THE UNIVERSITY OF
MELBOURNE

MCEN90028 Robotics Systems
The Final Report

Group 3

Hai Dang Nguyen 860308
Say Ee See 813641
BhargavRam Chilukuri 964069

20/5/2021

Contents

1 Introduction **2**

2 Summary **3**

3 Robot and Task Integration **4**
3.1 Hardware and Software 4
3.2 Motion Planning 5
3.3 Task Implementation 5
3.4 Calibration 9

4 Mechanical Design **11**

5 Discussion **15**
5.1 Results 15
5.2 Limitation and Further Improvement 17

6 Conclusion **19**

List of Figures

1 The schematic of the robot arm 2
2 Pseudocode of the trajectory generation algorithm 3
3 List of moves to be completed in Task 1 4
4 Block diagram of robot in position control mode 6
5 Task space velocity control architecture 7
6 Task space velocity control block diagram for task 2 8
7 Pseudocode for motion control in task 2 8
8 Bishop frame and inertial frame 8
9 The illustration of the pseudocode for the knight 9
10 The assembly of the robot arm 11
12 Isometric view of the base 12
13 Cross-section of the base of the arm 13
14 Gripper 3D model 14
15 Plots of joint space displacement of robot vs. reference trajectory 15
16 Plots of task space pose vs. reference pose 16
17 Poor cut wood dowel 17

List of Tables

1 DH Table 3
2 Inverse Kinematics results 3
3 PID parameters value for each motor 6

1 Introduction

This project focuses on building a robotic manipulator to move chess pieces on the chess board: moving chess pieces from one square to the other, over other chess pieces and moving captured pieces out of the board. Over the previous reports, there are different studies providing fundamental bases for the design phase of the project, including Forward and Inverse Kinematics, Velocity and Pose relationship, trajectory generation.

This report will present the final mechanical and controller design of the chessbot and present the implementation of previous studies. The following objectives will be explicitly addressed throughout the report:

- To present in detail the implementation of previous studies into the physical design of the chessbot
- To present the overall outcome of the project
- To evaluate the first two objectives and propose further improvements

Section 2 will present a summary of the previous studies. Table 1 presents the Denavit-Hartenberg (DH) table and parameters, used for calculating the Transformation Matrix of the end-effector. Whereas, Table 2 presents the Inverse Kinematics result. The relation of task space and joint space velocity will also be presented in Equation 1. Lastly, Figure 2 presents the pseudocode of the trajectory generation algorithm which will be using in the final phase of this project.

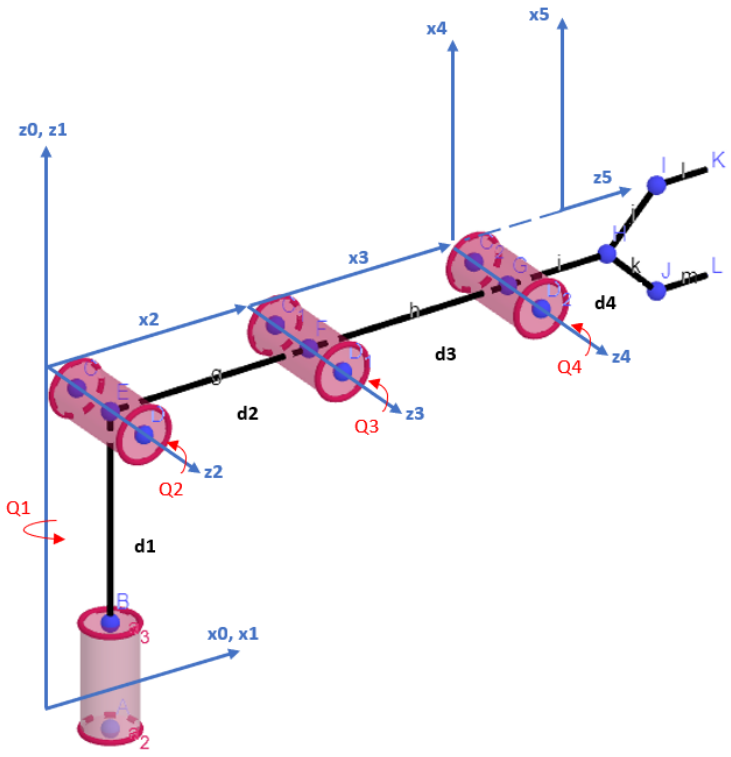


Figure 1: The schematic of the robot arm

2 Summary

Design and kinematics of the robot
A chess board of 400mm x 400mm was considered for the project, of which the playing area has a dimension of 320mm x 320mm. Hence the inter-square side is 40mm.

To obtain the optimal link lengths for covering the entire workspace, the furthest and closest location on the chessboard were taken into consideration, using geometry, the range of suitable link length was identified. From there the final link length was determined as $[d1, d2, d3, d4] = [100, 300, 300, 80]$ (mm)

Using the Denavit-Hartenberg convention (Table 1) to obtain the Transformation matrix, a concise presentation of the end-effector pose and orientation in the inertial frame. Table 2 shows the Inverse Kinematics result, calculating the joint displacement from the task space position of the end-effector.

Table 1: DH Table

i	a_{i-1}	α_{i-1}	d_i	Q_i
1	0	0	d_1	Q_1
2	0	$\pi/2$	0	Q_2
3	d_2	0	0	Q_3
4	d_3	0	0	$Q_4 + \pi/2$
E	0	$\pi/2$	d_4	0

Table 2: Inverse Kinematics results

Q_1	$\tan^{-1} y/x$
Q_2	$\tan^{-1}(z'/x') + \sin^{-1}(d_3 \sin\gamma_2/\sqrt{x'^2 + z'^2})$
Q_3	$-\cos^{-1}((x'^2 + z'^2 - (d_2^2 + d_3^2))/(2d_2d_3))$
Q_4	$Q_2 - Q_3 + pi/2$

Jacobian matrix
The relationship of the end-effector velocity and the joint angular rate is presented in Equation 1. The Jacobian matrix will be useful in task space or joint space velocity control.

$$\begin{bmatrix} {}^0\vec{v}_E^T & {}^0\vec{\omega}_E^T \end{bmatrix}^T = J(Q) \cdot \begin{bmatrix} \dot{Q}_1 & \dot{Q}_2 & \dot{Q}_3 & \dot{Q}_4 \end{bmatrix}^T \tag{1}$$

Trajectory Generation algorithm

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 2: Pseudocode of the trajectory generation algorithm

3 Robot and Task Integration

The main objective of the project consists of mechanically designing and manufacturing a robot that can execute 2 tasks listed below:

- *Task 1*: From a 'home' position, the robot is expected to move and pick up a piece from its current location to a target square. Then, the robot will return to its home position before attempting the next maneuver in the provided move list in Figure 3.

