Introduction to Serial Manipulators

LOURENÇO, Beatriz no. 93024; **PORTUGAL, Isabel** no. 93085
PALMA, Rita no. 93172

DEEC

December 30, 2022

Abstract: The development of the project is explained in this paper, which is based on solving a drawing task on a robotic arm with 5 rotational degrees of freedom (DOF), Scorbot ER-7, while demonstrating the importance of kinematic models. The goal of this project is to have the robot, which is grabbing a marker, draw an input image over a flat paper surface.
By taking an image of a one-line black-and-white drawing, it is expected to select a set of points that the marker must touch. In order to do so, various image processing techniques are used such as thinning, finding of key points, and sampling. Next, the coordinates are fed to the robot and saved using the appropriate commands. Finally, it is expected that the robot draws the reference image, as accurately as possible.
While developing the project, we reached a set of parameters. These are tuned in order to achieve the desired drawing, such decisions are described thoroughly. Additionally, besides only drawing the line, we implemented another feature, where the gripper rolls along the trajectory.
The robot was programmed using Python3 with the serial library and connected to a computer through a USB cable. The image processing and trajectory planning were also performed in Python, and the connection with the robot was ensured through the wait time to read each command.
Overall, the Scorbot ER-7 demonstrates strong potential as a drawing robot. However, the robot's memory could be compromised by too complex images that require heavy sampling. The robot's performance is evaluated in terms of accuracy and speed and was found to be generally on par with other similar robots.

Keywords: Scorbot ER-7, Serial Manipulator, Image Processing, One line drawing

1 Introduction

The Scorbot ER-7 [2] is a versatile and highly accurate robotic arm that can perform a variety of tasks, including drawing. Due to its 5 rotation degrees of freedom, it has multiple axes of movement and can be programmed to create complex drawings with high precision. Its adaptability allows it to be used in a wide range of industries, from manufacturing to art.

The serial manipulator Scorbot ER-7 is present in Figure 1.

Figure 1: Scorbot ER-7 Serial manipulator.

The goal of this project, as simple as it appears, required the investigation and solving of several significant robotics problems, including image processing, trajectory planning, and robot connection, which will be the primary focus of this report. Each decision's motivation is also included, as are the results with various parameters that lead to the best performance. The option of manually calculating the direct and inverse kinematics based on the points to be drawn was discarded given the robot's option of using Cartesian coordinates, only having to further provide the pitch and roll coordinates, if needed.

Before discussing the methodologies used, it is important to note that the code was written in Python3, even though the robot's language is Advance Control Language (ACL), which is an advanced, multi-tasking robotic programming language.

2 Methodology

In this section, it is provided the methodology thoroughly explained of the image processing, and trajectory planning made independently of any hardware

and the following connection with the robot.

2.1 Image Processing

To draw the desired figure, the image must be processed before determining which points the robot should save in the system. The goal is to have the minimum number of points that will accurately draw the input image. Image processing requires the use of algorithms and logic, which will be covered in detail in this section.

2.1.1 Pre-Processing

After reading and saving the image given, there are a few steps that are done in order to ease its processing: i) **Padding**, to guarantee that the windows used later in feature extraction don't go out of the image's limits; ii) **Transformation in binary image**, where pixels are assigned the value either 0 or 255; iii) **Thinning** takes the binary image and contracts the foreground until only single-pixel wide lines remain.

By these means, we get a binary image with only the representation of the core structure of the drawing, along with its coordinates. This will guarantee easier processing of the points and allow us to jump to the next step: feature extraction and line segmentation.

2.1.2 Feature Extraction and Line Segmentation

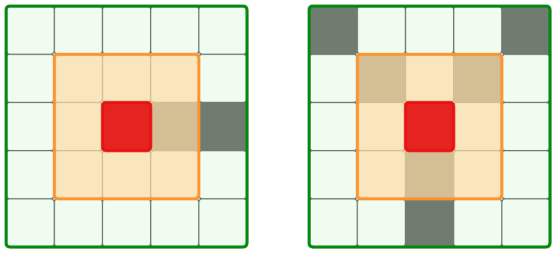
There are key points in the image to locate in order to achieve the desired drawing trajectory. These points can be divided into two categories:

- **Endpoints** - The beginning/end of the line;
- **Bifurcation points** - Points that represent an intersection of more than two branches/paths.

