THE UNIVERSITY OF
MELBOURNE

MCEN90028 Robotics Systems
Assignment 3 Report

Group 3

Hai Dang Nguyen 860308
Say Ee See 813641
BhargavRam Chilukuri 964069

28/4/2021

Contents

List of Figures

List of Tables

1 Introduction

In this part of the project, the report will focus on constructing the trajectories to move the end-effector from a given pose x_i with initial velocity $\dot{x}_i$ to a given final pose x_f with final velocity $\dot{x}_f$. An algorithm will be generated for generic cases and specifically applied in the problem of robot arm playing chess. These following objectives will be explicitly address throughout the report.

- To construct a trajectory generation algorithm, given the initial and fianl end-effector pose and velocities.
- To apply the trajectory generation algorithm to the chess playing robot problem, taking into account the context of the problem by considering the constraints and the goals.
- To relate the outcome of the trajectory generation algorithm to the material from previous report, by performing Inverse Kinematics on the resulting end-effector pose in the generated trajectory into a set of joint space trajectories, which will be fed as command into the controller of the robot.

2 Background

The project focuses on the design of the robot arm for picking up the chess piece, including the analysis of Forward Kinematics, Inverse Kinematics, and mechanical design considering the torque wrench due to structure weight. Previous analyses have been done with the consideration of a chosen chessboard size (130 x 130 mm). However, since there is a change in the chosen chessboard, the link length of the robot arm has been redesigned to make sure it can cover its new working space. Therefore, using the procedure in determining the link length presented in the report 1 with the new chessboard (320 x 320 mm), the new link length set has been chosen to be $[d_1, d_2, d_3, d_4] = [100, 300, 300, 80]$ (mm) and the base of robot arm will be located $240mm$ to the chessboard (as shown in Figure 1). Moreover, the Forward and Inverse kinematics results are identical in symbolic form, the torques generated due to structure mass are within the motor nominal torque range.