Move number	From	To
1	e2	e4
2	g1	f3
3	f1	b5
4	b1	c3
5	e1	g1
6	h1	f1
7	e4	d5
8	d5	captured pool
9	c3	d5
10	d5	captured pool
11	b5	c6
12	c2	c3
13	f3	g5
14	d2	d4
15	d1	f3
16	f3	d5
17	c1	captured pool
18	a1	c1
19	d5	captured pool
20	g1	h1
21	f2	f4
22	g5	h3
23	g2	g3
24	c1	d1
25	f1	e1

Figure 3: List of moves to be completed in Task 1

- *Task 2*: For various chess pieces, a human user is able to push the end-effector in the direction that is allowed for each respective piece, with the robot locking the displacement of the end-effector in all other directions. This task is to be performed on a rook, bishop and knight piece.

3.1 Hardware and Software

A project kit was provided, including some basic components with the required software describing its basic functionality. They are:

- 1 x Arduino Compatible Mega
- 1 x FE-URT-1 Serial to UART Board
- 1 x 7.5V 45W Power Supply & Cord

- 1 x 2.1mm DC Female to Leads
- 20 x M-F Jumper Leads
- 1 x Adafruit Analog 2-axis Joystick with Select Button
- 4 FeeTech SCS15 16.5kg.cm Servo Motors + Accessories
- 2 x FeeTech SCS009 0.76kg.cm Servo Motors + Accessories

For motor selection, four of SCS15 will be used for rotation movement of $Q_1,\ Q_2,\ Q_3,\ Q_4$, whereas SCS009 will be use for the end-effector/gripper (Q_5). Note that, movement of Q_5 is not taken into consideration for trajectory generation and controller design process. MATLAB was used for run the program controlling the motor as well as handling numerical calculations.

3.2 Motion Planning

Task 1

For motion planning, a database of each pick up position on the chessboard will firstly be constructed so that given a square on the chessboard, say *'a1'*, the function will return (260, 140, 60) mm as the pose of the end-effector for picking up a piece at *'a1'*. Moreover, a home position for the chessbot is defined to be (320, 0, 310) mm as where it will rest and wait for opponent's move when it finished its move.

Having got home, pick up and drop off poses, given one move ID, the robot is then need to perform three different paths starting from the home pose to the pick up pose (path #1) and deliver to the drop off pose (path #2) before returning back to the home pose (path #3). Applying the trajectory generation algorithm (from report 3) for each pair of poses, a list of setpoints will then be generated for each path, which the chessbot will need to perform in sequence. These setpoints are the different reference poses of the end-effector in the task space. The further implementation of this will be presented in Section 3.3.

Task 2

For task 2, the user should be able to move the end-effector of the chessbot along the path of the rook, bishop and knight projected on the chessboard. There will be no pre-defined trajectory for this task. Whenever the user drag the end-effector the chessbot will move in compliant with the indicated path. In addition, a pre-defined pose is chosen using position control mode on the motors before switching to velocity control mode for implementing this task.

3.3 Task Implementation

PID parameters

Table 3 presents the chosen parameters for the inner PID controller of the 5 motors implemented on the chessbot for task 1. These values were chosen based on trial-and-error basis and observation. Note that, motor ID from 1 to 4 respectively refer to joint 1 to 4 and motor ID 5 is the SCS009 implemented on the gripper.

Table 3: PID parameters value for each motor

Motor ID	K_P	K_I	K_D
1	7	7	1
2	7	5.5	0.5
3	7	2	0.5
4	4	4	1.5
5	10	1	0

Task 1

Having derived the components (i.e. Jacobian matrices & trajectory generation functions) necessary for executing the chess-playing tasks of the project, we can now implement linear controllers for our motor manipulators that will be required for both tasks. For task 1, we opted for the *position control* method - which involves first converting the target task-space displacement at each setpoint of the trajectory into joint angle displacements using the inverse kinematics derived in report 1. Then, the target joint angles are sent to the servo motors, whose internal encoders will allow for the motors to correct their positions and reach the desired angles without the need for an additional feedback regulator. In our trajectory generation, we utilised the algorithm developed in report 3 and split each segment (the path between 2 reference points) into 11 setpoints - which allowed for finer tuning of the motor angles and prevented the motor from having to increase its rotor speed drastically between each reference point. The block diagram below illustrates the position control process:

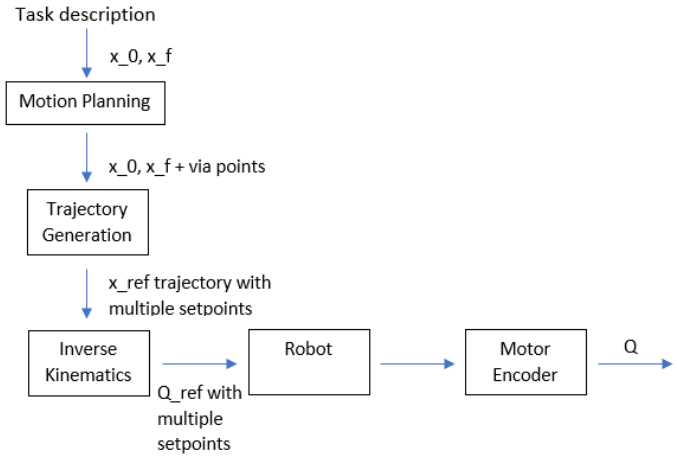


Figure 4: Block diagram of robot in position control mode

Note that this approach was chosen over alternative methods due to its reliability; task 1 required the robot to be able to execute a series of moves on the chessboard and place chess pieces of various sizes at their designated target squares. We found that utilising *velocity control* resulted in a significantly jerkier motion due to oscillations of the servo motors, which in turn yielded a less accurate performance by our robot.

