RWU Hochschule Ravensburg-Weingarten University of Applied Sciences

Degree:
Master Mechatronics

Scientific Project:
DRAWING WITH A 6-AXIS ROBOT

Guided by:

Dr. Prof. Konrad Wöllhaf
M.Sc. Vivien Glönkler

Semester:

SS2023

Authors:
Ajith Kumar Muthuswamy - 36129
Master Mechatronics

Atharva Mahindrakar - 36199
Master Mechatronics

Marel Alejandro Medina Morales - 36118
Master Mechatronics

Pranav Pandharpatte - 36209
Master Mechatronics

July 2, 2023

Abstract

This scientific report presents the results and analysis of a comprehensive project comprising three distinct tasks conducted with the KR4 R600 robot. The primary objective of the project was to explore the robot's capabilities in executing various drawing tasks, leveraging simulation software, programming techniques, and image processing methodologies.

The first task involved drawing simple figures, which required the development of precise motion plans using KUKA Sim Pro software. Through simulation, the robot's movements were optimized and validated before transferring the files to the main robot for autonomous execution. This task served as a foundation for understanding the robot's kinematics and programming capabilities.

In the second task, the focus shifted to replicating the RWU logo using the same robot. Building upon the knowledge gained from the previous task, meticulous programming was employed to precisely trace the logo's intricate design. Simulations were conducted to ensure accuracy and consistency between the virtual representation and the physical execution. The successful reproduction of the RWU logo showcased the robot's ability to recreate complex patterns with high fidelity.

The third task involved drawing custom images provided as input to the robot. To accomplish this, a Python code was developed to process the image, extract the necessary point information, and generate source and data files. These files were then imported to the main robot, enabling it to autonomously execute the drawing task. This task highlighted the integration of image processing techniques and programming to enable the robot to interpret and recreate custom artwork.

Throughout the project, careful attention was given to factors such as precision, safety, and efficiency. The findings from each task contributed to the field of robotic automation, demonstrating the versatility and potential of the KR4 R600 robot in executing diverse drawing tasks. The successful completion of the project underscores the importance of simulation, meticulous programming, and image processing methodologies in achieving accurate and reliable results.

This scientific report provides insights into the capabilities of the KR4 R600 robot for drawing tasks, offering valuable contributions to the field of robotic automation and paving the way for future advancements in robotic artwork and creative applications.

Contents

List of Figures

1 Introduction

Robotic automation has witnessed remarkable advancements in recent years, revolutionizing various industries and applications. One area of interest lies in the field of artistic endeavors, where robots are utilized to execute intricate drawings and reproduce complex images with precision and efficiency. This scientific report presents the findings of a project that aimed to explore the capabilities of the KR4 R600 robot in executing drawing tasks of increasing complexity.

The project consisted of three distinct tasks, each focusing on different aspects of robotic drawing. The first task involved the creation of simple figures using the KR4 R600 robot. The objective was to understand the robot's capabilities in executing basic drawings while optimizing motion plans and ensuring accurate reproduction. To accomplish this, simulations were conducted using the KUKA Sim Pro software, enabling the fine-tuning of motion paths and programming parameters. Once the simulations were deemed successful, the drawing files were transferred to the main robot for autonomous execution, highlighting the seamless integration between simulation and real-world implementation.

Building upon the knowledge gained from the initial task, the second task delved into reproducing the RWU logo, a more intricate and recognizable design. This task aimed to test the robot's ability to accurately trace complex patterns and reproduce them with precision. Extensive programming and meticulous coordination were required to ensure that each stroke and curvature of the logo was faithfully replicated. The simulation phase played a crucial role in validating the programming parameters and optimizing the robot's movements. The successful reproduction of the RWU logo showcased the robot's capability to execute intricate designs with high fidelity.

The third task focused on expanding the robot's repertoire by enabling it to draw custom images provided as input. To accomplish this, a Python code was developed to read the image, extract relevant data points, and generate source and data files. These files contained the necessary information to guide the robot in accurately reproducing the custom artwork. The integration of image processing techniques and programming further demonstrated the versatility and adaptability of the robot in executing a wide range of drawing tasks.

Through these tasks, the project aimed to shed light on the potential of the KR4 R600 robot as a creative tool in the field of robotic drawing. The successful completion of each task highlighted the importance of meticulous programming, simulation, and image processing methodologies in achieving accurate and reliable results. The findings of this project contribute to the ongoing exploration of robotic automation in artistic applications and provide insights into the capabilities of the KR4 R600 robot for executing diverse drawing tasks.

In the following sections of this report, we will delve into the methodologies, procedures, and results of each task, outlining the challenges encountered, the approaches taken, and the lessons learned. The outcomes of this project have implications for the broader field of robotics and automation, particularly in the realm of creative endeavors, and pave the way for further advancements in robotic artwork and related applications.

2 Task 1 - Creation of simple line drawings with linear and circular movements.

The initial task involves utilizing a robot to draw figures consisting of a rectangle, a scalene triangle, and a circle on a white paper. To ensure a smooth execution of the task, a simulation is first conducted using Kuka Sim Pro software. Within the simulation environment, a virtual robot and a white paper surface are set up, and the robot's parameters, such as arm length and joint limits, are adjusted to match those of the real robot. A virtual marker is added, configured with appropriate properties, and a program is developed to guide the robot's movements, specifying the coordinates and orientations for each shape. The simulation enables a comprehensive understanding of the robot's functionality and provides an opportunity to verify the accuracy of the drawings. Once confident with the simulated results, the task progresses to the use of a real robot. In this stage, the workspace is prepared by ensuring a clear area and assembling the necessary materials, including the robot, white paper, and marker. A program is then executed, guiding the robot's movements to draw the shapes on the physical paper. Safety precautions are diligently followed, and once the task is completed, the drawn figures are reviewed on the paper. This introductory task serves as a crucial step in acquainting oneself with the robot's capabilities and prepares for more advanced tasks ahead.

2.1 Introduction to KUKA SimPro

KUKA SimPro is an advanced simulation software developed by KUKA, a renowned leader in the field of industrial robotics and automation solutions. With its powerful features and capabilities, KUKA Sim enables users to design, program, and optimize robotic systems in a virtual environment, providing significant benefits throughout the entire development and deployment process. One of the key features of KUKA Sim is its ability to create a realistic 3D virtual environment where users can build digital representations of their robotic work cells. This includes the robot itself, along with fixtures, tools, and workpieces. By accurately simulating the robot's movements and actions within this virtual environment, users can gain confidence in the performance and behavior of their robot programs before implementing them in the real world. Collision detection is another critical aspect of KUKA Sim. The software incorporates advanced algorithms that detect potential collisions between robot components, workpieces, and the work cell environment. This early detection of collisions allows users to identify and resolve issues promptly, avoiding costly damages and ensuring the safety of the system during actual operations.
KUKA Sim also offers reachability analysis, which helps users determine whether the robot can reach desired positions within its workspace. By optimizing robot positioning and considering reachability constraints, users can enhance cycle times and improve overall system efficiency. One of the significant advantages of KUKA Sim is its support for offline programming. This means that users can develop and test robot programs without requiring access to a physical robot. This offline programming capability saves time and resources by eliminating the need for constant interaction with the robot during the programming phase. It allows programmers to refine their

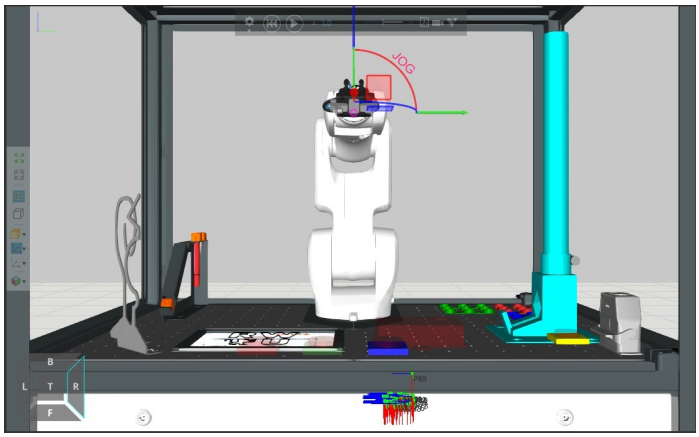

Figure 1: *Front view of Robot in KUKASimPro*

code, test different scenarios, and optimize robot movements more efficiently.
The visualization and analysis tools provided by KUKA Sim greatly contribute to the software's effectiveness. Users can visually observe and analyze robot movements, trajectories, and interactions within the virtual environment. This visual feedback aids in identifying potential issues, optimizing robot paths, and improving overall system performance. The benefits of using the KUKA Sim are numerous. First and foremost, it significantly reduces development time and costs by allowing users to simulate and validate robotic systems in a virtual environment. This eliminates the need for extensive physical testing and iterations, resulting in faster and more cost-effective development processes.

2.2 Teaching Tool and Base

Teaching the TCP and base for a KUKA robot is vital to achieve precise positioning, coordinate system alignment, calibration, compensation, task flexibility, and accurate programming. These factors contribute to the overall performance, reliability, and versatility of the robot in various applications. Teaching the tool and base is the first step to perform this task on simulation or on real robot.

2.3 Teaching The Tool using Four Point method

Measuring the tool, also known as teaching the tool, involves determining the position of the reference point of the tool, such as the tip of a welding tool. One commonly used method is the four-point method. This method involves aligning the tool tip to an arbitrary but fixed point from four different positions.By aligning the tool tip to these four positions, the reference point is consistently at the same distance from each respective position. Since the position of the robot flange is known from the position of the robot axes, there are four points in space (P1 to P4) that have the same distance from one reference point (PR). The reference point PR lies in a plane perpendicular to the connecting line between each pair of points P1 to P4 and is located in the middle between these pairs. To determine the reference point PR, intersection lines can be calculated for these planes. The point with the smallest distance between the intersection lines represents the previously unknown reference point PR. The tool

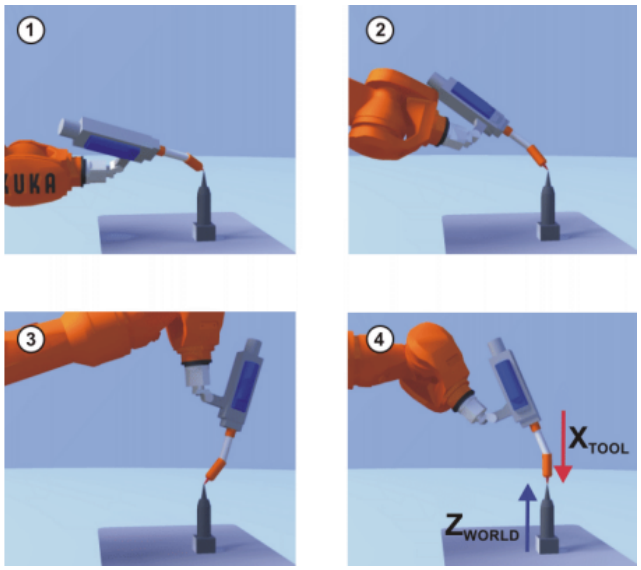

Figure 2: *4 point method*

center point (TCP) is then obtained by calculating the vector difference between one of the points P1 to P4 and the reference point PR.
The accuracy of the teaching process depends on the distance between the intersection lines. Smaller distances indicate more precise teaching. If the points P1 to P4 are very close together, the calculation of the reference point may be imprecise or not possible due to the poor condition of the system of equations.Knowing the exact position of the tool center point (TCP), such as the tip of a welding tool, is important for the robot's movements and rotations in Cartesian coordinates. If the TCP is known, the robot can be easily rotated around a point and moved in a specific direction. However, if the tool is attached at an angle to the flange, it may require iterative adjustments using Euler angles or successive movements in all spatial directions to accurately determine the TCP.
Measuring the tool is particularly important for teaching robot positions and executing programs that involve Cartesian coordinate movements, especially rotations. Having the precise position of the TCP is crucial for accurate calculations and orientation control.

In our case the tool was already taught on the robot so the resulting coordinates were imported into Kuka Sim. This allows you to use the taught tool for further tasks in the simulation. The tool used was a STABILO Trio marker, which was equipped with an attachment to facilitate proper gripping.

2.4 Steps in Configuring the tool load on a Kuka robot

1. Make sure that no program is selected.
2. Login as Administrator or Expert.
3. Navigate to Start up -> Tool/Base Management.

