Chess Playing Robot with EMG Signals for Paralyzed People RRR Serial Manipulator

ENG401

190412060 İslam AYDOĞMUŞ
190412081 Ozan AYDIN
180412047 Metehan KARAŞAHİN
190412023 Bilal YAMAN
180412066 Buse KARAKOÇ
150412014 Ege ALEMDAR

Interdisciplinary Engineering Project

December 1, 2022

Contents

1 Introduction

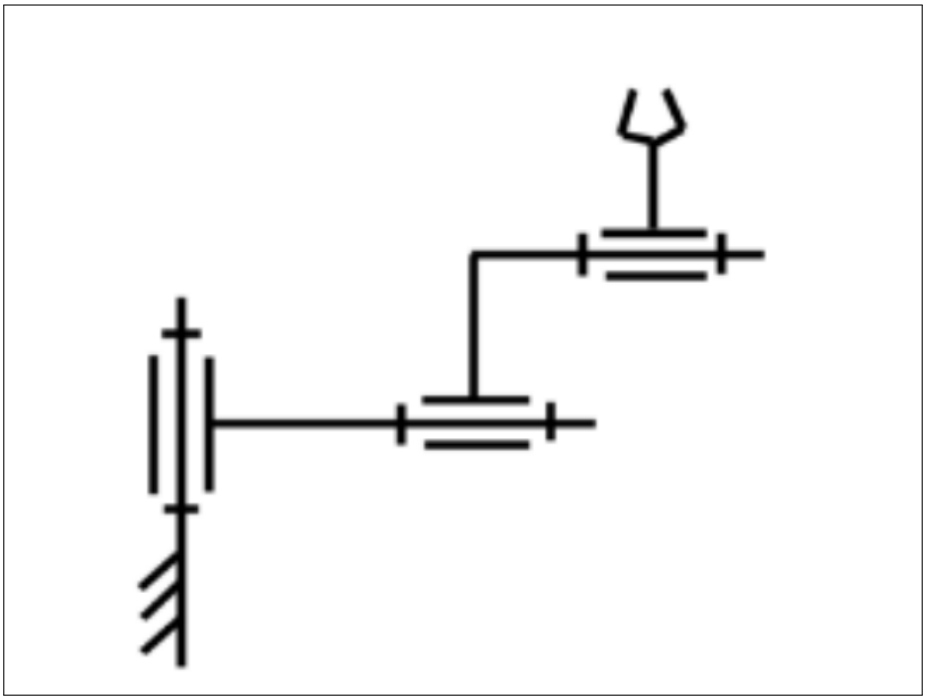


Figure 1: Kinematic representation of our RRR manipulator

Our robot enables individuals with physical disabilities to play chess with only EMG signals. The design of our robot has a task space to include a chessboard with an area of 54x54cm^2. It is aimed to have the feature of taking the stones at the points they reach to other stones by moving with 3 DoF in this space. In line with these goals, the robot design was designed with computer aided design and its kinematic analysis was carried out.

2 Analysis

2.1 Workspace Analysis

Workspace of a robot is the space which the defined end effector is capable of reaching with respect to our joint constraints. The forward path function can be easily found since we use a paralel manipulator as;

$$x = b\cos(\theta_1)\cos(\theta_2) + c\cos(\theta_1)\cos(\theta_3)$$
$$y = b\cos(\theta_2)\sin(\theta_1) + c\cos(\theta_2)\sin(\theta_1$$
$$z = a + b\sin(\theta_2) + c\sin(\theta_3)$$

where a, b and c our link lengths and will be defined as 0.1, 0.4 and 0.4 at the latter stages. Since we have 3 joints that is defined as RRR as can be seen by figure (1) we can define our joint constraints as given with respect to our potantiometer limits. The code that inspects

