordinates are recorded in a Comma-Separated Values (CSV) file. The format of the file stores the x-axis values in the first column, y-axis values in the second column, and uses (-1, -1) as a separator between paths to indicate lifting the pen.

3) Camera coordinate to World Coordinate: As mentioned in the Canvas Localization step, before each photo is taken, we first control the robotic arm's end-effector to move to a fixed position, ensuring that the gripper and camera face directly downward. Therefore, we can treat the camera mounted on the arm as a fixed camera in the world coordinate system, effectively configuring it in an eye-to-hand setup. This allows us to easily perform eye-to-hand calibration to obtain the transformation relationship from camera coordinates to world coordinates.

During eye-to-hand calibration, we predefine a 5x5 grid of 25 coordinates in the world coordinate system. The robotic arm picks up a wooden block and sequentially places it at each grid point on the table. After placing each block, the robotic arm returns to the fixed position described earlier, with the camera facing downward, and captures an image for storage. This process results in 25 photos, one for each placement. Once all 25 images are taken, we process each image by:

- Converting the image to grayscale,
- Applying thresholding and morphological operations to remove noise,
- Using Canny edge detection to identify contours, and
- Calculating the centroid of each contour using image moments.

The centroid positions are recorded and saved. Finally, using the 25 predefined grid points in the world coordinates and the 25 detected centroid positions in the images, we compute the rotation (R) and translation (t) vectors with cv2.solvePnP().

Algorithm 1 Image to Path Algorithm

Require: $Image[m][n]$: 2D grayscale image, $NextMask$: List of relitive position of next points, $Neighbor$: List of relitive position of neighbors, $MinPathLength$: Minimun length of Path
Ensure: $Paths$: List of continuous paths

1: Initialize $Paths \leftarrow []$
2: Initialize $Visited \leftarrow$ False
3: **function** FINDPATH(x, y)
4: Initialize $Path \leftarrow []$
5: Initialize $Momentum \leftarrow (0, 0)$
6: Initialize $Current \leftarrow (x, y)$
7: **while** True **do**
8: Initialize $MinDiff \leftarrow \infty$
9: Initialize $Next \leftarrow (0, 0)$
10: Append $Current$ to $Path$
11: **for** (nx, ny) in $Neighbor$ **do**
12: **if** $0 \leq x + nx < m$ AND $0 \leq y + ny < n$ **then**
13: $Visited[x + nx][y + ny] \leftarrow$ True
14: **end if**
15: **end for**
16: **for** (nx, ny) in $NextMask$ **do**
17: $Diff \leftarrow |nx - Momentum.x| + |ny - Momentum.y|$
18: **if** $0 \leq x+nx < m$ AND $0 \leq y+ny < n$ AND $Diff < MinDiff$ AND $Visited[x + nx][y + ny] =$ False **then**
19: $MinDiff \leftarrow Diff$
20: $Next \leftarrow (nx, ny)$
21: **end if**
22: **end for**
23: **if** $MinDiff = \infty$ **then**
24: Break
25: **end if**
26: $Momentum \leftarrow Next$
27: $Current \leftarrow (x + nx, y + ny)$
28: **end while**
29: **return** $Path$
30: **end function**
31: **for** $i = 0$ to $m - 1$ **do**
32: **for** $j = 0$ to $n - 1$ **do**
33: **if** $Visited[i][j] =$ False AND $Image[i][j] = 255$ **then**
34: $Path \leftarrow$ FindPath(i, j)
35: **if** $Path.length \geq MinPathLength$ **then**
36: Append $Path$ to $Paths$
37: **end if**
38: **end if**
39: **end for**
40: **end for**
41: **return** $Paths$

These vectors transform a 3D point expressed in the object coordinate frame into the camera coordinate frame, as represented by the following equation:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R|t]\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (1)$$

Where s is the scale factor, (u, v) are image points, K is the intrinsic matrix, $[R|t]$ are the extrinsic parameters, and (X, Y, Z) are the world coordinates of the image points. From equation (1), we can rewrite as following equation:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = K[R|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

Therefore, we can multiply the obtained intrinsic matrix, extrinsic parameters, and the 25 world coordinates. By extracting the values from the third row, we obtain 25 scale factors. These scale factors are then averaged to determine the final scale factor.