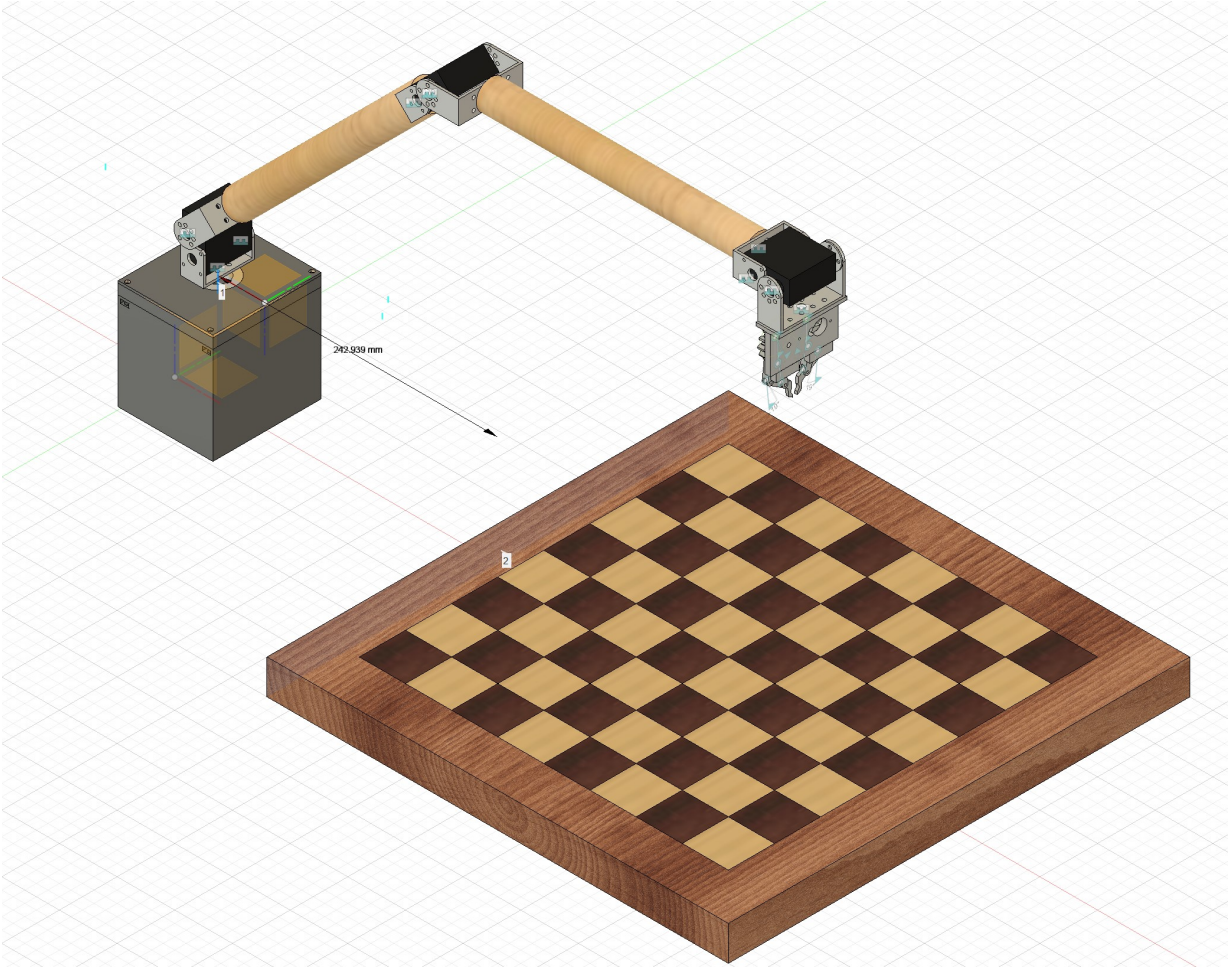


Figure 1: The layout, set up of the robot arm and chessboard

3 Trajectory Generation

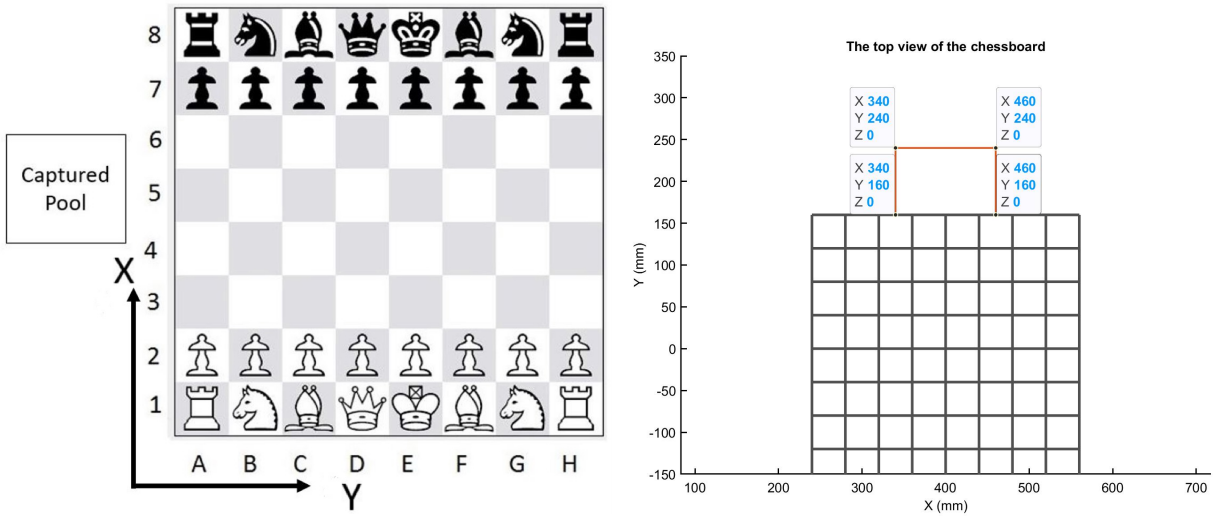
Different kinds of movement of the robot in playing chess are to be considered in this report. There are 8x8 locations on the chessboard the end-effector needs to reach, in addition to 2 others poses of the robot:

- The ready pose, where the robot will rest when not moving or wait for other player's turn.
- The location where the captured chess piece is placed outside of the board.

The structure of trajectory generation task will be conducted in sequence. In section 3.1, we will derive an algorithm to generate the coefficients of a cubic polynomial that describe the movement of the end-effector with respect to time given $(x_i, x_f, \dot{x}_i, \dot{x}_f)$ as initial pose, final pose, initial velocity, final velocity respectively. Since the robot arm is in 3D space, there will 3 different sets of coefficients associated with x, y, z to present poses of the end-effector at a given time. In addition to that, the orientation displacement of the robot arm will also be generated.

The chessboard layout is presented in Figure 2, where each square's side is $40mm$, the Y and X axis are respectively parallel with y_0, x_0 axis of the inertial frame. Assuming that the highest chess piece is $50mm$ high and the gripping pose of each chess piece on the board will be the point located $25mm$ vertically above the center of the square where it is located on. Therefore, the trajectory algorithm will take initial and final gripping point as inputs, along with their associated velocities. The length of time to finish the trajectory is assumed to be 5 seconds. The trajectory generation will be done using MATLAB.

Figure 2b presents the chessboard drawing in MATLAB where the captured pool is represented in orange with associated vertices. Assuming that the location for the robot arm to drop the piece into the captured pool is $(x, y, z) = (400, 20, 50)$ (mm). Moreover, for the '*rest*' pose, the location of the end-effector will be $(x, y, z) = (240, 0, 300)$ (mm) which corresponds to joint displacement $(Q_1, Q_2, Q_3, Q_4) = (0, 1.56, -1.5, -1.63)$ (rad), shown in Figure 3.