Joint	Interval
J1	30 to 150
J2	0 to 130
J3	-90 to 90

Table 1: Our joint constraints

all of the points in the workspace is as follows

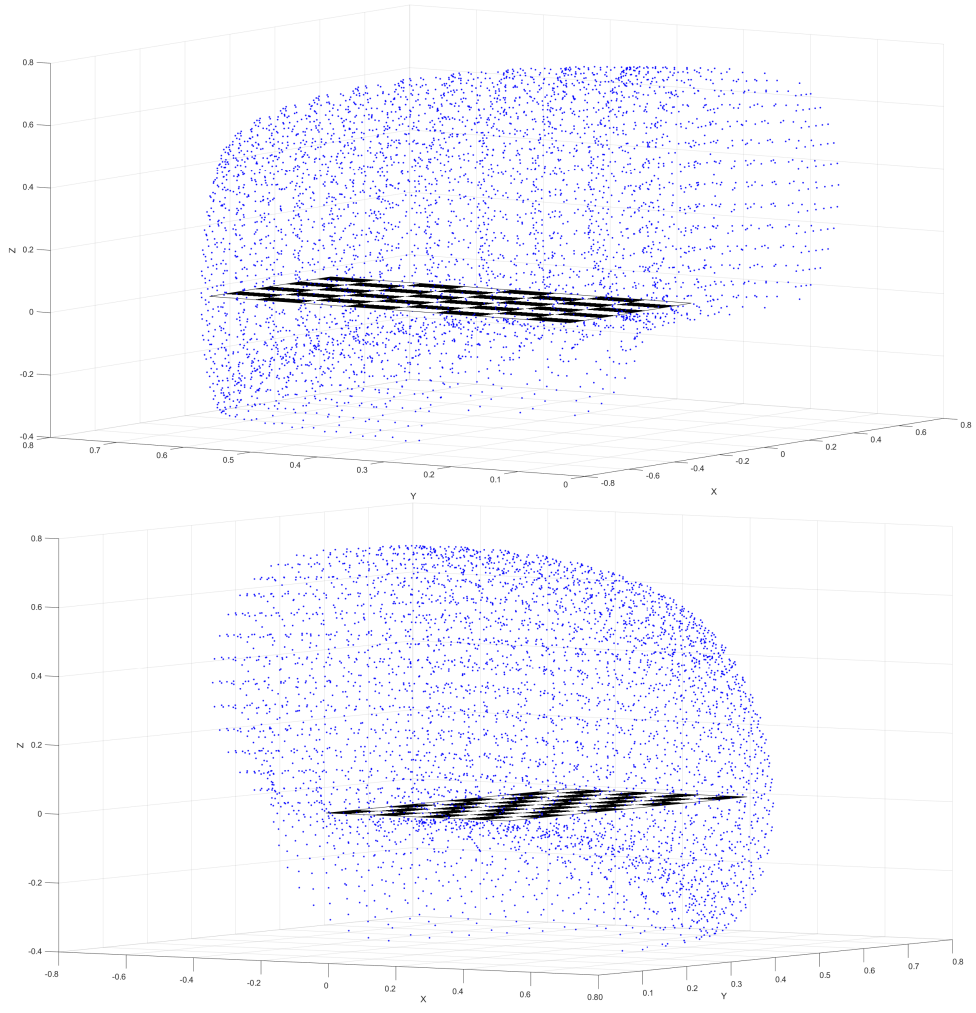
```
dt = 5;
b = 0.4;
c = 0.4;
a = 0;
theta_1_scope = pi/180*linspace(30,150,20); % 45:dt:315
theta_2_scope = pi/180*linspace(0,130,20); % 0:dt:270
theta_3_scope = pi/180*[linspace(0,90,10) linspace(270,360,10)];% linspace
    (135,360,16)]; % [0:dt:45 135:dt:360]

%length(45:dt:315)*length(0:dt:270)*length(0:dt:45 90:dt:360);
%x = eye(length(45:dt:315),1)
for theta_1 = theta_1_scope
    for theta_2 = theta_2_scope
        for theta_3 = theta_3_scope
            x = b*cos(theta_1)*cos(theta_2) + c*(cos(theta_3)*cos(theta_1));
            y = b*cos(theta_2)*sin(theta_1) + c*(sin(theta_1)*cos(theta_2));
            z = a + c*sin(theta_3) + b*sin(theta_2);
            n_vec = norm([x y]);
            if (not (n_vec < 0.17)) && (y > 0)
                plot3(x,y,z,'b.')
                hold on
            end
        end
```

```
    end
end
grid on

img = imread('Black_and_white_checkered_pattern.jpg');     % Load a sample image
xImage = [0.27 -0.27; 0.27 -0.27];   % The x data for the image corners
yImage = [0.2 0.2; 0.74 0.74];             % The y data for the image corners
zImage = [0 0; 0 0];   % The z data for the image corners
xlabel("X")
ylabel("Y")
zlabel("Z")
surf(xImage,yImage,zImage,...    % Plot the surface
     'CData',img,...
     'FaceColor','texturemap');
% Px = [0.169,0.639,0];
% Py = [0,0.47,0.20];
% Pz = [0.236,0.234,0];
% Px = [0.169,0,-0.236];
% Py = [0.639,0.47,0.234];
% Pz = [0.1,0.40,0.1];
% plot3(Px,Py,Pz,'r')
%i=imread('Black_and_white_checkered_pattern.jpg');
%patch( [-0.27 0.27 0.27 -0.27], [0.25 0.25 0.79 0.79],i,'r')
```

The Generated Workspace



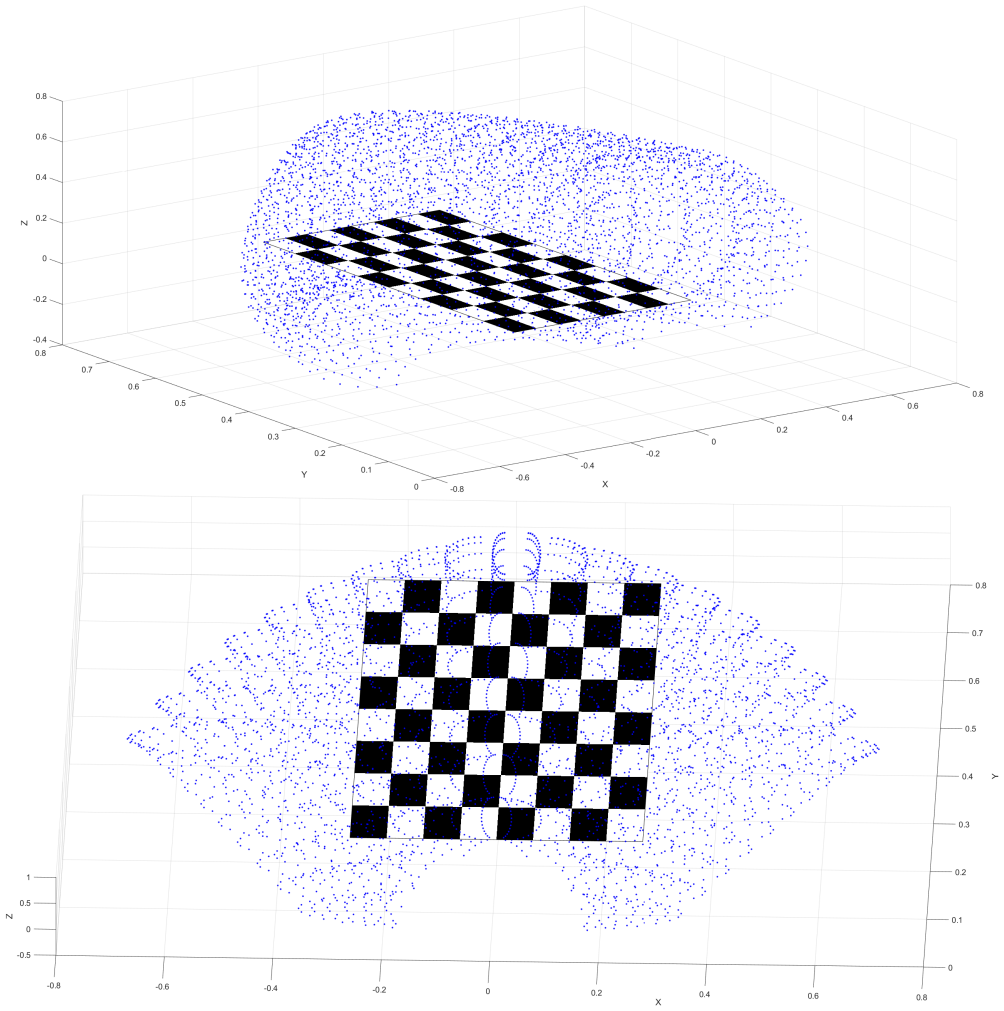
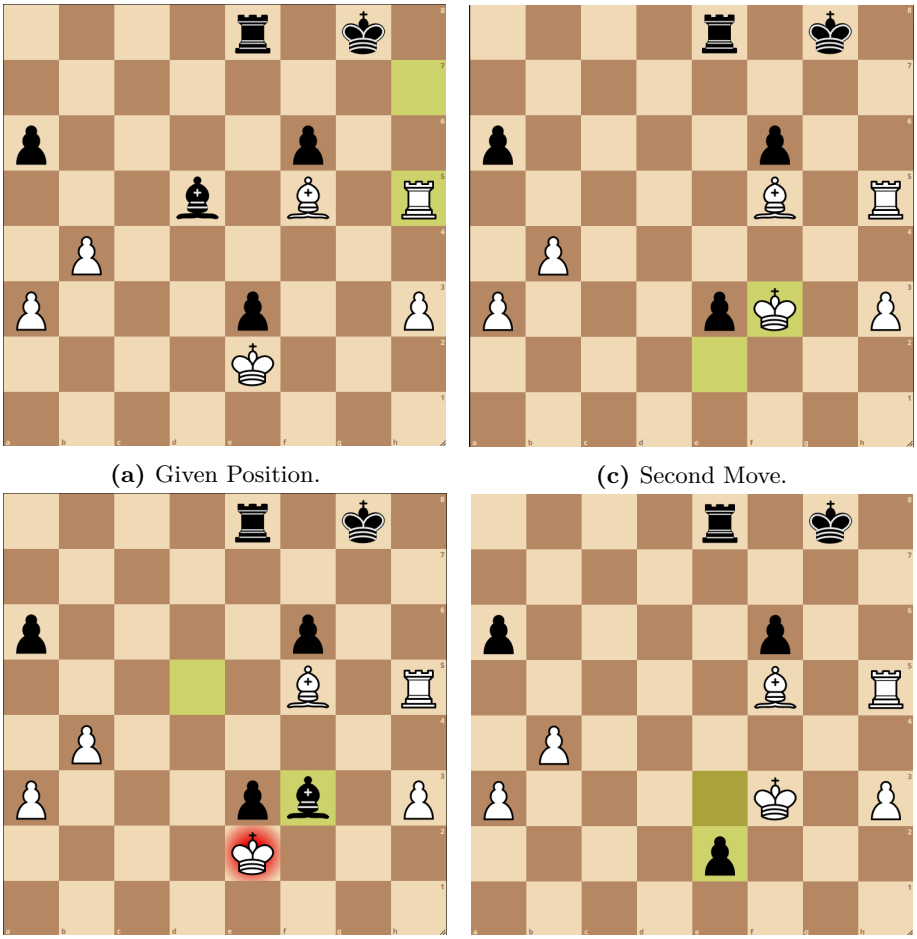


