Assignment3 Report AG03

Jarod Tang, Zac Lee, Xinyu Shen

May 2024

Contents

1 Introduction **2**
1.1 Overview 2

2 Mechanical Design **3**
2.1 Linkage Design 3
2.2 Joint Design 4
2.2.1 Joint 1 - Base 4
2.2.2 Joint 2 and 3 4
2.3 End Effector Design 6
2.4 Integration 9

3 Software Design **10**
3.1 Control Scheme 10
3.2 Controller 10
3.3 Task Specific Motion Planning 12
3.3.1 Task 1 12
3.3.2 Task 2 15
3.4 Calibration 17

4 Discussion **18**
4.1 Demonstration Performance and Evaluation 18
4.2 Potential Design Improvements 19

5 Bibliography **19**

6 Appendix **20**

1 Introduction

This report aims to demonstrate the logical arguments pertaining to the design justifications of the constructed robotic manipulator. The report will first present the reasoning for both the mechanical and software design. Afterwards, the performance of the robotic manipulator will be explored and potential improvements will be discussed.

1.1 Overview

The scope of this project is to build, design and program a robotic manipulator to move chess pieces on a chessboard. The robot is only required to realise legal chess moves. The configuration of the 4 DOF (degrees of freedom) robotic manipulator designed can be abstracted into a simple diagram shown in Figure 14a attached in Section 6. Note that l_{22} is not treated as a linkage.

Using the illustration depicted in Table 1, the Denavit-Hartenberg (DH) table and its subsequent parameters can be obtained. The table is essential in solving both the inverse and forward kinematics of the robot. Note that this configuration is slightly different to the one presented in the previous assignments, since more clearance between link 2 and 3 was required to achieve the correct reachable workspace - and thus an extra offset l_{22} is added. The table can be found in Section 6.

The robot's base is elevated 85 mm above the chessboard. Its position in the X-Y plane, shown as a red dot is depicted in Figure 14b. Note that Figure 14b is not to scale. The robotic manipulator has four linkages with the following lengths:

$$L_1 = 85\,\text{mm} \quad L_2 = 250\,\text{mm} \quad L_3 = 274\,\text{mm} \quad L_4 = 100\,\text{mm}$$

The dimensions of the chessboard are depicted in Figure 14b. Each chess piece has a radius of 1cm. The tallest piece is Queen at 56.3mm and shortest is Pawn at 31.85mm.

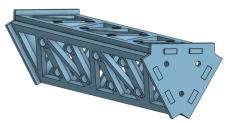
2 Mechanical Design

A conscience design decision was to have the parts discussed in Sections 2.1 and 2.2 made out of solely laser-cut material. Given that the software being implemented at the time was unstable and could potentially cause unintended collisions, it was important to design the robot such that it could be rebuilt and improved upon quickly in the event any particular part was destroyed or damaged. This approach minimised testing downtime and facilitated iterative development.

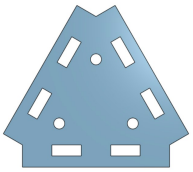
2.1 Linkage Design

The linkage design takes inspiration from construction cranes.

The robot linkage design, illustrated in 1a, is an assembly of three main components: the face plate shown in 1b, the mid pieces depicted in 1c, and the side structures represented in 1d. The face plate serves as the central connection point, providing structural integrity and housing multiple attachment points for other components. The mid pieces contribute to the linkage's overall length and flexibility, enabling a range of movements while maintaining stability. The side structures are designed to offer additional support and rigidity, ensuring the linkage operates smoothly under various loads. Together, these components create a robust and efficient mechanism capable of performing precise and reliable motions essential for the robot's functionality.



(a) Linkage Assembly



(b) Face Plate



(c) Mid Pieces



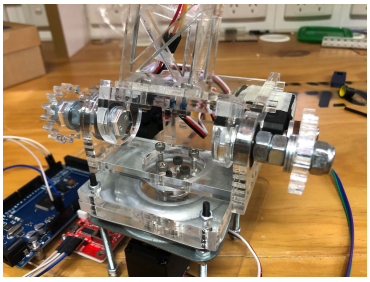
(d) Side Structures

Figure 1: Robot Linkage Components

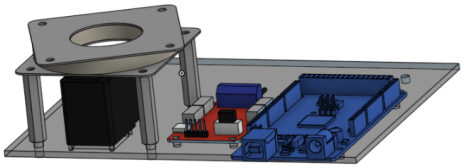
2.2 Joint Design

2.2.1 Joint 1 - Base

The design utilises a "Lazy Susan" bearing to enable rotation about the z_0 axis (refer to Figure 14a). The remainder of the robot is bolted down to an arcylic plate bolted to the lazy suzan. To transmit rotation, a STS3215 servo is directly attach to the arcylic plate. This is depicted in Figure 2a below.