Task 2

For task 2, the task space velocity controller will be implemented to control the pose of the end-effector in task space. The starting point for this task is to implement the control loop for controlling the chessbot to be stationary at a pre-defined pose. The motion control architecture will be built based on the model presented in Figure 5. Note that

- Controller: Proportional controller with gain K_p

- $x_{ref}(t)$: Reference pose of the end-effector in task space.
- $\dot{x}_{ref}(t)$: Reference velocity of the end-effector in task space, including the translational and orientational velocity
- e, $\dot{e}$: Pose and velocity errors respectively
- $\dot{x}_c(t)$: Task space velocity command - Control effort (output from the controller), not shown in Figure 5,

$$\dot{x}_c = \dot{x}_{ref} + K_p \cdot e \tag{2}$$

- $\dot{q}_c$: Joint space velocity command sent to the motors

$$\dot{q}_c = J^{-1} \cdot \dot{x}_c(t) \tag{3}$$

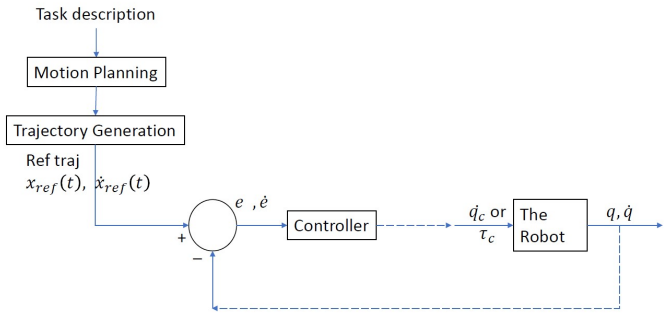


Figure 5: Task space velocity control architecture

In theory, the controller architecture was shown in continuous domain whereas in practice, the controller needs to be designed in discretized version with sampling time relied on the capabilities of chosen hardware. Having trying to implement the full control architecture as shown in Figure 5, we have got some issues in computational cost, which will be discussed in the following section. In discrete control, the motor needs to response in the small amount of time - high frequency to adjust the error in position due to the mechanical structural weight. Having a high sampling time means that the error is large and controller effort gets amplified to adjust for that, which in this case caused instability in the system. Therefore, given the limitation in time, mechanical design and hardware, we decided to reduce the complexity in computation to get faster sampling time.

In other words, we decided to reduce the number of degrees of freedom of the chessbot to 3 instead of 4. This means, we only control the first 3 joint Q_1, Q_2, Q_3 as Q_4 is entirely dependent on Q_2 and Q_3. Moreover, we only control the translational pose, hence the Jacobian matrix has reduce its size from 4-by-6 to 3-by-3 and the computational speed has increased up more than 10 times faster than before. The final motion control architecture is shown in Figure 6 with the pseudocode shown in Figure 7.

The controller K_p was chosen by trial-and-error method, starting with a small gain and incrementally increase to a value at which the chessbot can stay around the pre-defined pose. Initially, $K_p = 12$ for translational errors in x, y, z components, which results in oscillation with large amplitude in y direction, meaning that the gain for error in y should be reduced. Therefore, the new gain was chosen as $K_p = 12$ for x and z, $K_p = 1.8$ for y.

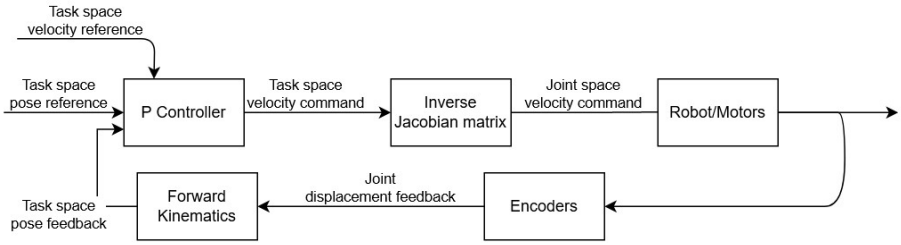


Figure 6: Task space velocity control block diagram for task 2

While (true)

Get feedback from motors, adjust with the offset

Calculate task space pose using Forward Kinematics

Calculate the pose error, get sampling time

Calculate control effort, task space velocity command

Calculate joint space velocity command using inverse Jacobian matrix

Convert velocity command to the ones fitted with the motors' setup

Send velocity command to motors

end

Figure 7: Pseudocode for motion control in task 2

For the task of moving the rook along X or Y direction, we follow the pseudocode in Figure 7 and we adjust the controller such that as there should not be any control effort to be made on the error in X or Y components depends on the type of movement. In other words, to be able to move along X axis, the gain for error in translation along X equals to 0, and similar for Y.

For moving the bishop along the diagonally on the chessboard, the idea will be as same as moving the rook along X or Y direction, except it will happen in another frame, i.e bishop's frame, shown in Figure 8. First of all, we can get the rotation matrix convert the reference and current pose from the inertial frame to the bishop's frame. Having switched to the bishop's frame, we apply the same controller for moving along X and Y direction of the rook to move the bishop along the diagonal direction on the chessboard. From different perspective, the bishop is just moving along X or Y axis of its frame.

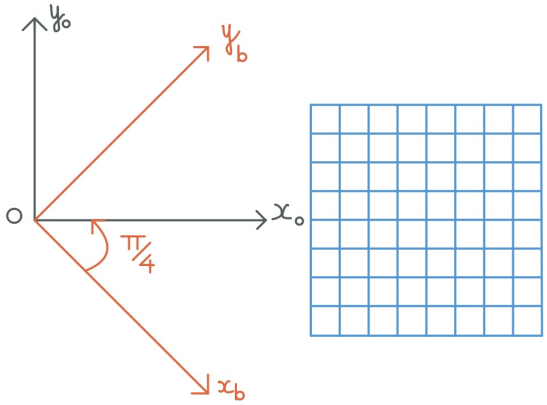


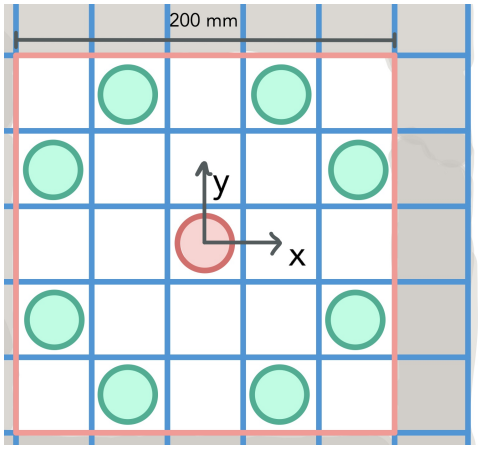
Figure 8: Bishop frame and inertial frame

Figure 9a shows the pseudocode for implement the knight movement, also visually illustrated in Figure 9.