Figure 2: Workspace of our Manipulator.

2.2 Kinematics

2.2.1 Task

Our task is to design a robot that is able to solve chess problems. We used the daily puzzle of the www.lichess.com for our puzzle.



(a) Given Position. **(c)** Second Move.

(b) First Move. **(d)** Third Move.

Figure 3: Daily Puzzle of the lichess.com at 1 December 2022

Or we can give the notation as

Figure 4: Chess notation

2.2.2 Task to Coordinate Transformation

Now we want an interface for us to convert the given chess board coordinates. For example when we want the coordinates of 'B_2' the function will give us $(x_{B_2}, y_{B_2},)$. So we wrote the script;

```
function [Sq] = findSqr_XY(given_pos)
%findSqr_XY Finds given square in terms of our coordinates
    given_x_pos = given_pos(1);
    given_y_pos = given_pos(2);

    if not (((0 < given_x_pos) && (given_x_pos< 9)) && ((0 < given_y_pos) && (
    given_y_pos< 9)))
        msg = 'Given indicies out of bound';
        error(msg);
    end
    %Const
    BOARD_HW = 0.54;
    BOARD_HALF = BOARD_HW/2;
    SQ_HW = BOARD_HW/8;
    BOARD_Y = 0.2;

    %Calculation
    Sq_x = ((given_x_pos-1)*SQ_HW-BOARD_HALF+SQ_HW/2);
    Sq_y = (given_y_pos-1)*SQ_HW+BOARD_Y+SQ_HW/2;
    Sq = [Sq_x,Sq_y];
end
```

Now that we find the coordinates of our squares we must interpolate a third point to construct a polynomial trajectory. This is utilized by following script;

```
function Pos = find_middle_P(P1,P2)
    P1_coord = findSqr_XY(P1);
    P2_coord = findSqr_XY(P2);

    P1_x = P1_coord(1);
    P1_y = P1_coord(2);
    P2_x = P2_coord(1);
    P2_y = P2_coord(2);

    Pos = [mean([P1_x,P2_x]),mean([P1_y,P2_y])];
end
```

2.2.3 Generate Trajectory

Polynomial that is going to represent our trajectory is constructed through x axis and then reverse projected on the plane that intersects with z-axis, P_1 and P_2. First we find the projection of our trajectory of x axis through 3 point 2nd order polynomial fit.

$$\underbrace{\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_i^2 & x_i & 1 \\ x_2^2 & x_2 & 1 \end{bmatrix}}_{\textbf{Known}} \quad \underbrace{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}_{\textbf{Coeff.Matrix}} = \underbrace{\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}}_{\textbf{Known}}$$