Figure 3: *Tool (STABILO Trio marker)*

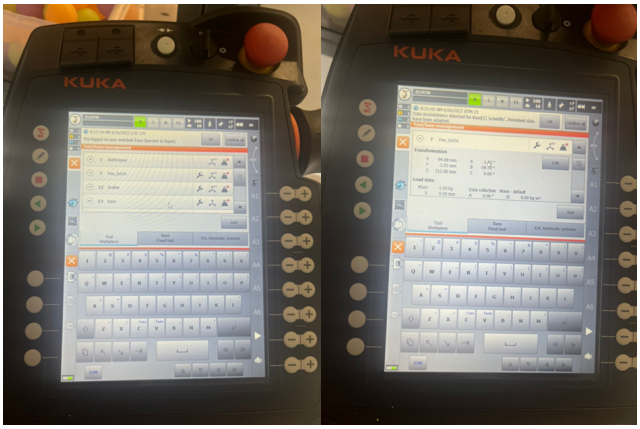

Figure 4: *(left)Tool Management Window (right)Coorinates of the tool*

2.5 Teaching the base

Base calibration plays a vital role in robot programming as it involves the creation of a coordinate system that is relative to the world coordinates. This calibration process provides a crucial reference point for jog motions and programmed positions, ensuring precise and accurate movements of the robot. The calibration is typically performed by determining the coordinate origin and defining the coordinate axes, which establishes the framework for subsequent programming tasks. One significant advantage of base calibration is the ability to jog the tool center point (TCP) along the edges of the work piece. With the coordinate system properly calibrated, the robot can smoothly

Methods	Description
3-point method	1. Definition of the origin 2. Definition of the positive X axis 3. Definition of the positive Y axis (XY plane)
Indirect method	The indirect method is used if it is not possible to move to the origin of the base, e.g. because it is inside a workpiece or outside the workspace of the robot. The TCP is moved to 4 points whose positions relative to the base that is to be calibrated are known and the coordinates of which must be known (CAD data). The robot controller calculates the base from these points.
Numeric input	Direct entry of the values for the distance from the world coordinate system (X, Y, Z) and the rotation (A, B, C).

Figure 5: *Methods to teach the Base*

follow the contours and edges of the work surface, enabling precise positioning and control during various tasks such as material handling, welding, or assembly. This capability is particularly valuable in applications where intricate manipulation or intricate path following is required. Another advantage of base calibration is the establishment of a reference coordinate system. By referencing taught points and programmed positions to the calibrated base coordinate system, the robot can consistently and accurately execute tasks in relation to the selected coordinate system. This eliminates ambiguity and ensures that the robot's movements are aligned with the desired reference frame, enhancing the overall accuracy and reliability of the system. We have used the 3 point method to teach the base

2.6 3 point Method

1. In the main menu, select Start-up > Calibrate > Base > 3-point.
2. Assign a number and a name for the base. Confirm with Next.
3. Enter the number of the tool whose TCP is to be used for base calibration. Confirm with Next.
4. Move the TCP to the origin of the new base. Press the Calibrate soft key and confirm the position
5. Move the TCP to a point on the positive X axis of the new base. Press Calibrate and confirm the position with Yes.
6. Move the TCP to a point in the XY plane with a positive Y value. Press Calibrate and confirm the position with Yes.
7. Press Save.
8. Close the menu.

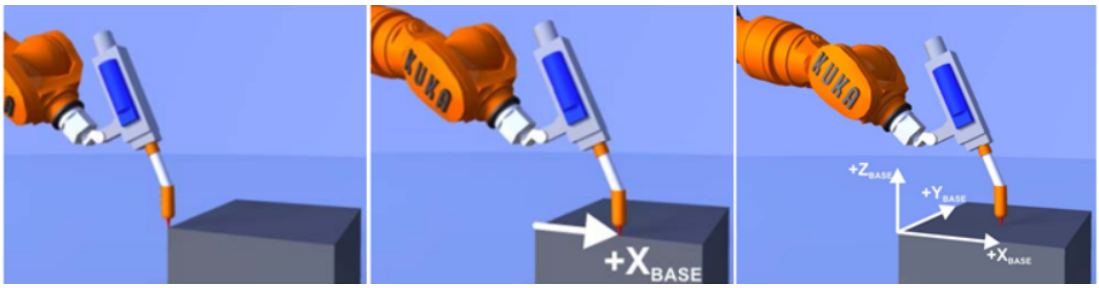

Figure 6: *(left) origin, (centre) X axis, (right)XY Plane*

2.7 Gripper

The gripper plays a vital role in facilitating the interaction between a robot and its surrounding objects. KUKA, renowned for its robotic systems, offers a diverse range of gripper options tailored to meet the specific needs of different applications. These grippers enable KUKA robots to grasp, manipulate, and release objects effectively. Whether it's electric grippers, powered by motors for precise control and delicate handling, or pneumatic grippers, utilizing compressed air for rapid operation and robust gripping forces, KUKA provides versatile solutions. By offering a variety of gripper options, KUKA empowers users to optimize their robots' capabilities and adapt them to a wide array of tasks and industries. For our project we use the pneumatic gripper-
Pneumatic Grippers:
Pneumatic grippers use compressed air to actuate the gripping mechanism. They are known for their fast operation, robustness, and high gripping forces. Pneumatic grippers are often used in applications that require rapid gripping and releasing of objects.

We use pneumatic gripper with appropriate attachment which helps the robot to pick up the pen properly. The attachment used is shown in the figure below-

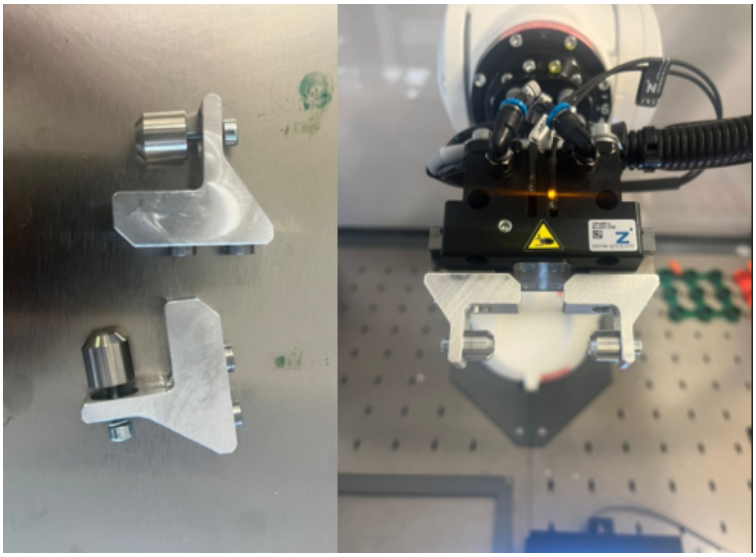

Figure 7: *Gripper*

2.8 Results

The subsequent section provides a detailed overview of the obtained results. In the initial phase, each image was meticulously drawn in isolation as part of the trial process, allowing for individual evaluation. Subsequently, an amalgamation of all three drawings was accomplished by commencing from a common starting point of the robot. This approach facilitated a comprehensive and systematic comparison, enabling a more in-depth analysis of the outcomes.

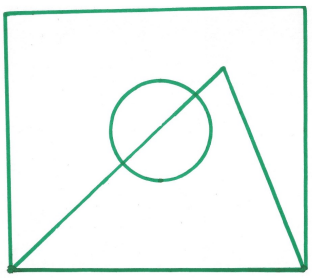

Figure 8: *Final Result*

2.9 Challenges Encountered

2.9.1 Using CIRC command

Initially, we encountered challenges while utilizing the command, which led to improper results. The confusion arose primarily from the order of assigning the start point, auxiliary point, and end point. It became evident that in order to draw a perfect circle, it was imperative to assign the start point and end point before the auxiliary point. Failure to adhere to this sequence resulted in the formation of curves that did not align to form a complete circle. To provide clarity, the correct steps for drawing a circle are outlined below.
Steps to draw a circle-
1. Choose the SCIRC command.
2. Choose the start point
3. Choose the endpoint and use touch up command
4. Choose the Auxiliary point and touch up Auxiliary command.

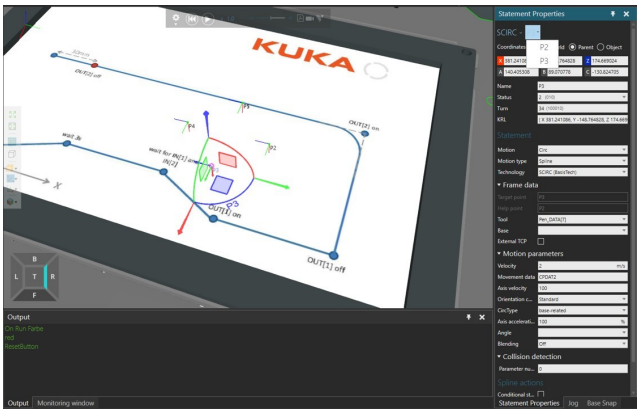

Figure 9: *Drawing a Circle*

2.9.2 Handling the Marker

To ensure consistent execution and streamline the process, two common tasks were identified: picking up the pen from its stand and returning the pen to its original position. To facilitate these tasks in each program, dedicated functions were defined: PenPickup() and PenReturn(). These functions are called within every program to ensure the pen is properly picked up and returned to its designated position, thus providing a standardized and efficient workflow for all subsequent drawing tasks. By encapsulating these tasks as functions, the code achieves modularity and reusability, contributing to a more structured and manageable programming approach. During the project, we encountered an issue with the gripping of the marker, where it was not positioned correctly, leading to improper closure of the gripper. As a result, we had to manually adjust the robot's position to ensure the marker was properly held. By carefully aligning the robot to the exact position and finalizing the point, we were able to address the issue and proceed with the drawing task effectively.

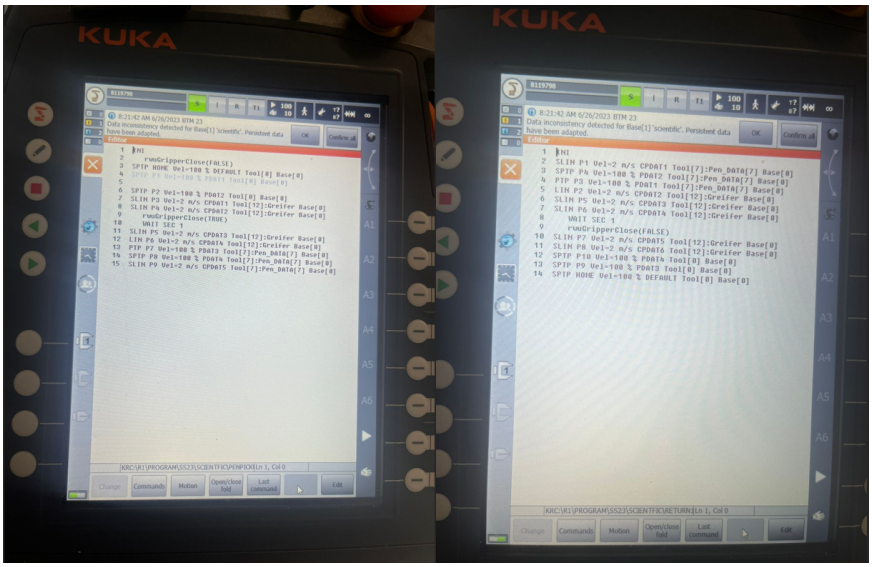

Figure 10: *(left)PenPickup() and (rigth)PenReturn() functions*

2.9.3 Determining the Z Co-ordinate

The process of determining the Z coordinate at which the marker makes contact with the paper proved to be challenging. Initially, a random point on the simulation was selected to represent this contact point, but the simulation's rwuTraceOn() function did not require actual contact between the tool and the base, leading to difficulties in finding an appropriate point. Another approach involved measuring the pen's length using a vernier caliper and adjusting the coordinate accordingly, but this method yielded less accurate results. Ultimately, the correct coordinate was identified by observing the real robot as it touched the paper with the marker. This precise co-ordinate, recorded in the dat file, was subsequently used in further programs. While simulations are useful, accurately replicating physical interactions and measurements can pose challenges, making real-world observations and measurements more reliable in such cases.

Figure 11: *Vernier Caliper Method*

3 Task 2 - Creation of the RWU Logo

The objective of this task was to utilize the KUKA Sim Pro software to accurately replicate our college logo, "RWU," using a robotic arm. The project involved a two-step process: simulating the program on KUKA Sim Pro and subsequently transferring the generated source (src) and data (dat) files to the physical robot. The robot was then programmed to draw the logo on a specified paper (base) surface using a designated pen (tool), mirroring the process employed during the first task.
This project exemplified the successful integration of simulation software and physical robotics, showcasing the ability to simulate and transfer complex programs to real-world applications. By achieving a faithful reproduction of the RWU logo, we demonstrated the feasibility and potential of utilizing robotic systems for artistic and precision tasks.