After obtaining the extrinsic parameters and the scale factor from the eye-to-hand calibration process, we can now rewrite equation (1) to establish the transformation relationship from camera coordinates to world coordinates:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \left(s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} K^{-1} - t \right) R^{-1} \tag{3}$$

Therefore, we can convert each point in the CSV file containing the saved camera coordinates into world coordinates using equation (3), and then send these coordinates as commands to the robotic arm to make it move. This will allow the arm to draw on a canvas in the world coordinate system.

IV. Experimental Setup

In this final project, our experiments and demonstrations were conducted in the space of the ER7 laboratory located in the basement of the De-Tian Building (CSIE Building) at National Taiwan University, using the TM5-900 robotic arm and computer in the lab. We also prepared a USB (Universal Serial Bus) microphone for audio recording. To ensure stable contact between the robotic arm and the canvas while drawing with a pen, we designed a custom pen-holding module with a spring mechanism, which was secured by the arm's gripper to facilitate drawing. The complete experimental setup is shown in Fig. 2. In terms of experiments, we focused mainly on testing the Text-to-Image model to achieve the best image generation results. Finally, we integrated the various steps and controlled the robotic arm using ROS 2 (Robot Operating System 2) [25] and Python subprocess [26], issuing different voice commands during the tests to observe the drawing outcomes.

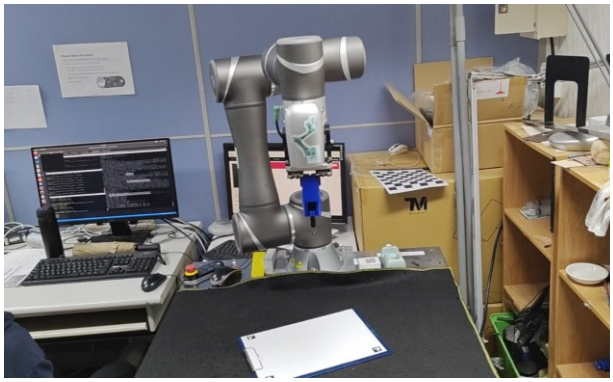

Fig. 2. Complete experimental setup

A. Hardwares

1) Robotic Arm and Computer: Our robotic arm, the TM5-900 (Fig. 3), is manufactured by Techman Robot Inc. It features a built-in intelligent vision system, enabling it to "see" and perceive its surroundings. This capability ensures stable performance in tasks such as automated assembly, vision inspection, and 3D pick-and-place. Designed for safe collaboration, the TM5-900 meets ISO 10218-1:2011 and ISO/TS 15066:2016 safety standards and includes collision detection and emergency stop mechanisms for enhanced operational safety. With a 946 mm reach and a 4 kg payload capacity, it offers efficiency and flexibility across industries like electronics, automotive, and food manufacturing.

The ER7 laboratory's computer runs on Ubuntu 18.04 and is equipped with an RTX 2060 graphics card. Pre-installed with ROS Melodic and ROS2 Dashing, it enables us to control the TM5-900 robotic arm via ROS Service. This setup supports seamless integration of various components and steps involved in our project research.

2) Microphone: We selected the ZIYA Omnidirectional Microphone [27] (Fig. 4) for this final project reserch, which is designed for business meetings, remote conferences, and versatile use in environments such as offices, hotel rooms, and home offices. With a compact size of 6 cm in diameter and 1.5 cm in height, it features a durable metal mesh exterior and an anti-slip rubber base to ensure stability during use. This microphone provides 360-degree, high-sensitivity sound pickup within a 2–3 meter radius. It supports a frequency range of 40–16,000 Hz and has a signal-to-noise ratio of 58 dB or higher. Weighing approximately 90 g, it connects via a USB plug-and-play interface, eliminating the need for driver installation. Due to these features, we selected this microphone to capture the user's voice input effectively.

3) Pen-Holding Module: To ensure the robotic arm can maintain stable contact with the canvas while drawing with a ballpoint pen, we designed and fabricated a custom pen-holding module (Fig. 5). The main structure is 3D-printed using PLA (Polylactic Acid) material, and components such as the pen tube and springs are repurposed from discarded ballpoint pens. The smooth, low-friction plastic material of the pen tube, along with its 3 mm outer diameter, makes it suitable for use as a linear guide rail, while its 1.9 mm inner diameter is perfectly sized to secure an M2 metric screw for fastening.