(a) Final Base Assembly



(b) Base Design CAD Assembly

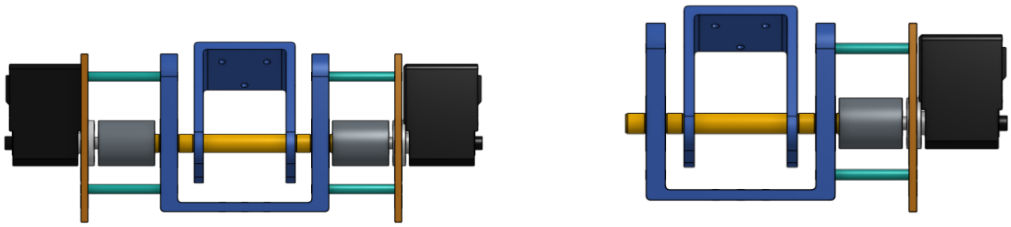
Figure 2: Final Base Design

2.2.2 Joint 2 and 3

Joints 2 and 3 utilised very similar designs. The principle of these joints is that they are connected by couplers that are driven by a motor. The motor used at both of these joints is an STS3215 servo. A commonality between both joints 2 and 3 is that the inner and outer couplers are connected by bearings and a shaft so that there is no translational DOF, but only 1 rotation DOF. Moreover, the inner coupler was placed inside the outer coupler, with the mount being mounted to the outer coupler.

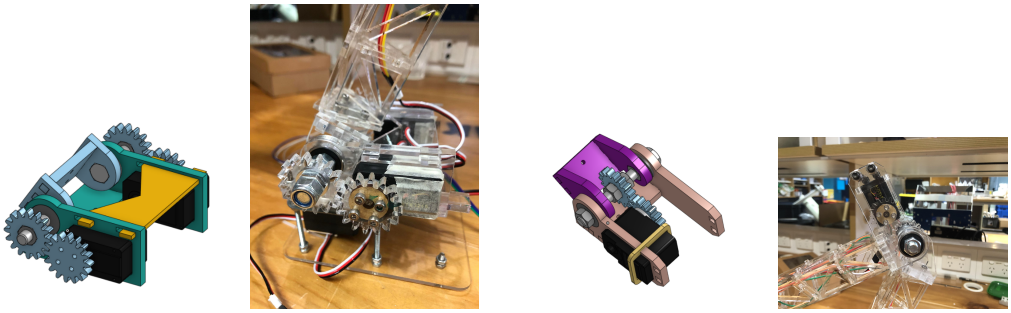
In the initial iteration of the design is displayed in Figure 3, the motor was directly attached to the shaft via a shaft coupler. The motor was then fastened to the outer coupler using a combination of bolts and nuts. Asides from being bulky and inelegant, it was also revealed that a severe flaw was present during testing, that being the connection between the shaft and inner coupler would fail under high load.

To make the connection between the inner coupler and shaft more secure, M8 bolts were used as the shaft (see Figure 4). They are readily available, fit snugly in the bearings, incredibly strong, and the hexagonal bolt head also



(a) Joint 2 Version 1 (b) Joint 3 Version 1

Figure 3: Old Joint Designs



(a) Joint 2 CAD (b) Joint 2 Final (c) Joint 3 CAD (d) Joint 3 Final

Joint 2 Joint 3

Figure 4: Joints 2 and 3

makes it possible to design the inner shaft such that the bolt head could be press-fitted into the design. Using a combination of spacers and jam nuts, the M8 shaft can secured and properly transmit rotation without slipping. In the final design, to reduce the amount of change to the code, a 1:1 gear ratio is used. The motors are press fitted into a snug cutout in the outer coupler. A brace would then fit over the motor to ensure that it would not slip out.

Joint 2 and 3 comparison As seen in figure 4, Joint 2 has motor mounts on both sides to house 2 motors, for added torque, whereas joint 3 only has 1 motor mount opening on one side. The mounting location of the bolt head on the inner coupler plate is shifted, to increase the range of motion of joint

3, as seen in figure 5.



(a) Joint 2 inner piece

(b) Joint 3 inner piece

Figure 5: Joints 2 and 3 inner coupler comparison

2.3 End Effector Design

With regards to the task discussed in Section 1.1, one major challenge presented is picking and placing the knight. Its complex and non-uniform geometry makes it difficult for an end effector to find a secure grip. Moreover, variations in the grip force may be required for different pieces, complicating the control system and increasing the likelihood of dropping or misplacing pieces.