(a) The top view of the chessboard

(b) The top view of the chessboard in MATLAB

Figure 2: The top view of the chessboard

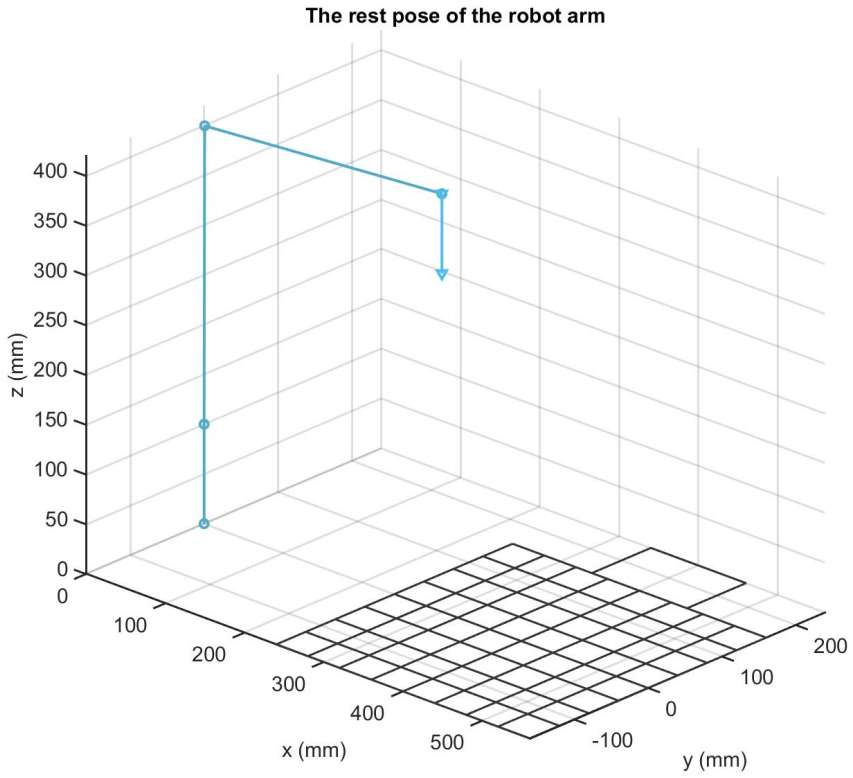


Figure 3: The rest pose of the robot arm

3.1 Determine sets of polynomial coefficients

3.1.1 Translational motion

First of all, we will look at the use of polynomials to describe the trajectory of a variable $x(t)$. Consider a cubic polynomial:

$$x(t) = a_3t^3 + a_2t^2 + a_1t + a_0 \tag{1}$$

$$\implies \dot{x}(t) = 3a_3t^2 + 2a_2t + a_1 \tag{2}$$

The initial, final position (x_i, x_f) and velocities $(\dot{x}_i, \dot{x}_f)$ are known, substitute into Equation 1 and 2, we have

$$\begin{aligned} x(t_i) &= a_3t_i^3 + a_2t_i^2 + a_1t_i + a_0 \\ \dot{x}(t_i) &= 3a_3t_i^2 + 2a_2t_i + a_1 \\ x(t_f) &= a_3t_f^3 + a_2t_f^2 + a_1t_f + a_0 \\ \dot{x}(t_f) &= 3a_3t_f^2 + 2a_2t_f + a_1 \end{aligned} \tag{3}$$

which can be presented in the matrix form as

$$\begin{bmatrix} x(t_i) \\ \dot{x}(t_i) \\ x(t_f) \\ \dot{x}(t_f) \end{bmatrix} = \begin{bmatrix} t_i^3 & t_i^2 & t_i & 1 \\ 3t_i^2 & 2t_i & 1 & 0 \\ t_f^3 & t_f^2 & t_f & 1 \\ 3t_f^2 & 2t_f & 1 & 0 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} \tag{4}$$

$$\implies \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} t_i^3 & t_i^2 & t_i & 1 \\ 3t_i^2 & 2t_i & 1 & 0 \\ t_f^3 & t_f^2 & t_f & 1 \\ 3t_f^2 & 2t_f & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} x(t_i) \\ \dot{x}(t_i) \\ x(t_f) \\ \dot{x}(t_f) \end{bmatrix} \tag{5}$$

This equation has been implemented in MATLAB as *solveForSpline()* function. To find the trajectory of x, y, z, we applied this function for each component x, y, z of both initial and final poses, with initial and final velocities are 0 from $t_0 = 0$ to $5s$. Hence we come up with 3 sets of polynomial coefficients for the trajectory for the 3 dof of the translational motion.

3.1.2 Orientation displacement

Any rotation can be written as one rotation about one axis with one angle. In order to study about the orientation of the movement of the robot arm, the first step is to identify the rotation axis $\hat{k}$ and the rotation angle θ_f from initial to final pose using Equation 6, 7 and 8 (according to [1]). The rotation angle θ is the reference angle of the rotation, so the initial angle $\theta_i = 0$ and final angle θ_f.

$${}^i_F R = {}^0_i R^T \cdot {}^0_F R \tag{6}$$

$$\theta_f = 2\cos^{-1}\left(\frac{1}{2}\sqrt{1 + r_{11} + r_{22} + r_{33}}\right) \tag{7}$$

$$\hat{k} = \begin{bmatrix} k_x \\ k_y \\ k_z \end{bmatrix} = \frac{1}{2\sin\theta_f} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \tag{8}$$

where

- ${}^0_i R$, ${}^0_F R$: are respectively rotational matrices from initial and final pose to the inertial frame. These matrices can be generated by projecting the axes of the end-effector frame on the inertial frame at the initial and final pose.