The module consists of three main parts: the pen clamp, the body, and the cover (Fig. 6). The pen clamp features three

Fig. 3. TM5-900

Fig. 4. ZIYA Omnidirectional Microphone

holes that allow the pen tubes to pass through, enabling smooth vertical movement with minimal deviation. It also uses a single set of M3 screws and nuts to tightly secure the ballpoint pen used for drawing. The body serves as the main structure for fixing the three pen tubes, functioning as linear guide rails. The cover not only secures the three pen tubes and connects to the body but also includes a gripping section designed to fit snugly with the robotic arm's gripper. This precise fit prevents any wobbling during movement, ensuring stability throughout the drawing process.

Between the pen clamp and the cover, three springs are installed around the pen tubes. These springs provide upward flexibility when the pen comes into contact with the canvas and exert a stable downward force. This design ensures consistent contact between the pen and the canvas, even when drawing on uneven surfaces or at varying heights. As a result, the robotic arm can easily produce the desired patterns with precision.

B. Text-to-Image

We initially sourced our fine-tuned models from Hugging Face. Our first attempt used the model "lora-library/B-LoRA-ink_sketch," which is based on Stable Diffusion XL 1.0 [28] and produces ink-sketch styles aligned with our needs. We began by setting the image size to 1200×800 (3:2 aspect ratio). However, the SDXL 1.0 model demanded more than 8GB of GPU memory, exceeding the 6GB VRAM limit of our RTX 2060 GPU.

To address the VRAM limitation, we switched to Stable Diffusion 1.5, which typically requires less than 7GB of VRAM. Even so, the memory usage remained slightly too high at our chosen resolution. By iteratively adjusting and testing different image sizes, we settled on 456×304, a resolution that allows the model to run without surpassing the 6GB VRAM limit. After finalizing the base model, we selected the fine-tuned model "yehiaserag/anime-pencil-diffusion" (based on SD1.5) due to its naturally pencil-like, line-art output style that matches our simplification goals.

In summary, our experiment setup involved:

- Selecting a suitable base model (SD1.5).
- Choosing a fine-tuned model that yields simpler, line-art style images.
- Adjusting the image resolution and other hyperparameters to fit within our hardware constraints.
- The final configuration utilizes SD1.5 along with the "yehiaserag/anime-pencil-diffusion" model and an image size of 456×304. This setup effectively produces acceptable, simplified line-art images while avoiding VRAM-related issues, ensuring smooth performance during processing.

To systematically evaluate our pipeline and compare different configurations, we designed four experimental conditions, followed by an optional post-processing step:

1) Base Model Only: Use the original Stable Diffusion 1.5 model and prompt it only with the object name (e.g., "fish" or "car").

2) Fine-Tuned Model Only: Replace the base model with the fine-tuned "yehiaserag/anime-pencil-diffusion" model, again using only the object name as the prompt.

3) Base Model with Designed Prompt: Keep the base model but use a carefully crafted prompt that emphasizes minimalistic, black-and-white line drawings without shading, ensuring that the resulting image has clean outlines.

4) Fine-Tuned Model with Designed Prompt: Combines both techniques by utilizing a fine-tuned model along with a carefully designed minimalistic line-art prompt. Together, they produce even simpler and clearer outputs, ensuring high-quality results with clean and precise details..

For each configuration, we generate 10 images—5 of "fish" and 5 of "car"—to evaluate performance across setups. All the output images are shown in Fig. 7 through Fig. 10. From Fig. 10, it can be observed that the lines become thinner, and no unusual objects are present compared to the results in the previous images. After generating images under the fourth configuration, we apply the previously described post-processing steps exclusively to this condition. These steps include Canny edge detection, bilateral filtering, and morphological thinning to further simplify the images into skeletonized line drawings. The final processed images are presented in Fig. 11. By examining these four configurations along with the subsequent post-processing, we can effectively assess the complexity, style, and quality of the generated images, ultimately identifying the most suitable approach for our robotic arm's drawing capabilities.

