Assignment 5: Robotics Systems

Group 22
Matthew Irwin - 837741
Jeygopi Panisilvam - 1068969
Yinrui Liang - 824346

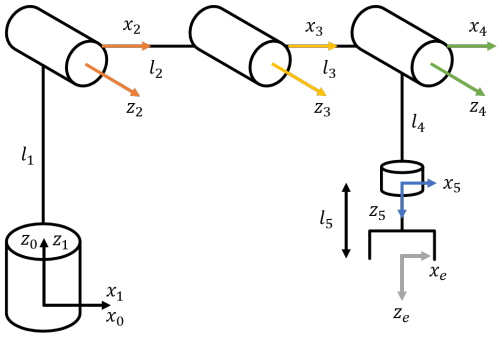
May 2021

Contents

1 Introduction

In the late 1930s the first pick and place robot was developed by Bill Taylor to stack blocks to build physical infrastructure. Since then pick and place robots have found applications in many industries, such as automotive, food processing and electronics. Pick and place robots are expected to maintain their tremendous growth into 2027 as industries seek to automate their industrial processes.

This report builds on the work completed in the previous three assignments. The main objective of this project is to develop a pick and place robot that is capable of moving chess pieces on a chess board using only revolt joints. The chosen chess board has a total length of 381.6mm, with a square length of 27.5mm, see figure 19 in Appendix. The kinematic diagram of the five degree of freedom robot is presented in figure 1a, the Denavit–Hartenberg Table is presented in table 1b and the measured parameters are in table 1.



(a) Kinetic Diagram of Robot

i	a_{i-1}	α_{i-1}	d_i	θ_i
1	l_1	0	0	Q_1
2	0	90°	0	Q_2
3	l_2	0	0	Q_3
4	l_3	0	0	Q_4
5	0	90°	l_4	Q_5
E	0	0	l_5	0

(b) Denavit–Hartenberg Table

Figure 1: Kinematic description of Robot

Parameter	Measured Angle	Parameter	Measured Distance
Q_1	-80°to 80°	l_1	24cm
Q_2	0°to 70°	l_2	28cm
Q_3	-160°to -20°	l_3	25.4cm
Q_4	$-Q_3 - Q_2$	l_4	19.8cm
		l_5	7.5cm

Table 1: Measured Parameters of robot

The first three assignments tackled trajectory generation, forward and inverse kinematics, as well as velocity and static wrench relationship. In this report the final mechanical design will be presented, as well as the development of both a position control controller and velocity control controller. The performance of the final design will also be analysed.

2 Integration

2.1 Mechanical Design

As shown in figure 2, the robot was created using a mix of additive manufacturing techniques, as well as carpentry. The chess board and the robot's base were secured to the same piece of wood to ensure that the relative position between the pieces would remain constant, a key requirement for any open loop trajectory design. The base of the robot is made out of wood and the limbs of the robot are made out of 3D filament to help limit the weight and thus the torque being placed on the motors. A counter weight was also added to further reduce the torque applied to motor two.

Motors are designed to withstand high levels of torque about the motor shaft, but are relatively fragile to other forces or torques. With the introduction of bearings it is possible to limit these extra forces and torques on the motors. As discussed in assignment two, motor 1 and motor 2 experience the highest wrench, as such a motor shaft and bearings were introduced to limit the total wrench applied to the motor itself. Motor 1 is

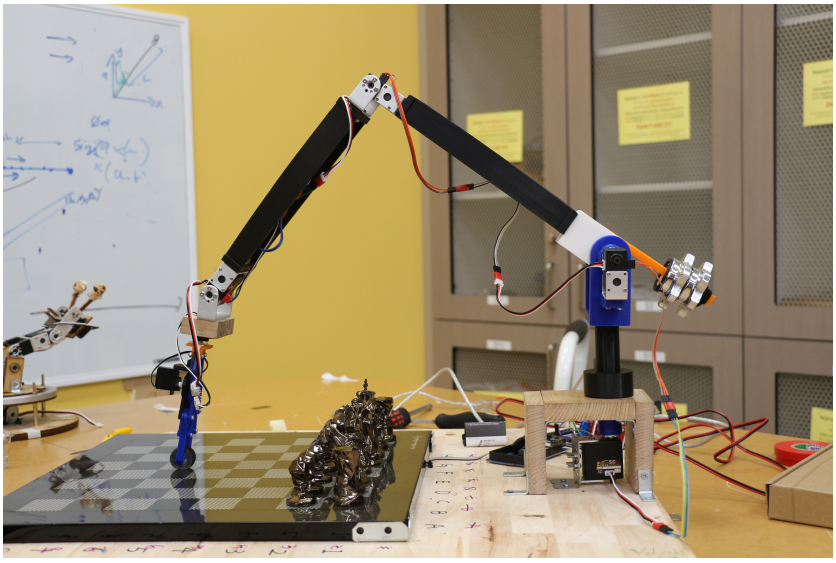
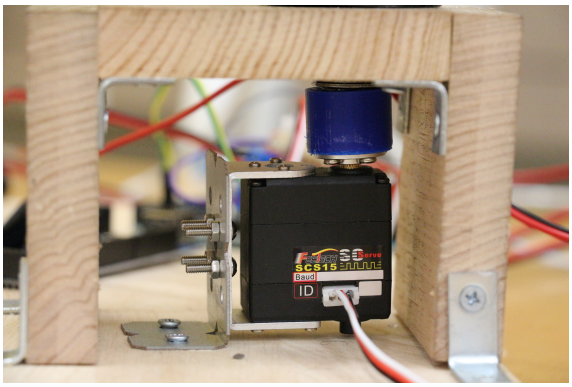
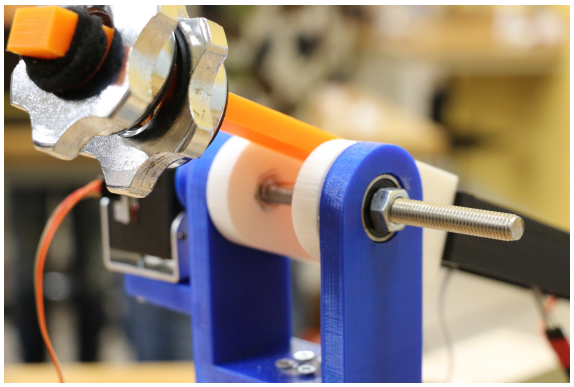


Figure 2: Mechanical Design of Robot

connected to a shaft using a custom 3D printed flange, as seen in figure 3a. This shaft passes through the wooden support, a thrust bearings and connects to length l_1 . This ensures that the wooden structure carries all the weight of the robot and most of the toque. The thrust bearing ensures smooth motion by limiting friction. Motor two is connected to a shaft made out of a 8mm bolt. This shaft passes through two ball bearings and has three nuts attached to it, see figure 3b. Two of these nuts are used to couple limb l_2 to the shaft, whereas the final nut is used to hold the system in place. This setup ensures that the motor is only resisting the applied torque about it's axis of rotation.