- r_{ij}: the element corresponding at row i and column j of the rotational matrix ${}^{i}_{F}R$.

Having obtained the value of θ_f, we can now interpolate the angular displacement using cubic polynomial such that

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{9}$$

From MATLAB, we can solve for the coefficients by using *solveForSpline()* function. Then we can have the description of the trajectory of θ from time $t_i = 0$ to t_f. We also have the value for $\theta(t)$ for every time instance. The interpolated $(\theta,\ \hat{k})$ can be converted into Euler parameters as quaternion form and will now provide the reference orientation at any given time t.

$$\begin{aligned} e_1(t) &= k_x \sin\frac{\theta(t)}{2} \\ e_2(t) &= k_y \sin\frac{\theta(t)}{2} \\ e_3(t) &= k_z \sin\frac{\theta(t)}{2} \\ e_4(t) &= \cos\frac{\theta(t)}{2} \end{aligned}$$

According to [1], we can convert the Euler parameters back to a rotational matrix ${}^{i}_{t}R$ at any given time t during the designed trajectory using the following equation

$${}^{i}_{t}R(t) = \begin{bmatrix} -2\,{e_2}^2 - 2\,{e_3}^2 + 1 & 2\,e_1\,e_2 - 2\,e_3\,e_4 & 2\,e_1\,e_3 + 2\,e_2\,e_4 \\ 2\,e_1\,e_2 + 2\,e_3\,e_4 & -2\,{e_1}^2 - 2\,{e_3}^2 + 1 & 2\,e_2\,e_3 - 2\,e_1\,e_4 \\ 2\,e_1\,e_3 - 2\,e_2\,e_4 & 2\,e_1\,e_4 + 2\,e_2\,e_3 & -2\,{e_1}^2 - 2\,{e_2}^2 + 1 \end{bmatrix} \tag{10}$$

$${}^{0}_{t}R(t) = {}^{0}_{i}R \cdot {}^{i}_{t}R(t) \tag{11}$$

The rotational matrix in Equation 10 can be expressed with respect to the inertial coordinate frame by using Equation 11. Hence, we have obtained the orientation displacement of the end-effector during a trajectory. This will be implemented in MATLAB as *InterpolatingOrientation()* function where inputs are initial, final poses and time instance, which gives the outputs of coefficients of interpolated $\theta(t)$, $\hat{k}$ and array of rotational matrices at each time instance.

3.1.3 The algorithm

Figure 5 presents the pseudocode of the algorithm, which is a generic solution covering the case where the trajectory has with more than 1 via point. For this section, we only consider initial and final pose of the end-effector, hence we only have one segment.

Algorithm

```
% get an initial and final pose from the label of the square on the chessboard
[XI, XF] = ChessBoardLocation(initial, final)

% list of points on the trajectory, including via points (if exist)
points = [XI, via_point_1, via_point_2, XF];

% list of time instance related to each point in the points array
time = getTimeArray(points, t_i, t_f);

for (each segment) in points:
    % iterate through each pair of points
    % iterate through each pair of time instance with the points

    for (each component x, y, z) in XI and XF:
        % solve for polynomial coefficients for each component
        % record the value x, y, z and velocity of the trajectory at each increment of time

        [coefficients, pose_vel_time] = solveForSpline(XI, XF, v_0, v_f, t_i, t_f)

        % solve for polynomial coefficients of the orientation displacement
        % record the value of phi, omega of the trajectory at each increment of time
        % solve for orientation of the rotation axis k_hat
        [coefficients, phi_omega_time, k_hat, R_t_array] = InterpolatingOrientation(x_i, x_f, t_i, t_f)

        (assign the results in a multi-dimension array for plotting)
    end
end
```

function
[timearray] = getTimeArray(points,t_i, t_f)

Input: points, t_i, t_f
Output: [timearray]

function
[XI, XF] = ChessBoardLocation(initial, final)

XI, XF [x; y; z]: Initial and Final gripping pose in 3D
initial: The initial square, ex: 'e4', 'a1',...
final: The final square, ex: 'f3',...

function

[coefficients, pose_vel_time] = solveForSpline(XI, XF, v_0, v_f, t_i, t_f)

Input: XI, XF, v_0, v_f, t_i, t_f

Output: [coefficients], [pose, velocity, time]

function:

[coefficients, phi_omega_time, k_hat, R_t_array] = InterpolatingOrientation(x_i, x_f, t_i, t_f)

Input: x_i, x_f, t_i, t_f

Output: [coefficients], [phi omega time], [k_hat], [R_t_array]

Figure 4: Pseudocode of the trajectory generation algorithm

3.2 Determine sets of polynomial coefficients for multiple segments

The trajectory generation equation presented in the previous part of the report can be applied to describe a linear trajectory in 3D space. However, as the chessboard has multiple chess pieces located on it, the trajectory of the end-effector should be designed such that the picked chess piece will not collide with any of the chess pieces remained on the board. To do so, we will include 2 via points into the trajectory, these via points take the role as checkpoints the end-effector needs to reach during moving towards the final pose. In order to make sure it can clear other pieces in the way, we assign 2 via points, one that is located 100mm upwards in z_0 axis from the initial pose whereas the other one locates 100mm upwards in z_0 axis from the final pose.