3.1 Tool and Base

The purpose of our project was to replicate our college logo, "RWU," using the KUKA Sim Pro software and the same tool and base as employed in the initial task. By utilizing the same tool and base, we were able to significantly reduce the time required for predefining new configurations. Additionally, this approach facilitated the seamless transfer of data to the physical robot.
To achieve this, we began by simulating the logo program within KUKA Sim Pro, leveraging the previously established tool and base configurations. The utilization of a shared tool and base, coupled with the direct import of data, underscores the efficiency and effectiveness of our methodology. This approach significantly reduced setup time and enhanced the overall workflow, reinforcing the feasibility of utilizing robotic systems for repetitive tasks with consistent requirements.

3.2 Importing the RWU logo (dxf) file for base precision

To facilitate the simulation process, we opted to overlay the RWU logo image file onto the base, allowing us to trace the points and perform the tracing more conveniently. However, importing the RWU image file presented a significant challenge.
1. Open the "Import Geometry" feature in the software.
2. A file explorer window will appear.
3. Browse and select the desired DXF file from your computer.
4. Click on the "Import" button.
5. The "Import Model" window will open.
6. In the "Structure" option, choose either "Feature" or "Component" based on your requirements.
7. Select the appropriate option for the imported file's position and orientation.
8. Click on the "Import" button to proceed.
9. Position and adjust the imported model to the desired location within the simulation.

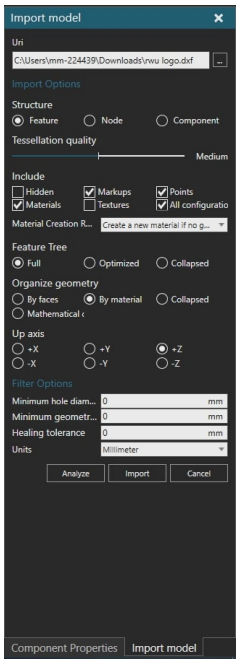

Figure 12: *Component Properties for Importing*

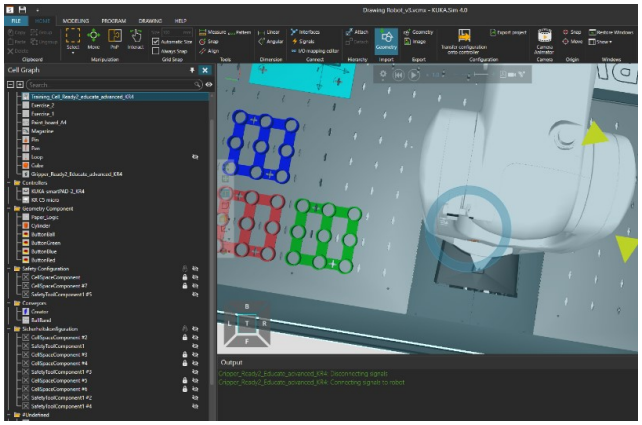

Figure 13: *Components of simulation*

Initially, we attempted to import the image as a DWG file, but encountered an error within the KUKA Sim Pro software. Subsequently, we explored various CAD file formats, employing a trial and error approach to identify a suitable format. Ultimately, we found that the DXF file extension yielded successful results, enabling us to load the image onto the KUKA Sim Pro simulation. Initially, the image

was randomly positioned beneath the robot. To achieve the desired simulation of the drawing process, we manually relocated the image to the intended base position, while simultaneously adjusting its size to ensure proper alignment with the tracing area. This method of overlaying the RWU logo image onto the base in KUKA Sim Pro simplified the tracing process by providing a visual guide. While encountering difficulties during the image import stage, our persistence and experimentation led

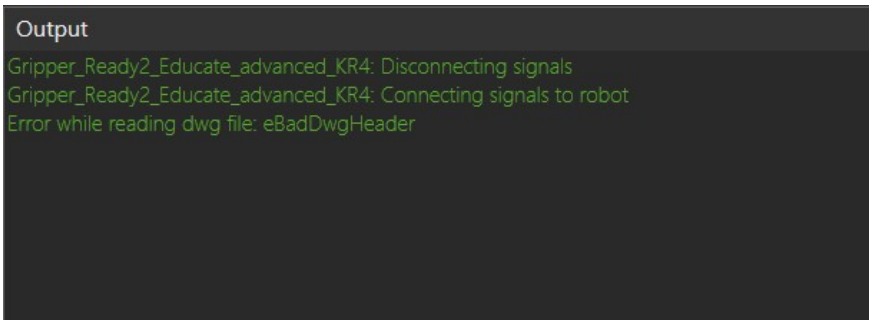

Figure 14: *Error when importing DWG file*

us to discover a compatible file format, demonstrating the importance of adaptability and problem-solving in the project.

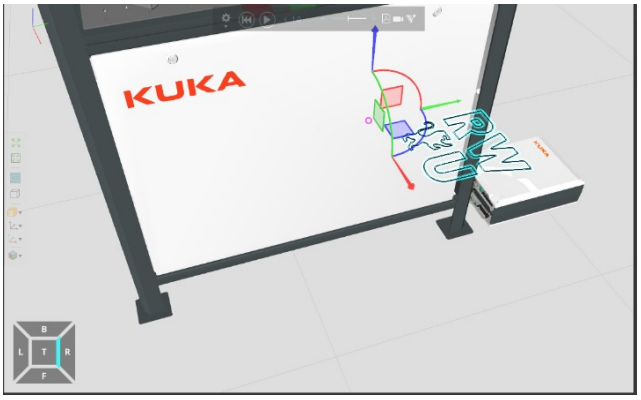

Figure 15: *Moving the logo to the desired position*

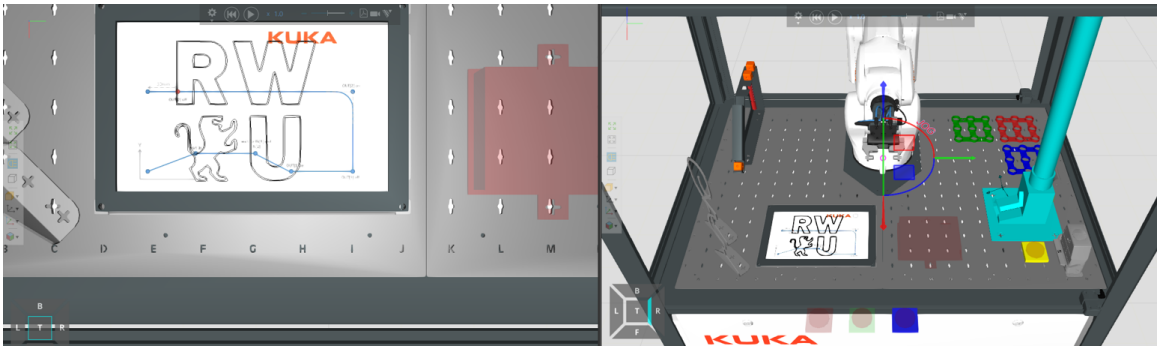

Figure 16: *Final position of the RWU logo*

3.3 Simulation program

In our project, we initiated the program by utilizing a series of commands, including "SLIN," "SCIRC," and "SPTP," to control the movement of the robotic arm. A total of 132 points were strategically defined to trace the complete logo based on the provided image superimposed on the base. Although the task itself was straightforward, it demanded a substantial amount of time due to the intricate nature of the logo. Each point encompassed specific coordinates in the x, y, and z axes, along with corresponding angles a, b, and c, ensuring precise positioning of the robot's end-effector.

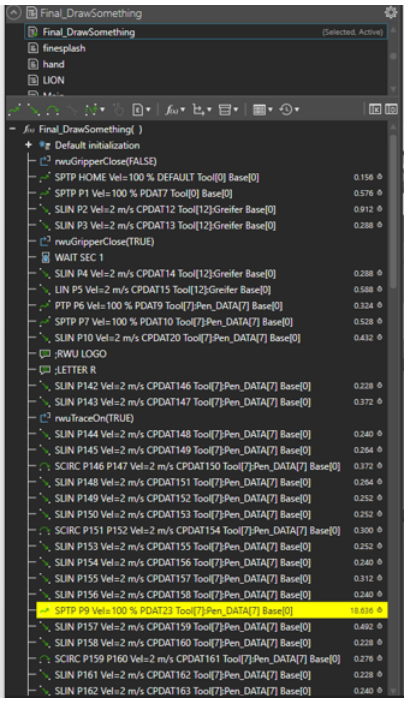

Figure 17: *Points described in the simulation*

The points drawn on the simulation are given below -

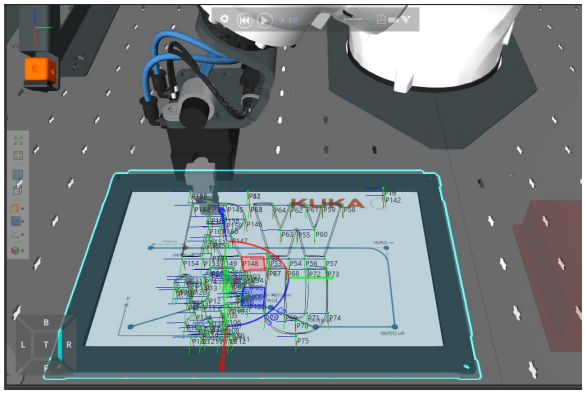

Figure 18: *Points described on the base*

It is noteworthy that, to draw the logo accurately, we encountered the requirement of lifting the tool and relocating it to the next letter of the logo, thereby continuing the drawing process seamlessly. This necessitated careful coordination and synchronization between the robot's movements and the transfer of the tool between the letters.

3.4 Results

After successful completion of the program, we proceeded to test the code on the physical robot. The entire logo was faithfully reproduced, mirroring the results obtained during the simulation phase. This outcome highlights the effectiveness and accuracy of the simulation, as well as the seamless integration between the virtual and real-world execution.

By meticulously defining the movement commands and executing them on the real robot, we achieved precise and consistent results, validating the feasibility and reliability of our approach in replicating complex drawings with the assistance of robotics.

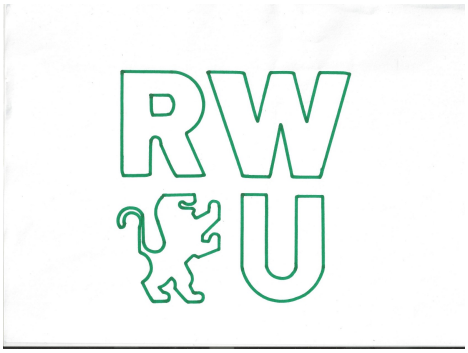

Figure 19: *Final Result (1)*

3.5 Challenges Encountered

During the execution of the project, we encountered an anomaly in the lion part of the logo, specifically related to the angle 'a' that was functioning correctly in the simulation. However, upon transferring the program to the physical robot, we observed that the tool pen was not functioning as intended, resulting in improper drawing of the lion image. This prompted us to conduct a thorough investigation to identify the root cause of the issue.

Upon inspection, we discovered that all the points within the lion part had a different angle 'a' when compared to the other letters. Recognizing the discrepancy, we promptly adjusted the angle 'a' for the affected points to align with the requirements of the physical robot. Subsequently, we executed the modified program, and this time the robot accurately reproduced the lion image on paper, resolving the issue.

9 The unintended occurrence took place while teaching the points and manipulating the gripper. Due to an inadvertent rotation of the gripper, the robot underwent an unintended rotation. This incident went unnoticed initially, as our focus was primarily

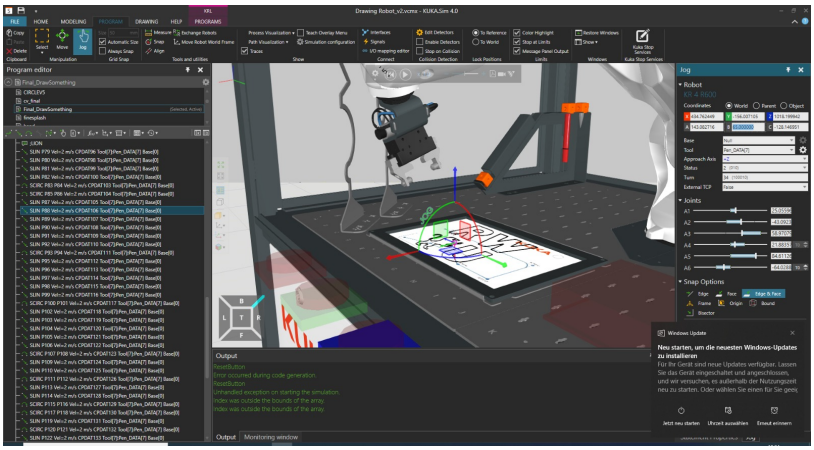

Figure 20: *Error position*

on teaching the points from a top-down perspective. As a consequence, the gripping position of the marker was not aligned properly, resulting in an inadequate closure of the gripper. However, upon recognizing the discrepancy, we manually readjusted the robot's position to precisely align the marker and finalize the desired points. This precautionary action was crucial to ensure accurate execution and control throughout the drawing process.