The algorithms to find these two kinds of points are similar and are based on looking for the neighbor points around the point under study. The point in red is analyzed in Figure 2 by applying a window (study point surrounded by one-pixel window width) and counting the value of the 8 pixels on it. The same is done for a second window (2 pixels away from the point, with a width of one pixel), where it is possible to analyze $8 \times 2 = 16$ pixels, and so on.

The logic for endpoints is to find only one pixel different than zero in the first or second window. As a confirmation, until the tenth window, the algorithm examines each window to see if it contains more than two pixels that differ from zero; if so, the point is not considered an endpoint. The Figure 2a shows the initial analysis of a point that will be considered an endpoint.

The same idea is used to find bifurcation points, but the rule now requires the same or more number of pixels as seen in the first window. In fact, in Figure 2b each window has three pixels different than zero, two in the top corners and one in the bottom center and accordingly the window of size two also has three filled pixels. Thus, the point analyzed is a bifurcation point of three branches.



(a) (b)

Figure 2: The pixel being examined (in red); Looking around the pixel, a window for one pixel (in orange) is created, followed by two pixels around the point under study (green window). Illustrations centered on (a) endpoint. (b) bifurcation point.

These points are crucial in order to accurately draw the reference image. Indeed, the start or end of a drawing must be rigorously chosen in order to avoid losing the "tale" of a line. Moreover, bifurcation points are where branches intersect, thus if there is a need to return to another line having to pass through a bifurcation point this must be well defined in order to avoid a mismatch in the drawing. Consequently, these points must be saved and preserved even after sampling the image.

After finding these key points, and preceding the trajectory choice, the drawing can be split into segments. This is useful as it provides a more clear understanding of the path to take. Provided these points are defined each line either starts or finishes with an endpoint or with a bifurcation point. After finding a bifurcation point we erase the pixels in the area around it with a certain threshold, to guarantee isolated segments.

2.1.3 Find Contours

The isolated segments are noticed through the function `cv2.findContours` from the OpenCV library. This allows the detection of contours within an image - contours are closed curves that define the boundaries

of a segment. The function returns a list of contours found in the image, with each contour represented as an organized list of points (where the order defines the path).

What is left is to order each of the segments such that it occurs in a continuous motion.

2.1.4 Order Path

The segments were ordered by proximity. The proximity threshold is defined in consensus with the number of pixels deleted in the segmentation phase. Every time closeness is not ensured the line is backtracked from a previously detected straight path. This prioritizes the seamless superimposed effect we intend to achieve in the final drawing. If backtracking were to be performed on curved lines the robots' unstable line control would be highlighted. This algorithm is not optimal since we do not explore the different path options beforehand. An optimal solution would require the implementation of a search algorithm that goes beyond the scope of this assignment.

2.1.5 Sampling

At this point, we have specified the route that the robot must take in order to complete the drawing. We have a large number of (consecutive) points that we want to minimize without compromising the accuracy of the drawing.

Points that are consecutive may introduce an error as, on a very small scale, may seem that the trajectory is erratic and does not develop smoothly from one point to the next. As not to account for consecutive points an initial sampling with a certain interval.

Next, the goal is to only account for points that are necessary to perform the trajectory. In other words, for a straight line, we want to include only its loose ends. For a curved line, we created an algorithm that regards the angle between consecutive points.

In fact, the angle is measured by making use of the inner product of vectors. The method considers every three consecutive points, it checks the angle between the vectors formed by the first and second points and the second and third points. If it is bigger than a certain threshold, then it means there is a significant shift in the trajectory, thus the second point must be included, this is illustrated in Figure 3. On the contrary, taking the example of a straight line since the angle is smaller, the middle point is discarded. In this way, we expect to reduce the number of points, maintaining the ones that are crucial to perform the trajectory.