(a) Thrust Bearing Q_1



(b) Radial Ball Bearing Q_2

Figure 3: Images of joints with bearings

There are many different designs for robotic manipulators. There are variations in the number of contact points, materials and movement. A two contact point design was chosen, as this would be simple to implement and would also be relatively light. It was also decided that the two "fingers" of the gripper should adjust their relative distance rather than their relative angle, this was to reduce the likelihood of the gripper knocking over the piece if it were not positioned correctly, see figure 4. As shown in figure 5, when the relative position rather than the angle is changed, the force is applied closer to the center of mass, thus reducing the moment placed on the piece, which in turn reduces the likelihood of the piece toppling over. The manipulator was design by adapting a reference design from the internet.

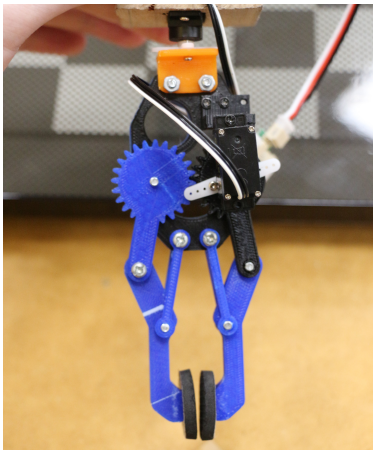


Figure 4: Manipulator design

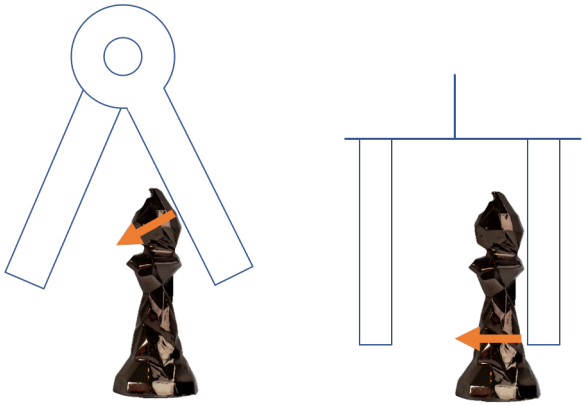


Figure 5: Forces applied to piece based on manipulator design

2.2 Robot and Task integration

The following concepts from the previous assignments were used in the final design of the project to produce the required motion to complete the chess robot task:

- Assignment 1:
  - **Forward kinematics**: This was used to determine the current position of the robot in task space after reading the joint angles from the servo motors using the function readFB. This was used at several points in order to check for errors in position in both velocity and position control during debugging.
  - **Inverse Kinematics**: This was used to determine what joint angles the servo motors needed in order to reach specific positions in task space. As all the positions of the chess pieces on the chess board are presented in task space, this was a crucial step in task 1 where position control was used.
- Assignment 2:
  - **Jacobian**: This was primarily used in order to transform joint space velocities into task space velocities. As only the motor angles were available as feedback from the servo motors, this was an essential step in order to produce the required task space results in task 2.

- **Inverse Jacobian**: This was used in order to transform the task space velocities that were needed to be sent into joint space velocities. As the motors only accepted velocity commands sent in joint space, this was also crucial to the completion of task 2.

- Assignment 3:
  - **Trajectory generation for position**: This was used in task 1, where several via points were created and a trajectory generated so that the robot can be sent specific joint space angles at specific time intervals.
  - **Trajectory generation for velocity**: This was used in task 2 where the robot needed to hold a particular position in task space, so that it could be freely manipulated by a user. In order to hold the robot in a particular position a velocity trajectory needed to be generated.

3 Task 1: Position Control

3.1 Position Control

3.1.1 Structure of the Position Control System

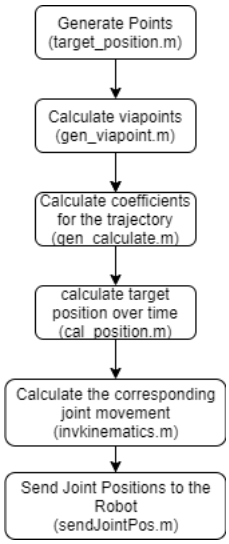


Figure 6: Structure of Position Control System

As can be seen in figure 6, task 1 is driven by six parts: generate the required start and end points, calculate via points, calculate coefficients for the generated trajectory, calculate target position over time, calculate the corresponding joint movement, and send joint positions to send to the robot. The corresponding Matlab code functions are shown in brackets.

Generate Points is a process that generate the exact x, y, z values for the pickup position and the drop position. To obtain the values for the pickup and drop process, previous measurements of the chess board and the associated pieces have been taken from assignment 1. For instance, target_position.m takes two inputs, which are row number and column number respectively. The column number will generate a y-value and the row number will generate an x-value. Both of these values will be calculated using the length of the chess board, along with other relevant measurements such as the distance of the robot to the chessboard.

After obtaining two end points of the whole trajectory, the via points can be calculated according to the start point and the end point. The method that has been used to calculate via points has been provided in assignment 3. The key idea is that the two via points are located directly above the start and end points of the chess piece. This is to ensure that the movement of the chess piece does not knock over any other chess

piece. Using the function that has been created, the generation of the via points can be automated.

As a result of the two via points, the entire trajectory can be separated into three parts. The code will iterate across each part and calculate the coefficients for each part. This is done by gen_calculate.m. The calculation of the coefficients is explained in assignment 3. Specifically, a cubic trajectory was used for this assignment, as only velocity and position needed to be controlled.

After the relationships between the trajectory and time are obtained, it is possible to calculate the corresponding position of the end-effector during operation of the robot. Cal_position.m is used for calculating the corresponding x,y,z value of end-effector. These positions will then be passed to invkinematics.m to use the inverse kinematics equations obtained from assignment 1 so that the joint space position commands can be obtained according to the corresponding required task space position. Finally, the joint position of the robot will be sent to the robot through sendJointPos.m.

3.2 Controller Design and Tuning

The position control mode has an in built PD controller which is implemented inside of the Arduino code. The in built controller for position control is used to main the position of each joint displacement. Due to the properties of the robot, the proportional and differentiation coefficients were used to provide a better performance. By using the controlled variable method, the differentiation coefficient is kept constant and the proportional coefficient is changed to determine the best performance. Through testing, the proportional coefficients of 30, 10, 5, and 1 were tested. It was found that when the proportional coefficient is 5, it has the least amount overshoot and does not have oscillation. Therefore, this was chosen as the coefficient for the controller. Integral control was not used in position control. This is mainly due to the fact as there appeared to be no steady state error, and the PD controller performance was sufficient to provide good performance on task 1.

3.3 Home Position

The home position must meet the following requirements:

- The position of the end-effector needs to be higher than any possible position of the pick up point and drop point so that it will not hit any chess piece.
- The position of the end-effector should be above the position of the chess board so that it can have a shorter traveling distance from its home position to the required pick up point.
- The position needs to be relatively close to the base of the robotic arm so that the centre of mass of the robotic arm will be closer to the base of the robot which reduces the load for the servo motors.

As a result, the chosen home position for the robot is $\{0°, 70°, 0°, -Q2 - Q3, 0°\}$ in joint space.