This occurrence underscored the need for diligent parameterization and highlighted the potential disparities that can arise between simulation and real-world execution. Moving forward, we will exercise greater caution and precision in defining the robot's parameters, thereby mitigating any potential discrepancies and optimizing the overall performance of the drawing program.

The task at hand posed a considerable time-consuming aspect, as it involved manually entering 132 points into the simulation. However, in Task 3, our objective is to streamline this process and minimize manual effort. To achieve this, we aim to automate the generation of points based on the input image. By implementing an algorithm that analyzes the image and extracts relevant data points, we can eliminate the need for manual input. This automation not only saves time but also reduces the potential for errors, ensuring a more accurate and efficient drawing process.

4 Task 3 - Schematic drawing of any motifs that are available as image data.

This part of the task presents an in-depth exploration of the generation of schematic drawings of motifs using image data and sophisticated image processing programs. The investigation encompasses the utilization of software tools such as KUKA SimPro 4.0, Python, OpenCV, and Visual Studio (IDE) to facilitate the entire process. The integration of image processing techniques and robotic systems has garnered substantial attention in various domains, including industrial automation, digital fabrication, and artistic endeavours. By harnessing the capabilities of these software tools, the acquired image data is meticulously analysed, enhanced, and transformed into a format compatible with robotic systems, enabling the precise replication of intricate motifs. The utilization of KUKA SimPro 4.0, a powerful robot simulation software, allows for the development and testing of robotic programs in a virtual environment. This ensures a streamlined workflow and facilitates thorough 2 of the proposed methodologies before their physical implementation.

Python, a versatile programming language, serves as a flexible and efficient platform for implementing complex image processing algorithms. The incorporation of OpenCV, a popular open-source computer vision library, further enhances the image processing capabilities, providing a vast array of functions for image enhancement, feature extraction, and pattern recognition. Visual Studio, a comprehensive integrated development environment, offers a seamless coding experience, facilitating efficient software development and integration. The findings of this study have far-reaching implications, showcasing the potential of integrating image processing programs with robotic systems for the automated generation of schematic drawings. The successful implementation of this interdisciplinary approach opens avenues for advancements in industrial automation, digital fabrication, and creative arts. By streamlining the workflow, reducing human intervention, and ensuring high precision, this research contributes to the advancement of image-based automation systems and their seamless integration with cutting-edge robotic technologies.

4.1 Algorithm Approach of the program

The **Ramer-Douglas-Peucker** algorithm is a technique used for line simplification in computational geometry. It aims to reduce the number of points required to represent a curve or polyline while preserving its overall shape. The algorithm works by iteratively identifying the point farthest from the line segment connecting the start and end points. If the distance between this point and the line segment is above a specified threshold, it is considered significant and retained. Otherwise, it is discarded, effectively simplifying the curve. This process is recursively applied to the resulting line segments until the desired level of simplification is achieved. The Ramer-Douglas-Peucker algorithm is widely used in various applications, such as map simplification, trajectory compression, and image processing, where reducing data complexity is crucial for efficiency and storage purposes.

Algorithm 1 Algorithm for Plotter Program

```
Import the required libraries
Open a file dialog box to select an image file
Check if the selected image fits within the dimensions of an A4 paper.
if Size of image is too big then
    Display an error message and exit
end if
Read and resize the image
Convert the image to grayscale
Apply Gaussian blur to reduce noise
Use the Canny edge detection algorithm to find edges
Find contours in the edge-detected image
Simplify the contours using the Ramer-Douglas-Peucker algorithm
Convert the simplified contour points to millimeters
Calculate the center of the A4 paper and adjust the contour points to fit within its dimensions
Create files with commands for a robotic arm
Display the original image and simplified contour points
Create a visualization of the contour points on an A4 paper
Display additional information about the image processing and file generation
Show the plots and close the figures
```

4.2 Code description

1. Import necessary libraries:
'cv2', 'numpy', 'matplotlib.pyplot', 'matplotlib.gridspec', 'sys', 'os', 'Tk', 'simpledialog', and 'messagebox' for image processing, numerical computing, plotting, system-related operations, file operations, and dialog box creation.
2. Open a file dialog to select an image file:
'askopenfilename' to allow the user to choose an image file with specified extensions from a specified initial directory.
3. Read and resize the selected image:
Reading the image using 'cv2.imread' and resizing it using 'cv2.resize' to double its size.
4. Check if the image fits in A4 size:
Calculating the image dimensions in millimetres and compare them to the A4 paper size. Display an error message and exit if the image is too large.
5. Convert the image to grayscale and apply Gaussian blur:
Converting the image to grayscale using 'cv2.cvtColor' and applying Gaussian blur using 'cv2.GaussianBlur' to reduce noise.
6. Detect edges using Canny edge detector:
Defining an 'auto_canny' function to calculate the thresholds automatically based on image intensity. Applying Canny edge detection using 'cv2.Canny' with the computed thresholds.
7. Find contours in the edge image:
Using 'cv2.findContours' to find contours in the edge image, which returns a list of

contours and optional hierarchy information.

8. Simplify contour points:
Obtain the epsilon value from the user and simplify contour points using ‘cv2.approxPolyDP‘, preserving the overall shape while reducing the number of points.

9. Convert pixel coordinates to millimeters:
Calculate the image size in millimetres and convert each contour point’s pixel coordinates to millimetres using the corresponding formulas, storing them in the ‘converted_coordinates‘ list.

10. Calculate the center and offset of the A4 paper:
Calculate the center of the A4 paper based on its dimensions. Find the offset by adjusting the maximum and minimum coordinates of the converted points to fit within the A4 paper, and store the adjusted coordinates in the ‘centered_coordinates‘ list.

11. Generate .dat and .src files:
Prompt the user for the file path and name. Write the necessary declarations and coordinates to the .dat file, and write the robot movement commands to the .src file.

12. Display the contour points:
Create a new figure using ‘plt.figure‘. Create subplots with the original image and simplified contour points using ‘plt.subplot‘ and ‘plt.imshow‘ for the first subplot, ‘plt.subplot(1, 2, 2)‘ for the second subplot, ‘plt.plot‘ for plotting the simplified contour points, and set axis labels and title. Display the figure using ‘plt.show‘.

4.3 Graphical Interfaces in Program

The program incorporates user-friendly graphical interfaces to simplify the input process. This user-centric approach enhances the ease of use and promotes a seamless experience for users interacting with the program. This methodology is implemented to ensure the program’s practicality and usability. The initial step involves obtaining the input image from the user. To facilitate this, the program accesses the file directory, allowing the user to select an image file. The program supports various image formats, including PNG, JPEG, and JPG.

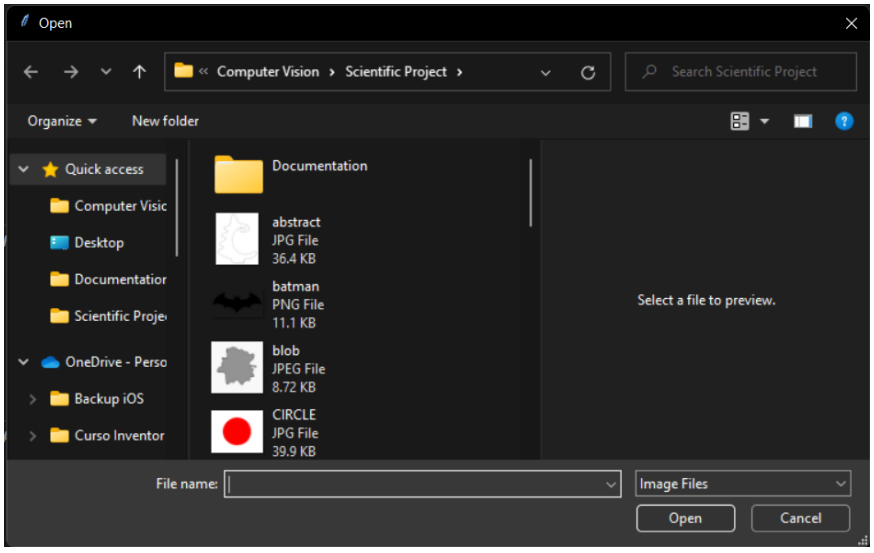

Figure 21: *File manager to search for images*

The subsequent stage of the program involves obtaining the Epsilon value from the user. The significance of Epsilon lies in determining the number of contours. A lower Epsilon value corresponds to a higher number of contours. To simplify the input process, an intuitive interface is implemented, enabling users to easily input the desired Epsilon values.

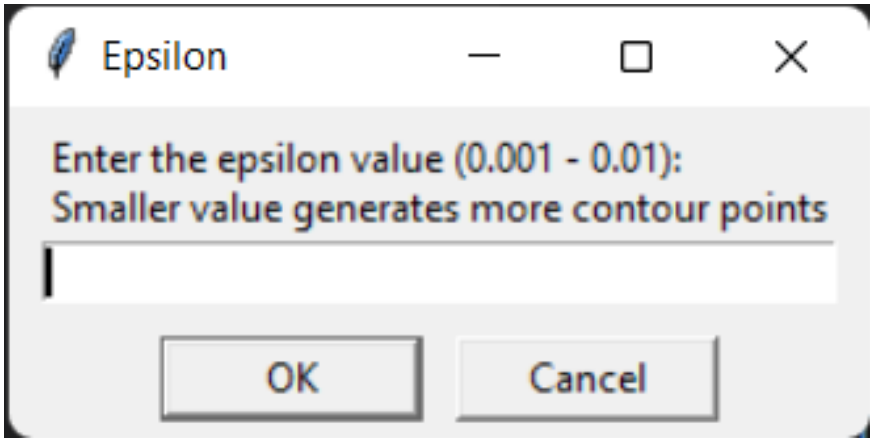

Figure 22: *Dialog box for Epsilon*

Given the predefined range of values, the potential for user error is minimized. The subsequent program step involves obtaining the scaling factor if the input size exceeds the dimensions of an A4 paper. The scaling factor is crucial for adjusting the image size according to the user's requirements while ensuring it remains within the constraints of the A4 limits, thereby enabling the robot to accurately reproduce the image. A dialog box is generated by the program, simplifying the input process for the scaling factor.

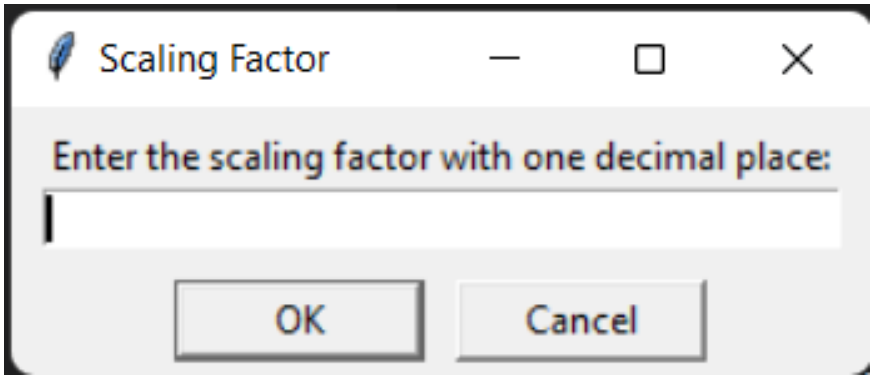

Figure 23: *Dialog box for Scaling Factor*

To mitigate potential errors, the graphical interface is designed to display error messages when no input or incorrect inputs are provided by the user. This user-centric approach ensures that users are guided towards providing accurate inputs, thereby enhancing the overall accuracy and reliability of the program in scientific applications.