Fig. 5. Pen-holding module

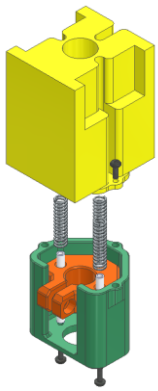

Fig. 6. Exploded view of the pen-holding module

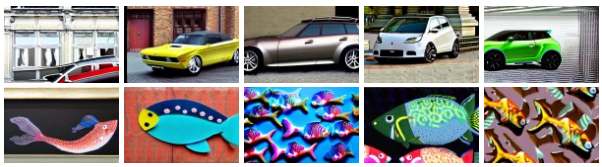

Fig. 7. Output pictures of Base Model Only

Fig. 8. Output pictures of Fine-Tuned Model Only

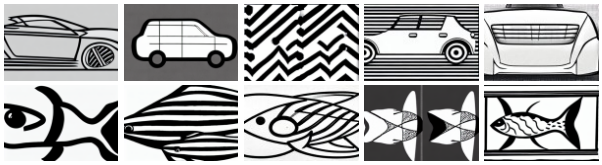

Fig. 9. Output pictures of Base Model with Designed Prompt

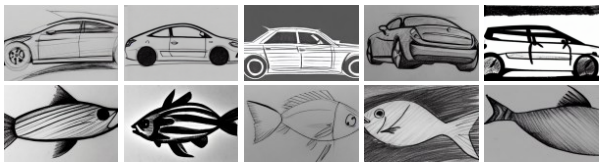

Fig. 10. Output pictures of Fine-Tuned Model with Designed Prompt

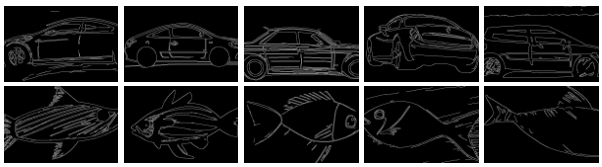

Fig. 11. Output pictures of Fine-Tuned Model with Designed Prompt and post-processing

C. System and Process Integration

In the previous Methodology section, we introduced the four main techniques used in our project—Speech-to-Text, Text-to-Image, Canvas Localization, and Path Generation. Earlier sections also covered the hardware components we used and the experiments conducted to fine-tune the Text-to-Image model. Next, we focus on integrating these components and conducting tests to achieve our ultimate goal—Real-World AI Artist, where the robotic arm draws the desired image based on voice commands given by the user.

1) System Integration: In this final project, our program is primarily built on the ROS2 framework and written in Python. Each step of the process is implemented and tested as a separate module (class). Every step reads input from one file and outputs results to another file. Detailed descriptions of file inputs and outputs will be provided later. This modular approach allows for clearer task division, improved efficiency, and easier collaboration within our team. Each member can focus on developing and testing their assigned modules independently. Once testing is complete, the modules can be quickly integrated into the main program. In the main program, tasks are executed simply by calling the functions of each module.

For the Speech-to-Text and Text-to-Image processes, several less common libraries, such as pyaudio, whisper, noisereduce, openai, and diffusers, were required. Some of these libraries specifically require Python 3.10, whereas the ROS2 system in the ER7 laboratory computer runs on Python 3.6. To avoid interfering with the computer's native Python environment and resolve this version compatibility issue, we utilized Anaconda to create a dedicated environment for downloading these libraries and running our programs. However, since this custom environment is incompatible with ROS2, we employed the Python subprocess module to execute these two steps in separate subprocesses outside the main program. Each subprocess activates our custom environment to execute the corresponding step. Given that our implementation handles input and output through files at each stage, this approach ensures seamless integration with other steps in the main program without any conflicts.

For robotic arm control, we use ROS2 Services to send commands, including:

- Moving the arm to a specified position and orientation (X, Y, Z, Rx, Ry, Rz).
- Controlling the gripper at the end effector to open and close.
- Capturing images using the camera mounted on the end effector and returning the images.

Additionally, we use ROS2 Topic Subscription to receive and store the captured images.

2) Process Integration: The process of main program consists of the following steps:

a) Setup and Initialization: The canvas is first secured onto a clipboard and placed on the table. The program is then started to initialize the system. The robotic arm automatically moves to a preset position. Once the arm has stopped, the 3D-printed pen-holding module is attached to the gripper, and the gripper is closed to secure the module by pressing any key.

b) Command Selection: After initialization, the system prompts the user to select one of the available commands:

- auto(1): Execute the full automated process.
- get_canvas(2): Capture and process the canvas image.
- record_audio(3): Record a voice command.
- generate_image(4): Generate an image based on the voice command.
- generate_path(5): Generate the drawing path.
- draw(6): Execute the drawing process.
- exit(q): Exit the program. .

c) Automated Process: Selecting "auto" (command 1) executes the entire workflow automatically, including capturing the canvas, recording audio, generating the image, calculating the drawing path, and performing the drawing. Each process is outlined below.

d) Canvas Localization: Selecting "get_canvas" (command 2) begins the canvas localization process. The system captures an image of the canvas when any key is pressed. It then calculates the four corners of the canvas and displays the processed image on-screen. Pressing any key again closes the image display.