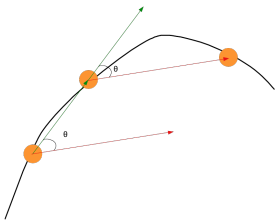


Figure 3: Sampling Illustration.

2.1.6 Normalization

The size of the drawing on paper must be determined. The drawing area chosen is 1.2 times larger than an A5 piece of paper. As a result, the width of the paper in Figure 4 is 2100 x 1.2 tenths of mm and the height is 1480 x 1.2 tenths of mm. An example of this operation is illustrated in Figure 4 with a scale of 6 by 3 instead.

Due to the high resolution of the received image, the scale is clearly larger than the paper, implying that the image must be normalized. Normalization requires two evaluations:

- When the height of the image exceeds the height of the paper, the height of the image must be lowered to match the height of the paper. However, there are some cases where it is possible, as shown in Figure 4, in the third drawing, that even when the image's height is equal to the paper's, its width is still greater. In this instance, as shown in the fourth drawing, normalization must be based on the width proportion rather than the height proportion.

- If the width of the image exceeds the width of the paper, the same thing occurs; in that case, the normalization is based on the width proportions. If the height is greater than that limit, the normalization is based on the height proportion.

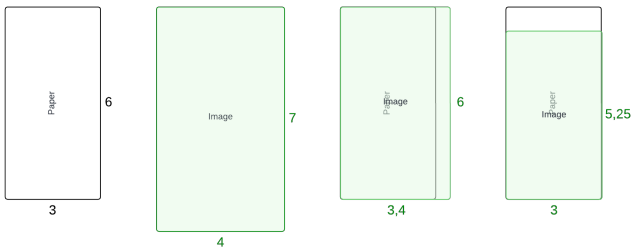


Figure 4: Normalization based on height proportion.

The following transformations of the corner coordi-

nates occur in the case depicted in Figure 4:

$$(0,0) \rightarrow (0,0)$$
$$(0,7) \rightarrow (0,5.25)$$
$$(4,0) \rightarrow (3,0)$$
$$(4,7) \rightarrow (3,5.25)$$

2.2 Trajectory Planning

This step describes some decisions regarding the robot's point selection. Beginning with the robotic arm's initial position follows the choice made about the pen's position and the additional feature of rolling the gripper to increase friction with the paper.

2.2.1 Calibration

Before starting to draw with the Scorbot ER-7, it is important to initialize the robot to a safe position. This ensures that the robot has enough space to move and perform the necessary drawing motions without hitting any obstacles or exceeding its physical or usage limits. We provide two options in which this can be performed:

1. **Manually** - where the robot's test bench is used to move the robot's end effector to a safe and accessible location away yet perpendicular from the workspace (A3 white paper);
2. **Non manually** - where the robot's end effector moves to a pre-defined safe point again perpendicular and away from the paper. This point was measured during a lab session. To achieve a more accurate and reliable latter motion this point is defined using joint coordinates so as to avoid the drawing to be affected or interrupted by factors such as joint limits or stiffness.

Furthermore, the option of homing the robot is set for non-manual initialization. Homing is the process of moving the robot to its default, say "home" position. It is important to home the robot before starting to use it because it ensures that it is in a predictable and controllable state.

Once the robot's initial point is set, we consider the pen's position while loading the points onto the robot.

2.2.2 Pen's position

We discovered that loading the points while the pen is resting on the workspace results in unnecessary pressure on the paper as a stain of tint in the draw. A possible solution is to lift the pen from the paper until all of the points are loaded. However, there is a disadvantage to this, since by loading the points in a lifted position, there is a need to assign the values of Z to touch the paper during the draw, this increases roughly one-third of the robot's processing time. Nevertheless, our choice was to maintain the pen off the paper until the drawing begins, in order to achieve a smooth clean illustration.

