J-COSINE (Journal of Computer Science and Informatics Engineering)
Vol. 8, No. 2, December 2024 E-ISSN:2541-0806
Accredited Sinta-4 by RISTEKDIKTI Decree No. 79/E/KPT/2023 P-ISSN:2540-8895

# Enhancing Robot Autonomy: Path Planning via Hand-Drawn Sketches and Computer Vision Integration

Mohamedalmogtaba Abdelrahman*, Mietra Anggara
Mechanical Engineering Dept, Sumbawa University of Technology
Sumbawa, Indonesia
*Email:* mo.mo.mojtaba.01@gmail.com, mietra.anggara@gmail.com

****Corresponding Author***

***Abstract*** **This investigation delves into an advanced method for enhancing robot autonomy by integrating hand-drawn sketches with computer vision systems to optimize route planning effectively. The procedure starts with the capture of image data, which is then processed through multiple stages until the sketches are turned into exact coordinates that guide a robot. Canny edge detection, a specific computer vision technique, is employed to detect the edges of the drawn lines, facilitating precise and dependable navigation for autonomous robot operations. Initially, the image is subjected to noise reduction and edge enhancement before converting the sketch lines into Cartesian coordinates. This step is succeeded by point filtering and ordering to ascertain accurate path adherence by the robot, with no coordinates overlooked. Scale adjustments are also made to match digital image coordinates with the real-world setting, thereby improving the robot's interaction based on the received visual inputs. The program has been tested in Robotic Operating System 2 (ROS) using MoveIt2 and also in RoboDK software, where it was used to verify the results. It is important to note that path planning, which focuses on determining an optimal route from start to finish, is a subset of motion planning, which also considers the dynamics and kinematics of the robot's movement along that route.**

***Keywords--robot autonomy, path planning, computer vision, robotic arm, image processing***

## I. INTRODUCTION

The robot is a machine programmed by a computer, capable of executing a series of commands or complex tasks automatically, often requiring precision or speed. The twentieth century marks the beginning of the modern era of robots, characterized by the development of the first industrial robots in the 1950s. The field of robotics has witnessed rapid advancement driven by technological developments in electronics and computers, leading to artificial intelligence. This has enhanced the efficiency, intelligence, and capability of robots to perform increasingly complex tasks[1].

Robots have contributed to various fields. In industry, robotic arms have automated repetitive tasks, improving the efficiency and accuracy of final products. They are used in assembly and production lines, enabling the manufacture of large quantities of specific products through several robotic arms working together, as seen in automobile assembly plants. Robots are also employed in metal painting, welding, and industrial machinery, and in automating the transportation of pieces across different parts of factories using mobile robots.

Nowadays, service robots tailored to aid humans have become widespread, integrating into daily routines in areas such as cleaning, security, education, entertainment, and more. Examples include robotic vacuum cleaners, drones, and self-driving vehicles used for transportation and delivery services[2].

The advent of robots offers numerous advantages but also poses societal challenges. Questions arise regarding job prospects, mechanization, and the replacement of human labor. Ethical issues concerning safety and confidentiality also emerge with the increasing prevalence of robots in society.

Motion is a fundamental characteristic of all industrial and service robots. Therefore, motion planning is crucial, involving the determination of a series of valid configurations for the robot to move from its current position to the desired goal while avoiding obstacles. This process comprises two main elements: path planning and trajectory planning. Path planning focuses on finding a feasible path from the robot's initial configuration to the target configuration, considering surrounding obstacles and manufacturing constraints. Trajectory planning involves creating a smooth and feasible path along the planned trajectory, taking into account dynamic constraints such as speed and acceleration[3].

Our research introduces a new and rapid method for path planning using paper sketching through computer vision technology. A person draws on a sheet of specific dimensions (square in our case), then takes a picture of this sheet for image processing to extract the coordinate points that constitute the drawn path. The image undergoes three stages of processing: identifying the paper boundaries and cropping excess edges, detecting lines and removing noise to obtain a black and white result where the white part represents the path for the robot arm.

In the second stage of image processing, the drawn path is converted into a set of points by selecting the central region of the drawn line's thickness. This process adjusts the number of points to balance detail and efficiency.

The third stage involves determining the order of these points based on their appearance, starting from the nearest point to the robot's initial position. The X and Y coordinates are then extracted (Z-coordinate remains stable, though future work may use depth cameras for 3D processing). These coordinates are scaled to match the real working environment, followed by inverse kinematics to calculate joint movements for the robot.

WE used RoboDK software to test and verify the system responsible for the robot's movement. The results of these tests are discussed in the Discussion section, highlighting the system's effectiveness and potential improvements.