In consideration of these problems, two solutions were conceptualised: a two fingered grippper and a latching end effector, inspired by the Canadaarm2 Latching End Effector (LEE). Whilst a two fingered gripper offers versatility and precision for a wide range of applications, a latching end effector offers greater consistency, reliability and efficiency for a chess playing robot. The final assembly is depicted below in Figure 6. The Canadaarm2 LEE is also attached below for reference.

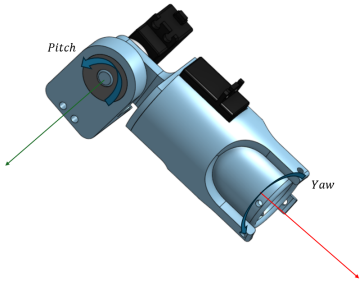
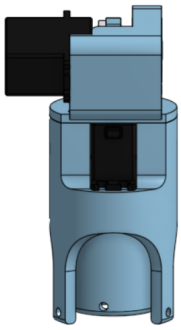
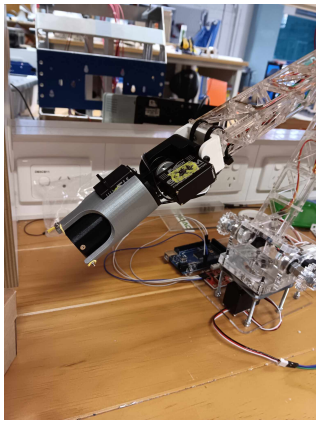


Figure 7: End Effector Rotational Motion



(a) End Effector Front View



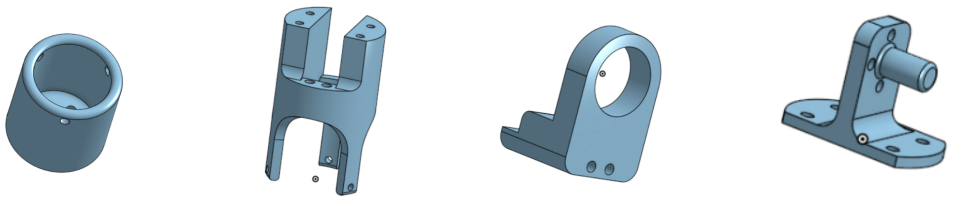
(b) Final End Effector Side Profile



(c) Canadaarm2 LEE (Canadian Space Agency, 2018)

Figure 6: Final End Effector Design

The latching end effector is composed of seven components: an inner and outer sleeve, a joint and sleeve mount, an elastic band and two STS3045M servo motors. These two motors provide the end-effector pitch and inner sleeve rotation. This is depicted below in Figure 7. Each of these components excluding the elastic band and motors is depicted below in Figure 8. All of the depicted components were 3D printed using polylactide (PLA).



(a) Inner Sleeve (b) Outer Sleeve (c) Joint Mount (d) Connector

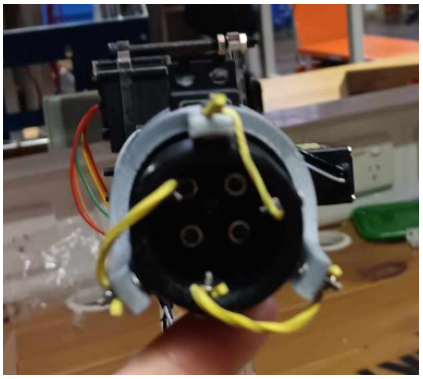
Figure 8: 3D Printed End Effector Components

In the design and manufacturing of the end effector, it was crucial to create an assembly that was both simple and easily modifiable. This approach was essential because the required software implementation could potentially be unstable, necessitating multiple redesigns and remanufacturing of the end effector. To achieve this, a minimal number of components were used in the construction. This strategy ensured fewer points of failure and streamlined the assembly process, making it quicker and simpler. Additionally, using fewer components facilitated easier maintenance and reduced the overall cost, allowing for rapid prototyping and iterative testing to refine the design based on software and operational feedback. This emphasis on simplicity and modularity ensured that the end effector could be efficiently adapted to evolving requirements.

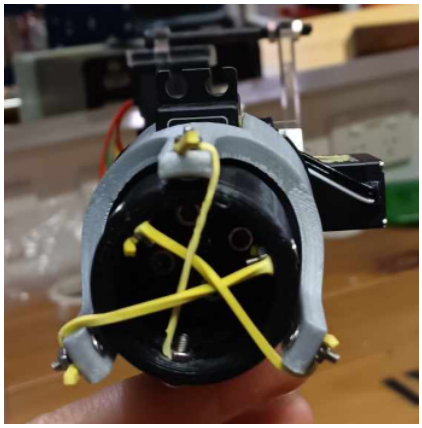
The inner and outer sleeves were designed with a diameter matching the chessboard square width. This reduces the precision required, allowing the end effector to grasp pieces even if slightly offset from the intended position.