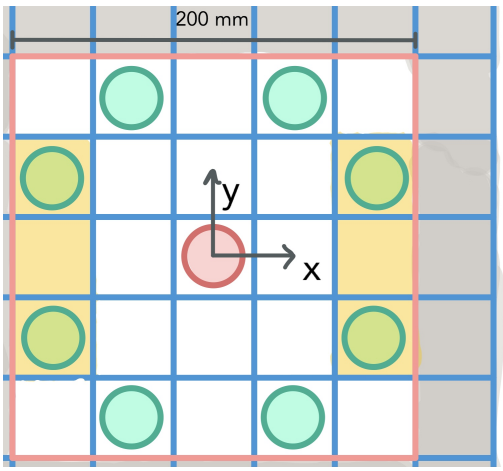
Figure 9b shows a part of the chessboard where the knight is located at the center, indicated by a red node. Its range of movement is defined by the red square and green nodes. From the pseudocode, the idea of moving the knight is to continuously keep track of the x component of the current pose of the end-effector, to make sure that (1) the end-effector cannot go beyond the red square, (2) adjust the gain for y movement and set the appropriate condition of the error in y component for the end-effector to move depends on which range the end-effector is located (yellow shaded region in Figure 9c and 9d).

```
While (true)

    Get feedback from motors, adjust with the offset

    Calculate task space pose using Forward Kinematics

    Calculate the pose error, get sampling time

    ---------- Move the Knight ----------
    Set Kp for x = 0
        if abs(error of x) > 0.1, restrict x movement
        if abs(error of x) > 0.6, allow y movement
            if abs(error of y) > 0.6, restrict y movement
        else if abs(error of x) > 0.2, allow y movement
            if abs(error of y) > 0.1 restrict y movement
    ------------------------------------------
    Calculate control effort, task space velocity command

    Calculate joint space velocity command using inverse Jacobian matrix

    Convert velocity command to the ones fitted with the motors' setup

    Send velocity command to motors

end
```

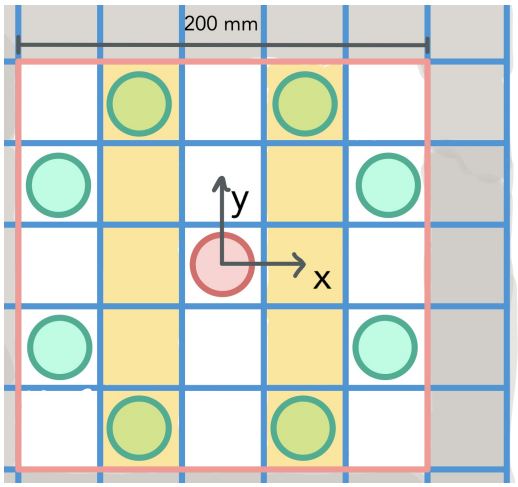
(a) Pseudocode for moving the knight



(b)



(c)



(d)

Figure 9: The illustration of the pseudocode for the knight

3.4 Calibration

Several calibration measures were taken to improve the performance accuracy of our robot for task 1.

While the position control method is superior to velocity control when executing task 1, there are still some notable drawbacks - i.e. the poor robustness resulting from the lack of feedback control. In

addition, limitations encountered during the mechanical assembly process (discussed in the following section) also contributed to the robot being less accurate at executing the chess moves. To mitigate these issues, we fine-tuned the trajectories generated for each specific chess move, to account for the difference in height of the corresponding chess pieces and the glossy & slippery condition of our chessboard. In addition, we calibrated the gains for the PID controller that is present in the motors themselves, and produced a different set of $K_P, K_I \& k_D$ gains for each motor as shown in table 3.

We also had to correct for how the motors were oriented relative to one another when assembled, and the slight joint angle errors present ($Q_{error} = Q_{e1}, Q_{e2}...Q_{e4}$) when the robot was in its home position (i.e. default motor angle configuration when the robot is idle) due to how it was constructed. Therefore, we included the following offsets to the reference angles Q_{ref} generated by our inverse kinematics before passing them on to the motor encoder:

$$\hat{Q}_{ref1} = -Q_{ref1} + Q_{e1}$$
$$\hat{Q}_{ref2} = Q_{ref2} + \pi/2 + Q_{e2}$$
$$\hat{Q}_{ref3} = Q_{ref3} - \pi/4 + Q_{e3}$$
$$\hat{Q}_{ref4} = Q_{ref4} + Q_{e4}$$

The motor for link d_1 was installed in a way such that the direction of rotation was always opposite to the input trajectory, hence we multiplied the input by -1 and added the angle error Q_{e1} to account for this. As link d_2 is aligned with the vertical axis by default, any input trajectory for Q_{ref2} should be offset by an angle of $\pi/2$ radians plus the respective angle error Q_{e1}. For link d_3, we found that the joint angle formed with link d_2 was always shifted by $\pi/4$ radians from the intended value, so we corrected for that and the angle error Q_{e2}.

We also remeasured the lengths of our fabricated robot components, to account for possible deviations of their dimensions from the design specifications. The actual robot link lengths were found to be:

$$\hat{d}_1 = 204\,mm$$
$$\hat{d}_2 = 290\,mm$$
$$\hat{d}_3 = 290\,mm$$
$$\hat{d}_4 = 120\,mm$$

4 Mechanical Design

For the mechanical components of our robot manipulator, we chose a variety of different materials that would suit the needs for each specific part of the robot. For the joint links, we decided on using a cylindrical pine wood shaft as the main connector mainly due to its ease of procurement and desirable properties. Pine wood is both a durable and lightweight material, which satisfies the requirements for a link needing to both be able to support the weight of connected components; and be light enough such that the attached motor can generate sufficient torque to move it. Utilising a wooden link also meant that we could directly secure the motor brackets with a common drill - all without the need for proprietary adapters or the use of any machining equipment as would be the case for an aluminium link. The following figures provide a clearer picture of this:

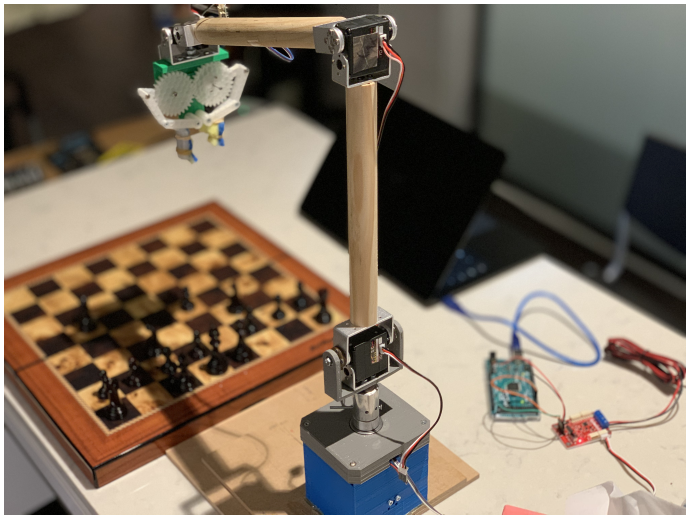
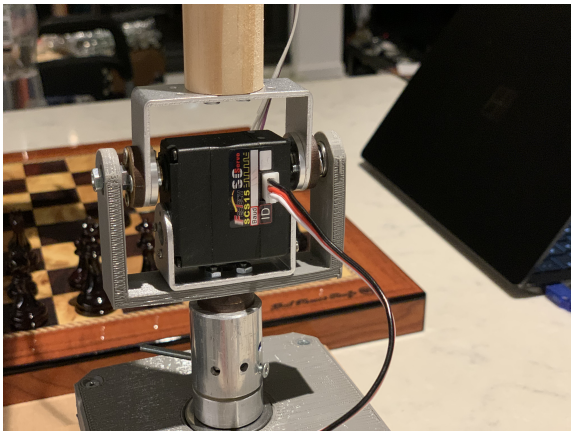


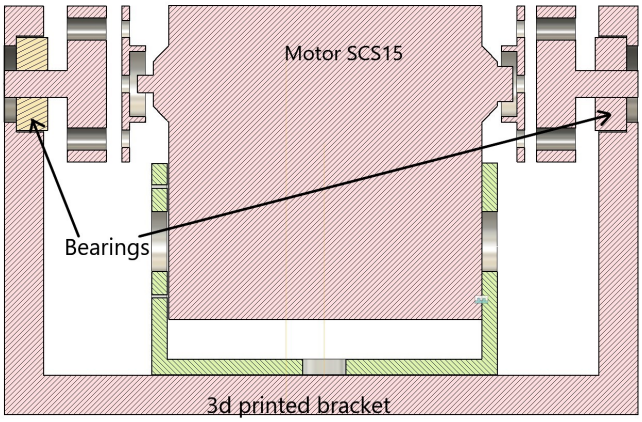
Figure 10: The assembly of the robot arm

To mitigate the transverse load applied on the motor at the second joint (which will bear the most weight), we designed and 3D-printed mounting plates that would allow bearings to be secured on both sides of the motor. The final design consists of 2 main components - an outer bracket connected to the lower motor bracket that would have openings at the sides for inserting the bearings; and 2 mounting plates installed both sides of the upper motor bracket that would allow a screw shaft to be secured with the inner ring of the bearing. This implies that the outer ring of the bearing will be stationary while the motor is rotating, hence offsetting any non-axial load imposed on the motor.

Figure 11a and 11b illustrate the actual assembly and the cross-section of the CAD model at the second joint respectively. Initially, the outer brackets were designed for 3D printed, yet they were not durable and strong enough to take the reaction force during operation. Hence, an improvement in the design was using a laser-cut-3mm-MDF bracket with a screw to form an outer bracket, the center screw takes a role as a shaft sitting on bearings, whereas four screws were tightened from the opposite side so attach the bracket to the motor brackets and stabilise the "shaft-screw". This has minimised the offset of the chessbot, allowing smoother movement and less oscillation.



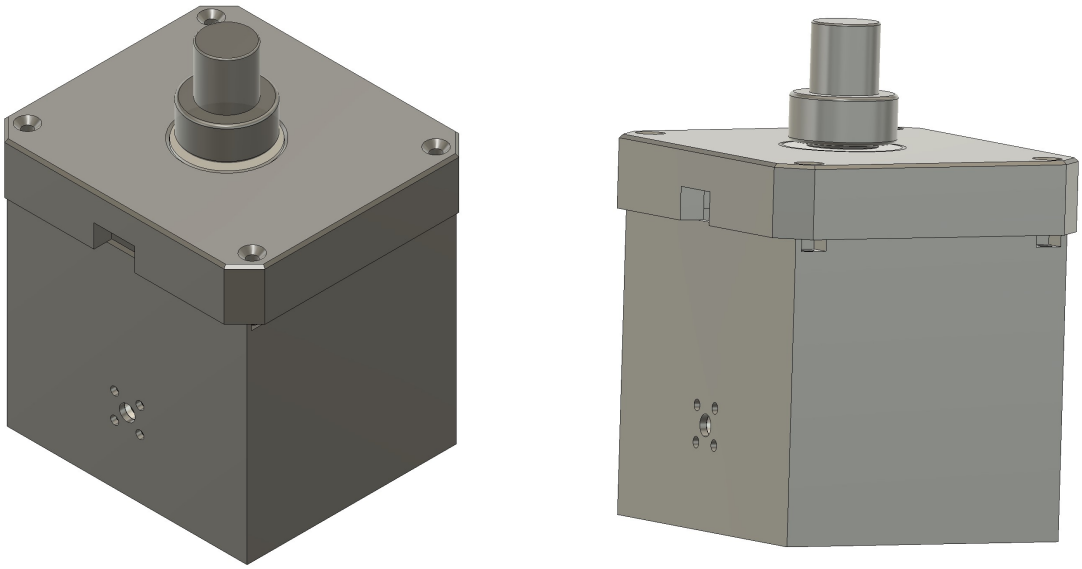
(a) The assembly of the second joint



(b) Cross-section of the second joint

For the joint 1 of the chessbot, an aluminum shaft was machined to be attached with the shaft of the motor rotating about z_0 axis (Figure 13). Due to the structural weight, shaft d_1 will take both bending moment and axial load on it. To ensure smooth and accurate rotation, a thrust bearing was inserted at the lid of the base, whose inner diameter was fitted with the aluminum shaft, to take on the axial load and radial load, protecting the motor shaft from being damaged. The lid also take the role as a housing for the bearing, linked with the motor's housing via four screws at the corner. These screws will ensure the lid stay in place, allowing the alignment of motor shaft and bearing, shown in Figure 12.

There are four slots for hex nuts (Figure 12b), allowing to tighten the screws inserted from the top of the lid. For avoiding stress concentration on the 3D printed lid and having more pre-tension force on the screws, the edges were chamfer (Figure 12a). There are small slot and holes for connecting the wires with the motor, as well as assembling it to the housing. Then, the base is sticked onto a 240 x 240 mm, 6 mm thick MDF sheet to ensure the whole structure will not fall down due to its weight when the arm is spanning towards the edge of its workspace.



(a) **(b)**

Figure 12: Isometric view of the base

Figure 13 show the cross-section of the base, as well as the assembly of the aluminum shaft with

thrust bearing.