e) Voice Command Input: Selecting "record_audio" (command 3) initiates the voice recording process. After displaying the prompt "Recording..." on the terminal, the system records audio for 5 seconds. The user can issue a command such as "I want to draw a flower." or "I want to draw a fish." The recorded audio is then automatically translated into English, displayed on-screen, and saved as text file.

f) Image Generation: Selecting "generate_image" (command 4) initiates the image generation process. The system creates an image based on the text file generated from the voice command input in the previous step and saves the resulting image file.

g) Path Generation: Selecting "generate_path" (command 5) begins the path generation process. The program calculates an optimized drawing path based on the generated image and the four corners of the canvas identified in the Canvas Localization step. The calculated path is saved as a CSV file.

h) Drawing Execution: Selecting "draw" (command 6) executes the final drawing process. The robotic arm follows the pre-calculated path to draw the generated image onto the canvas automatically.

This structured procedure ensures that each step is modular, facilitating ease of testing and integration while enabling the user to operate the system either fully automatically or step-by-step based on specific requirements.

3) Full Integration Experiment Results: After completing the full integration, we tested the system by inputting various object commands to observe the drawing performance of the robotic arm. The tested objects included a flower, fish, cup, tree, apple, and car. All the images drawn by the robotic arm are presented in Fig. 12.

V. Results

The developed system demonstrates significant capabilities in translating voice commands into robotic artworks. It successfully interprets spoken instructions with high accuracy through its speech-to-text and semantic extraction modules. The fine-tuned text-to-image generation produces simplified visuals suitable for robotic execution, reducing drawing time while maintaining artistic integrity. The canvas localization module achieves precise alignment, ensuring that generated images are accurately transferred onto the canvas. In testing, the system effectively handled diverse command styles and varying levels of drawing complexity, producing smooth and visually appealing outputs. These results validate the robustness of the integrated modules and their ability to perform in real-world scenarios.

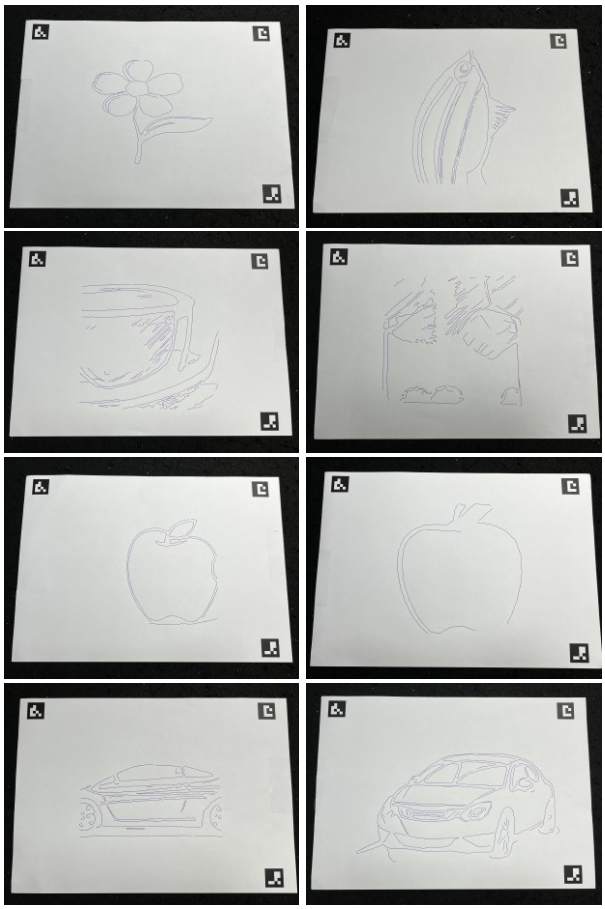

Fig. 12. Result images drawn by the robotic arm

VI. Conclusions

This research highlights the integration of AI and robotics to enable voice-controlled robotic drawing systems, addressing challenges in natural language understanding, image processing, and trajectory optimization. The system's ability to autonomously create precise and visually interpretable artworks demonstrates its potential for practical and creative applications. Beyond showcasing technical feasibility, this work emphasizes the synergy between AI-driven generative models and robotics for user-specific tasks. Future efforts could expand the system's adaptability to more complex artistic styles, enhance real-time responsiveness, and explore its use in education, entertainment, and collaborative art-making.