Then we converted the calculated x projection to calculated plane and translated with required distances. But this method does not work for purely y axis trajectories, so we made an exepction for that case. The script that has been written considered these knowledge is written as follows:

```
function v_out = gen_traj(C1,C2)
    %Const
    BOARD_HW = 0.54;
    BOARD_HALF = BOARD_HW/2;
    SQ_HW = BOARD_HW/8;
    BOARD_Y = 0.2;
    GRIP_Z = 0.05;
    GRIP_Zh = 0.4;
    dt = 0.001;

    if C1 == C2
        msg = 'C1 == C2!!';
        error(msg);
    end

    P1 = [findSqr_XY(C1),GRIP_Z];
    Pinter = [find_middle_P(C1,C2), GRIP_Zh];
    P2 = [findSqr_XY(C2),GRIP_Z];
    mat = [P1;Pinter;P2];

    if (C1(1) == C2(1)) % If contains only y movement
        cofmat = [P1(2)^2 P1(2) 1;Pinter(2)^2,Pinter(2),1;P2(2)^2 P2(2) 1]\[GRIP_Z
    ;GRIP_Zh;GRIP_Z];
        if P2(1) > P1(1)
            x = P1(2):dt:P2(2);
        else
            x = P2(2):dt:P1(2);
        end
        y = cofmat(1).*x.^2 + cofmat(2).*x + cofmat(3);
        v = [P1(1)*ones(1,length(x));x;y];
    else
        cofmat = [P1(1)^2 P1(1) 1;Pinter(1)^2,Pinter(1),1;P2(1)^2 P2(1) 1]\[GRIP_Z
    ;GRIP_Zh;GRIP_Z];
        if P2(1) > P1(1)
            x = P1(1):dt:P2(1);
        else
            x = P2(1):dt:P1(1);
```

```
        end
        y = cofmat(1).*x.^2 + cofmat(2).*x + cofmat(3);

        %find necessary plane
        angle = atan2(P2(2)-P1(2),P2(1)-P1(1));
        v = eye(3,length(x));
        for i = 1:length(x)
            y_comp = x(i)*tan(angle);
            v(:,i) = [x(i);y_comp;y(i)];
        end

        % Convert angle to 2pi proximity
        while angle < 0
            angle = angle + 2*pi;
        end

        val_y = min(v(2,:));
        v(2,:) = v(2,:) + abs(val_y) + min(mat(:,2));
    end
    v_out = v;
end
```

2.3 Denavit Hartenberg Parameters

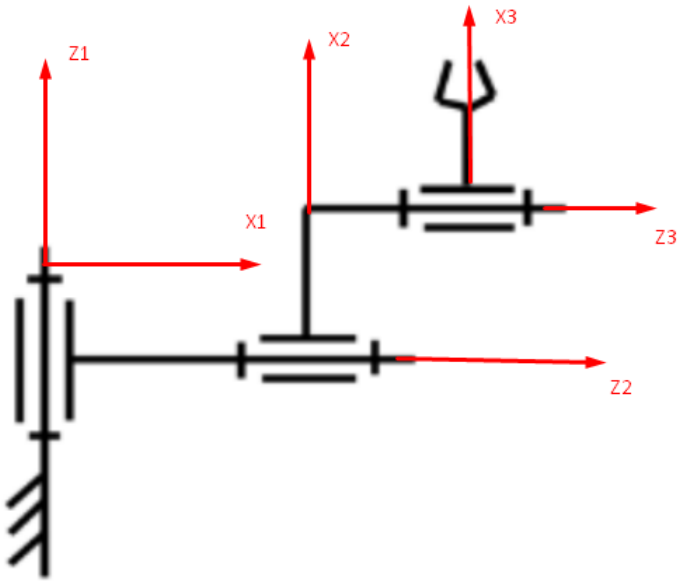


Figure 5: Given z and x axis

Denavit–Hartenberg parameters (also called DH parameters) are the four parameters associated with a particular convention for attaching reference frames to the links of a spatial kinematic chain, or robot manipulator.
The DH table of our manipulator that can be seen in fig. (5) is constructed with respect to above axes is as following:

i	$\alpha_{i-1,i}$	$a_{i-1,i}$	d_i	θ_i
1	0	0	0.1	θ_1
2	$\pi/2$	0	0	θ_2
3	0	0.4	0	θ_3
4	0	0.4	0	θ_4

Table 2: DH Table

Above table is used in all kinematic and dynamic equations.

2.4 Numeric Inverse Kinematics

Inverse kinematics is solved by robotics toolbox with the help of Peter Corke Robotics Toolbox for MATLAB. This toolbox allowed us to generate our robot from our DH table. All end points generated in the previous section is converted into the joint angles. The following code is used to generate this utility:

```
L(1) = Link([0 0.1 0 0],'standart');
L(2) = Link([0 0 0 pi/2],'standart');
L(3) = Link([0 0 0.4 0],'standart');
L(3).qlim = [-90 90]*pi/180;
L(4) = Link([0 0 0.4 0],'standart');

q0 = [0 -pi/2 -pi/4 0];
R = SerialLink(L,'name','robot');

time_step = 0.2;

x = gen_traj([4,5],[6,3]);
%x = gen_traj([5,2],[6,3]);
%x = gen_traj([5,3],[5,2]);
[n,m] = size(x);

%% some variables
error_ = 0;
err = 0;
err_i = 1;
x_1 = 0;
y_1 = 0;

for i=1:m
        mat = transl(x(1,i),x(2,i),x(3,i));
        HG_matrix(:,:,i) = mat;

        %% ikine is numerical inverse kinematics solver
        % it outputs joint angles
        angle_ = R.ikine(HG_matrix(:,:,i),'q0',[pi/2 pi/2 pi/2 pi/2],'ilimit',200,
    'tol',1e-2, 'quiet','alpha',1,'rlimit', 200,'mask',[1 1 1 0 0 0]);
        J_angles(i,:) = angle_;
end
```

For checking the feasibility of this result we generated an animation script and locus of the en effector by following equation:

```
%plot3(x(1,i),x(2,i),x(3,i),'.r')
img = imread('Black_and_white_checkered_pattern.jpg');     % Load a sample image
xImage = [0.27 -0.27; 0.27 -0.27];   % The x data for the image corners
yImage = [0.2 0.2; 0.74 0.74];             % The y data for the image corners
zImage = [0 0; 0 0];   % The z data for the image corners
xlabel("X")
ylabel("Y")
zlabel("Z")

while 1
    time_step = 0.2;
```

```
    speed_multiplier = 10;
for i=1:m
%%figure(2);
plot3(x(1,i),x(2,i),x(3,i),'.r')
xlim([-1 1])
ylim([-1 1])
zlim([-1 1])
hold on
b = J_angles(i,:);
R.plot(b,'delay', time_step/speed_multiplier,'ortho','jointcolor', [0.1 0.5 0.6],'
    basewidth',6,'linkcolor','black')
surf(xImage,yImage,zImage,...    % Plot the surface
     'CData',img,...
     'FaceColor','texturemap');
end
pause(0.5);
i = 1;
end
```