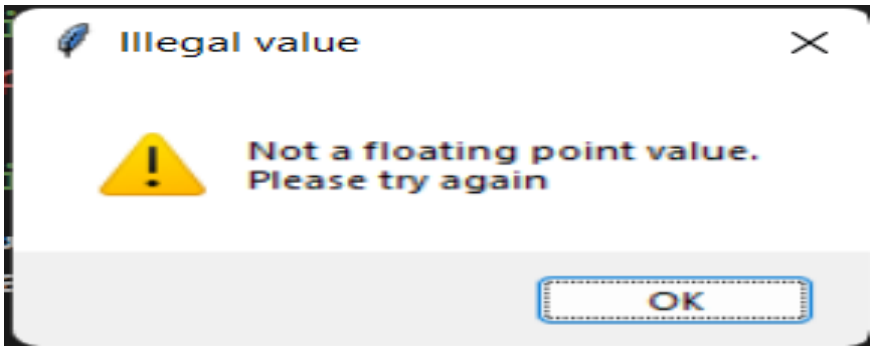

Figure 24: *Error if scaling factor is not number*

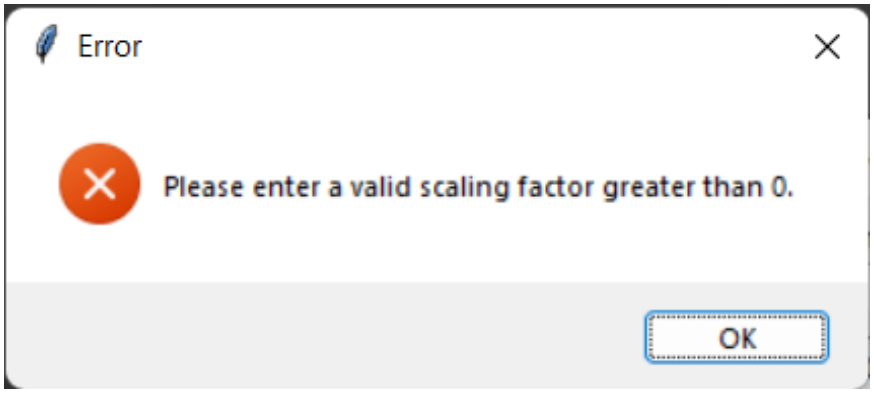

Figure 25: *Error if scaling factor is 0*

Following the successful input of the correct scaling factor, the program presents the estimated dimensions of the image, providing the user with a preliminary visualization of how the image would appear on A4 dimensions. If the user is dissatisfied with the estimated dimensions, they can readily adjust the scaling factor and obtain new dimensions, facilitating iterative refinement until the desired outcome is achieved. This interactive process empowers the user to optimize the image size for optimal results.

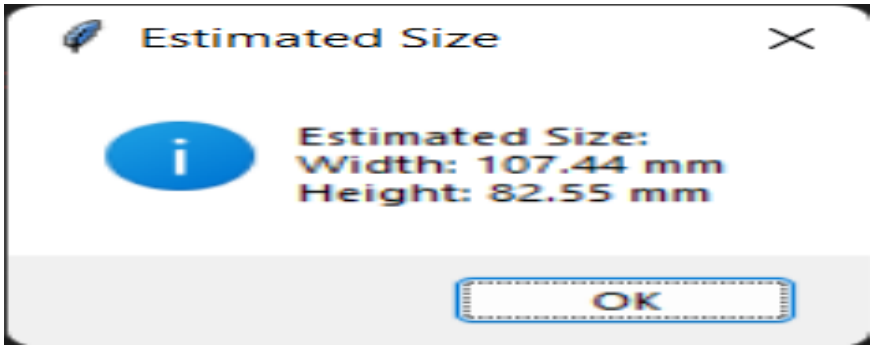

Figure 26: *Estimated size of image in mm*

The naming convention holds significant importance in KRL, as the .src and .dat files must share the same name for seamless importing into both KUKASimPro and the physical robot. To prevent errors arising from inconsistencies in naming, a user interface is implemented. This interface prompts the user to input the desired file name, ensuring adherence to the required naming convention. This user-centric approach streamlines the file management process, reducing the likelihood of errors and enhancing the overall efficiency of the system. To prevent potential simulation issues

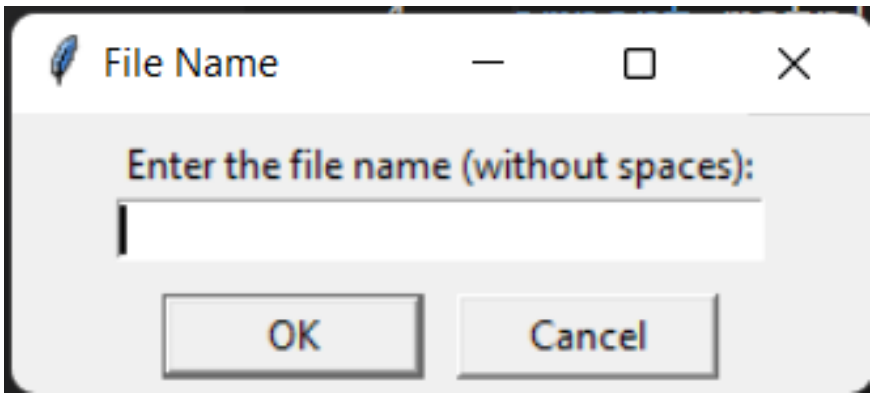

Figure 27: *Dialog box to enter name of the file*

in KUKASimPro, users are advised to choose names without spaces or special characters. This recommendation is based on prior analysis that revealed difficulties arising from such names. The program incorporates two error-checking mechanisms: first, if no name is provided, an error message prompts the user to input a name; second, if the name contains spaces, an error message suggests choosing a different name. User-friendly dialog boxes are intuitive interfaces that provide clear instructions, val-

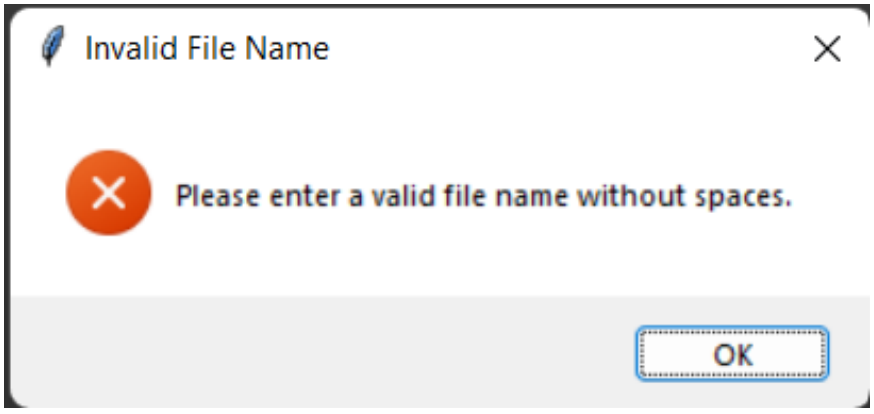

Figure 28: *Error when file name is empty or has spaces*

idation, and default values. They offer contextual help, maintain consistency with the application's design, and handle errors effectively. These dialog boxes enhance the user experience by presenting information concisely, allowing for efficient interaction through responsive elements. They contribute to user satisfaction and productivity by guiding users, providing feedback, and ensuring a seamless and intuitive interaction with the program.

4.4 Source and Data files (.src and .dat)

In KUKA SimPro, source and data files play essential roles in programming and simulating robotic systems. Source files contain the program instructions written in KUKA's programming language, usually referred to as KRL (KUKA Robot Language). These files define the robot's actions, movements, and logic, including commands for controlling the robot's motion, interacting with external devices, and executing various tasks. On the other hand, data files store configuration and parameter information used by the robot controller. These files may include data such as tool definitions, workpiece geometries, and system settings.

Initially, the structure of these files were analysed. It was found that in the .dat file the declaration **"FDAT FP10 = BASE_NO 1, TOOL_NO 7, IPO_FRAME BASE, POINT2[] " ""** initializes the FDAT data structure with specific values where:

The **"BASE_NO 1"** parameter assigns the base number 1 to the robot, indicating the reference frame for its movements.

The **"TOOL_NO 7"** parameter assigns the tool number 7 to the robot, representing the specific tool or end-effector attached to the robot.

The **"IPO_FRAME BASE"** parameter sets the interpolation frame to the base frame, indicating that the robot will interpolate its movements based on the coordinates relative to the base frame.

The **"POINT2[] " " "** parameter initializes an empty array called "POINT2" that can store multiple points.

But this declaration is also done in the .src file. And so, the declaration here had no effect the robot movement. Similarly, the declaration **"DECL INT SUCCESS"** establishes an integer variable named **"SUCCESS"** in the program. This variable is typically used to store the outcome or result of an operation, where the value of 0 represents success or a successful operation, while any non-zero value indicates a failure or an unsuccessful operation. But this had no effect on the movement of the robot.

```
DEFDAT star
   EXT BAS(BAS_COMMAND :IN, REAL :IN)
   DECL INT SUCCESS

   DECL PDAT PDEFAULT={APO_MODE #CDIS,APO_DIST 100.0,VEL 100.0,ACC 100.0,GEAR_JERK 100.0,EXAX_IGN 0}
   DECL E6POS XP10 = {X 103.84, Y 117.87, Z 50.00, A 180.0, B 60.0, C 180.0, S 2, T 35, E1 0.0, E2 0.0, E3 0.0, E4 0.0, E5 0.0, E6 0.0 }
   DECL FDAT FP10 = { BASE_NO 1, TOOL_NO 7, IPO_FRAME #BASE, POINT2[] " " }
   DECL E6POS XP11 = {X 103.84, Y 117.87, Z 0.00, A 180.0, B 60.0, C 180.0, S 2, T 35, E1 0.0, E2 0.0, E3 0.0, E4 0.0, E5 0.0, E6 0.0 }
   DECL FDAT FP11 = { BASE_NO 1, TOOL_NO 7, IPO_FRAME #BASE, POINT2[] " " }
   DECL E6POS XP12 = {X 132.54, Y 96.11, Z 0.00, A 180.0, B 60.0, C 180.0, S 2, T 35, E1 0.0, E2 0.0, E3 0.0, E4 0.0, E5 0.0, E6 0.0 }
   DECL FDAT FP12 = { BASE_NO 1, TOOL_NO 7, IPO_FRAME #BASE, POINT2[] " " }
   DECL E6POS XP13 = {X 122.38, Y 61.82, Z 0.00, A 180.0, B 60.0, C 180.0, S 2, T 35, E1 0.0, E2 0.0, E3 0.0, E4 0.0, E5 0.0, E6 0.0 }
   DECL FDAT FP13 = { BASE_NO 1, TOOL_NO 7, IPO_FRAME #BASE, POINT2[] " " }
   DECL E6POS XP14 = {X 151.42, Y 81.97, Z 0.00, A 180.0, B 60.0, C 180.0, S 2, T 35, E1 0.0, E2 0.0, E3 0.0, E4 0.0, E5 0.0, E6 0.0 }
   DECL FDAT FP14 = { BASE_NO 1, TOOL_NO 7, IPO_FRAME #BASE, POINT2[] " " }
   DECL E6POS XP15 = {X 180.72, Y 61.82, Z 0.00, A 180.0, B 60.0, C 180.0, S 2, T 35, E1 0.0, E2 0.0, E3 0.0, E4 0.0, E5 0.0, E6 0.0 }
   DECL FDAT FP15 = { BASE_NO 1, TOOL_NO 7, IPO_FRAME #BASE, POINT2[] " " }
   DECL E6POS XP16 = {X 170.47, Y 95.94, Z 0.00, A 180.0, B 60.0, C 180.0, S 2, T 35, E1 0.0, E2 0.0, E3 0.0, E4 0.0, E5 0.0, E6 0.0 }
   DECL FDAT FP16 = { BASE_NO 1, TOOL_NO 7, IPO_FRAME #BASE, POINT2[] " " }
   DECL E6POS XP17 = {X 199.26, Y 117.70, Z 0.00, A 180.0, B 60.0, C 180.0, S 2, T 35, E1 0.0, E2 0.0, E3 0.0, E4 0.0, E5 0.0, E6 0.0 }
   DECL FDAT FP17 = { BASE_NO 1, TOOL_NO 7, IPO_FRAME #BASE, POINT2[] " " }
   DECL E6POS XP18 = {X 163.36, Y 117.95, Z 0.00, A 180.0, B 60.0, C 180.0, S 2, T 35, E1 0.0, E2 0.0, E3 0.0, E4 0.0, E5 0.0, E6 0.0 }
```