In the following sections, WE will review three published works on robots and motion planning. WE will also review techniques for integrating robots with computer vision for various tasks. Subsequently, WE will detail Our methodology, present Our results, and discuss possible avenues for future development and practical applications of this method.

## II. Literature Review

Significant breakthroughs in robotics have been achieved through the integration of computer vision. This advancement enhances robots' perception and interaction with their surroundings, leading to improved overall performance. With computer vision, robots can process visual information to perform functions such as navigation, object recognition, and interaction within their environment. The adaptable framework provided by modern robotic control systems further enhances communication and control, resulting in smarter and more efficient robotic platforms. The combination of computer vision with these control systems opens opportunities for revolutionary developments in robotics technology.

Current researches reflects these advancements and highlights various innovative approaches:

Ahmed Abdulsahib's[4] research focuses on self-directing mobile work machines with articulated frame steering (AFS) for path following and motion control, aiming to eliminate human intervention in environments like mines and construction sites. Abdulsahib's method uses a ROS2-based path-following control system that generates machine velocities and converts them into commands. This approach assumes vehicles follow an imaginary point moving along a path, akin to Reza Ghabcheloo's algorithms[5]. The system was tested against a modified pure pursuit technique from ROS2 navigation, demonstrating practicality in improving path following in heavy-duty autonomous vehicles.

Peng Zhou's[6] research aims to augment robotic arc welding using point cloud models to overcome the limitations of older methods. Zhou's solution incorporates advanced techniques such as seam detection and tracking, adapting to changes in the working environment. Using hand signals, 3D model reconstruction, and de-noising techniques, the system produces reliable welding plans for different joint types and positions. While effective for complex seam patterns, this method requires a sophisticated setup due to its reliance on point cloud mining.

Nazar Kais AL-Karkhi's[7] study explores intelligent robotic welding through computer vision technology, focusing on precise welding line edge detection to ensure accurate positioning at the start of the welding process. The research employs image processing techniques, including edge detection and top-hat transformation algorithms, alongside the adaptive neuro-fuzzy inference system (ANFIS) to control the robot's kinematics. The results indicate satisfactory accuracy in tracking the welding path, with ANFIS effectively predicting the robot arm's movements.

In contrast to the aforementioned methods, our study introduces a rapid path planning method using computer vision to extract coordinates from a 2D sketch. This approach involves photographing a sketched course, processing the image to identify boundaries and detect lines, and transforming these lines into point sets that form a route. These points are then ordered and scaled for inverse kinematics-based robot movement control. Our method addresses gaps in previous research by offering a simpler, more intuitive solution for dynamic path planning, enhancing flexibility and adaptability in various environmental contexts.

Our experimental setup employs both ROS2 MoveIt and RoboDK software to test and verify the effectiveness of our approach. ROS2 MoveIt facilitates motion planning and trajectory control for our robotic arm, allowing us to simulate and refine the movement paths generated from the computer vision algorithm. RoboDK software provides a platform to validate these paths in a virtual environment, ensuring the accuracy and feasibility of the planned trajectories before real-world application.

The following sections will provide a comprehensive review of these studies, detail our methodology, present the results obtained from using both ROS2 MoveIt and RoboDK, and discuss potential future developments and practical applications of our method. By integrating these advanced tools, our research demonstrates significant improvements in path planning efficiency and robot autonomy.

## III. Methodology

Before delving into the specifics of our computer vision system, it's crucial to understand the foundational principles and recent advancements in this field. Computer vision, operating within the sphere of artificial intelligence, intends to replicate human perception in comprehending visual world intricacies. Humans effortlessly identify elements in their three-dimensional surroundings like types of cars and recognize individuals

simply by glancing at them. Significant advancements have occurred in computer vision concerning the reconstruction of the shape and appearance of three-dimensional objects from images, the creation of precise models, and the utilization of techniques for tracking objects. At present, computer vision is applied in optical character recognition (OCR), machine inspection, and medical imaging. Nonetheless, attaining a level of image interpretation similar to human perception continues to present notable challenges. Nevertheless, through the utilization of mathematical algorithms and strategies of deep learning, computer vision systems can extract valuable insights from visual data, offering substantial advantages across various industries, including healthcare, manufacturing, and security[8].

The world of robots has been totally revolutionized by recent advances in computer vision technology. Robots can be described as machines that are independent and able to carry out tasks according to a definition from the Cambridge Workshop. This definition does not include human-operated robots, which stresses out the necessity for autonomy and interaction with the environment. Previously, robotics concentrated on autonomous abilities of robots achieved through developing control algorithms to help in completing tasks. These algorithms range from basic wall-following techniques to more complex ones such as path planning and learning algorithms. Nevertheless, if a robot cannot perceive a task or its surrounding then it becomes difficult to ascertain if it can accomplish this task autonomously. This problem has led to growing interest in robotic perception enabling them to acquire information about their surroundings through observing similar to what people do. To elaborate further on interacting with the environment, real task completion by a robot is only possible when it touches tangible things in order that bring about an end-goal situation. It dives into the area of robotic manipulation, which shares borders with AI planning and relies heavily on computer vision techniques[9], [10].