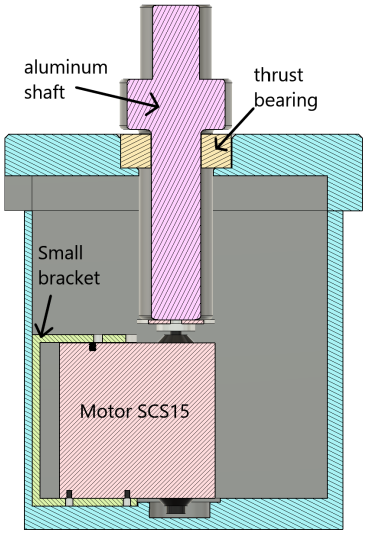
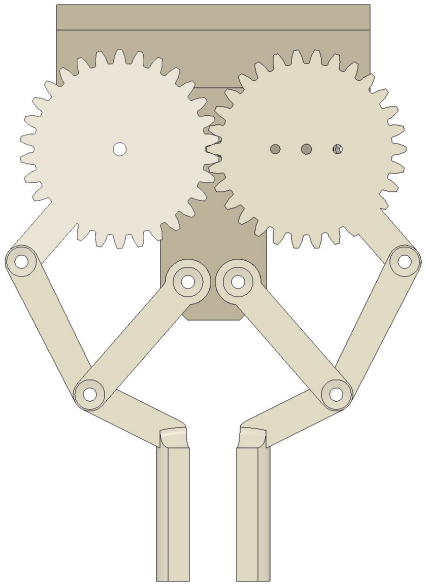


Figure 13: Cross-section of the base of the arm

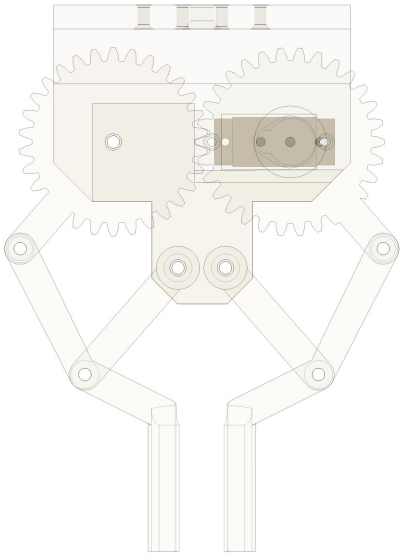
Gripping Mechanism

Figure 14 presents the 3D CAD model of the gripper used for picking up the chess piece. This design features parallel linkage, using 2 3D printed spur gears to transmit torque produced from one gear to the other. Figure 14c shows the back of the gripper, having a close look at how the SCS009 motor was mounted to the gripper. There is a small wheel bracket connect with the motor's shaft and that bracket is connected at the back of the spur gear using M2 screw.

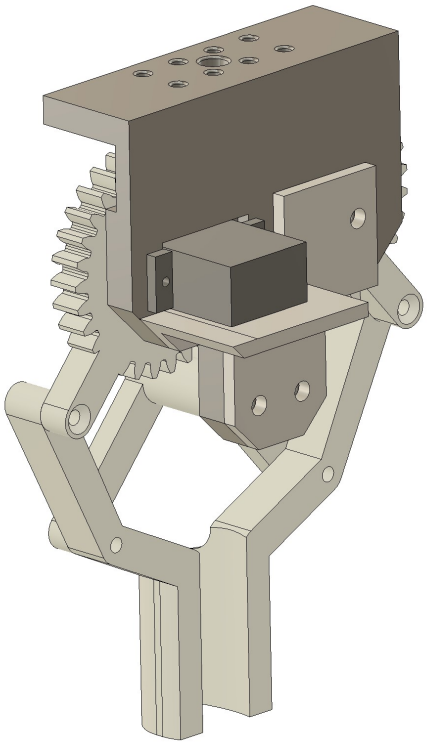
The idea was to have a long gripper so that the chessbot does not have to lower down too much on the Z_0 direction as then it will require more torque to moving back to its home position. Moreover, with this design, we can make sure that the gripper will not touch on other adjacent chess pieces to the target one. However, this design has one drawback, that the chessbot needs to be precisely controlled to where the chess piece is located, which our chessbot was lack of.



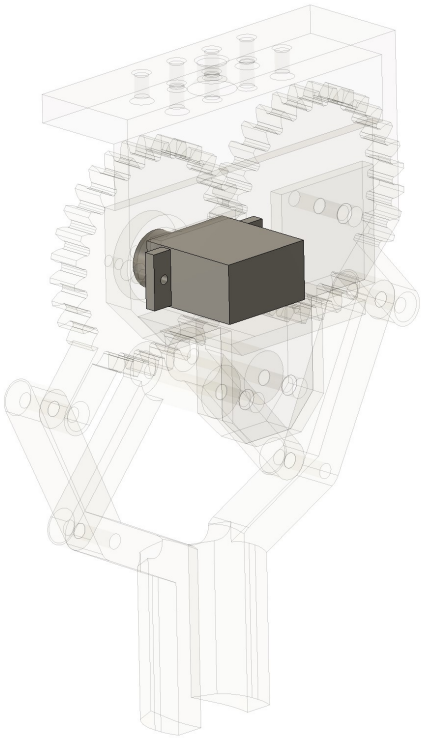
(a) Front View



(b) Front View



(c) Isometric View



(d) Isometric View

Figure 14: Gripper 3D model

5 Discussion

5.1 Results

Task 1

The following plots illustrate the joint space displacement of our robot relative to the reference trajectory angles for several moves in task 1.