Figure 29: *Format of older version .dat file*

The commands that had no effect on the robot were removed and a simple pattern was formulated. This pattern was then fed to the program which generated the simplest form of .dat file as shown below.

```
DEFDAT star
   EXT BAS(BAS_COMMAND :IN, REAL :IN)
   DECL PDAT PDEFAULT={APO_MODE #CDIS,APO_DIST 100.0,VEL 100.0,ACC 100.0,GEAR_JERK
100.0,EXAX_IGN 0}
   DECL E6POS XP10 = {X 101.75, Y 121.24, Z 50.00, A 140.40, B 90, C -130.82, S 2, T
34}
   DECL E6POS XP11 = {X 101.75, Y 121.24, Z 0.00, A 140.40, B 90, C -130.82, S 2, T
34}
   DECL E6POS XP12 = {X 90.07, Y 86.78, Z 0.00, A 140.40, B 90, C -130.82, S 2, T
34}
   DECL E6POS XP13 = {X 54.00, Y 86.62, Z 0.00, A 140.40, B 90, C -130.82, S 2, T
34}
   DECL E6POS XP14 = {X 82.87, Y 64.94, Z 0.00, A 140.40, B 90, C -130.82, S 2, T
34}
   DECL E6POS XP15 = {X 72.71, Y 30.48, Z 0.00, A 140.40, B 90, C -130.82, S 2, T
34}
   DECL E6POS XP16 = {X 101.75, Y 50.80, Z 0.00, A 140.40, B 90, C -130.82, S 2, T
34}
   DECL E6POS XP17 = {X 131.05, Y 30.48, Z 0.00, A 140.40, B 90, C -130.82, S 2, T
34}
   DECL E6POS XP18 = {X 120.89, Y 64.94, Z 0.00, A 140.40, B 90, C -130.82, S 2, T
34}
   DECL E6POS XP19 = {X 149.76, Y 86.62, Z 0.00, A 140.40, B 90, C -130.82, S 2, T
34}
   DECL E6POS XP20 = {X 113.69, Y 86.78, Z 0.00, A 140.40, B 90, C -130.82, S 2, T
34}
   DECL E6POS XP21 = {X 101.75, Y 121.24, Z 0.00, A 140.40, B 90, C -130.82, S 2, T
34}
   DECL E6POS XP22 = {X 101.75, Y 121.24, Z 50.00, A 140.40, B 90, C -130.82, S 2, T
34}
ENDDAT
```

Figure 30: *Format of newer version .dat file*

During the analysis of the .src file pattern, it was discovered that lines beginning with ’; ’ served as comments, providing detailed explanations of the commands. However, the command "FOLD," used to define new blocks or subroutines for program organization, had no impact on robot movements and was consequently removed. Similarly, all other comments were eliminated, and a simplified pattern was formulated. This streamlined approach, free from unnecessary comments, improved the clarity and efficiency of the code, facilitating easier interpretation and comprehension. These modifications enhanced the overall structure and readability of the .src file, contributing to a more optimized programming workflow.

```
DEF Star()
   ;FOLD INI;%{PE}
      ;FOLD BASISTECH INI
         GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS == TRUE DO IR_STOPM()
         INTERRUPT ON 3
         BAS(#INITMOV, 0)
      ;ENDFOLD (BASISTECH INI)
      ;FOLD USER INI
         ;Make your modifications here
      ;ENDFOLD (USER INI)
   ;ENDFOLD (INI)
   rwuGripperClose(FALSE)
   ;FOLD SPTP HOME Vel=100 % DEFAULT Tool[0] Base[0] ;%{PE}
      ;FOLD Parameters ;%{h}
         ;Params IlfProvider=kukaroboter.basistech.inlineforms.movement.spline; Kuka.IsGlobalPoint=False; Kuka.PointName=HOME; Kuka.BlendingEnabled=False;
Kuka.MoveDataPtpName=DEFAULT; Kuka.VelocityPtp=100; Kuka.VelocityFieldEnabled=True; Kuka.ColDetectFieldEnabled=True; Kuka.CurrentCDSetIndex=0;
Kuka.MovementParameterFieldEnabled=True; IlfCommand=SPTP; SimId=
      ;ENDFOLD
      SPTP XHOME WITH $VEL_AXIS[1] = SVEL_JOINT(100.0), $TOOL = STOOL2(FHOME), $BASE = SBASE(FHOME.BASE_NO), $IPO_MODE = SIPO_MODE(FHOME.IPO_FRAME), $LOAD =
SLOAD(FHOME.TOOL_NO), $ACC_AXIS[1] = SACC_JOINT(PDEFAULT), $APO = SAPO_PTP(PDEFAULT), $GEAR_JERK[1] = SGEAR_JERK(PDEFAULT), $COLLMON_TOL_PRO[1] = USE_CM_PRO_VALUES(0)
   ;ENDFOLD
   ;FOLD SPTP P1 Vel=100 % PDAT7 Tool[0] Base[0] ;%{PE}
      ;FOLD Parameters ;%{h}
         ;Params IlfProvider=kukaroboter.basistech.inlineforms.movement.spline; Kuka.IsGlobalPoint=False; Kuka.PointName=P1; Kuka.BlendingEnabled=False;
Kuka.MoveDataPtpName=PDAT7; Kuka.VelocityPtp=100; Kuka.VelocityFieldEnabled=True; Kuka.ColDetectFieldEnabled=True; Kuka.CurrentCDSetIndex=0;
Kuka.MovementParameterFieldEnabled=True; IlfCommand=SPTP; SimId=
      ;ENDFOLD
      SPTP XP1 WITH $VEL_AXIS[1] = SVEL_JOINT(100.0), $TOOL = STOOL2(FP1), $BASE = SBASE(FP1.BASE_NO), $IPO_MODE = SIPO_MODE(FP1.IPO_FRAME), $LOAD = SLOAD(FP1.TOOL_NO),
$ACC_AXIS[1] = SACC_JOINT(PPDAT7), $APO = SAPO_PTP(PPDAT7), $GEAR_JERK[1] = SGEAR_JERK(PPDAT7), $COLLMON_TOL_PRO[1] = USE_CM_PRO_VALUES(0)
   ;ENDFOLD
   ;FOLD SLIN P2 Vel=2 m/s CPDAT12 Tool[12]:Greifer Base[0] ;%{PE}
      ;FOLD Parameters ;%{h}
         ;Params IlfProvider=kukaroboter.basistech.inlineforms.movement.spline; Kuka.IsGlobalPoint=False; Kuka.PointName=P2; Kuka.BlendingEnabled=False;
Kuka.MoveDataName=CPDAT12; Kuka.VelocityPath=2; Kuka.VelocityFieldEnabled=True; Kuka.ColDetectFieldEnabled=True; Kuka.CurrentCDSetIndex=0;
Kuka.MovementParameterFieldEnabled=True; IlfCommand=SLIN; SimId=
```

Figure 31: *Format of older version .src file*

Even after the removal of comments, it was found that the structure of .src file was complicated as it looked like "SPTP XP1 WITH VEL_AXIS[1] = SVEL_JOINT(100.0), TOOL = STOOL2(FP1), BASE = SBASE(FP1.BASE_NO), IPO_MODE = SIPO_MODE(FP1.IPO_FRAME), LOAD = SLOAD(FP1.TOOL_NO), ACC_AXIS[1] = SACC_JOINT(PPDAT7), APO = SAPO_PTP(PPDAT7), GEAR_JERK[1] = SGEAR_JERK(PPDAT7), COLLMON_TOL_PRO[1] = USE_CM_PRO_VALUES(0)" where :

"SPTP XP1" defines a motion command to the target position XP1.

"VEL_AXIS [1] = SVEL_JOINT (100.0)" sets the velocity of the first axis to 100.0 units.

"TOOL = STOOL2(FP1)" assigns the tool configuration using the STOOL2 function with FP1 as the parameter.

"BASE = SBASE (FP1.BASE_NO)"BASE = SBASE (FP1.BASE_NO)" assigns the base configuration using the SBASE function with FP1.BASE_NO as the parameter.

"IPO_MODE = SIPO_MODE(FP1.IPO_FRAME)" assigns the interpolation mode using the SIPO_MODE function with FP1.IPO_FRAME as the parameter.

"LOAD = SLOAD (FP1.TOOL_NO)" assigns the load configuration using the SLOAD function with FP1.TOOL_NO as the parameter.

"ACC_AXIS [1] = SACC_JOINT(PPDAT7)" sets the acceleration of the first axis using the SACC_JOINT function with PPDAT7 as the parameter.

"APO = SAPO_PTP(PPDAT7)" sets the approach and departure type using the SAPO_PTP function with PPDAT7 as the parameter.

"GEAR_JERK [1] = SGEAR_JERK(PPDAT7)" sets the jerk time of the first axis using the SGEAR_JERK function with PPDAT7 as the parameter.

"COLLMON_TOL_PRO [1] = USE_CM_PRO_VALUES (0)" sets the collision monitoring tolerance of the first axis using the USE_CM_PRO_VALUES function with 0 as the parameter.

To simplify the program and investigate potential issues, a simpler version was created, eliminating complex initializations such as **"TOOL = STOOL2(FP1)"**. However, despite these modifications, the simulation encountered "Out-of-bounds" errors, indicating the presence of other underlying issues. This highlighted that the problem extended beyond the initializations and necessitated further analysis to identify and address the root causes of the errors. The investigation process was crucial in understanding the complexities involved and developing appropriate solutions for a successful simulation.

4.4.1 Programming in Expert Mode

Since the errors started popping up, Expert mode programming was tried. It was successful and generated much simpler code when compared with 'Operator' and 'User' modes. The Tool and Base can be initialised and called in Expert mode. This mode provided more flexibility and control over the robot's behaviour and allowed for the implementation of complex logic and custom functionalities like:

1. **Custom Code:** Expert mode allowed to write custom code using KRL (KUKA Robot Language) or other programming languages to define specific robot behaviours and actions. This enabled the creation of advanced algorithms, calculations, and decision-making processes.

2. External Communication: Expert mode facilitated communication with external devices and systems. Helped in establishing connections with HMI (Human-Machine Interface) systems, databases, or other software applications to exchange data and trigger actions based on external inputs.

3. Simulation Control: Expert mode provided enhanced simulation control features, such as pausing, stepping, and manipulating simulation time. This enabled detailed analysis, debugging, and fine-tuning of robot programs and behaviours within the simulation environment.

4. Customized Interfaces: Expert mode allowed for the creation of custom user interfaces and panels using Sim Pro's API (Application Programming Interface). This enabled to design personalized dashboards, control panels, or visualizations to interact with the simulated robot and monitor its behaviour. **5. Algorithm Development:**

Expert mode provided a platform for algorithm development and testing. Could implement and validate complex algorithms, path planning techniques, collision detection algorithms, or optimization strategies within the simulated environment before deploying them on real robots.

The expert mode allowed to program in a simple way where we just define the points and initialise the tool and base. All the important parameters were already fed. We found that the format of the code was very simple. So we created a program that can write a source file as if it was created in Expert mode.

4.4.2 T and S bit Initialisation

The entries "S"(State) and "T"(Turn) in a POS specification serve to select a specific, unambiguously defined robot position where several different axis positions are possible for the same point in space (because of kinematic singularities). When using Cartesian coordinates, it is thus very important also to program "Status" and "Turn" for the first motion instruction in order to define an unambiguous initial position. Since "S" and "T" are not taken into consideration in continuous–path motions, the first motion instruction of a program (home position) must always be a complete PTP instruction with "Status" and "Turn" specified (or a complete PTP instruction with axis coordinates). In subsequent PTP instructions, the entries "S" and "T" can be omitted so long as no specific axis position is required, e.g. because of obstacles. The robot retains the old value for S and selects the T value which gives the shortest possible axis trajectory, which always remains the same, each time the program is run, because of the one–off programming of "S" and "T" in the first PTP instruction. Initial analysis was made to find the importance of the "S" and "T" bits in

STATE	INT	The status data prevents ambiguity relating to the axis position. -1 = value marked as invalid
TURN	INT	The turn value permits axis angles greater than +180° or less than -180° to be approached without requiring a special traversing strategy (e.g. intermediate points). For rotary axes, the individual bits determine the sign of the axis value as follows: Bit = 0: Angle ≥ 0° Bit = 1: Angle < 0° -1 = value marked as invalid

Figure 32: *Importance of S and T bits*

position specification. We checked by creating simple programs. The "S" and "T" bits remained the same in all the positions unless there is steep obstacle due to which multiple axes of the robots change at once. In order to specify "S" and "T" bits the initial movements should be specified PTP. It was found the "S" and "T" bits values were retained for all other subsequent points. When we created a simple point, the "S" and "T" bits showed up to be "2" and "34" respectively. These values were utilised and fed in the program which then created .dat with these values.