The edge of both the inner and outer sleeves has three holes. With a nut and bolt, pieces of an elastic band are secured in place, with each end of the elastic band attached to both the outer and inner sleeve. This is observed in Figure 9. The grasping mechanism is as follows and is depicted in Figure 9.

1. The end effector is aligned to the designated position.
2. The inner sleeve is rotated clockwise 60 degrees.



(a) Snare Open



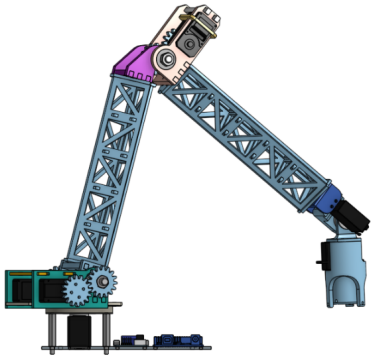
(b) Snare Closed

Figure 9: End Effector Snare Action

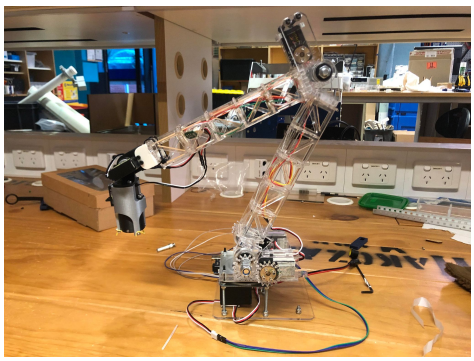
Moreover, irrespective of the type of piece being grasped (pawn, queen, knight, etc) or its orientation, the grasping mechanism accommodates these variations. The snare action will capture the payload in the centre of inner sleeve.

2.4 Integration

The final design and assembly is depicted below by Figures 10a and 10b.



(a) Final Robot CAD



(b) Final Robot Build

Figure 10: Final Robot Assembly and Build

3 Software Design

The robot's control software is written in a *MATLAB live script* and consists of four sections: initialisation and connection to motors, variable assignment and object creation, control code proper, and helper functions. Most of the calculations and variable manipulation is handled by a custom *MATLAB class* named "calculate", to produce neater code through abstraction.

3.1 Control Scheme

As recommended by the teaching team as well as to produce (theoretically) smoother robot operation and avoid the use of inverse kinematics: Task Space Velocity (TSV) control was chosen to be implemented as the primary control scheme. Originally, a full 4DOF TSV control was implemented to control the pose of the arm at the tip of the *end-effector* - however, due to performance issues outlined in Section 3.2 the position and orientation were decoupled into a 3DOF TSV control on the *wrist* point (origin of frame 4, see Figure 14a) and a simple PID controller on joint 4 (and thus the overall pitch orientation of the arm). This can be achieved by realising that the arm's gross position is controlled by joints 1 to 3, gross orientation by joint 4, and that in the *end-effector down* orientation the wrist position is always directly above the tip and thus can be used as a proxy for the end-effector position. Note that the end-effector actuation motor was operated in "joint-space position control" meaning that the *open* and *close* positions were set by sending joint position commands.

3.2 Controller

As discussed in Section 3.1, the coupled position and orientation control as in a 4DOF TSV controller produced significant performance issues. Again, a simple PID controller of the form shown in Equation 1 was used to produce a task space command velocity, $\dot{X}_c$, as the sum of a reference velocity, $\dot{X}_{ref}$, and a linear combination of various forms of the task space position error, e.

$$\dot{X}_c = \dot{X}_{ref} + K_P e + K_I e_{accum} + K_D \frac{e_{curr} - e_{prev}}{\Delta t} \tag{1}$$

Where e_{accum} is the accumulative error accrued, e_{curr} is the error at the current timestep, and e_{prev} is the error at the previous timestep