Our computer vision system undergoes four distinct stages from image capture to processing and converting the drawing into a path with coordinates usable by the robot. During the development stages of this system, we drew three different paths on paper to test the software in various scenarios. We selected square sheets measuring 3x3 inches to represent a miniature version of the robot's designated workspace.

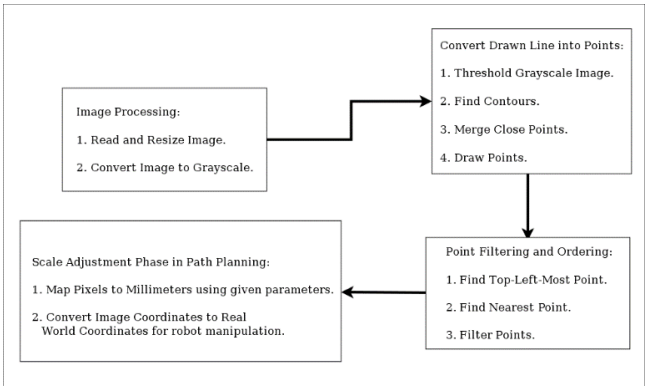


Fig 1. Image Processing Workflow

### *A. Image Processing*

The first stage of the program begins with a Python script designed to remove noise from the image and convert the edges of the drawn line on the paper. This is achieved through a series of steps, which can be summarized as follows:

*1) Reading and Resizing Image:*

The algorithm begins by reading the input image and then resizing it to a fixed size of 480x480 pixels. This resizing step ensures consistent processing and manageable computational complexity by standardizing the input size, making subsequent analysis and processing more efficient and uniform.

*2) Converting an Image to Grayscale:*

A colored input image is converted into a grayscale image. The image gets reduced to one intensity channel, making the subsequent processing steps simpler by eliminating color information and emphasizing variance of light intensity. This simplification is vital as it enables ease in analyzing and processing the picture, thus aiding extraction of necessary features relevant for later stages in computer vision workflow.

*3) Initialization of Parameters:*

Default values are set for edge detection and image processing, including:

- *Low and High Thresholds*: These parameters are crucial for Canny edge detection[11]. Pixels with gradient values lower than the low threshold are discarded, those greater than the high threshold are considered strong edges, and pixels with gradient values between the two thresholds are regarded as weak edges unless they connect to strong edges.
- *Blur Kernel Size*: This parameter determines the size of the Gaussian blur kernel[12]. Gaussian blur smooths an image by averaging pixel values within a specific neighborhood, with the kernel size controlling the amount of smoothing applied.
- *Morph Kernel Size*: This parameter determines the size of the kernel used for morphological operations[13] such as dilation and erosion. These operations manipulate the shapes and structures of objects in an image, with the kernel

size affecting the magnitude of the morphological operation.

- *Merge Loops*: This parameter deatermines the size of the kernel used in the morphological operation to merge closed loops. Closed loops in edges can occur due to noise or imperfections in the image, and merging these loops helps obtain cleaner edge maps.

### B. Convert Drawn Line into Points

After image processing, the program proceeds to the second stage, converting the drawn line into a set of Cartesian coordinates (X, Y) points. These points will be used in plotting the robot's path through inverse kinematics equations.

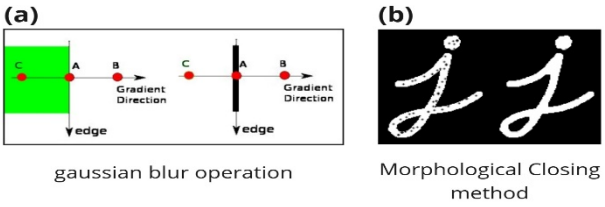


Fig 2. gaussian blur and Morphological

*1) Thresholding:*

This step isolates the white curve in the image. The function `cv2.threshold` takes a grayscale image, a threshold value (240 in this case), a maximum value (255), and a threshold type (`cv2.THRESH_BINARY`). It transforms the grayscale image into a binary image where pixels with intensity values higher than 240 are set to white (255), and those below this threshold are set to black (0). The output is a binary image.