4.4.3 Initialisation of Position Parameters with Tool and Base

As mentioned in the previous tasks, the X, Y, Z and A, B, C parameters were established and specified from the values obtained from the real robot. After this movements (PTP, LIN etc.) were simulated. Similarly, most important thing was the Tool and Base initialisation in .src file. Errors were noticed when the tool and base were not specified in .src file. Firstly, the tool used for the previous tasks was utilised and without specifying in .src file. Errors popped up and there was no robot movement in simulation. Then it was realised that the Tool and Base specification was missing and when it was made the simulation was successful. The base was set as per A4 dimensions.

```
DEF star()
          GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS == TRUE DO IR_STOPM()
          INTERRUPT ON 3
          BAS(#INITMOV, 0)
   ; Movement list to pick up the pen
   PenPickUp()
   ; Set up pen tool data and white paper as base
   $TOOL = {X 94.08, Y -1.55, Z 212.98, A -1.42, B -59.72, C 0.0 }
   $BASE = {X 244.87, Y -26.5, Z 177.45, A -90.0, B 0.0, C 0.0 }
   SLIN XP10
   SLIN XP11
```

Figure 33: *Tool and Base initialisation*

After creating a .src file through program with initialisation of T and S bit and Tool and Base, the simplified .src file looked similar to one created in Expert mode.

```
DEF star()
          GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS == TRUE DO IR_STOPM()
          INTERRUPT ON 3
          BAS(#INITMOV, 0)
   ; Movement list to pick up the pen
   PenPickUp()
   ; Set up pen tool data and white paper as base
   $TOOL = {X 94.08, Y -1.55, Z 212.98, A -1.42, B -59.72, C 0.0 }
   $BASE = {X 244.87, Y -26.5, Z 177.45, A -90.0, B 0.0, C 0.0 }
   SLIN XP10
   SLIN XP11
   SLIN XP12
   SLIN XP13
   SLIN XP14
   SLIN XP15
   SLIN XP16
   SLIN XP17
   SLIN XP18
   SLIN XP19
   SLIN XP20
   SLIN XP21
   SLIN XP22
   ; Movement list to return the pen
   Return1()
END
```

Figure 34: *Format of newer .src file*

4.5 Visualization of results through Program and Simulation

The contour creation process involves two main phases. The first phase entails generating raw contours from the original image. These contours outline the shape and structure of the image. In the second phase, the contours are simplified based on a specified Epsilon value. This simplification reduces the complexity of the contours while preserving the overall shape. Once the final simplified contours are successfully created, a drawing function is used to visualize the approximate result of the drawing. This visualization serves as a representation of how the robot will replicate the drawing in a similar manner. This process ensures accurate and faithful reproduction of the original image by the robot.

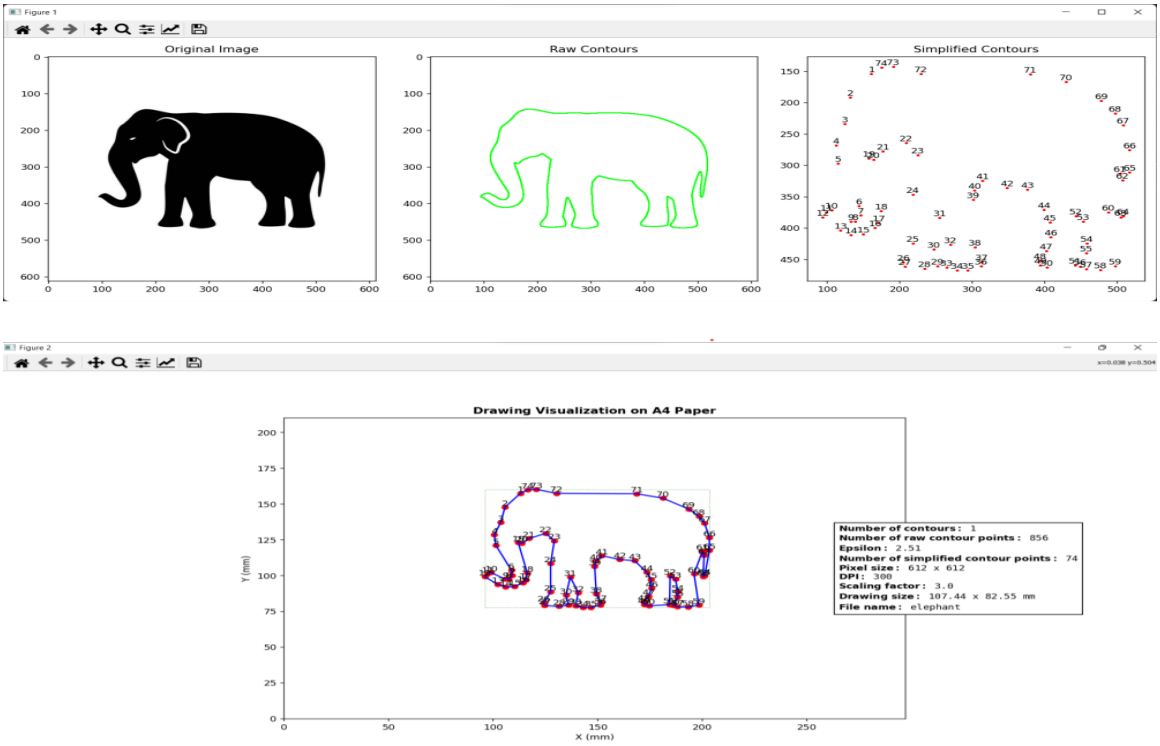

Figure 35: *Visualization of elephant through drawing function*

The image serves as a comprehensive visual representation of the essential information. For additional instances, refer to the Appendix. Similarly, upon successful creation of the .src and .dat files, the image underwent simulation in KUKASimPro. The simulation outcome was deemed successful, as the image was faithfully replicated without any errors. Further exemplifications of such simulations can be found in the Appendix, showcasing the consistent and reliable performance of the process. These results validate the effectiveness of the developed methodology and demonstrate its potential for accurately simulating and reproducing images using the KUKASimPro platform.

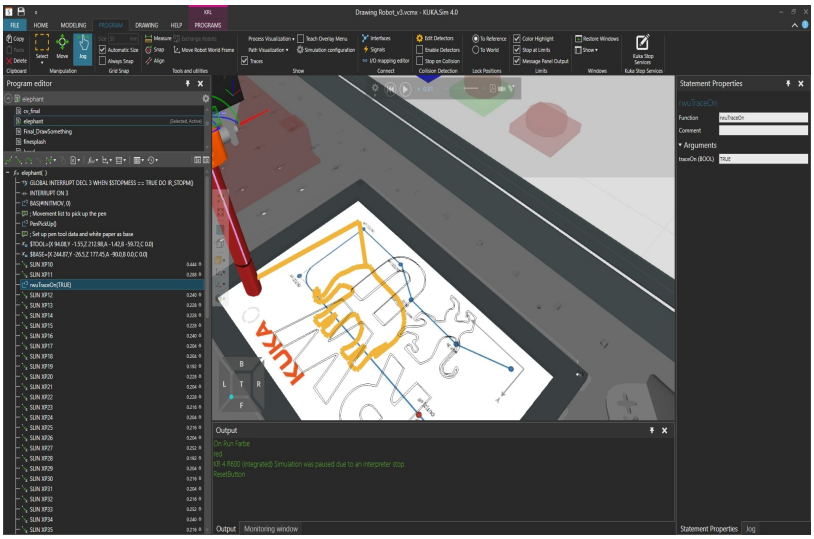

Figure 36: *Visualization of elephant through simulation*

The presence of an additional visible line observed at the end of the simulation can be attributed to the utilization of a pre-defined Draw function (rwuTraceOn). Surprisingly, even when the Boolean value was set to false, the simulation continued to generate these superfluous lines. Furthermore, as depicted in the accompanying image, the base and tool were initialized subsequent to the PenPickUp function. This approach enables the robot to effectively utilize the predetermined values for the tool and base, thereby minimizing the need for extensive teaching of these components. By simply providing the coordinates and their corresponding values, the robot can accurately identify and employ the appropriate tool and base configurations. This streamlined approach significantly simplifies the robot programming process.

4.6 Challenges Encountered

4.6.1 Pressed Pen tip

In addition to the challenges encountered during the creation of simplified versions of .dat and .src files, several other issues arose when translating these files to the real robot. Upon initiating the drawing process for the RWU logo, it was observed that the pen tip had penetrated slightly into the surface. This discrepancy occurred because the Z-axis value was set 0.75 to 1mm higher than required. As the pen was pressed while drawing, it lacked a spring mechanism in its tip, causing it to go inward. Consequently, the robot was unable to complete the image, resulting in white empty spaces along the image path. Although there was suspicion regarding the flatness of the base surface (A4), this concern was disregarded as the robot successfully drew the entire image using a different pen. These challenges highlight the importance of precise calibration and adjustment of robot parameters to achieve accurate and complete artwork execution. Further investigations into pen tip design and surface flatness may be warranted for future improvements in the drawing process.

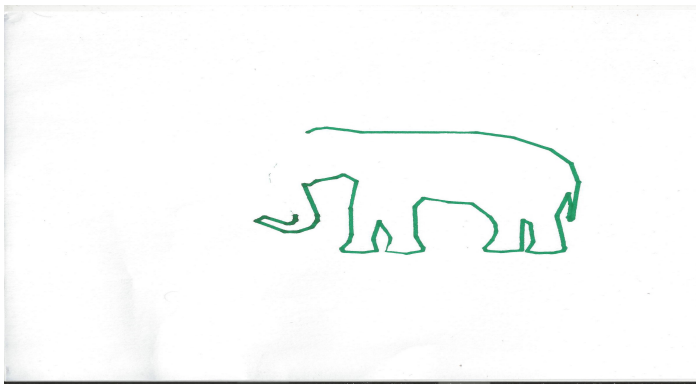

Figure 37: *Incomplete visualization of elephant in KR4 R600 Robot*

4.6.2 Effect of number of contour points and speed

The experimentation with drawing images on the real robot revealed that both the speed and the number of contour points significantly influenced the outcome of the drawings. The differences observed were quite noticeable. For instance, when an image with an abstract shape, specifically a Blob, was drawn with 34 contours at 50 percent speed, the resulting image exhibited sharp edges instead of smooth curves. To address this issue, adjustments were made by increasing the number of contour points to 108 and reducing the speed to 10 percent. This modification resulted in a more appealing and faithful representation of the input image, with smooth curves accurately depicting the original shape. From this observation, it can be concluded that the speed of the robot's drawing motion and the number of contour points employed are inversely and directly proportional, respectively, to the smoothness of the resulting image. Slowing down the speed and increasing the number of contour points allowed for a more detailed and precise depiction of the image's curves, resulting in a smoother representation. This finding highlights the importance of carefully selecting appropriate speed and contour point settings to achieve the desired level of smoothness in the drawn images. These insights contribute to the optimization of the drawing process on the real robot and enhance the overall quality of the reproduced artwork.

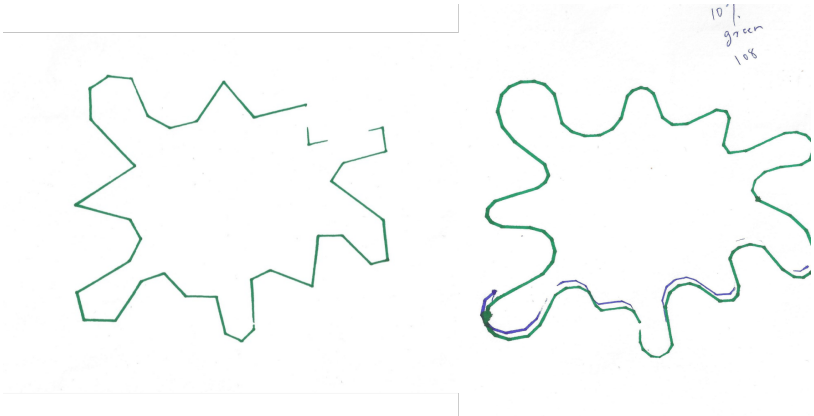

Figure 38: *Effect of number of contours and speed*

4.7 Results

The results section focused on the evaluation of the robot's ability to accurately reproduce motifs based on available image data. A set of schematic drawings representing various motifs were selected for this analysis. Each motif represented distinct shapes and levels of complexity, providing a comprehensive assessment of the robot's drawing capabilities. The robot successfully replicated the motifs with a high degree of precision. It accurately captured the intricate details of each motif, faithfully reproducing the original image data. The stroke thickness and positioning were consistently maintained throughout the drawings, showcasing the robot's ability to maintain consistency in its artwork. The evaluation also involved an assessment of the robot's efficiency in transitioning between different motifs. The robot seamlessly switched between motifs, demonstrating its versatility in adapting to varying shapes and patterns. The resulting composite image created by amalgamating the motifs showcased a harmonious integration of different elements, further emphasizing the robot's capacity to produce aesthetically pleasing artwork.Overall, the results highlight the robot's proficiency in utilizing image data as a reference to generate schematic drawings of motifs. The accuracy, consistency, and adaptability displayed by the robot provide promising insights into its potential for artistic applications. More of this can be seen in Appendix