By the nature of using the full 4DOF TSV controller, the joint velocity commands, $\dot{Q}_c$, are calculated by multiplying $\dot{X}_c$ by the 6x4 inverse Jacobian matrix consisting of *both* the translational and rotational velocity components. This means that the position of the end-effector in task space is affected by both the gross translation of the arm (controlled chiefly by motors 1 to 3) *and* rotation (controlled chiefly by motor 4) of the arm (and similarly for the orientation of the end-effector). The main problem with this architecture is that having links 2 and 3 relatively long compared with link 4 (end-effector length): a large translational change can be achieved with little motor movement, as opposed to a larger required motor movement to affect a large *rotational* change at joint 4 plus compensation for the change in orientation induced by the change in position (as well as joint 4 consisting of the smaller servo which had an empirically slower response time). This results in the reference position being reached much earlier than the reference orientation, and since they are coupled, the orientation correction causes a change in the end-effector position, and thus creates a translational error which must be compensated for - which creates a back and forth motion of the arm as it tries to correct for both components simultaneously. Furthermore, by tuning the PID gains for the rotational component higher to achieve faster response results in a decrease in performance in the translational component (which would then be compensated for with different gains, consequently negating the change made to the rotational components, and so the cycle repeats). Thus, tuning the controller using the full 4DOF TSV control scheme was deemed infeasible.

Following this, a 3DOF TSV controller was implemented to control the *wrist* position instead of the *end-effector* position - which was determined to have no negative consequences due to the constraint on having the end effector always point towards the chessboard, meaning the the (x, y) location of the wrist is identical to that of the end-effector bar an offset in z of L_4 (length of end-effector). This proved highly advantageous as the original translational component already had good performance when ignoring the orientation, which remained after decoupling. Then, with the orientation no longer dependent on the position (and vice versa), a more aggressive orientation controller could be implemented to improve the response time without negatively affecting the controller tuned for the position.

3.3 Task Specific Motion Planning

3.3.1 Task 1

The goal of Task 1 was to "play" a simulated predefined chess game using a pick-and-place style robotic arm, the mechanical and high level software design of which was to be completed entirely from scratch. As discussed in the previous sections, a task space velocity control scheme was chosen to achieve the desired trajectories in a theoretically smooth manner. The block diagram of the control code is shown in Figure 11, with the pseudocode outlined below:

1. Get the current move's via points,
2. Generate the trajectory which passes through the via points,
3. Discretise the trajectory into timesteps and iterate through the list at a regular interval,
4. With the trajectory at the current timestep as the reference position, calculate the difference between the measured and actual task space position as the *error*,
5. Using this error and the reference velocity at the current timestep as the feedforward velocity, calculate the task space command velocity through the PID control law described in Equation 1,
6. Convert the task space command velocity to a joint space command velocity through the matrix multiplication with the inverse Jacobian calculated for the the current robot pose,
7. Send the joint velocity commands to the motors,
8. Repeat steps 4 - 7 using subsequent timesteps until the final trajectory position is achieved to some predefined degree of accuracy,
9. Operate the end-effector as necessary to pick-up/place a chess piece,
10. Return to step 1 and repeat for all subsequent moves until the chess game concludes.

Note that concurrent to this control loop, a simple PID controller is run to track the orientation reference throughout the positional trajectory. This is shown in block diagram form in Figure 12.

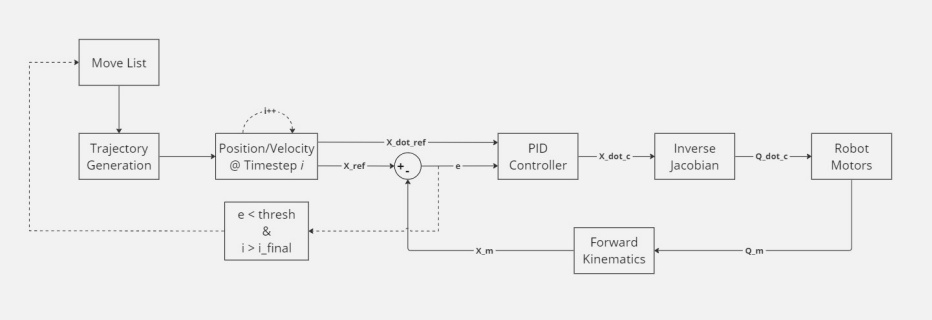


Figure 11: Block Diagram for Task Space Velocity Controller

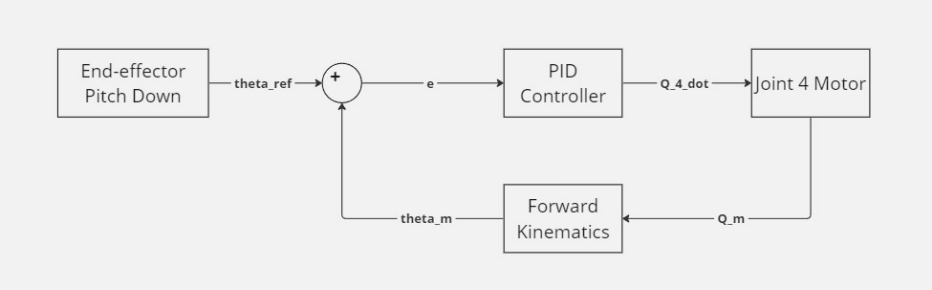


Figure 12: Block Diagram for Orientation Controller

The trajectory planning algorithm implemented is a simple quintic polynomial interpolation between each via point using "zero-velocity" and "zero-acceleration" boundary conditions. This is to ensure the arm comes to a complete stop at each via point thus reducing the risk of knocking pieces over. Each move in the chess game is divided into a sequence of seven sub-moves (defined by via points in the trajectory):