3.4 Gripper Position

The gripper position was picked to be parallel to the x-axis regardless of the position of the end effector. This is necessary because of the unique shape of the knight. To enable the knight to be picked up consistently and steadily, the piece must be picked up from its sides, where it has the largest amount of surface area. Therefore, during the entirety of task 1, to maintain consistency, all the chess pieces will be picked up from their sides. This is also beneficial to other pieces, as most of the prescribed movements have a component along the x axis, adding extra stability by preventing rotational movement. Allowing the gripper to be in this orientation can also reduce the effect of interference due to vibrational movement after placing the chess piece.

3.5 Calibration and adjustments

Due to the slightly loose connections between links in the joint, the inverse kinematics is not 100% accurate through the operation time. Therefore, calibration and minor adjustments are required. Information about

the design alterations will be mentioned below in the discussion section.

Calibration is essential every time when the robot is carried from one place to another because when it has a rapid movement, some of the joints can become looser and are shifted slightly. Although the distance change can be minor, it is not negligible. Small changes in the flexibility of the joints can cause a difference in the result of the inverse kinematics. As a result, the lengths of l1,l2,l3,l4,l5 were measured every time in order to obtain the most accurate inverse kinematics variables.

Due to imperfections in 3D printing and the fact that joint angles may change every iteration due to connections being secured by screws, there are issues with the robot achieving perfect accuracy after a large transportation has occured from our home to the lab environment for example. Although the servo motor can rotate accurately to the commanded angles, it sometimes encounters issues of not reaching the desired position as commanded. This has caused a problem that can be solved by implementing some minor changes for specific moves. This is more likely to happen when the target position is further away from the base of the robot as most of the error is caused by the joint with the second bearing Q2, due to it's construction being susceptible to transportation wear. As a result, minor adjustments were implemented throughout the tasks when required. For example, if the y value of column D is varying compared to the last execution at a different location, an offset Δy value of D1 will be added to compensate for the open loop system. A similar principle is also applied to the x axis.

As Q2 and Q3 are susceptible to small angle errors, some of this error is propagated to Q4 because Q4 is designed to be calculated by using the fact that it should be facing in the opposite direction of the z axis in task space (-Q2-Q3 as derived in assignment 1). However, since the internal angle readings of Q2 and Q3 are sometimes inaccurate due to a rotational allowance in these joints, the angle of Q4 will have a specific fixed error which is dependent on the fixed error propagated from the joint Q2 specifically. This is a problem especially when the gripper needs to pick up the and drop a chess piece. Therefore, two variables called offset1 and offset2 are used to adjust the desired position of the gripper respectively so that the gripper can be perpendicular to the ground which makes the picking up and dropping functions work normally. By using some effective programming solutions, the performance of the robot has been corrected so that it can provide repeatable and robust results without changing the mechanical design of the robot significantly. Even though this is not the most systematic method to solve the problem, it still results in good performance while also meeting the time constraints of the project.

Our testing to ensure the metric was met included trying to same move 5 times in a row to ensure that the robot was able to repeatedly pick up and place a piece given the same initial conditions. It was successful in completing this task for all moves, and hence we concluded that the robot was robust enough for the specified task.

3.6 Final Results and Comparison

An example of the output of move 2, as compared to the planned trajectory is presented below. The comparisons for each axis are also presented in separate graphs.