Work Division

TABLE I. Work division

Name	Work
Hao-Yu Liu	Robotic arm control, pen-holding module design, system and process integration, report organization, and video editing.
Shao-Yang Chang	Speech-to-Text and report organization.
Chi-Han Tsai	Text-to-Image
Ding-Shan Chen	Path generation
Sin-Ruei Yang	Canvas localization

Source Code and Demo Video

Source code:

https://github.com/HaoYuLiu0725/Robotics2024_FinalProject_Team15

Demo video:

https://youtu.be/O0C3AmE9UMM

References

[1] Lin, C.-Y., L.-W. Chuang, and T. T. Mac. (2009). Human portrait generation system for robot arm drawing. 2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics,

[2] Jean-Pierre, G., and Z. SaId. (2012). The artist robot: A robot drawing like a human artist. 2012 IEEE International Conference on Industrial Technology,

[3] Giannakopoulos, T. (2015). pyaudioanalysis: An open-source python library for audio signal analysis. PloS one, 10(12), e0144610.

[4] Tim Sainburg, noisereduce, [Online]. Available: https://github.com/timsainb/noisereduce

[5] Radford, A., J. W. Kim, T. Xu, G. Brockman, C. Mcleavey, and I. Sutskever. (2023). Robust Speech Recognition via Large-Scale Weak Supervision. Proceedings of the 40th International Conference on Machine Learning, Proceedings of Machine Learning Research. 28492--28518.

[6] OpenAI, GPT-4o-mini: Advancing Cost-Efficient Intelligence, [Online]. Available: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/

[7] R. Rombach, A. Blattmann, D. Lorenz, et al., "High-Resolution Image Synthesis with Latent Diffusion Models," arXiv preprint arXiv:2112.10752, 2021. [Online]. Available: https://arxiv.org/abs/2112.10752

[8] OpenCV, "Image file reading and writing," OpenCV 3.4 Documentation, 2019. [Online]. Available: https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html

[9] OpenCV, "Color conversions," OpenCV 3.4 Documentation, 2019. [Online]. Available:

https://docs.opencv.org/3.4/d8/d01/group__imgproc__color__conversions.html

[10] OpenCV, “Filtering images,” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html

[11] OpenCV, “Canny edge detection tutorial,” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

[12] OpenCV, “Structuring element creation and filtering,” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html

[13] OpenCV, “Morphological operations: MORPH_CROSS,” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

[14] TM5-900. [Online]. Available: https://www.tm-robot.com/zh-hant/tm5-900/

[15] OpenCV, “findChessboardCorners(),” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga93efa9b0aa890de240ca32b11253dd4a

[16] OpenCV, “calibrateCamera(),” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga3207604e4b1a1758aa66acb6ed5aa65d

[17] OpenCV, “Detection of ArUco Markers,” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

[18] GeeksforGeeks, “Parallelogram Law of Vector Addition”. [Online]. Available: https://www.geeksforgeeks.org/parallelogram-law/

[19] OpenCV, “DICT_4X4_50,” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/3.4/d9/d6a/group__aruco.html#ggac84398a9ed9dd01306592dd616c2c975ada8e830ff0024e839e93c01f5fed0c55

[20] OpenCV, “DetectorParameters(),” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/4.x/d1/dcd/structcv_1_1aruco_1_1DetectorParameters.html#a12c71ac3314cb086054c963be50d9eeb

[21] OpenCV, “ArucoDetector(),” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/4.x/d2/d1a/classcv_1_1aruco_1_1ArucoDetector.html#aebb849a70f107f72ff5b0504d89120de

[22] OpenCV, “solvePnP(),” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d

[23] OpenCV, “projectPoints(),” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga1019495a2c8d1743ed5cc23fa0daff8c

[24] OpenCV, “findHomography(),” OpenCV 4.x Documentation, 2021. [Online]. Available: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780

[25] Robot Operating System (ROS), “ROS 2 Documentation,” ROS 2 Documentation: Dashing documentation. [Online]. Available: https://docs.ros.org/en/dashing/index.html

[26] Python, “subprocess — Subprocess management,” Python 3.13.1 documentation. [Online]. Available: https://docs.python.org/3/library/subprocess.html

[27] ZIYA Omnidirectional Microphone. [Online]. Available: https://24h.pchome.com.tw/prod/DCBA83-A900B0F3V

[28] [9] S. M. M. Sari et al., “SDXL: Advancing Diffusion-Based Image Generation at Scale,” arXiv preprint arXiv:2307.01952, 2023. [Online]. Available: https://arxiv.org/abs/2307.01952