The resulting animations is as follows

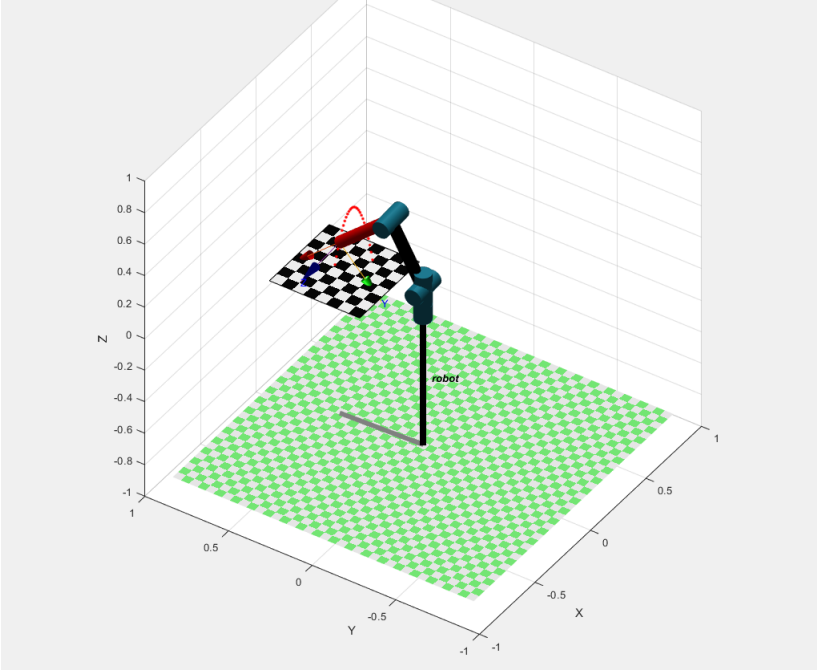


Figure 6: Movement on fig. (3b)

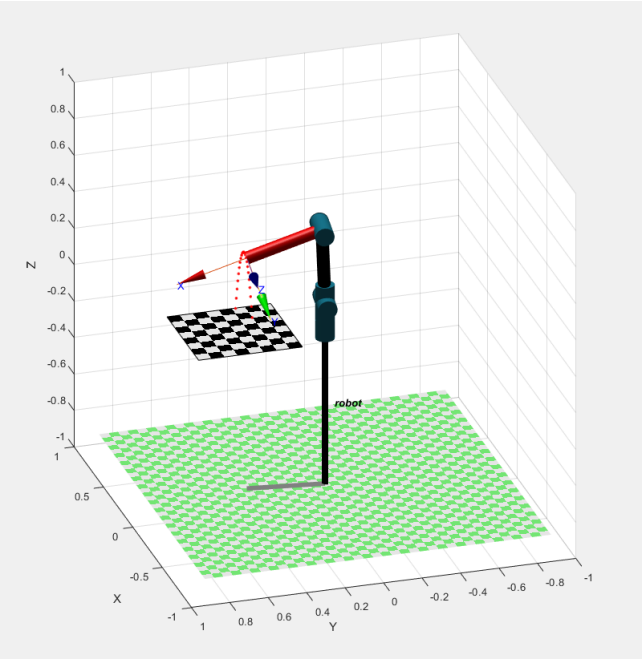


Figure 7: Movement on fig. (3c)

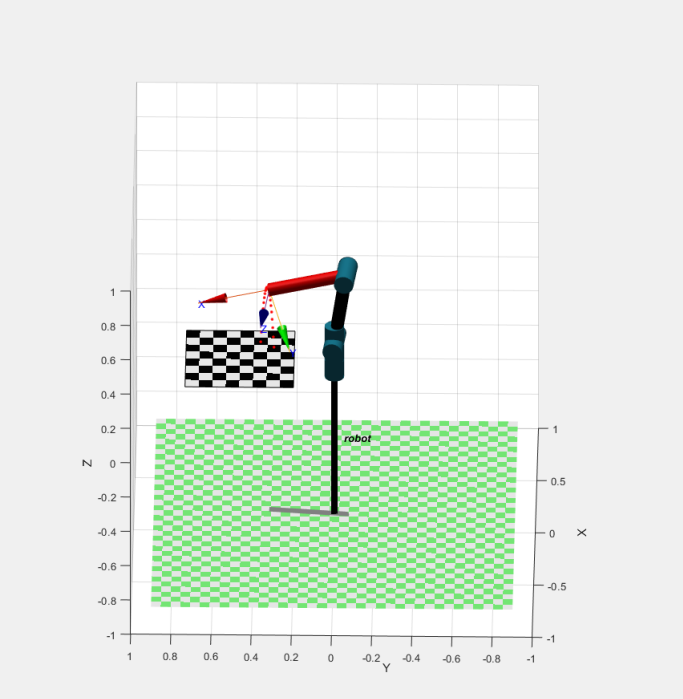


Figure 8: Movement on fig. (3d)

3 Dynamic Analysis

We used the motion analysis tool of the Solidworks for this results. Motor torques on a determined trajectory were calculated by the program and graphed. The results are shown in the three graphs below.

3.1 Ground Motor

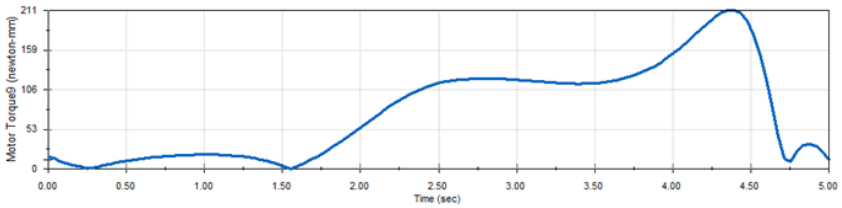


Figure 9: Moment vs time of Ground Motor

It is seen that a maximum torque of 211Nmm is required on ground motor, since all series arms are loaded on ground motor, its torque was expected to be this high. When the robot arms are opened and closed, it can be seen that the torque changes a lot because the diameter also changes. The RMS of our first motor is 101.1 Nmm.