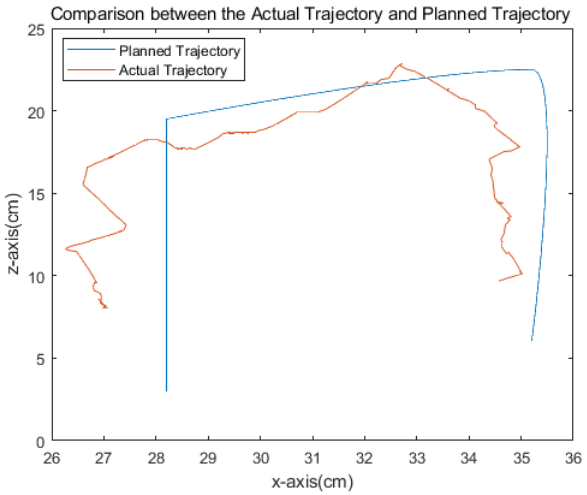


Figure 7: x vs z Value Comparison in Move 2

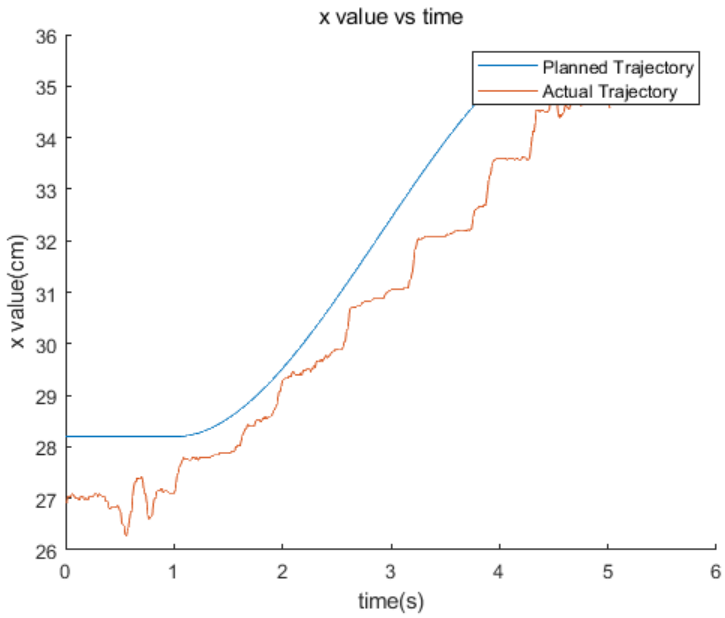


Figure 8: x value Comparison in Move 2

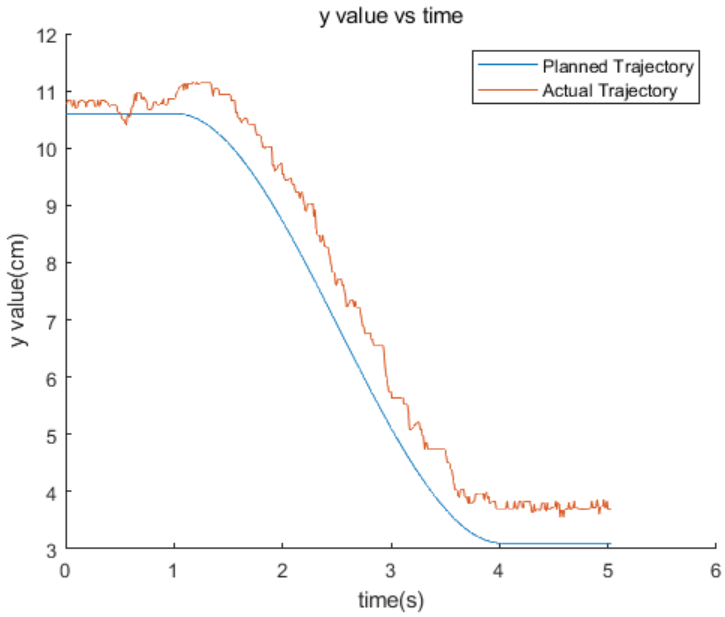


Figure 9: y value Comparison in Move 2

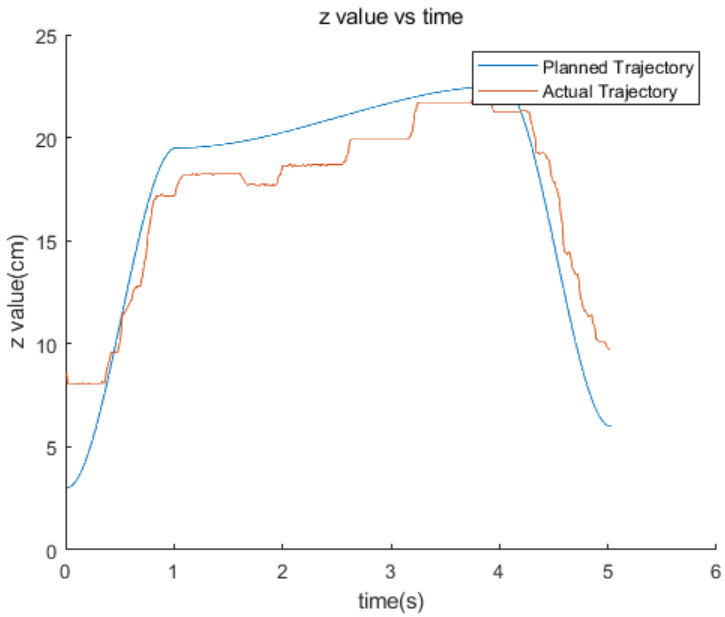


Figure 10: z value Comparison in Move 2

As represented by the figures above, it can be seen that although the general shape that each axis should take is correct, they still have the difference between the planned trajectory and the actual trajectory. There are many different factors that may cause this difference, which will be discussed below in the discussion section. Comparing figure 8, figure 9, and figure 10, it can be found that the worst case is the control of the x-axis. The motors that are responsible for movement in the X axis are mostly Q2 and Q3, due to the fact that movement of pieces is usually forward, along the x axis. A few of the reasons for the inaccuracies would be:

- The amount of rotation that Q2 can exhibit is limited by the motor torque, as Q2 has to support the most torque, it struggles occasionally, despite accurate calculated design requirements from assignment

2.

- The amount of flex in joint Q2 due to the double bearing design, can result in a larger amount of torque required to move this torque compared to a solution without a bearing
- A change in centre of mass over time can result in a poorer performing Q2 motor as the robot moves further away from it's rotational base. Regardless of the rest of the robot design, this is a nonlinear variable, that should realistically be modelled in order to achieve optimal results in terms of error propagation.

These issues are discussed in detail in section 5.3, and solutions to alleviate these issues are also proposed.

4 Task 2: Velocity Control

Task 2 consisted of allowing free moving elements in only one axis, and restricting movement in other axes, similar to how a piece would move on the actual chessboard. The specifics behind the actions taken to implement certain moves will be discussed in their respective subsections.

4.1 Sending Task Space velocity without feedback

In order to allow the robot to only move in one axis, a velocity trajectory was generated using the trajectory generation methods presented in assignment 3. The velocity trajectory was to lift the robot vertically up, and then hold that position. At this point it allowed for the end effector to be manipulated by a human user.

In a continuous for loop, the trajectory velocity commands were sent to the robot in velocity control mode. Since the trajectory velocity is generated in task space, these needed to be converted into joint space.

There were several attempts at creating an inverse for the Jacobian, these were presented in a paper by Buss (2004). Singular value decomposition and the pseudo inverse worked equally as well in the final results.

Singular value decomposition is found by first reconstructing the Jacobian in the following form:

$$J = UDV^T \tag{1}$$

Where $J \in \mathbb{R}^{m\times n}$, $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ and $D \in \mathbb{R}^{m\times n}$. D is a diagonal with only non zero entries in the diagonal. As discussed in the aforementioned paper, the pseudoinverse can be calculated using the following:

$$J^\dagger = VD^\dagger U^T \tag{2}$$

The pseudo inverse of D can be determined by the $n \times m$ diagonal matrix where each nonzero value is inverted.

The pseudo inverse can also more simply be calculated by using the direct pseudo inverse formula, which is as follows:

$$J^\dagger = J^T(JJ^T)^-1 \tag{3}$$

The expression to transform velocity and angular velocity into joint angle velocities is presented below:

$$J^\dagger \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \dot{Q}1 \\ \dot{Q}2 \\ \dot{Q}3 \\ \dot{Q}4 \\ \dot{Q}5 \end{bmatrix} \tag{4}$$

By generating the desired positional trajectory by using the results in assignment 3, in conjunction with the inverse kinematics approach presented in assignment 1, explicit joint angles at each time step are also able to be generated. These joint angles are required in order to use the jacobian matrix, and as a result the jacobian needs to be recalculated at each time step.

At this point the joint velocity commands can be directly sent to the motors. The next section will discuss how feedback can be incorporated into the design.

4.2 Task Space velocity with feedback

As discussed in the previous section, once the desired reference velocity has been generated, it's possible to send these commands to the motor.

Since the motors provided are servo motors, it's also possible to read the angles that the motors currently have back to the MATLAB code. By calculating the difference between successive joint angles in the for loop where the system runs and dividing by the time the code takes to execute one loop, it's possible to find a discrete representation of the velocity of the system.

Due to the fact that there is an nonzero amount of jitter in the motors even at rest, along with small movements getting amplified by the small (¡0.03) dt value, the joint velocities resulted in very noisy signals. To filter out the jitter that occured at rest values, a mode filter which selects the most common value among a set of values was used. The window size used here was 4. In addition, a moving average filter of window size 4 was used in order to obtain more accurate velocity values.

After the filtering has occured, the joint velocity values are transformed into task space using the forward Jacobian matrix, as shown in assignment 2. This results in the actual task space velocities for the system.

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = J \begin{bmatrix} \dot{Q}1 \\ \dot{Q}2 \\ \dot{Q}3 \\ \dot{Q}4 \\ \dot{Q}5 \end{bmatrix} \tag{5}$$

The Jacobian matrix joint angles are also known, as they can be directly read from the servo motor feedback.

Since the reference velocity and position have been generated already as discussed above, the error between the reference velocity and the actual velocity in each of the task space dimensions can be directly calculated. This error value was then combined with the PID equation as described below:

$$\dot{x_C} = \dot{x_{ref}} + K_p e + K_I \int_{t=0}^{t_k} e\, dt + K_d \frac{de}{dt} \tag{6}$$

The same equation is true for the y and z axes as well.

The PID controller was tuned by first increasing the K_p gain until the system started to oscillate. At this point the K_p gain was slightly reduced, and the K_d gain was increased until the oscillations stopped. After this the K_p and K_d gain were increased in conjunction to improve the response time of the system. The last step was increasing the K_I gain to account for any fixed constant offset error. The final values used for each gain values in each task space axis direction are presented in the table below:

Gain Values	K_p	K_I	K_d
Axis			
x	2.5	0.05	0.33
y	2.4	0.05	0.33
z	4.3	0.07	0.33

Table 2: A table representing the Gain values that worked best for operation of the robot in task 2.