1. *Home* to *pick-up location* in $z_{move\ layer}$,
2. Vertically down at *pick-up location* to a predefined $z_{place\ layer}$ based on the height of the piece,
3. Vertically up again to $z_{move\ layer}$,

4-6. Repeat of the previous three moves but from the *pick-up location* to the *place location*,

7. And finally back to *home*.

Note that $z_{move\ layer}$ and $z_{place\ layer}$ were chosen such that when the end-effector has a piece grasped the base doesn't knock over any other piece, the wires of the LEE intersect at a desirable location on each piece, and the placing height is aligned such that the pieces aren't being dropped or forced into the chessboard. Further, the *home* position is defined as the location where the arm rests to allow the opponent to play their move - noting that the move to *home* in step 7 is omitted when the move is a "capture" since the replacing of the captured piece is defined as one singular move as opposed to two distinct moves requiring the arm to return to the *home* position.

The trajectories described above are generated as the coefficients of 3 standard quintic polynomials defining the reference x, y and z positions as a function of time (used for the feedback error). As such, these can be differentiated in time to obtain the reference velocities (used for the feedforward velocity). These equations are shown in Equations 2 and 3, where $a_i, i \in \mathbb{R} \in [0, 5]$ are the calculated coefficients.

$$X(t) = a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0 \tag{2}$$

$$\dot{X}(t) = 5a_5t^4 + 4a_4t^3 + 3a_3t^2 + 2a_2t + a_1 \tag{3}$$

Since the motors only return current position as feedback (that is using the provided low level control code; the motors do have the functionality to return velocity measurement, but it was not used in this project): the task space position error term in the control law shown in Equation 1 must be calculated from the measured joint positions through the robot's forward kinematics. This is the translational component calculated from the transformation matrix from the inertial frame to the point of interest (in this case the wrist position).

Finally, as seen in Figure 12, joint 4 (which commands the gross orientation of the end-effector) is controlled with a basic PID controller. Unlike with the position controller, the only desired orientation is end-effector pointing towards the chessboard and so no trajectory needs to be generated (i.e. through the use of quaternions) - thus, no feedforward velocity reference is included in the control law (see Equation 4). Instead, the joint 4 velocity is calculated

solely on the error between the reference orientation (end-effector down) and the current measured orientation (which is achieved using the transformation matrix from the inertial frame to the end-effector frame). This error is calculated using the standard "cross-product" method as shown in Equation 5. Note that since the arm can only affect the pitch of the end-effector using joint 4 and the yaw using joint 1, and that only a pitch of $-\frac{\pi}{2}$ is required to define the reference orientation: only the y component (e_{or_y}) is used to calculate the orientation error (assuming that rotation in yaw is calculated based on the (x, y) position of the end-effector, and rotation in roll is identity).

$$\dot{Q}_4 = K_P e + K_I e_{accum} + K_D \frac{e_{curr} - e_{prev}}{\Delta t} \tag{4}$$

$$e_{or} = \frac{1}{2}[(\hat{x}_E \times \hat{x}_{ref}) + (\hat{y}_E \times \hat{y}_{ref}) + (\hat{z}_E \times \hat{z}_{ref})] \tag{5}$$

3.3.2 Task 2

The goal of Task 2 was to "teach" someone how a select number of chess pieces move in the game including the rook, bishop and knight. This was achieved through operating the robotic arm in a "compliant control" mode whereby it allowed an external force to move the end-effector freely in some axes, but resist movement in the others - where the axes correspond to the directions each piece can legally move. Although typically in industry robotic arm compliant control is achieved using force/torque or impedance control - due to the unavailability of any force transducer or any easy way to measure impedance (current) - compliant control in this context was performed using task space velocity control.