Therefore, the trajectory will be separated into 3 different segments, each of which have 4 sets of polynomial coefficients to perform the move. In order to implement this in MATLAB, we have combined the 2 via points into an array called *points* where the initial and final pose will locate at the first and end indices of the array, shown in the algorithm in Section 3.1.3. Then, we will loop through the array and take a pair of poses and treat them as initial and final pose and substitute into the function *solveForSpline()* and *InterpolatingOrientation()* to solve for the 4 sets of polynomial coefficients. Each pair of poses represents each segment in the trajectory, giving N points, we always have $N-1$ segments, total we will have $(N-1)*4$ sets of polynomial coefficients. In addition to that, the amount of time going through each segment of the end-effector is assumed to be proportional to the ratio of the distance of that segment over the overall distance of the trajectory. The reasons behind this is to make sure the acceleration on the end-effector will not go beyond a certain threshold value, which can result in the chesspiece to be dropped off along the trajectory. Having to spend a small amount of time on a long distance will require the motor to provide much more effort to accelerate to a desirable velocity to reach to the final pose in time. Moreover, this is also to minimize the control error in the motor, preventing the chess piece from being dropped off at the incorrect square.

The algorithm for constructing the trajectory with multiple via points is as same as the one presented in Section 3.1.3

Algorithm

```
% get an initial and final pose from the label of the square on the chessboard
[XI, XF] = ChessBoardLocation(initial, final)

% list of points on the trajectory, including via points (if exist)
points = [XI, via_point_1, via_point_2, XF];

% list of time instance related to each point in the points array
time = getTimeArray(points, t_i, t_f);

for (each segment) in points:
    % iterate through each pair of points
    % iterate through each pair of time instance with the points

    for (each component x, y, z) in XI and XF:
        % solve for polynomial coefficients for each component
        % record the value x, y, z and velocity of the trajectory at each increment of time

        [coefficients, pose_vel_time] = solveForSpline(XI, XF, v_0, v_f, t_i, t_f)

        % solve for polynomial coefficients of the orientation displacement
        % record the value of phi, omega of the trajectory at each increment of time
        % solve for orientation of the rotation axis k_hat
        [coefficients, phi_omega_time, k_hat, R_t_array] = InterpolatingOrientation(x_i, x_f, t_i, t_f)

        (assign the results in a multi-dimension array for plotting)
    end
end
```

function
[timearray] = getTimeArray(points,t_i, t_f)

Input: points, t_i, t_f
Output: [timearray]

function
[XI, XF] = ChessBoardLocation(initial, final)

XI, XF [x; y; z]: Initial and Final gripping pose in 3D
initial: The initial square, ex: 'e4', 'a1',...
final: The final square, ex: 'f3',...

function

[coefficients, pose_vel_time] = solveForSpline(XI, XF, v_0, v_f, t_i, t_f)

Input: XI, XF, v_0, v_f, t_i, t_f

Output: [coefficients], [pose, velocity, time]

function:

[coefficients, phi_omega_time, k_hat, R_t_array] = InterpolatingOrientation(x_i, x_f, t_i, t_f)

Input: x_i, x_f, t_i, t_f

Output: [coefficients], [phi omega time], [k_hat], [R_t_array]

Figure 5: Pseudocode of the trajectory generation algorithm

3.3 Resulting trajectories

This section will focus on visualize the results by implementing the method presented in the previous sections. The trajectory will be moving a rook chess piece from one square to another that is 4 squares away in the positive Y direction as shown in Figure 2a. There will be another knight chess piece located in the square immediately adjacent the square of the initial pose. Specifically, the initial and final pose are chosen to be $(x_i, y_i, z_i) = (380, -60, 50)(mm)$ and $(x_f, y_f, z_f) = (380, 100, 50)(mm)$ respectively. To ensure that our robot can lift the rook to a height that allows it to clear the knight that is in the way, we will select 2 via-points directly above the initial and final pose coordinates at $(x_{s1}, y_{s1}, z_{s1}) = (380, -60, 150)(mm)$ and $(x_{s2}, y_{s2}, z_{s2}) = (380, 100, 150)(mm)$ respectively. Having defined our 3 segments, we can then solve for the 4 sets of polynomial coefficients for each segment and plot the resulting x,y and z displacements of the end-effector over time. Figure 7 illustrates the location of the end-effector from $t = 0$ to $t = t_f$ along with the timestamp and coordinates of the via points on each respective axis.