*2) Finding Contours:*

function `cv2.findContours` is used to obtain the contours that represent the boundary of the white area in the binary image. It takes the binary image, a contour retrieval mode (`cv2.RETR_EXTERNAL` to retrieve only external contours), and a contour approximation method (`cv2.CHAIN_APPROX_SIMPLE` to compress horizontal, vertical, and diagonal segments and leave only their end points). The output is a list of contours found in Figure 3.

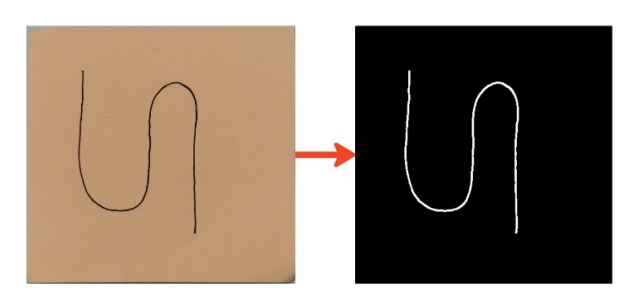

Fig 3. Original Image to Contour

*3) Merging Close Points:*

The function `merge_close_points` processes points within each contour to merge those in close proximity. The parameter `merge_var` determines the distance at which points are considered close enough to be merged. This function iterates through all points in each contour and groups together points that are close, effectively reducing the number of points and ensuring a clean visualization.

*4) Drawing Points:*

Circles are drawn at the locations of the merged points on a blank canvas. A blank image is initialized to have a black background with the same dimensions as the original image. The function `cv2.circle` is then used to draw a circle at each point's location, taking parameters for the image, the point's coordinates, the radius of the circle, the color (in this case, green `(0, 255, 0)`), and the thickness of the circle. The result is an image with circles representing all the available points, approximating the initial curve.

### C. Point Filtering and Ordering

After converting the line into a set of points, the program proceeds to the Point Filtering and Ordering stage. This stage involves extracting the coordinates of these points in the necessary order for the robot to move along the drawn path without bypassing any point or starting from a random point. The points are handled as follows:

*1) find_top_left_point(points):*

This function identifies the top-left-most point from a list of points. It takes a list of points as input and uses a sorting mechanism that prioritizes the y-coordinate and then the x-coordinate to find the top-left point. This step is crucial for establishing a consistent starting point for reordering the points.

*2) find_nearest_point(reference_point, points):*

This function calculates the nearest point to a given reference point. It takes a reference point and a list of points, using the Euclidean distance formula to find the closest point. This is vital for sequential reordering, ensuring that each subsequent point is the nearest to the current point.

*3) filter_points(points, min_distance=5):*

This function filters out points that are too close to each other, with a default minimum distance parameter set to 5 units. It takes a list of points and an optional minimum distance value, removing points that fall within this distance from each other. This step reduces noise from the point detection process, ensuring that the reordering process works with distinct, relevant points.

This stage culminates in obtaining an array containing the coordinates of all points constituting the desired path. These coordinates are scaled according to the image processing steps, requiring a magnification scale

adjustment based on the dimensions of the workpiece targeted for robot operation.

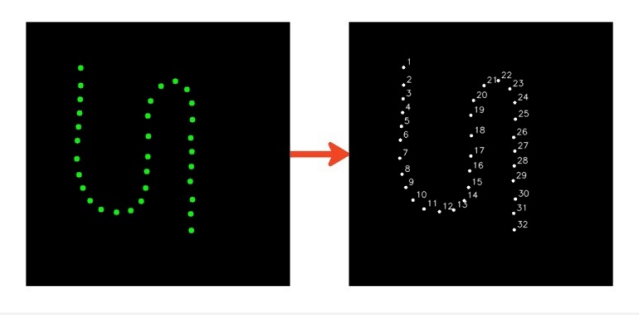

Fig 4. Converted Line and Ordering Points

### *D. Scale Adjustment Phase in Path Planning*

By carefully calibrating the scale and appropriately mapping the coordinates, tools that operate in physical space (like robotic arms) can accurately interact with targets recognized through digital imaging. The function 'map_pixels_to_mm' converts coordinates from a digital image's pixel-based system to a physical coordinate system measured in millimeters. This translation is crucial for applications where precise physical positioning based on image data is required, such as in robotic arm manipulation or precisely targeting areas in an automated inspection system.