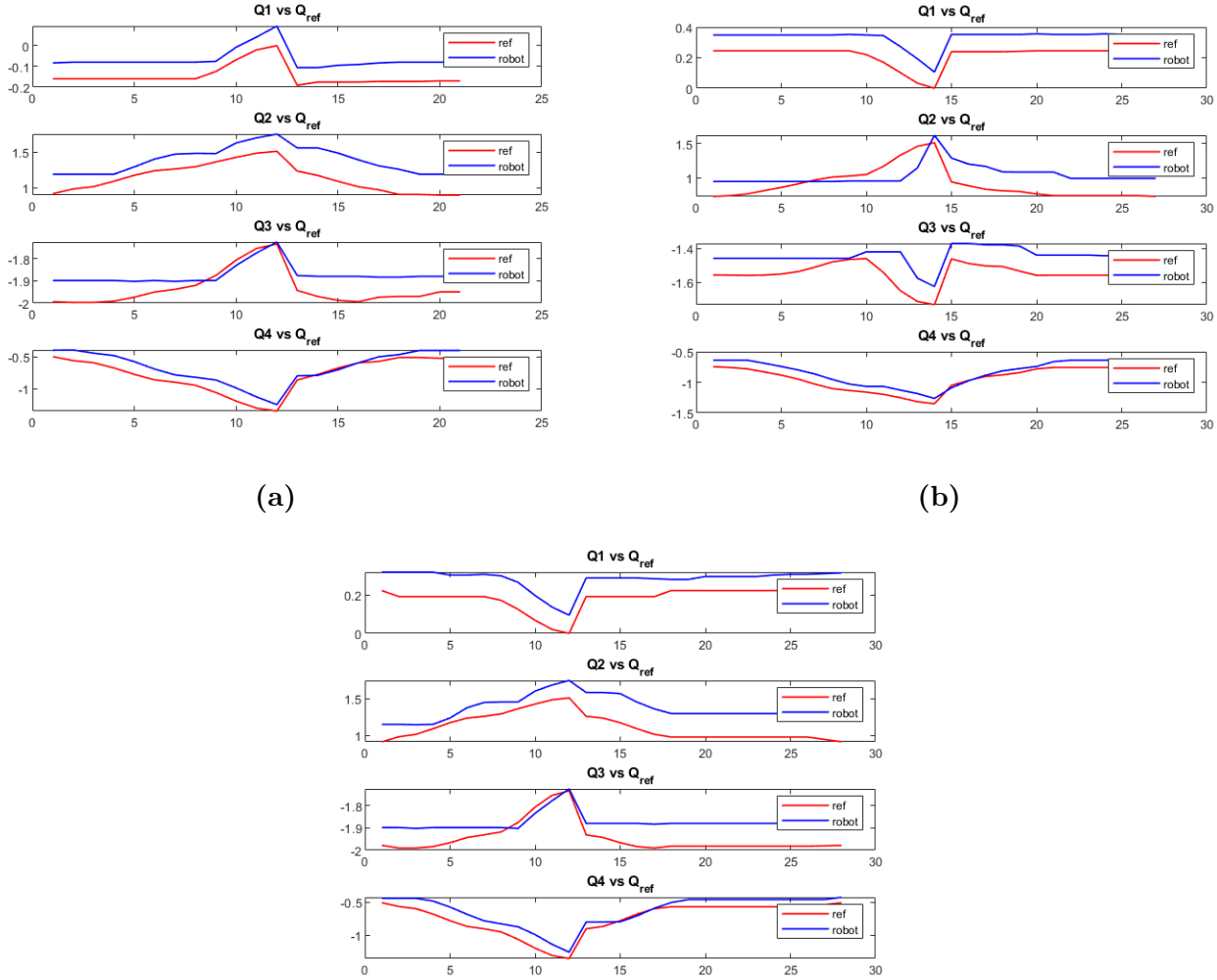
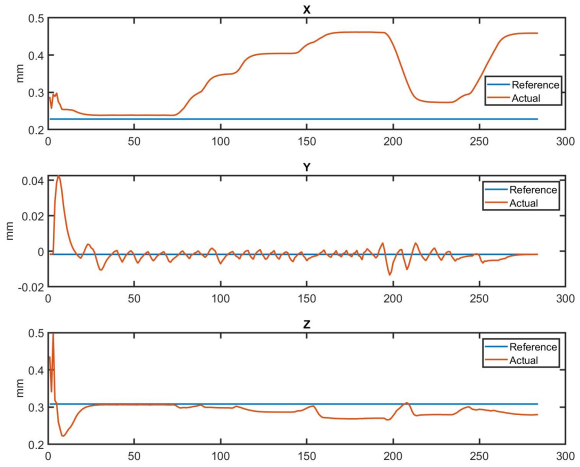


Figure 15: Plots of joint space displacement of robot vs. reference trajectory

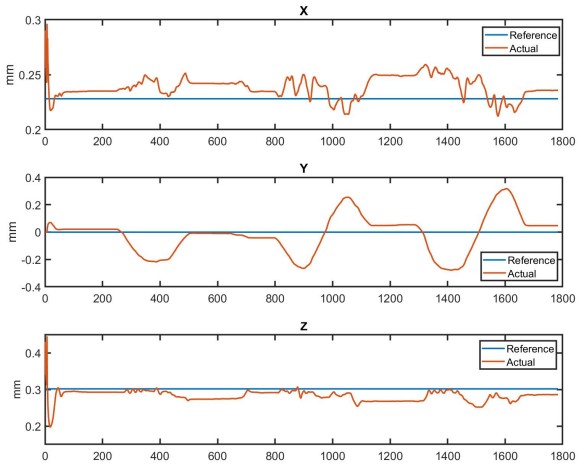
From figure 15 above, it is evident that the robot is able to successfully track the provided reference trajectory for all 4 joint angles Q_1, Q_2, Q_3, Q_4 with no oscillations present. However, there is still a persistent steady-state error at times when the trajectory angles are constant - especially for joints not directly attached to the end-effector (Q_1, Q_2, Q_3). This can be attributed to a lack of feedback integral control, and imperfections in the mechanical construct of the robot which could lead to some unavoidable deviation from the reference angle. Nevertheless, the robot still manages to perform well for various different moves, and the joint angle errors are within an acceptable bound.

Task 2

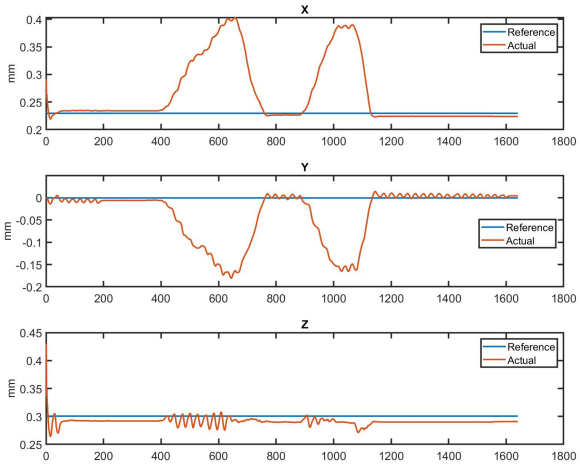
Figure 16a, 16b and 16c show the joint Q_4 pose in the task space using Forward Kinematics, in comparison with the reference pose for Rook's movement in X, Y and Bishop's movement along X_b respectively.