The underlying concept is that by setting the gains of a pure proportional controller on the task space error in the axis required to have compliance: any task space error accrued in that axis would not be compensated for in the control law, and thus would allow the end-effector free movement in that axis. Similarly, by having non-zero gains in the non-compliant axes, accrued task space error *would* be compensated for and so resist any external effort to move the end-effector in those axes. However, since at every timestep the compliant controller runs there is necessarily some finite real number calculated as the command velocity: the motors will always receive some ref-

erence velocity which the inner motor controller will try to follow. As such, in order to create *any* error in *any* axis the external force must overcome the torque generated by the motors trying to match the reference velocity at the previous timestep. These motors have particularly high gear ratio and thus high output torque (peak at $19kgcm$), thus making it very difficult to deviate the arm from its resting position - even with the controller gain set to 0. Regardless, the code logic for the compliant controller for each piece is described below.

Rook: The rook can move "up or down vertically on any file... [and] left or right horizontally on any rank" (Chess.com, n.d.). This movement is exactly parallel with the chessboard's x and y axes, meaning that compliance can be achieved by setting either the x, y or both components of the proportional gain matrix to zero depending on where the end-effector currently is. If it's located within the starting square then the rook can move in either the x or the y direction, and so both gain matrix components are set to zero. If the end-effector has moved out of the starting square in the x direction: compliance should be removed from the y axis (setting the gain matrix y component to be non-zero) and retained in the x axis (leaving the gain as zero) - and vice versa for movement in the y axis. A further layer can be included which prevents the end-effector from exceeding the bounds of the chessboard itself by having compliance in only one direction of the x or y axes (check whether the error is positive or negative as required and set the compliance accordingly).

Bishop: The bishop can move "only on diagonals" (Chess.com, n.d.). This movement is very similar to the rook except rotated 45°, and so a similar controller can be used. Moving the bishop is identified to be equivalent to running the rook code with the arm mounted at the corner of the chessboard rather than square to the sides (assuming no offset of the robot from the chessboard as in the case of this one). To achieve this in software, an extra frame can be defined 45° to the inertial frame - and then run all the TSV code relative to this frame rather than the inertial frame. This would entail recalculating all the forward kinematics and Jacobians to be relative to this rotated inertial frame - but then the compliance control code for the rook can be recycled. Again, a further layer of calculating the chessboard bounds can be done to fully constrain the movement.

Knight: The knight can move either "up or down one square vertically and over two squares horizontally OR up or down two squares vertically and over one square horizontally" (Chess.com, n.d.). These movements - like the rook - are aligned parallel to the x and y axes of the chessboard, and so once again the rook control code can be used with just a few extra constraints. Similarly to the chessboard boundary limit logic, the same process can be done for the shape of the knight movement. For instance, if the end-effector has moved out of the starting square in the positive x direction: allow only x compliance for two squares forwards, after which y compliance is added and x compliance is restricted to only the negative direction (the top of the L-shape), then once the end-effector has moved in either the positive or negative y direction out of the corner square allow only y compliance, and so forth. This logic is then repeated for all movement combinations of the knight while still remaining in the boundaries of the chessboard.

3.4 Calibration

To test whether the robot is executing the intended script, initially it is first tested facing outwards from the edge of a table. Ensuring there is ample empty space around the robot is important, so that the robot does not smash into its surrounding and damaging itself.

To test the function of the script, a sample script involving picking up and placing a chess piece is ran (exact location is not very important at this stage), and the movement of the end-effector is observed. The z-offset is set to be fairly high initially, this can be lowered later on to the desired height. Once it is certain that the robot will not smash itself into the table, it can be transferred onto the chessboard for further Z-axis calibration.

The sample script is repeated and the z-offset of the end-effector is modified until a desired height for the end-effector is obtained. The same process is repeated for all other chess pieces. It may seem like defining the z-offset based on the raw height of the chess pieces is more efficient, however, the ideal gripping location for each chess piece for this EE design is at the chess piece's lip. As indicated in figure 13, the z-offset should be set to accommodate the lip height of each piece, rather than its raw height.

Now to calibrate the movement in the X-Y direction, with exact coordinates of the starting and finish chess grid now considered. The sample script is

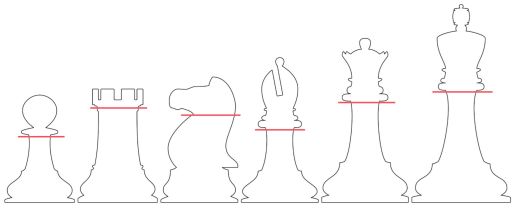


Figure 13: Lip of Chess pieces

now modified to move a pawn from D4 to A1. By observing whether the end-effector is missing the piece, and by how much, as well as where the gripper is releasing the chess piece, x-offset and y-offset can be fine tuned until desirable performance is obtained.