The computational procedures employed to obtain path coordinates on the required spatial scale can be elucidated as follows:

$$mappedValue = \left(\frac{value - input_{min}}{input_{max} - input_{min}}\right) \times (output_{max} - output_{min}) + output_{min} \quad (1)$$

- $x_{pixel}$ $y_{pixel}$ : are the pixel coordinates.
- $x_{img\_min}$ , $x_{img\_max}$ *and* $y_{img\_min}$ , $y_{img\_max}$*: are represent the dimensions of the image utilized, which were 480x480 pixels. So, in our case* $x_{img\_min} = 0$ , $x_{img\_max\,=\,480}$ and same for y.
- $x_{robot\_top}$ , $x_{robot\_bottom}$ , and $y_{robot\_top}$ , $y_{robot\_bottom}$ : These four values represent the coordinates specific to the workspace of the robotic arm, where the upper-left edge and the lower-right edge are utilized to match them with the dimensions of the image later according to the following equation:

After obtaining the coordinates of the points for the robot, an equation is needed to undertake the task of transferring the robot's motion between one point and another. This is where the interpolation equation comes into play.

The interpolation formula plays a crucial role in smoothly transitioning between different robot positions. It helps ensure that the robot moves seamlessly from one point to another, improving overall motion quality. Without interpolation, sudden shifts in posture or alignment could lead to jerky or unstable movements. Additionally, interpolation helps prevent kinematic singularities, which are specific robot configurations that can cause control issues. By allowing for gradual changes in joint angles, interpolation aids in avoiding these problematic scenarios. Moreover, interpolation is essential for effective path planning, controlling the speed of movement between positions within safe limits. By adjusting the pace and acceleration of the robot's motions, interpolation minimizes mechanical stress on the robot's components, reducing the risk of damage. Furthermore, interpolation provides detailed control over the robot's motion profile, enabling precise movements crucial for tasks such as assembly lines or pick-and-place operations.

The interpolation equation works by calculating intermediate poses between two given poses of the robot using this equation:

$$intermediatePose = startPose \times ((steps - i) \times stepSize) + endPose \times (i \times stepSize) \quad (2)$$

Where ***'i'*** iterates from ***0*** to steps. The *stepSize* is calculated as $\frac{1.0}{steps}$.

## IV. Results and Discussion

The image processing system effectively transformed a drawn line into Cartesian coordinates, which were then used to guide a robotic arm along the corresponding path. Initially, the input image was read and resized to a standard size of 480x480 pixels to ensure consistent processing. The image was then converted to grayscale, simplifying the analysis by emphasizing light intensity variance. Default parameters for edge detection were set, enhancing edge detection and noise reduction. The resulting binary image highlighted the edges of the drawn line, facilitating further processing stages.

In the next stage, the binary image was thresholded to isolate the white curve. Contours representing the boundary of the white area were found, and points in close proximity within each contour were merged to reduce the number of points and ensure clean visualization. Circles were drawn at the merged points' locations on a blank canvas, creating an image that approximated the initial curve with distinct points. The final set of Cartesian coordinates was scaled to match the robot's workspace dimensions, ensuring accurate physical representation of the path.

The shape and form of this image processing implementation were intentionally chosen to test the resilience of the system. The path was selected with several curves and different line widths so as to evaluate how well program could handle imperfections such as noise or irregularities in the drawn line. We, therefore,

wanted to establish if the algorithm was successful when it came to admitting real-life imperfect inputs which ensure that it is reliable in a number of circumstances.

*1) ROS2 and Moveit:*

For ROS2 testing, we employed a KUKA robotic arm and scaled the coordinates to fit the robot's workspace. Using a ROS2 action server, we implemented the coordinates and executed the movement commands. A C++ script utilizing the MoveIt interface[14] facilitated precise movement of the robotic arm. The `move_to_position` function set joint values and generated a motion planning plan using the MoveGroupInterface, ensuring smooth and efficient motion planning and execution. The ROS2 framework provided robust real-time communication and control, confirming the system's ability to translate visual data into accurate robotic actions. This integration demonstrated the feasibility of our approach, with results visualized in Figure 5.

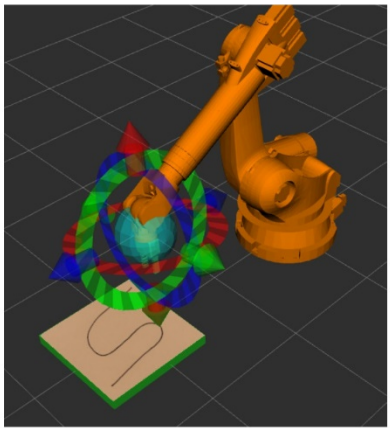

Fig 5. Ros2 Implementation

*2) RoboDK:*

In RoboDK testing, we simulated the robotic arm's movements using a Python script and the RoboDK Python API. The script mapped image coordinates to the robot's coordinate system, interpolated motions between points, and controlled the UR10 robotic arm. The `interpolate_motion` function calculated intermediate poses, ensuring smooth transitions between start and end positions. This virtual testing validated the planned trajectories' accuracy and feasibility, with the setup illustrated in Figure 6. This testing phase confirmed the effectiveness of our approach in both virtual and physical environments, providing a comprehensive assessment of the system's capabilities.