(a) Rook's movement: along X



(b) Rook's movement: along Y



(c) Bishop's movement: diagonal, along X_b

Figure 16: Plots of task space pose vs. reference pose

For Figure 16a, the proportional controller gain was $(K_{px}, K_{py}, K_{pz}) = (0, 1.8, 12)$. From observation in demonstration, the chessbot was not entirely stablised in that control mode. It seems to oscillate about z_0 axis. Moreover, as user moved the end-effector further along x axis, the chessbot could not maintain the height for the end-effector. This can be caused due to the backlash in motor or due to excessive weight of the structure. The result from graph has shown similar point. The sawtooth response on y axis reasonably matched with the oscillation behaviour of the chessbot. Whereas, for z axis, as we move further along x axis, the z components of the pose also reduced and diverged away from the reference, which peak error occurred when the end-effector was furthest in x direction to the reference pose. However, as the encoder can capture this divergence on z component, it is more likely that there is an excessive amount of torque due to mechanical structure that has gone beyond motors' capabilities. Similar issues have been observed from other movements.

5.2 Limitation and Further Improvement

Mechanical design

There are several factors that has affected the performance of the chessbot. First of all, there is free-play in the assembly of the chessbot, especially at the connection of d_1 shaft and motor associated with joint Q_2 in Figure 11a. The motor connected with a coupling by 4 M3 screws, d_1 shaft and coupling are connected by 1 M3 screw and that was not enough to prevent free play. Moreover, due to difference in inner diameter of the coupling and d_1 shaft, misalignment can happen and that can cause the offset of the origin of frame 2 from z_0 axis, referring to the schematic of the chessbot (Figure 1).

Another factor comes from mechanical design was that, the wood dowel used for connecting joints, was cut by hand, meanings that the final shape of it may not be a symmetrical cylinder (Figure 17). Without using any measurement tools or equipment, we can even see that the top radial plane is not perpendicular to the symmetrical axis of the dowel.

Figure 17: Poor cut wood dowel

, poor fabrication has led to a big offset of the end-effector from the $x_1 - y_1$ plane, resulted in errors in poses at different location on the chessboard. The Forward and Inverse Kinematics were calculated assuming that X_{i-1} has to intersect Z_{i-1} and Z_i in perpendicular manner, which was not met with poor fabrication. Due to time constraint and complicated symbolic expression of the end-effector's pose in task space, the solution was to cut another dowel with less inaccuracy and test on the chessbot to move to each individual square. From there, we adjust the pose of that square such that the chessbot can fully pick and drop the piece closest to the center of the square.

Without time constraint, our second prototype will be using a different material for the link with changes in the design to reduce weight and switch to another fabrication method. Pine wood is light, yet it is soft, reducing its size may not be the optimal solution in this case. New design will also aim for reduction in number of screws and bolts using to connect 2 joints to reduce weight. Hence, reducing weight can solve two issues that contributed to the error in the end-effector's pose, (1) the offset due to poor fabrication, (2) excessive torque due to weight.

Gripping mechanism

One issue with the gripper was that, the end-effector needs to be precisely controlled to the location of the chess piece, which our controller cannot satisfy with the current deisgn. Moreover, the gripper tip was made of plastic PLA, and the chess piece was made of plastic, the gripping force is lack of between the two to make sure that the gripper can capture the piece even it is a bit offset from the exact position.

We have tested multiple cases on this issue and from observation, we found that rather than tuning the controller, a simpler solution was that we can mount a small piece of sponge on each tip of the gripper to improve its gripping force on the chess piece.

Hardware and Software Limitation

Having testing running the loop in MATLAB to only obtain the feedback angle from the motors from their encoders, we observed that the frequency for MATLAB to finish the loop was 60Hz. In the case where we have to substitute values into the symbolic expression of the Forward Kinematics and Jacobian matrix, this frequency has a significant downturn to 1.7Hz, i.e 0.6s in sampling time. There were 3 different solutions for resolving this issue:

1. Derive Equations of motions of the system, linearize and workout the bandwidth for the discreted controller.
2. Implement the Integral and Derivative action for the controller, aiming for a critical damped response.
3. Reduce the complexity in the calculation.

Given limitation in time for testing different potential solutions and hardware selection, we have decided to go with solution (3) by reducing the number of degrees of freedom, which has been mentioned in Section 3.2 that the frequency was increased to 40Hz. The actual implementation has shown that the chessbot was able to stay in place with minor oscillation about Q_1. Having implemented Integral and Derivative action can minimise oscillation, improving its performance.

Another issue was observed while working with the chessbot was that it seems the Serial communication cannot respond within the Timeout period such that the control loop was freezed for an amount of time, at the start of which the chessbot started to behave abnormally before getting back to its reference pose or completely going unstable. Therefore, we was not able to generate the final result for the bishop to move along its Y_b axis.

A proposed solution for optimal design of the chessbot was to derive its equations of motion (EoM) at the beginning, along with that is the detailed 3D CAD model of the chessbot to ensure the reliability of the EoM before manufacturing it. Torque control should be the inner loop control before sending velocity command to motors, outer loop could be the task-space velocity controller. Having got the model of the robot, we can workout and simulate different version of the discretized controller with the appropriate bandwidth for the inner loop and outer loop. This process will save so much time for getting an optimal solution rather than testing different value of the controller on an fully assembled chessbot, which poses high risk of mechanical design failure or injuries to the tester. Assuming that we are design a big scale robot arm for industrial application, this concept would be approached to identify the safety range, at which we can start trial-and-error test to get optimal solution.

6 Conclusion

This report covered the implementation of motion planning and control methods to enable our chessbot to successfully execute tasks 1 & 2. For task 1, the position control method was utilised to enable to robot end-effector to navigate to the required target square on the chessboard, and pick up or place the respective chess piece. In contrast, task-space velocity control was used for task 2, where the end-effector of the robot should be restricted to only move along the allowed trajectories for a specific chess piece. Calibrations were made to improve the accuracy of the robot for task 1, and to correct for any offsets as a result of the robot's construction.

The joint space displacements of the robot were then recorded for several moves in task 1, then plotted and compared with their respective reference angle trajectories. The results were satisfactory, albeit with slight deviations from the reference. For task 2, we collected the pose of the end-effector in task space when moved along the allowed trajectory for both the rook and bishop pieces. In this regard, the robot was successfully able to correct for deviations from the restricted path, but performance was hampered by issues such as backlash and insufficient torque from the motors. In addition, we covered the mechanical design of the robot in detail, and discussed possible limitations and areas for further improvement.

References