To improve the sent velocity results, the $\dot{x_C}, \dot{y_C}, \dot{z_C}$ were also passed through a moving average filter, with a window size of 3. The inverse jacobian is used to transform the task space velocity commands back into the joint space velocity commands. As the inverse jacobian operation is an approximation, there is a slight amount of error that results from this operation.

Once the joint space velocities are known, these values are then passed to the motors directly through the in built MATLAB function provided with the assessment. The full implementation of the system is represented in a block diagram format below:

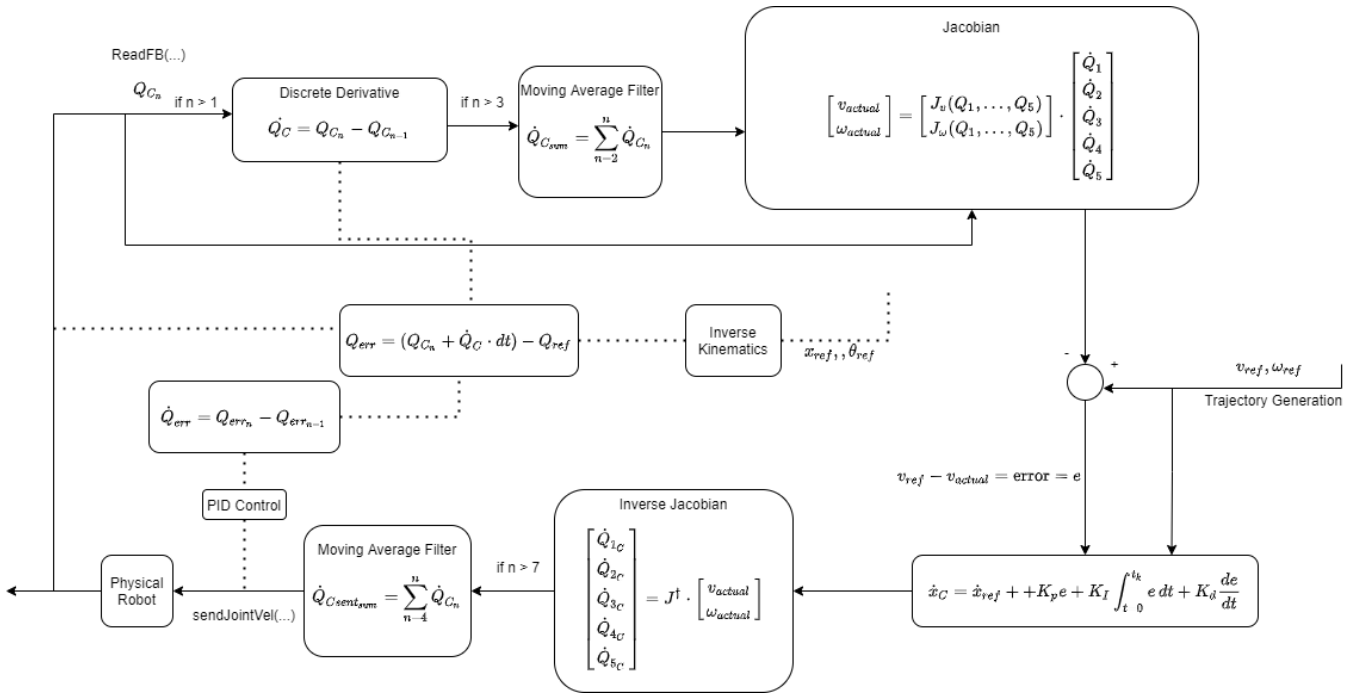


Figure 11: Full block diagram architecture of control system used, Note: Dotted lines indicate joint space error calculation and control, these were not nessecary in our implementation due to fast execution time of the code, in different implementations it may be nessecary, and hence was included.

As the orientation of the gripper was specifically meant to face exactly down, this was the reference angular velocity was initially set to be [0,0,0], assuming that this was also the initial orientation of the gripper. A more complex version of this was implemented in future implementations towards the demo, as the original orientation was not able to be guaranteed. The reference orientation was generated by reading in the original angular displacement of the end effector which was given by the rest of the system. After this, the result from assignment 3 was used in order to calculate the rotation matrix needed to move from the original rotation frame to the final rotation frame. At this point the euler parameters were generated and then used as the reference angular rotation.

Using these results, the robot that was designed was able to maintain specific desired positional trajectories, using velocity based commands.

4.3 Moving the Rook

To allow movement of the end effector in directions allowed by the rook chess piece, the following modifications were made:

- The control system in the x and y axis is originally set to gains of all 0.
- While the velocity in either the x or y axis higher than a set threshold which is also greater than the ambient noise value of the velocity is detected, the system enters a separate loop. In this loop the gains of the axis where a velocity is not detected are set to the values mentioned in table 2.
- If the velocity in the respective axis returns to a value below the specified threshold, both axis are unlocked once again, and the system will be able to move in either the x or y direction again.

The basic architecture of how the gains are set depending on the state of the system is provided in the block diagram below:

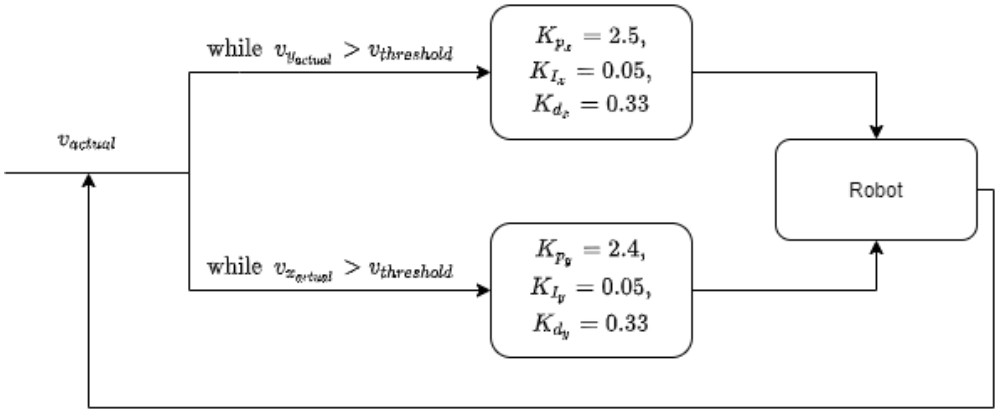


Figure 12: Architecture of how the gains for the rook are set depending on the state of the system

4.3.1 Results for Rook

The following graphs exhibit the results for the implementation:

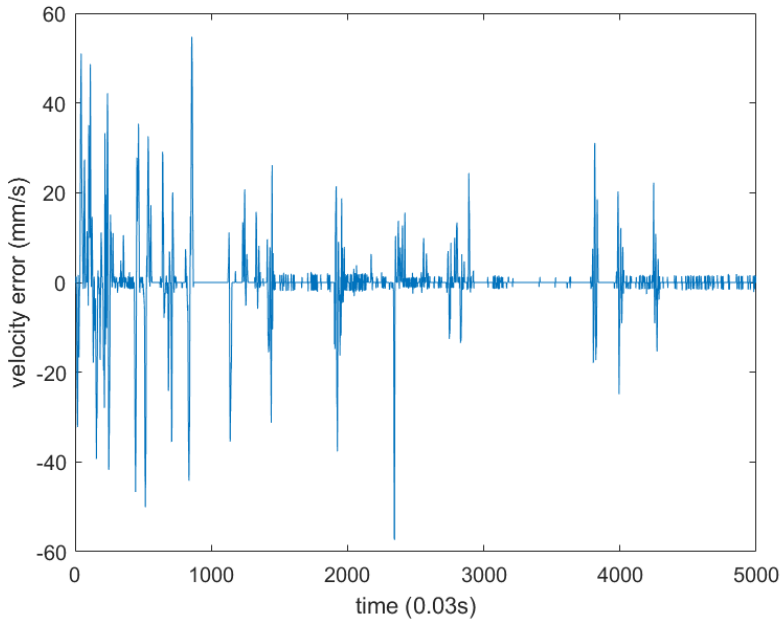


Figure 13: A representation of the velocity error of the robot in the z axis, when using the algorithm for the rook described above.

As it can be seen in the graph, when a disturbance is originally applied to the system, there is a large amount of compensation, this is eventually reduced towards the middle of the graph when the system settles in the z direction.

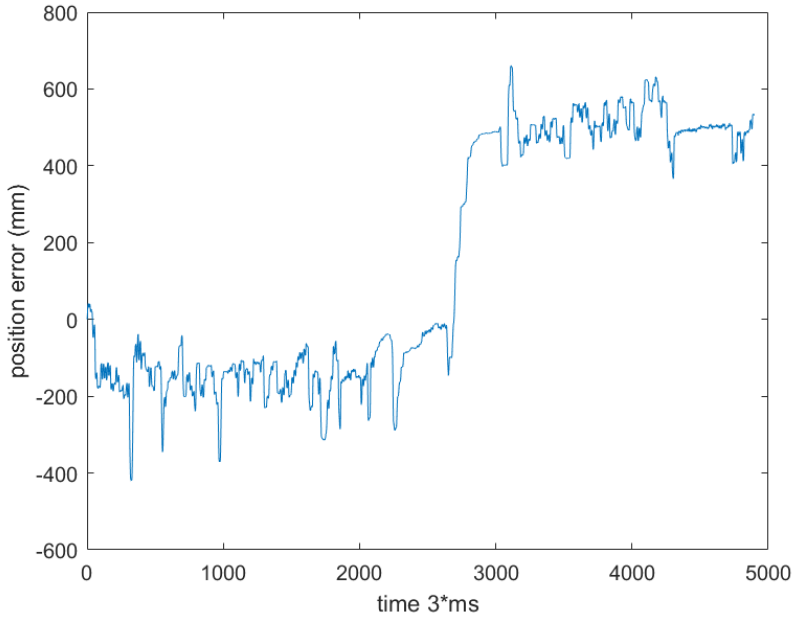


Figure 14: A representation of the position of the robot in the y axis, when the robot is restricted to only move in the x axis, when using the algorithm for the rook described above.

As can be seen from the data between [0,3000], There is a correction in the y axis direction even when the robot is pushed continiously to one direction. This shows that the algorithm is correcting actively for the displacement. There is some amount of noise, as the pseudo jacobian matrix is an approxmation of the inverse, in addition to the dt value used to calculate the velocity magnifying any jitter in the original motor angle readings. After this point the robot was moved to another position after the code was exited, and this was recorded in the results, to display that the robot can move to a different direction in comparison when there is no control algorithm operating.

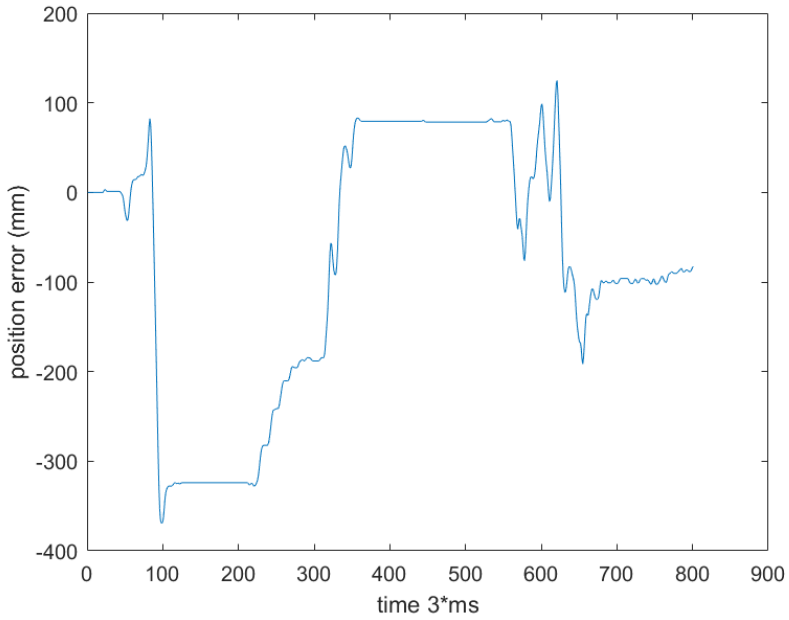


Figure 15: A representation of the position of the robot in the z axis, when the robot is restricted to only move in the x axis, when using the algorithm for the rook described above.

As can be seen from Figure 15, a similar correcting effect can be seen when the robot attempts to correct for a displacement in the z axis, this correction effect is much more pronounced than the correction in the y axis, which appears to suffer from some jitter. This will be discussed further in the discussion section in this report.

4.4 Moving the Bishop

The movement of the bishop piece is similar to the rook piece. The only exception is the fact that the movement is rotated 45° around the original task space.

To implement this movement, the initial start point of the robot is set to be a corner space, this can be implemented by generating the start point of a trajectory to be one of the corners of the system.

In the control loop, the control system is rotated in task space before the control loop is implemented. This ensures that the system always attempts to correct for errors around the axis of the bishop, which is 45 from the axis of the rook. The following rotation matrix is used:

$$ {}^{B}_{R}R = \begin{bmatrix} cos(\frac{\pi}{4}) & -sin(\frac{\pi}{4}) & 0 \\ sin(\frac{\pi}{4}) & cos(\frac{\pi}{4}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7} $$

The basic architecture of how the gains are set depending on the state of the system is provided in the block diagram below:

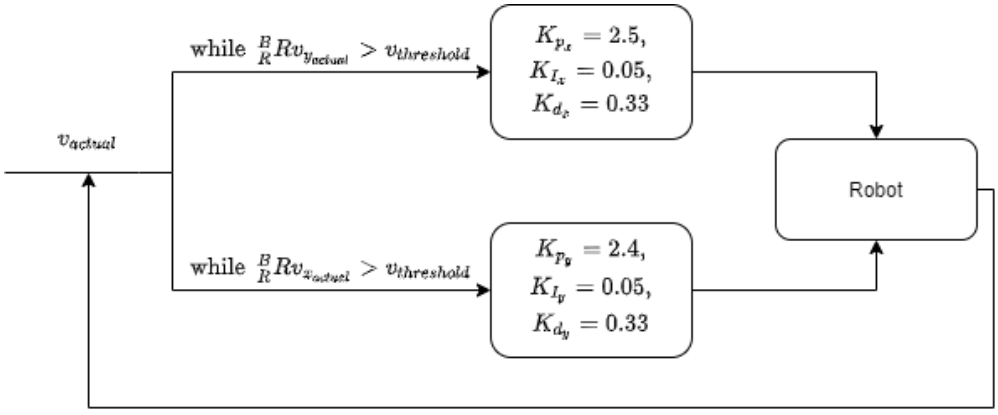


Figure 16: Architecture of how the gains for the bishop are set depending on the state of the system

4.4.1 Results for Bishop

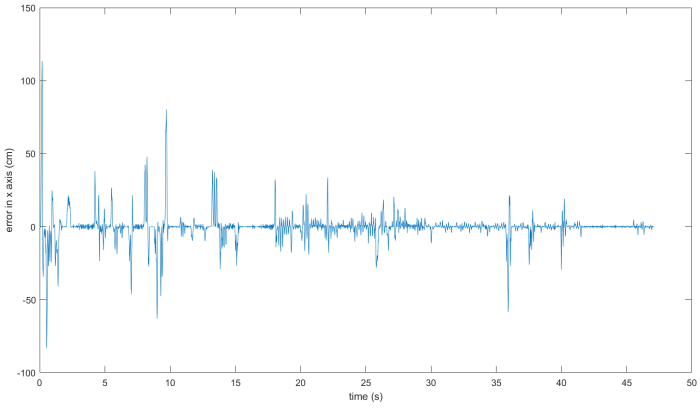


Figure 17: A resulting graph of the error in velocity from moving the bishop. As can be seen, the velocity quickly returns to zero after the target position has been reached and the error has been corrected.

As can be seen from the graph above, the velocity returns to close to zero after an error has been applied, but before that the velocity quickly shoots up to correct for the error. The small amount of residual velocity is due to several factors. The main cause is residual oscillations from the control loop applied, and the secondary cause is error being magnified from the feedback reading due to the aforementioned jitter being present.

4.5 Moving the Knight

Applying the above principles to moving the knight resulted in mixed success. To implement the movement that a knight can take the following steps were taken:

- The axis of the control system was returned to the axis used in movement of the rook
- By integrating the actual velocity that was being sent, it was possible to obtain an approximation of the distance that the knight piece was moved in any direction, this was used as the metric to control how far the knight can move.
- Initially, only movement in one axis is allowed, this is done by setting the gains in one axis to the gains presented in table 2. When the movement has exceeded a fixed amount which is equal to 2 chess squares (82.4mm), the system swaps the x and y gains.
- When the gains are swapped, a displacement in only the opposite axis is allowed. Once again, when the displacement exceeds the length of one square (41.2mm), the gains in the x and y axis are set to the values in table one, no longer allowing any movement.

The basic architecture of how the gains are set depending on the state of the system is provided in the block diagram below:

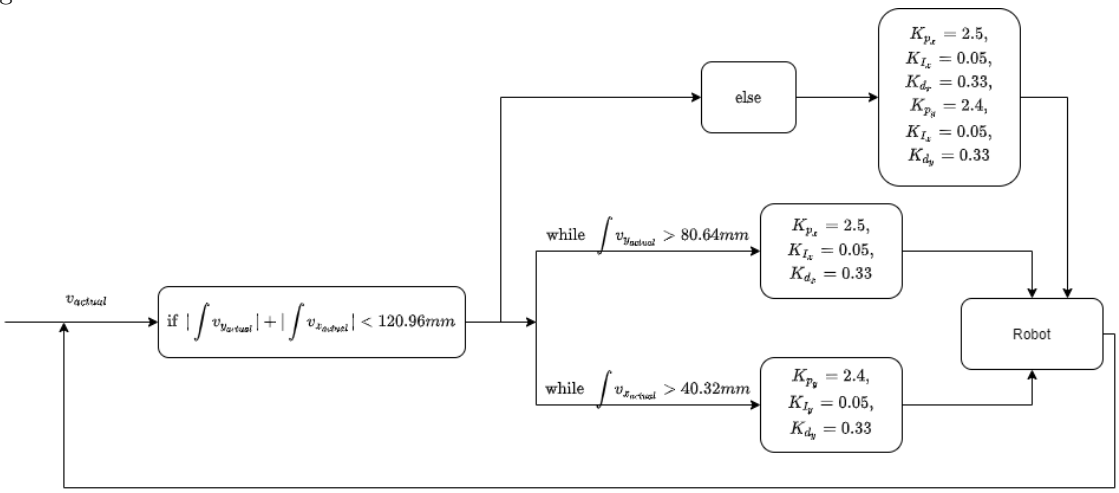


Figure 18: Architecture of how the gains for the knight are set depending on the state of the system

One of the key issues with implementation of the knight, was the fact that the movement relied on an integrated distance measurement. Since velocity data was very noisy even after being put through a filter, as shown in Figure 13, it was very difficult to accurately determine how much forwards the knight needed to move before the direction is locked. This instability along with the failure of a motor during the demonstration resulted in a lack of results for the knight. The tutor was informed of this issue during the lab.

5 Discussion

Discussion and results pertaining to specific tasks are provided in the text above, this section will focus on issues and aspects associated with the operation of the robot, and how they could be improved in the future.

5.1 Mechanical Issues

There were a few issues with the design the mechanical design that impacted both the performance of task 1 and task 2. As both tasks were implemented using an open loop model of the robot, the quality of the joints plays a vital role in the performance, because any subtle change in the physical design can significantly affect the projection in task space.

The links are connected by 3D printed joints. These joints were secured using M3 screws. These screws did not have washers and thus had no vibration dampening. Throughout our testing it was noted that the joints had a tendency to loosen, thus affecting the relative position of the joints. As this is a moving object, it future it would be wise to use washers.

Another observed issue was that the flanges that were used to connect the motors to the shafts exhibited plastic deformation. This indicates that the choice of material was not appropriate for the forces that the flange would be subjected to. This introduced some play to these joints, thus decreasing the accuracy of the model. Time permitting it would be wise to replace the shafts and flanges with custom machined parts, to ensure a high quality coupling between the motors and the limbs.

For the base, 8mm thrust bearings were chosen to match the size of the ball bearings. This was a bit of an oversight as a wider bearing would have been better able to resist the moment created by the robot arm. This put more strain on the motor, and also contributed to the sometimes jerky movement of the robot.

5.2 Manipulator

With all else being equal, the motion of the manipulator was very reliable, accurate and precise. The gripper would always close correctly and if placed correctly around a chess piece would pick it up and deposit it accurately with no drift. Unfortunately all else was never equal, the manipulator is located at the end of the robot and as such is the most susceptible to changes earlier in the chain. Due to the mechanical issues presented above, the gripper would never be in the same place even if the same sequence of commands were sent to the motors. This lead to some significant issues. While the design was able to correct for alignment errors in line with the clamping action, it would not be able to correct for misalignment in the other direction. Though, it was noted that while the design would fail to pick up pieces, it would rarely if ever knock pieces over, as intended. A future design could incorporate a tri-axis gripper. This design would be more robust at picking up pieces as it would be able to correct for errors in both the x and y direction, unlike the current design. The existing design did not max out the torque rating of the gripper motor, as such there is extra headroom for a more elaborate design.