3.2 First Motor

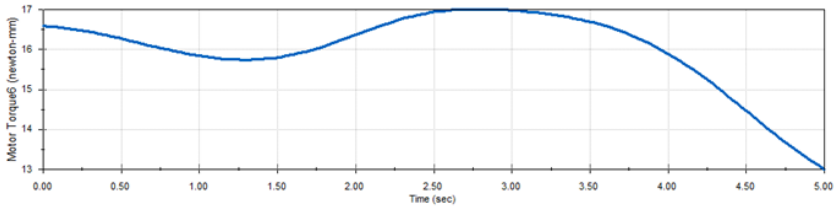


Figure 10: Moment vs time of First Motor

It can be observed that motor 1 has less torque than ground motor. this is because engine two carries less load and the farthest arm is usually closer to the engine. in this way, this motor can be controlled with less torque as the diameter is smaller. The RMS of our first motor is 16.0 Nmm.

3.3 Pulley Motor

Since this motor has the least load, we can control it with a very small amount of torque as can be seen.
The above values are taken over the most extended trajectory on the chessboard (a trajectory on the diagonal of the chessboard). Therefore, if a robot is to be built that only works within the specified limits, it will perform the task as long as the motors that meet the above values are installed. But if it is operated in an area other than the working area,

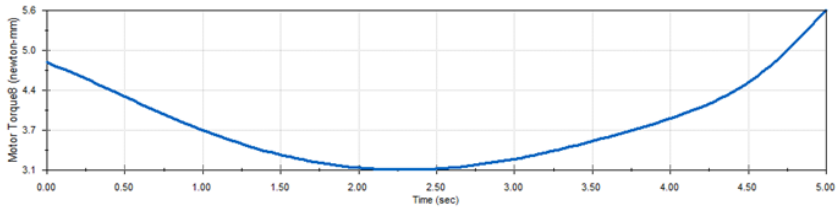


Figure 11: Moment vs time of Pulley Motor

these torque values will change. So we may lose control over the arm.

3.4 About Motor Selection

When the values are extracted from the plots above and the RMS values are found, we see that the RMS torque values for our motors are 101.05, 15.95 and 3.94, respectively. The normal torque values of our motors can be selected accordingly. but the max torque values of the motors should also provide the maximum torques in the graphs above. therefore motor 1 must be chosen carefully. because the motor has a high amount of change on moment, while the RMS torque value is low.

With these considerations we use the table from reprap.org to chose our motor. NEMA 17 17HS08-1104S Stepper Motor has 13 Ncm torque potential. Even though this is sufficient for us to satisfy the RMS torque valeus, our max torque is climbs up to 211 Nmm from time to time so we suggest that we should use the motor of NEMA 17 42HS02. Specs. of NEMA 17 motors we chose are as follows:

Model	Holding Torque	Step Angle	Rated Current	Inductance
17HS08-1104S	13 Ncm	1.8 Degree	1 A	4.5±20% mH
42HS02	22 Ncm	1.8 Degree	0.4 A	21±20% mH

Table 3: NEMA 17 motor specification with respect to their models

4 Design

4.1 Motors

Figure 12: NEMA 17 Motor

We used NEMA 17 step motors to control the mechanism since it's very easy to find it. The motors. They are very capable of producing great torques since they have an inner gear reduction system. The following technical drawing is used for our designs.

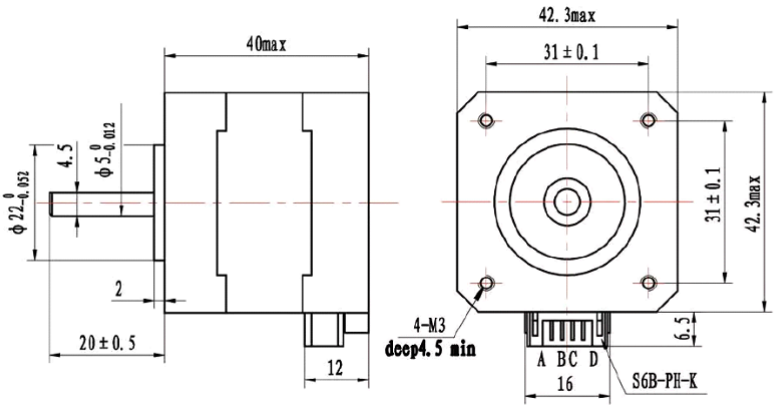


Figure 13: NEMA 17 Motor Dimensions

4.2 Feedback

Figure 14: Rotary Potentiometer

Angular position information of each joint is gathered through a rotary potentiometer. These potentiometers usually have a 270 degrees of rotation angle. This is clearly a restriction for us but the task space is still contained the chess board after potentiometer restrictions.

4.3 Ground

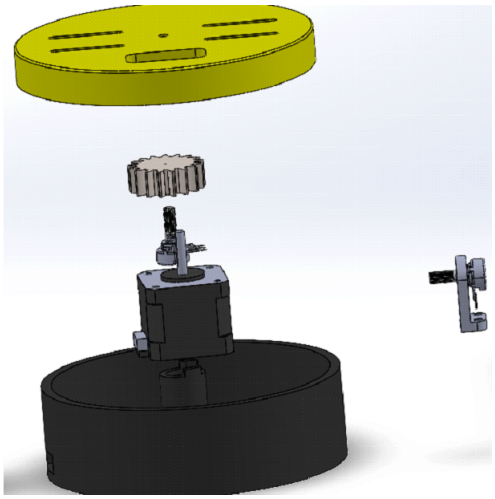


Figure 15: Our ground assembly

Ground consists of 2 main parts. These parts are main ground and plate. The first part is the main ground, which is connected to the earth and the second part is the plate. In the main part, there is the motor housing, the motor corresponding to the slot and the potentiometer bed where we can get feedback and measure the position. There is a gear in the connection of the plate part with the ground and the motor on the ground and the gear engage each other. There is another gear in order to receive feedback and the potentiometer

is connected to the inner hole of this gear. On the other side of the plate, there are the motors that control the links and their slots, and the slots for the support elements.