The extent to which the arm must be calibrated is likely due to overall backlash leading to unmeasured joint movement, inexact link geometries generating discrepancies from the DH table, and imperfect controller tuning leading to slight nondeterministic performance between repeated movements (see section 4).

4 Discussion

4.1 Demonstration Performance and Evaluation

Overall, the robotic manipulator performed the required tasks satisfactorily, with some minor issues. With regards to realising the intended trajectory, the robot mostly followed the planned path, with slight deviations occurring occasionally. The deviations from the path were caused by vibrations and wobbles that stemmed from the physical structure. As discussed in Section 2, a major design choice was to make the joints out of laser cut material. This has caused an issue of gears not meshing perfectly, causing noticeable backlash. As already stated, the robot is wobbly when picking up and releasing pieces. The wobbles causes issues in calibration, as well as impact the effectiveness of the PID controller. In turn, this would cause the end effector to sometimes miss the piece it intended to snare.

Additionally, there have been instances where one of the motors at joint 2 would somehow lose connection, so that one of the motors still receives commands to rotate, whereas the other locks in place. This difference in

joint angle has caused joint 2 of the robot to fail several times. The motor at joint 3 also occasionally loses power and collapses downwards.

4.2 Potential Design Improvements

An alternative to laser cutting the parts is to 3D print the components of the robot arm. The motor mounts can be designed to encase the motor so that they are more secure. 3D printing the gears would ensure that they mesh better to reduce backlash. The benefits of 3D printed components is that it can produce a more refined final product, but if a component fails, it might take a few days to get another one ready. Laser cut materials on the other hand can be produced very quickly, but the overall quality of the robot suffers.

If the issue of backlash cannot be resolved, then an alternative linkage design could be explored. Although the current linkages are quite sturdy, relative to joints 2 and 3 (the main perpetuators of vibrations), it is quite skinny. Furthremore, typically skinner structures have a lower natural frequency, making them more susceptible to vibrations, whereas a wider structure, due to a larger cross-sectional area and greater stiffness would not be.

Moreover, to address the issue of the motors being disconnected, the connection pin should be improved to prevent this from occurring.

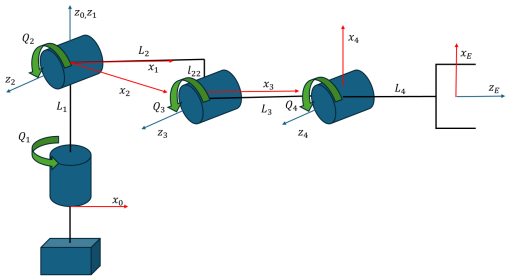
5 Bibliography

1. Canadian Space Agency (2018). Canadarm2 Latching End Effector (LEE) demonstration. [online] YouTube. Available at: `https://www.youtube.com/watch?v=QqgxfFlQ3D0` [Accessed 24 May 2024].
2. Chess.com (n.d.). Bishop - Chess Terms. [online] Chess.com. Available at: `https://www.chess.com/terms/chess-bishop`.
3. Chess.com (n.d.). Knight - Chess Terms. [online] Chess.com. Available at: `https://www.chess.com/terms/chess-knight`.
4. Chess.com (n.d.). Rook - Chess Terms. [online] Chess.com. Available at: `https://www.chess.com/terms/chess-rook`.

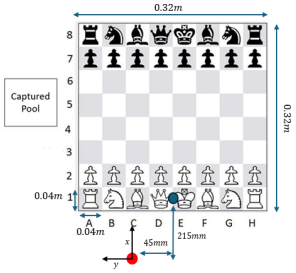
5. Government of Canada (2018). About Canadarm2. [online] www.asc-csa.gc.ca. Available at: `https://www.asc-csa.gc.ca/eng/iss/canadarm2/about.asp`.

6. Stack Exchange (n.d.). What are the operational details of the Canadarm Latching End Effectors? [online] Space Exploration Stack Exchange. Available at: `https://space.stackexchange.com/questions/55753/what-are-the-operational-details-of-the-canadarm-latching-end-effectors` [Accessed 24 May 2024].

6 Appendix



(a) Robot Configuration



(b) Robot Position Relative to Chessboard

Figure 14: Robot Configuration and Position

i	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	0	L_1	Q_1
2	0	$\frac{\pi}{2}$	0	$Q_2 - atan(l_{22}/L_2)$
3	$\sqrt{L_2^2 + l_{22}^2}$	0	0	Q_3
4	L_3	0	0	$Q_4 + \frac{\pi}{2}$
E	0	$\frac{\pi}{2}$	L_4	0

Table 1: DH Table