The orientation angle of the end-effector is also plotted in Figure 6, the rotation axis was determined to be $\hat{k}_2 = (0, 0, -1)$ on Segment 2, while there were no rotation occurring during Segment 1 and 3. On each segment, $\theta(t)$ was calculated and it is the reference angle to the initial angle of the starting point of that segment, which is always equal to 0. Therefore, to obtain the reference $\theta(t)$ to the initial pose of the trajectory, on each new segment, we have to add the final θ value of the previous segment. In Figure 6, during the first segment, the angle remains unchanged which is reasonable as the end-effector moves vertically upwards along the z_0-axis. This has also happened during Segment 3 where the end-effector moves vertically downwards along z_0-axis, which was validated by the results of $\hat{k}_1$ and $\hat{k}_3$. During Segment 2, although the end-effector translated horizontally along y_0-axis, its coordinate system has changed referring to the inertial frame. Considering the orientation, this change can be interpreted as a rotation about $\hat{k}_2 = (0, 0, -1)$ with an angle of 0.41 rad within 2.23 seconds.

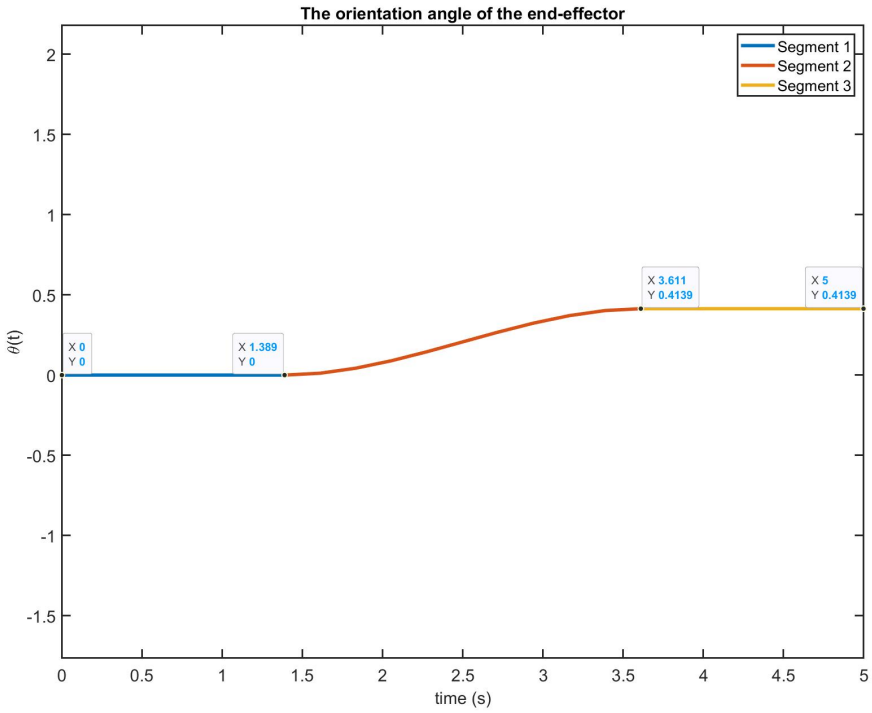


Figure 6: The interpolated orientation of the end-effector

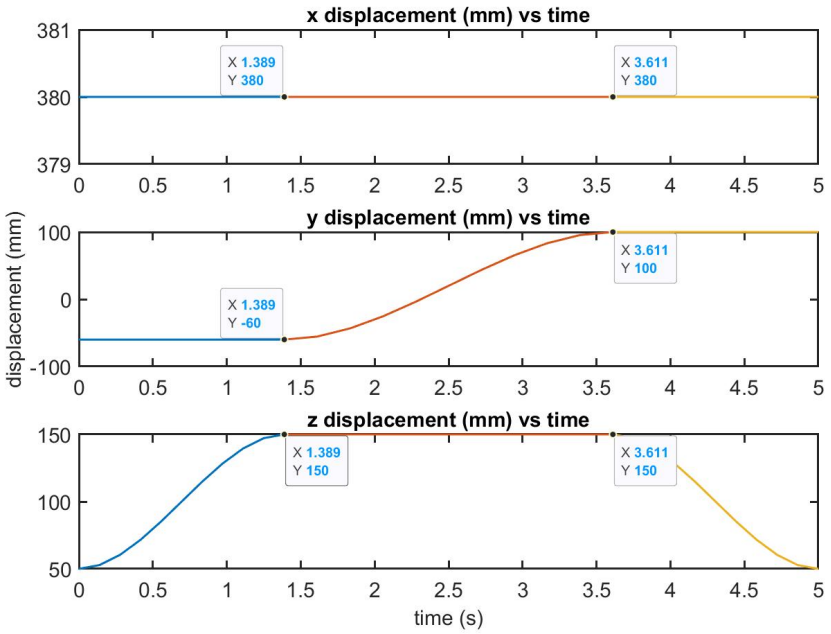


Figure 7: x,y,z displacement of end-effector over time

To better illustrate the spatial motion of the end-effector, we will also plot its trajectory in the XY and ZY plane.