4.4 Link 1

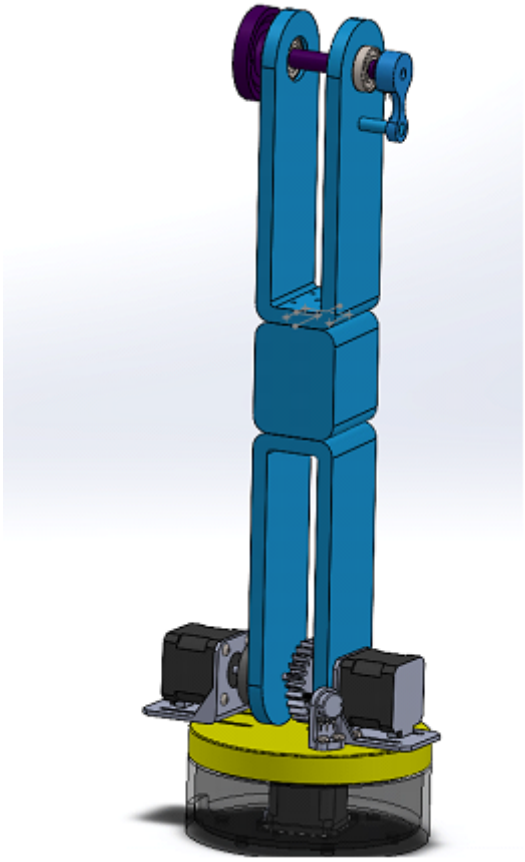


Figure 16: Link 1 of our robot

The first link is directly controlled by the motor. In order to measure the position, there is a gear which is connected to the motor shaft, and this gear rotates with the other gear, allowing us to receive feedback. In addition, there is an auxiliary element to support the potentiometer on the first link for position measurement and feedback of the other link.

4.5 End Link

The control of the end link is controlled by the motor on the ground through the belt pulley. The pulley is screwed together with the end link and there is a bearing at this end of the end link. The end link has a potentiometer for position measurement and feedback, and this potentiometer gets its support from the first link. On the other end, there is the gripper that will take the chess pieces and release them.

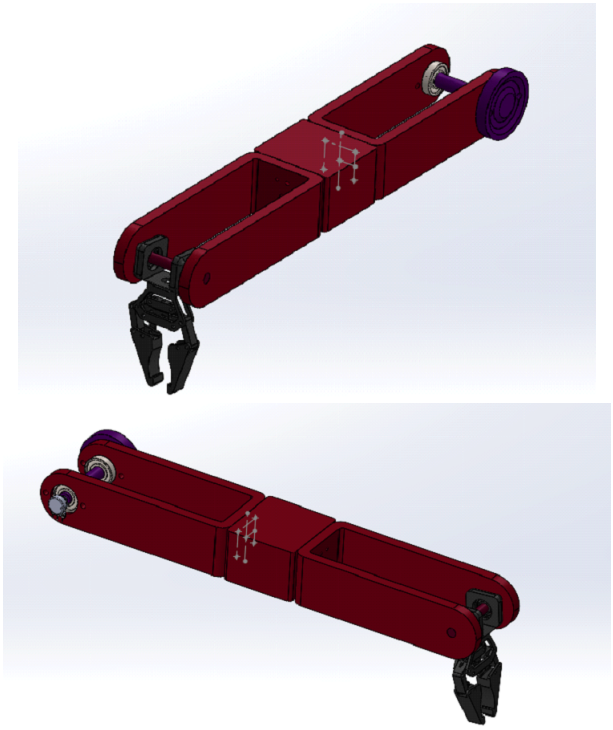


Figure 17: End link of our with the gripper

4.6 Final Design

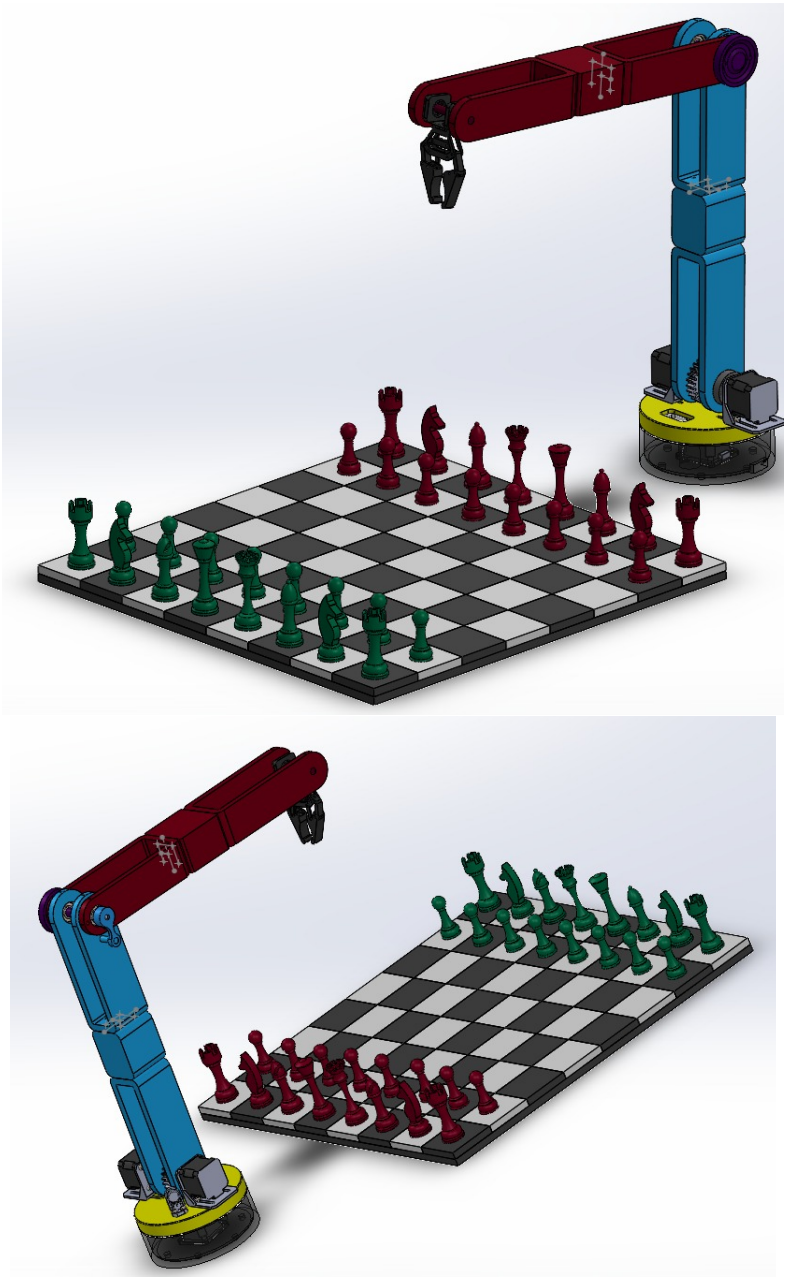


Figure 18: Our robot with the chess board

5 Gripper

5.1 Objective

The task in this section is to design a passive gripper that can move all chess pieces smoothly. The size of the squares on the chessboard is 6x6cm and the largest piece is 3.8cm