In the same way, after drawing all of the points, there is an extra point with the same coordinates as the previous one but a different Z. This choice seemed more obvious since its advantages could be achieved by adding only one point (which is approximately 6 seconds).

Overall, with the choices made, the line can be smooth and free of visible points at the beginning and end of the drawing.

2.2.3 Drawing Trajectory

To represent this trajectory, we use the processed data to create a vector that contains the Cartesian coordinates of each waypoint. The size of the vector is based on the sampling done during the image processing stage, and we use the `DIMP` function in ACL to initialize the vector.

Next, we iteratively specify the X, Y, and Z coordinates for each waypoint using the `SETPVC` function in ACL. These points are then input into the robot.

The robot's built-in kinematic algorithms take care of the necessary transformations to execute the desired motion and maintain a perpendicular pitch throughout the process. Finally, we use the `MOVES` function in ACL to smoothly interpolate between the points in the vector, resulting in a more realistic drawing motion.

2.2.4 Roll coordinates

To maintain a better grip on the paper while drawing, we decided to change the robot's roll coordinates over time. By adjusting these coordinates in the direction we were heading, we were able to maintain a more stable grip on the paper and prevent the pen from slipping (as shown in Figure 5).

(a) (b)

Figure 5: Drawing with the robot's gripper (a) parallel to the draw. (b) perpendicular to the draw.

It is also worth noting that the predicted trajectory is used to calculate the roll angle. In other words, an analysis is performed by calculating the `arctangent2` function with the difference in Y coordinates between the next point to be drawn (at time $k+1$) and the point under consideration (at time k), under the same for X coordinates. This calculation can be described as follows

$$\mathrm{R} = \texttt{atan2}\left(\frac{y_{k+1} - y_k}{x_{k+1} - x_k}\right)$$

2.3 Connection with the robot

The connection between the computer and the robotic arm is established using a USB cable and the `serial` library. To ensure reliable communication between the two devices, we implement a wait time to read each command. We also make sure to delete all created variables at the end of the process before disconnecting. To maintain the connection when uploading the trajectory we implement a loop that continuously sent the required commands to the robot whilst the feedback is printed in the user's console. This allows us to monitor the robot's status and make any necessary adjustments to the drawing trajectory.

2.3.1 Software and tools

The code for the robot was developed using Python and stored in a Git repository [1] hosted on GitHub .

3 Results

The results reached, the analysis of the tradeoff of parameters, as well as its evaluation, limitations, and future work, can be found further.

3.1 Overview of the Results

The overview of the results reached is present in Figure 6 and 7. To achieve these results, we modify one or two parameters, allowing us to analyze the data and select the best options for our solution.

Figure 6 presents three versions of the first test image.

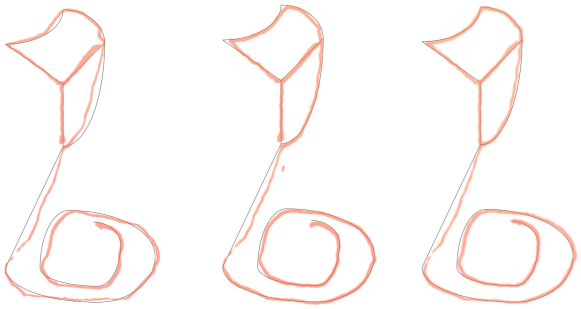
(a) (b) (c)

Figure 6: Drawings from `test_draw_1.png` (a) `SAMPLING` = 200 (b) `SAMPLING` = 150 (c) using roll's coordinates.

By visualizing Figure 6a and 6b, it is possible to state that a decrease in the interval of sampling, produces more points, and results in a smoother line. In fact, the drawing shifts less from the reference. This however comes with the drawback of an increase in the number of points, thus a higher processing time.