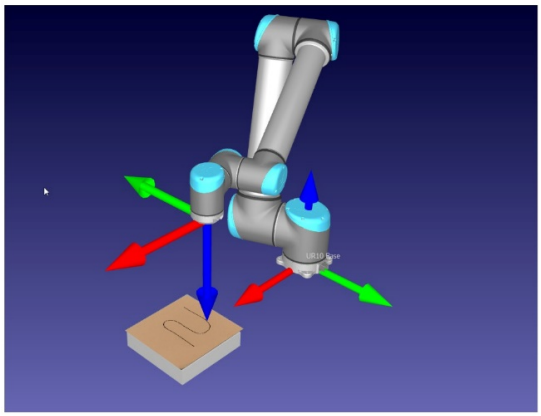

Fig 6. Robodk Implementation

*3) Comparison of ROS2 and RoboDK*

ROS2 is a robust software framework designed for handling complex, real-time robotics applications. It excels in environments that require advanced motion planning, real-time communication, and integration with various sensors and actuators. In practical implementations, it has been shown to be effective in managing complex tasks and dynamic scenarios due to its modular architecture and extensive libraries of ROS2. This system's wide adoption in both industry and research shows how flexible it can be since it was developed for use case high performance use where precise control and real time are important.

Contrastingly, RoboDK specializes simulation and offline programming for robotic systems. An intuitive interface is provided by the company that allows virtual testing as well as validation of robot motions before they are physically deployed. RoboDK is easy to use for motion planning and trajectory optimization especially in educational setups or prototyping contexts. However, while not very suitable for real-time applications compared with ROS2, RoboDK continues to be an excellent tool for validating and refining robot trajectories thereby conferring significant merit on simulation-based exercises at early-stage development.

For our system, ROS2 appears to be better option because of its versatility as well as ease of integration with custom control boards like ESP32. The modularity and many hardware components that ROS 2 supports make it applicable in the context of university projects and experimental setups. Because it can easily be paired up with custom controllers, such flexibility is desirable when developing new systems for trials. In contrast, RoboDK mostly connects to particular robot controller boards by providing native compatibility with a range of leading brands like ABB, FANUC, KUKA and Universal Robots. Although RoboDK excels in integrating these established controllers together into one cohesive whole, it does not naturally support consumer boards such as ESP 32. Even though some solutions may be found to connect RoboDK with non-standard controllers, the built-in adaptability and multiple device compatibility offered by ROS 2 are much

more in line with the requirements and objectives of our system.

*4) Image Processing Results*

The image processing results effectively isolated and converted the drawn line into a set of Cartesian coordinates. This robust approach ensured that the actual path was accurately captured for further analysis. To enhance the visualization and interpretation of the processing pipeline, several improvements are recommended.

Firstly, incorporating color-coded points before and after scaling would significantly enhance the clarity of the transformation process. Using different colors to represent the original and scaled coordinates can provide immediate visual feedback on the accuracy of the scaling operation. For example, plotting the original coordinates in blue and the scaled coordinates in red would allow for a clear comparison between the initial and transformed data.

Additionally, overlaying the processed image with the original image can help verify that the points accurately follow the drawn line. This overlay can be further enhanced by adding labels or markers to indicate key points, such as the start and end of the path. This would not only improve visual verification but also help in identifying any discrepancies between the path planned and the actual path.

To further refine the image processing results, using dynamic visualization tools can offer interactive exploration of the processed image and coordinates. Implementing features like sliders to adjust threshold values and kernel sizes in real-time can assist in fine-tuning the image processing parameters for optimal results. This interactive approach can be invaluable in adjusting parameters to achieve the best possible fit between the drawn line and the processed path.

Lastly, integrating these visualizations into a unified interface, where users can view the raw image, processed image, and the final coordinates, will facilitate a comprehensive understanding and debugging of the image processing pipeline. Such an interface would not only provide a clearer picture of the data but also improve the accuracy and reliability of the robotic path planning by ensuring that all steps of the image processing and coordinate transformation are thoroughly validated and refined.

By implementing these enhancements, the visualization and interpretation of the processed image and coordinate scaling can be significantly improved, leading to more accurate and reliable robotic path planning.

*5) Analysis of Path Accuracy and Efficiency*

In the evaluation of different point groups used for path planning, we focused on assessing ***fit accuracy*** to understand how well the generated paths align with the actual paths defined by binary images. Fit accuracy is a critical metric for determining the precision of a generated trajectory and its adherence to the desired path.