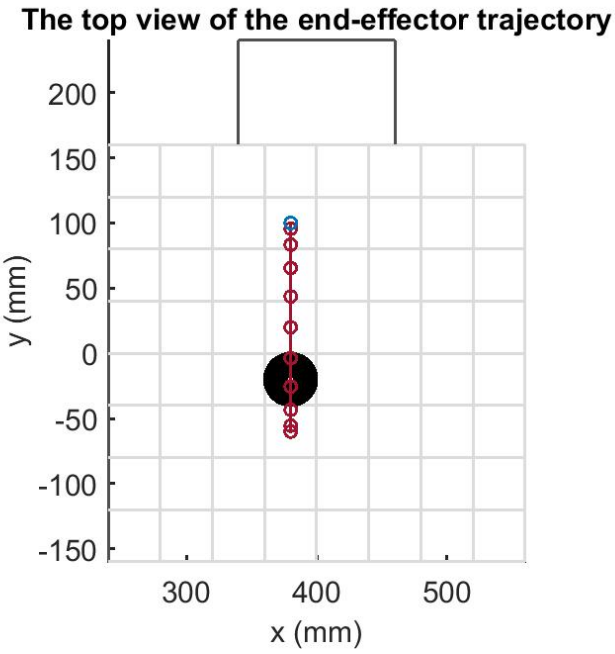


Figure 8: Top view of end-effector trajectory

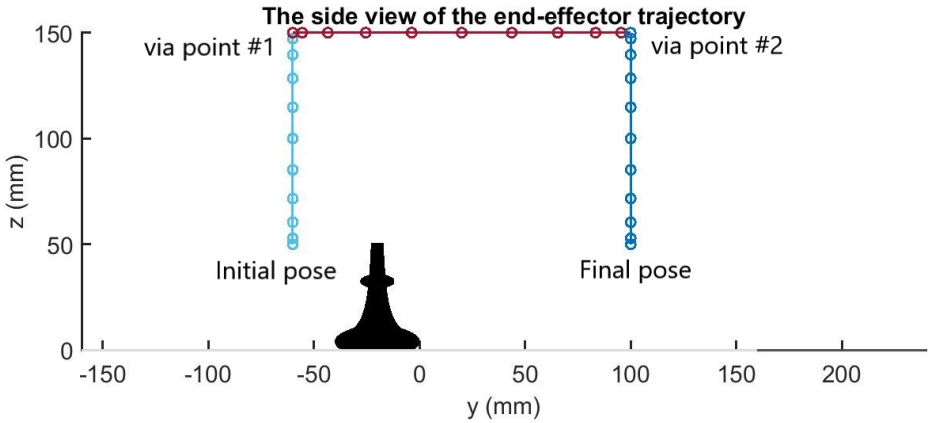


Figure 9: Side view of end-effector trajectory

From Figure 8, it can be inferred that the movement of the end-effector will simply be a straight line along the y-axis when viewed from above. In contrast, the side view in Figure 9 demonstrates how the end-effector will first move along the z-axis to the height of the first via-point where it can confidently clear the knight piece, before moving along the y-axis and begin lowering to its final position after it reaches the second via-point.

We can now superimpose upon the end-effector trajectory its corresponding joint configuration at time steps $t = kt_f/10$, where $k = 0, 1, 2...10$. Using Inverse Kinematics, we can obtain the joint displacement associated with each pose during the trajectory, from which we can represent the robot arm as a stick plot, shown in Figure 10:

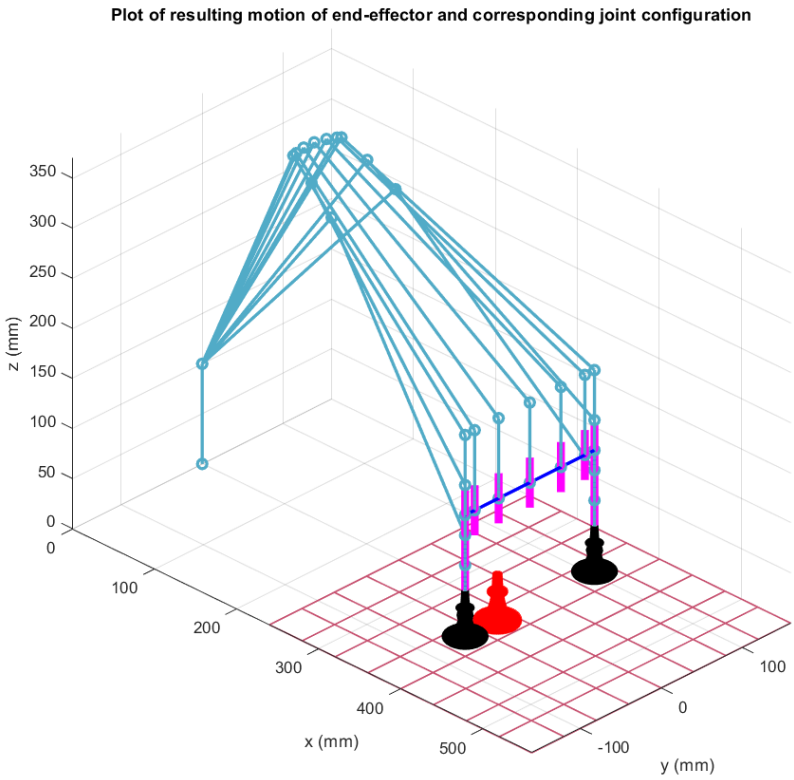


Figure 10: Resulting motion of end-effector and corresponding joint configuration