The design was also a little cumbersome and struggled to fit in-between some pieces. This was partly due to the wide "fingers" on the end of the manipulator. With a better choice of material, it would be possible to reduce the width of the "finger" and thus make the robot more nimble. On the other hand, this would also allow for a wider opening and thus would allow the manipulator to cover a wider or the whole axial distance of the chess board squares. The trade-off being that a wider area would have a higher chance of interfering with other pieces, but would be more robust when correcting for misalignment's.

Finally, the release of pieces that were not collected correctly presented some issues, with pieces regularly toppling over. This was most likely due to the sudden release of the pieces. A solution would be to use velocity control to ensure the gradual release of the pieces, thus minimising impact forces on the piece that could produce a moment and knock it over. Also focusing on picking up the pieces near their center of gravity would limit the effect these forces have on the stability of the pieces as the moments would be less.

As discussed above in section 3.5, to ensure robustness of the design, and the repeatability of the experiments, each move was tested 5 times in a row. If a move was able to be completed 5 times in a row, it indicated to us that the system was robust, and the moves were repeatable given the same initial conditions.

5.3 Discussion and Improvements suggested for Task 1

Although the robotic arm was successfully able to complete all the moves required by task 1, there are still aspects that can be improved. Firstly, it was found that when the end-effector is far away from the rotational base, the performance decreases. This is because when the centre of mass of the robotic arm moves away from the base of the robot, the required torque from the motor will increase. Eventually, the servo motor will have insufficient torque to support the links to hold in position. A counter weight was implemented to decrease some of the negative effects of this issue. The best way to solve this problem is to change the servo motor to a motor with a higher torque or change the design of the robotic arm to have a different centre of mass or less weight overall.
The most efficient way to solve this problem however, is to model the amount of proportional gain required to lift the robotic arm at different lengths away from the base, and make the proportional gain a function of the distance from the rotational base. As the required proportional gain will be nonlinear due to the 3 degree of freedom robotic arm, this is the best way to approximate the result to ensure the performance is consistent throughout the range of operation. The main issue with the performance decrease over the range of operation was the fact that it made the output a bit more inconsistent. By adding in a proportional controller that scales based on the calculated distance of the robot, it's possible to reduce some of the variance associated with overshoot in the controller and produce more reliable and consistent operation.

5.4 Discussion and Improvements suggested for Task 2

It was noted that one of the key issues with completing task 2 was in relation to the fact that the velocity sent to the robot did not exactly correspond to the actual velocity received by the robot. While the solution presented in Figure 11 by the dotted line can alleviate some of the issues in relation to the sent and received velocity, it doesn't show much effect for low sampling times, as experienced during testing. An alternative solution would be to program some of the code using Arduino directly, or a faster compiled coding language such as C++. Increasing the sampling time would reduce overshoot as the robot would be moving in an incorrect direction for a lesser amount of time.

Another aspect that would directly improve the performance of task 2 while not drastically changing the equipment would be to use two motors at the base of the robot. It was noted during testing that the proportional gain needed to fix errors in the z axis were much more significant than the proportional gain in the x and y axis. This was because the motor located at the joint Q_2 needed the most amount of torque in order to lift the rest of the robot. The key issue with this however, is that to reduce the height of the robot, the motor does not need as much torque as it has gravity assisting it. This resulted in larger errors when manually decreasing the height of the robot. A stronger torque at that joint could significantly improve performance, as a lower amount of proportional gain would be needed to account for errors.

The final improvement that can be made in regards to velocity control would be in relation to the discrete velocity calculated as mentioned in section 4.3. Due to the jitter in the motor readings during operation, the transformed velocity readings had a large amount of variance. To reduce this effect, if the motors had a form of internal velocity readings, a much more effective velocity control system would be able to be implemented.

6 Conclusion

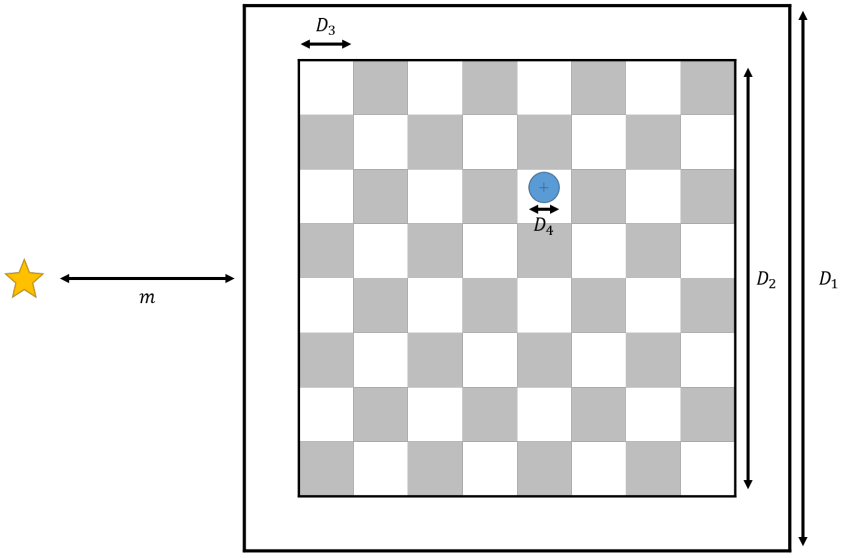
In conclusion, the robot that was created over the duration of this course was able to perform a variety of chess moves provided by the project parameters. The solution for task 1 was robust, and able to pick up the chess piece from several positions, despite the inherently open loop design. In task two, velocity commands were used to restrict the movement of the robot end effector to specific planes. Overall the results considering the task description have been excellent.

Improvements to the output of the project in the future would be primarily changing some of the mechanical design, and the end effector design, to ensure a higher success rate in both tasks 1 and 2. In addition to this, adding several robustness features to the mechanical design could also improve performance. Reducing the amount of free movement about joints would improve the accuracy of the programming, especially in task 2 where this is critical.

7 References

1. Buss, S. R. (2004). Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods (No. 1). https://groups.csail.mit.edu/drl/journal$_c$*lub*/*papers*/033005/*buss*−2004.*pdf*

8 Appendix



(a) Diagram of chess board with robot base

Parameters	
D_1	381.6 mm
D_2	329.6 mm
D_3	41.2 mm
D_4	$\approx$ 27.5 mm
m	mm

(b) Parameters

Figure 19: Dal Rossi Chess Board Dimensions

Parameter	Designed Value
Q_1	-80°to 80°
Q_2	0°to 70°
Q_3	-160°to -20°
Q_4	$-Q_3 - Q_2$
l_1	34cm
l_2	34cm
l_3	34cm
l_4	7.5cm
l_5	7.5cm

Table 3: Final design lengths and angles