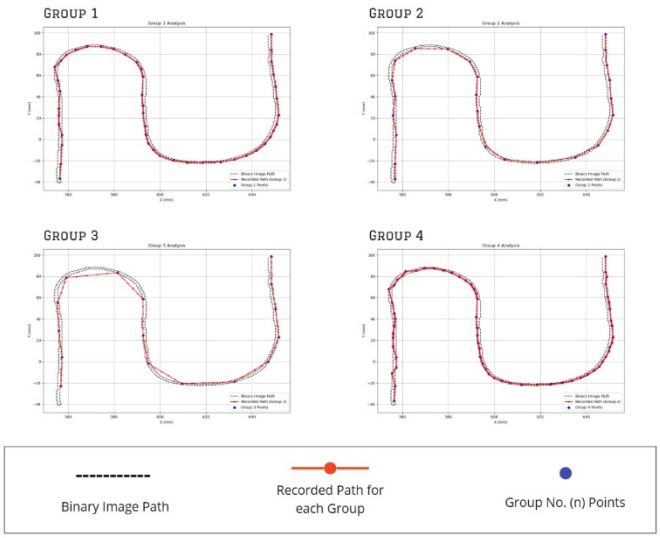


Fig 7. Drawn Paths vs. Robot Paths: Accuracy and Coverage Comparison.

To quantify the fit accuracy, we employed the following methodology:

1. **Nearest Distance Calculation**: For each generated point $(x_i, y_i)$ along the robot's path, we compute the Euclidean distance to each actual point $(x_i, y_i)$ from the binary image path:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (3)$$

   Here, $d_{ij}$ represents the distance between the $i^{th}$ generated point and the $j^{th}$ actual point. This step involves determining how close each generated point is to any actual point on the true path.
2. **Minimum Distance for Each Generated Point**: For each generated point $x_i, y_i$ we select the minimum distance to any actual point:

$$distances_i = min_j d_{ij} \quad (4)$$

   This ensures that we are measuring the shortest distance between each generated point and the closest actual point on the path.
3. **Average Fit Accuracy**: The final fit accuracy is obtained by averaging these minimum distances across all generated points:

$$Fit\ Accuracy = \frac{1}{N}\sum_{i=1}^{N} distances_i \quad (5)$$

   Where $N$ denotes the total number of generated points. This average value reflects how well the entire set of generated points matches the actual path.

*a) **Analysis of Results***

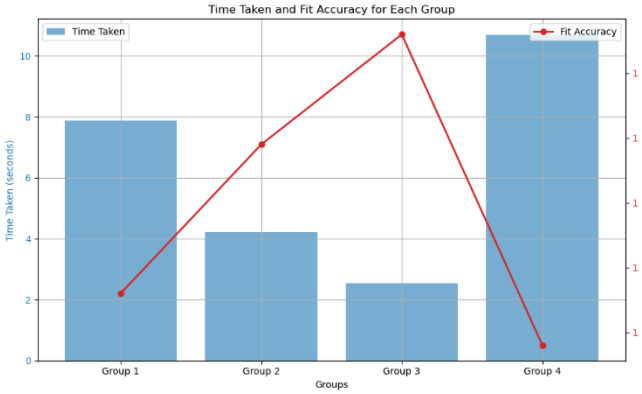


Fig 8. Time Taken and Fit Accuracy for Each Group.

The results of our experiments highlight the impact of the number of points on the fit accuracy and the overall performance of the path planning system. We analyzed four different groups with varying numbers of points:

- Group 1: This group, consisting of 46 points, yielded a fit accuracy of 1.06 mm and took 7.87 seconds to complete the path. The relatively high number of points provided a good balance between accuracy and execution time.
- Group 2: With 26 points, this group achieved a fit accuracy of 1.29 mm and a total execution time of 4.23 seconds. Although it had fewer points compared to Group 1, resulting in slightly lower accuracy, it completed the path more quickly.
- Group 3: Featuring only 16 points, this group had a fit accuracy of 1.46 mm and the shortest execution time of 2.54 seconds. The reduced number of points led to less accurate path generation, reflecting a trade-off between path detail and speed.
- Group 4: This group used 64 points, the highest number of points tested, achieving the best fit accuracy of 0.98 mm. However, it also had the longest execution time at 10.68 seconds. The increased number of points allowed for finer resolution and more precise alignment with the actual path but required more processing time.

From these results, it is clear that increasing the number of points generally improves fit accuracy by providing a more detailed representation of the path. Group 4's superior accuracy demonstrates how additional points can enhance path fidelity. However, this improvement in accuracy comes with increased computational time, as observed with the longer execution time for Group 4.

Conversely, fewer points, as seen in Group 3, result in faster execution times but with lower fit accuracy. This indicates a trade-off between the level of detail captured and the computational efficiency of the path planning process.