Additionally, the output using the roll coordinates, described in the last section is shown in Figure 6c. It may be seen that the drawing is less smooth since it stops at each point to perform the roll, which produces a large number of stains during the trajectory. Moreover, because we are saving double the points than before, it also takes double the time for the robot to process them. We realized that although in theory, this seemed like a good approach, in practice it proved to not perform that well and be very time-consuming.

Figure 7 presents three performances of the second test image.



(a) (b) (c)

Figure 7: Drawings from `test_draw_2.png` (a) `TRESHOLD` = 20 (b) `TRESHOLD` = 10 (c) with good endpoints.

In fact, comparing Figures 7a and 7b, it is possible to realize that a lower threshold for the angle, since is more sensitive to shifts in trajectory, will produce more points, thus, it will more accurately follow the reference. However, this comes with a weakness: the increased amount of points takes longer for the robot to process. Nevertheless, we decided that the accurate reference following was more advantageous over the time spent to process.

Lastly, we realized from the previous drawings that the pen wasn't reaching the bifurcation points. This issue was solved in Figure 7c.

Although we select a set of parameters to provide a robust output, the values used may ultimately depend on the drawing task's specific goals.

3.2 Evaluation and Analysis

Comparing the performance of the Scorbot ER-7 to relevant benchmarks or standards, we found that the accuracy and speed were generally on par with other similar robots. In terms of accuracy, the robot was able to draw the one-line drawing with a high degree of precision, with an average error of less than 0.2 cm. This is comparable to the repeatability of 0.1 mm for the Universal Robots UR5 [3], which is widely used in manufacturing and assembly tasks.

Overall, the performance of the robotic arm compares favorably to relevant benchmarks and standards in the field, and there is potential for further improvement through continued research and development.

3.3 Limitations and Future Work

There are a few limitations to the performance of the Scorbot drawing robot that should be considered when analyzing the results. One limitation is that too complex images may require too many waypoints to accurately represent the drawing trajectory, which could compromise the robot's memory and computational capabilities. This could potentially be addressed by implementing more efficient trajectory planning algorithms, that require less points, or using more powerful hardware.

Additionally, the robot is not using sensors to gather real-time information about its environment. Instead, it is relying on predefined coordinates that are coded into the control software. This means that the robot's motion is dependent on the current setup and is not able to adapt to changes in the environment or unexpected obstacles.

Another limitation is that we are working with Cartesian rather than joint coordinates and are not performing any kinematic calculations. While this approach has some benefits, it could potentially compromise the joints of the robot if the joint limits are exceeded or the robot is subjected to external forces. In the future, it may be worth exploring the use of joint coordinates or implementing kinematic calculations to address this potential issue.

Despite these limitations, the precision and scalability of this robot are promising, and this technology could potentially be applied to a wide range of applications where precise and repeatable motions are required, such as:

1. **Manufacturing and assembly**: The robot's precision and repeatability make it well-suited for tasks such as picking and placing components, soldering, or painting.

2. **Medical and laboratory**: It could be used for tasks such as handling and testing samples, performing surgeries, or dispensing medications.

3. **Art and design**: The robot could be used to create complex or large-scale drawings, sculptures, or modern installations.

These are just to name a few potential applications of this technology.

4 Conclusion

The Scorbot ER-7 has proven to be a reliable and effective tool for drawing one-line drawings.

Further research and development could focus on exploring the use of sensors and other specialized equipment to enhance its capabilities.

Contents

References

[1] Beatriz Lourenço, Isabel Portugal, and Rita Palma. *Laboratory Assignment Repository*. `https://github.com/blourenco217/robotics-1st-lab-assignment`. Accessed: 26-Dec-2022. 2022.

[2] Eshed Roboted. *Scorbot-ER7 User Manual*. Address, 1996. URL: `https://www.scribd.com/document/117998942/Manual-Scorbot-ER-VII`.

[3] Universal Robots. *UR5 Collaborative Industrial Robot*. `https://www.universal-robots.com/products/ur5-robot/`. Accessed: 26-Dec-2022.