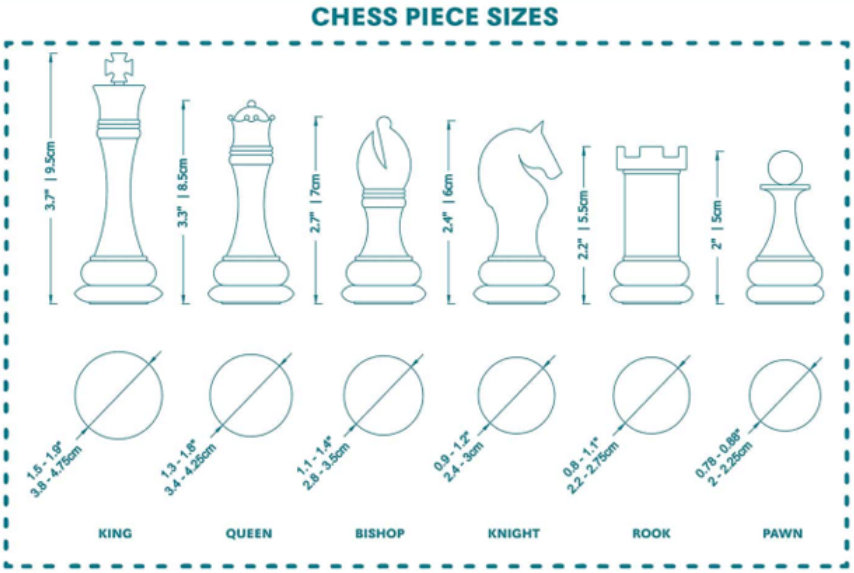


Figure 19: Original Chess Piece Size According to FIDE

in diameter. When the calculation is made using these data, there is a minimum operation area of 4.4 cm. In the figure shown above, the most optimal grasping areas of the chess

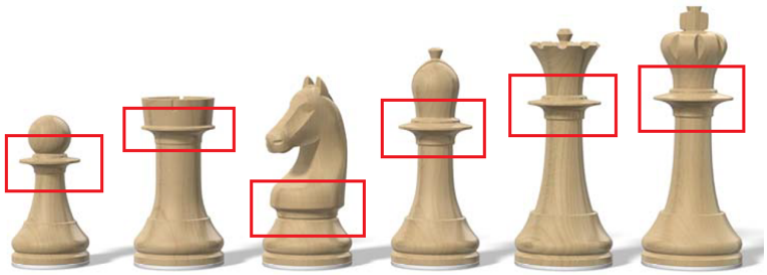


Figure 20: Targeted grip points

pieces are shown with red squares. When the above regions are selected, a comment can be made about the gripper height. Among the chess pieces in figure, the most difficult to reach is the knight. This measurement from the horse's head to the grip part corresponds to approximately 5.15 cm. So the gripper length must be longer than 5.15 cm.

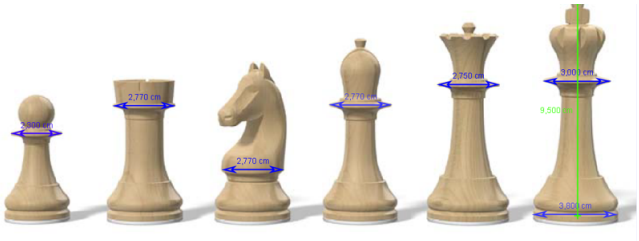


Figure 21: Approximate Grip Diameters of Pieces

As a result of the above analysis, it is deduced that the diameter measurements required for gripping should be in the range of 2.2 cm <Dg <2.3 cm. These obtained data provide enough information to design a gripper.

5.2 Gripper Design

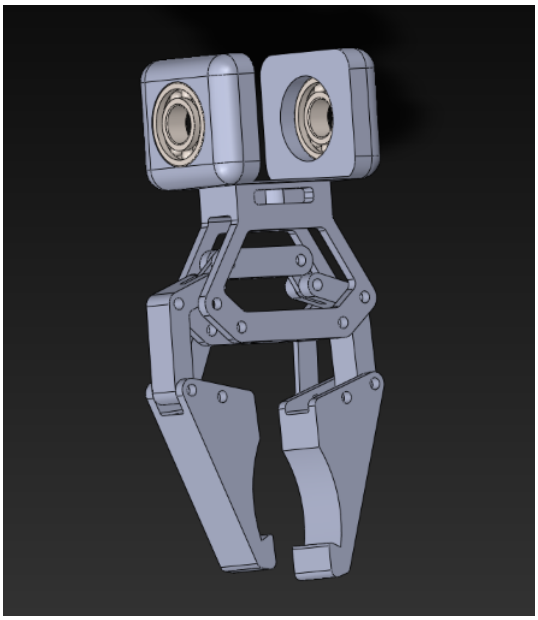


Figure 22: CAD of our gripper.

The Gripper's design was made with certain criteria in mind. These are functionality, budget or feasibility, and appearance. For functionality, we first had to think of a gripper for our task. While lifting the chess piece, it is important that it does not hit other pieces, that the piece does not fall while lifting and lowering the piece, and that it can hold all the pieces in the same way. After providing these, it had to take up little space and perform the task with minimum effort. The gripper, which will look down with the weight given by the motor, will open and close with high precision with the leadscrew mechanism. Secondly, it was considered that the parts of the design we made were either accessible or could be printed from the 3D printer properly. Finally, aesthetics and appearance, It was considered that the robot arm does not look very absurd and different from other parts.