In summary, the number of points significantly influences the fit accuracy of the generated path. While a greater number of points enhances accuracy, it also increases the time required to complete the path. The choice of point density should therefore be guided by the specific needs of the application, balancing the requirements for precision and efficiency.

## V. Conclusions

In this initiative, we directed efforts towards constructing a computer vision system capable of converting visual data into precise instructions for robotic operations. We began by applying sophisticated techniques in image processing to refine and decode the visuals, ensuring that the paths depicted on paper were replicated accurately by the robotic arm. By translating these images into coordinate points and adjusting their scale to the operational parameters of the robot, we achieved remarkable precision. Our system incorporates the Robotics Operating System (ROS) and RoboDK, both known for their effectiveness in managing intricate robotic functions. Extensive tests and calibration, especially in regulating the robot arm's motion and path planning, allowed us to reach an impressive level of accuracy. This framework not only boosted the functionality of our robotic system but also expanded our capabilities concerning accuracy and practical application. The successful methodologies we adopted have broad implications for future enhancements in various robotic systems and are instrumental in advancing autonomous robotics.

Currently, the program only processes 2D images, thus ignoring variations in elevation or other three-dimensional aspects. Our goal is to augment the system with support for 3D drawing. This enhancement will enable the robot to navigate and interact within a more complex environment, elevating its adaptability and operational efficiency. By addressing these future enhancements, we aim to significantly broaden the scope and applicability of our system.

The integration of 3D processing capabilities will enhance the robot's ability to handle more complex tasks, making it suitable for a wider range of industrial applications. As we continue to develop and refine this system, we anticipate it will play a crucial role in advancing the field of autonomous robotics, offering innovative solutions to real-world challenges[15].

## References

[1] M. T. Mason, “Creation myths: The beginnings of robotics research,” IEEE Robot Autom Mag, vol. 19, no. 2, pp. 72–77, 2012.

[2] A. Nayyar and A. Kumar, A roadmap to industry 4.0: Smart production, sharp business and sustainable development. Springer, 2020.
[3] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path Planning and Trajectory Planning Algorithms: A General Overview," Mechanisms and Machine Science, vol. 29, pp. 3–27, Mar. 2015, doi: 10.1007/978-3-319-14705-5_1.
[4] Abdulsahib Ahmed, "Path Following and Motion Control for Articulated Frame Steering Mobile Working Machine Using ROS2," Master of Science Thesis, Tampere University, 2023.
[5] R. Ghabcheloo, "Coordinated Path Following," PhD thesis, 2007.
[6] P. Zhou, R. Peng, M. Xu, V. Wu, and D. Navarro-Alarcon, "Path Planning with Automatic Seam Extraction over Point Cloud Models for Robotic Arc Welding," Nov. 2020, [Online]. Available: http://arxiv.org/abs/2011.11951
[7] N. K. Al-Karkhi, W. T. Abbood, E. A. Khalid, A. N. Jameel Al-Tamimi, A. A. Kudhair, and O. I. Abdullah, "Intelligent Robotic Welding Based on a Computer Vision Technology Approach," Computers, vol. 11, no. 11, Nov. 2022, doi: 10.3390/computers11110155.
[8] R. Szeliski, Computer vision: algorithms and applications. Springer Nature, 2022.
[9] K. Bayoudh, R. Knani, F. Hamdaoui, and A. Mtibaa, "A survey on deep multimodal learning for computer vision: advances, trends, applications, and datasets," Vis Comput, vol. 38, no. 8, pp. 2939–2970, 2022.
[10] L. F. P. Oliveira, A. P. Moreira, and M. F. Silva, "Advances in agriculture robotics: A state-of-the-art review and challenges ahead," Robotics, vol. 10, no. 2, p. 52, 2021.
[11] N. D. Lynn, A. I. Sourav, and A. J. Santoso, "Implementation of Real-Time Edge Detection Using Canny and Sobel Algorithms," IOP Conf Ser Mater Sci Eng, vol. 1096, no. 1, p. 012079, Mar. 2021, doi: 10.1088/1757-899x/1096/1/012079.
[12] "OpenCV: Smoothing images." [Online]. Available: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html
[13] "OpenCV: Morphological Transformations." [Online]. Available: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html
[14] "Move Group C++ Interface — moveit_tutorials Kinetic documentation." [Online]. Available: http://docs.ros.org/en/kinetic/api/moveit_tutorials/html/doc/move_group_interface/move_group_interface_tutorial.html
[15] R. Alterovitz, S. Koenig, and M. Likhachev, "Robot planning in the real world: Research challenges and opportunities," AI Mag, vol. 37, no. 2, pp. 76–84, 2016.