Note that the 2 black chess pieces represent the initial and final location of our rook, and the red chess piece represents the knight that is located in the square immediately adjacent the initial pose. The blue line illustrates the trajectory of the end-effector in 3-dimensions, whereas the magenta bars are representations of the rook at each time step t. We will assume the extreme case where the rook has a height of the largest chess piece at 50mm. From Figure 10 we see that the selected trajectory is successful at enabling the end-effector to maneuver the rook to its final location without hitting the knight that is in its direct path, as the magenta bars do not intersect with the red chess piece at any given time. Figure 11 also shows the robot arm pose at each time instance.

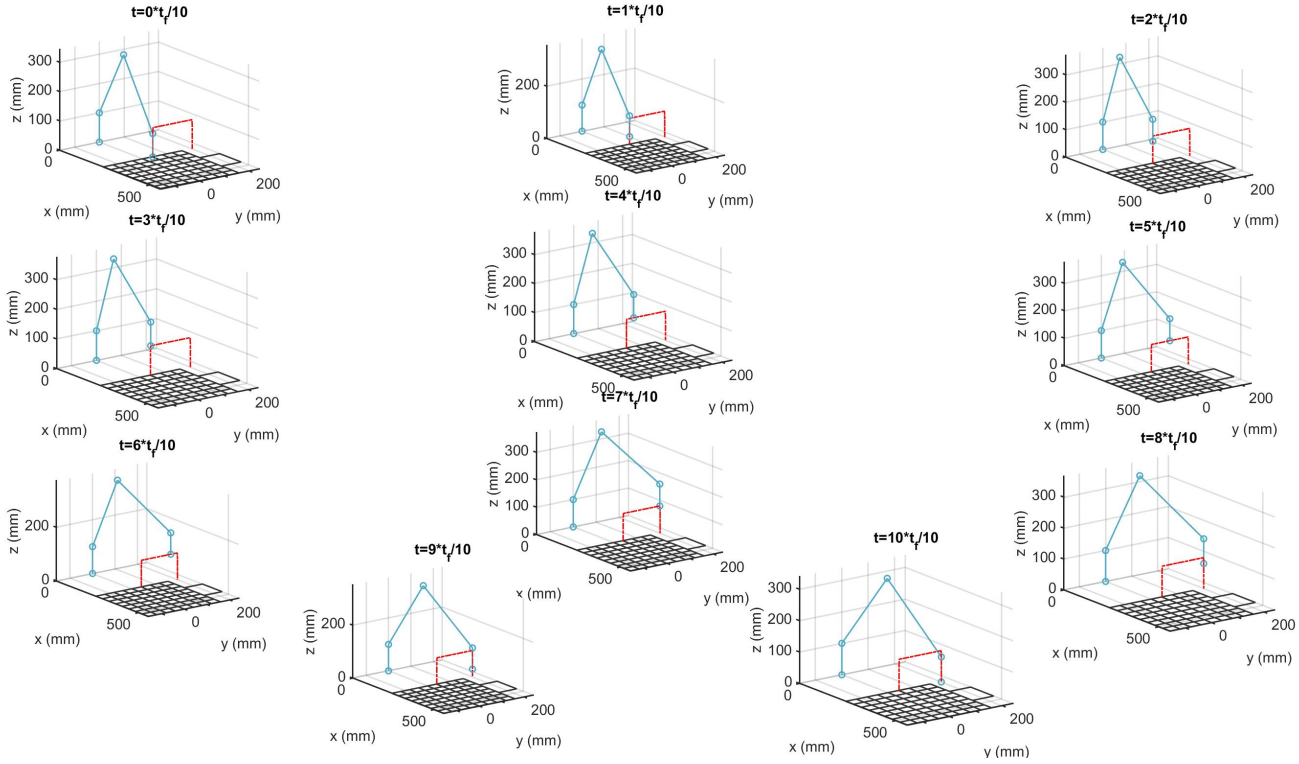


Figure 11: The robot arm pose at each time instance

4 Conclusion

This report covered the construction of trajectories with which to move the end-effector from a given initial pose and velocity $[x_0, \dot{x}_0]$ to a given final pose and velocity $[x_f, \dot{x}_f]$. The algorithm for determining the sets of polynomial coefficients for translational motion and orientation displacement was derived, and further improvements were made to take into account the use of via-points. From there, we then plotted the sample trajectories for the end-effector and its corresponding joint configuration to justify if our selection of via-points can allow the end-effector to move the rook chess piece to its final position without hitting the knight in its path. The evaluation shows that our robot can successfully accomplish this task for via points $(x_{s1}, y_{s1}, z_{s1}) = (380, -60, 150)(mm)$ and $(x_{s2}, y_{s2}, z_{s2}) = (380, 100, 150)(mm)$.

References

[1] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 2nd ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1989.