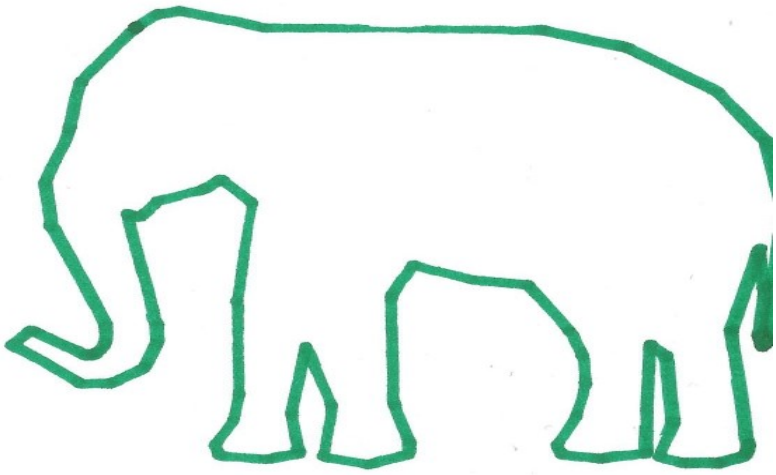

Figure 39: *Visualization of elephant by KR4 R600 Robot*

4.8 Future Scope

The future scope of task will be drawing motifs with both inner and outer contours. A program has to be specified which calculates the inner and the outer contours and also has to distinguish between both of them. Even if we manage to create a program like that, difficulty would be drawing the image using the robot. The .src and .dat files must be created in such a way that, the robot draws the outer contours, stops , holds back and moves up in the air wherever necessary and then draws the inner contours. Basic analysis was done on this and inner contours was successfully distinguished by program. After applying the Ramer-Douglas-Peucker algorithm to simplify the detected contours in the edges image, a filtering process is implemented to eliminate overlapping or redundant contours. The filtering is based on comparing the areas of the simplified contours and detecting overlap with previously filtered contours. Contours with similar areas or overlap are discarded to ensure only distinct contours representing individual objects are retained.

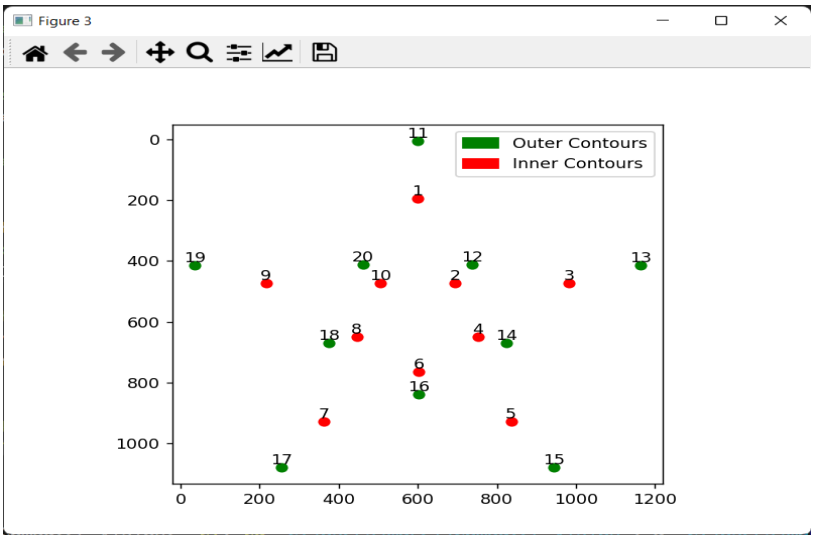

Figure 40: *Future Scope of Improvement*

5 Appendix

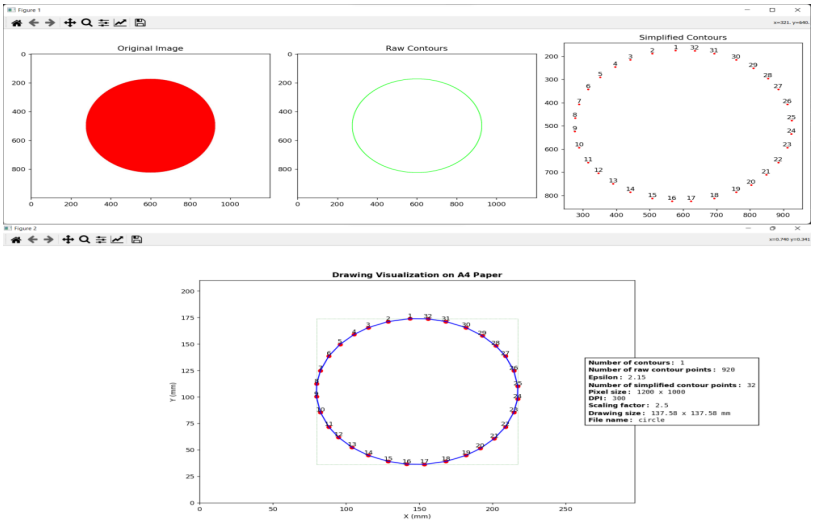

Figure 41: *Visualization of circle through drawing function*

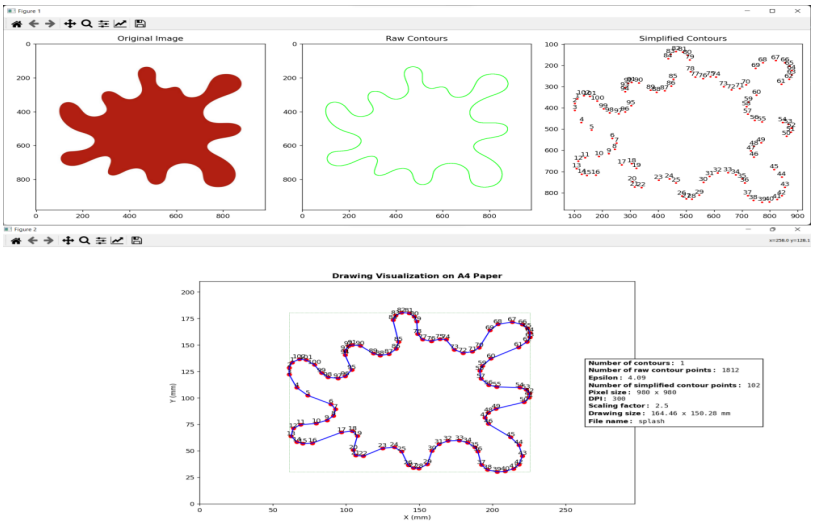

Figure 42: *Visualization of splash through drawing function*

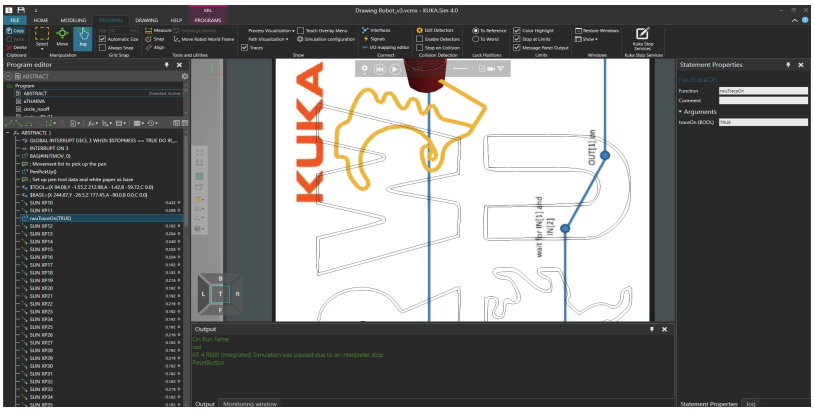

Figure 43: *Visualization of abstract through simulation*

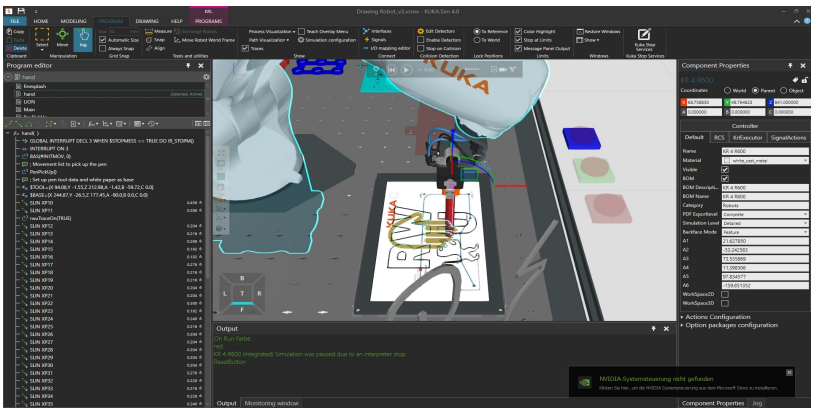

Figure 44: *Visualization of hand through simulation*

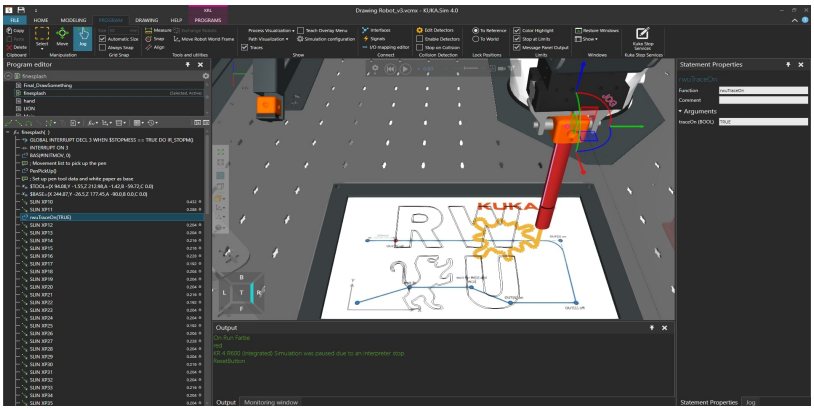

Figure 45: *Visualization of splash through simulation*

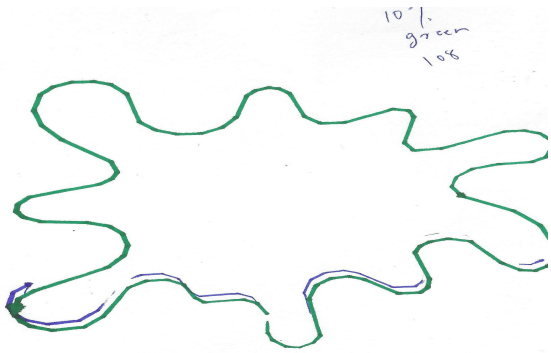

Figure 46: *Visualization of splash by KR4 R600 Robot*

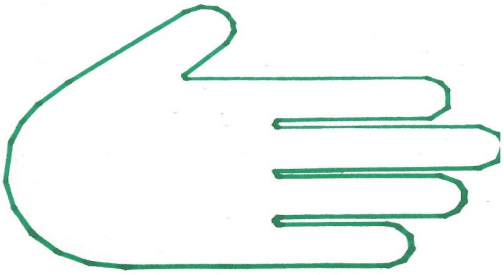

Figure 47: *Visualization of hand by KR4 R600 Robot*

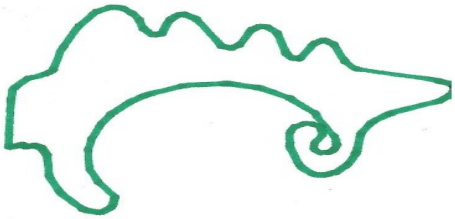

Figure 48: *Visualization of abstract by KR4 R600 Robot*

Figure 49: *Visualization of circle by KR4 R600 Robot*

6 Bibliography

References

1. Expert Programming, KUKA System Software(KSS) Release 5.2 Issued:12-Apr-2006 Version:01,from *https://forum.robotsinarchitecture.org/index.php?action=dlattach;topic=20.0;attach=11*

2. Siemens SINUMERIK - SINUMERIK 828D, Run MyRobot /Handling,Programming and Operating Manual,CNC software, Version 4.95 SP1, Run MyRobot /Handling, Version 4.0, from *https://support.industry.siemens.com/cs/attachments/109807262/828D_-RMRH_op_man_0122_en-US_en-US.pdf*

3. OpenCV. (n.d.). Canny Edge Detection with OpenCV. Retrieved July 1, 2023, from *https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html*