```matlab
%% Script1---- Homogenous Transformation
syms a1 a2 a3 aq4 a5 a6
a1 = 6;
a2 = 3;
a3 = 3;
a4 = 0.5;
a5 = 1.5;
a6 = 1.5;

%% DH Parameters(theta, d, alpha, offset)
% position joints

H1 = Link([0,a1,0,pi/2,0]);
H1.qlim = pi/180*[-90 90];
H2 = Link([0,0,a2,pi/2,0,pi/2]);
H2.qlim = pi/180*[-90 90];
H3 = Link([0,0,0,0,1,a3]);
H3.qlim = [0,3];

% orientation joints

H4 = Link([0,a4,0,3*pi/2,0]);
H4.qlim = pi/180*[-90 90];
H5 = Link([0,0,0,pi/2,0,3*pi/2]);
H5.qlim = pi/180*[0 90];
H6 = Link([0,a5+a6,0,0,0]);
H6.qlim = pi/180*[-90 90];

NS = SerialLink([H1 H2 H3 H4 H5 H6],'name','6 DOF arm')
NS.plot([0 0 0 0 0 0],'workspace',[-15 15 -15 15 -15 15])
NS.teach
%% Script2---- Forward and Inverse kinematics
% Define DH Parameters (theta, d, alpha, offset)
syms a1 a2 a3 a4 a5 a6;
a1 = 6;
a2 = 3;
a3 = 3;
a4 = 0.5;
a5 = 1.5;
a6 = 1.5;

% Define Robot Links
H1 = Link([0,a1,0,pi/2,0]);
H1.qlim = pi/180*[-90 90];
H2 = Link([0,0,a2,pi/2,0,pi/2]);
H2.qlim = pi/180*[-90 90];
H3 = Link([0,0,0,0,1,a3]);
H3.qlim = [0,3];
H4 = Link([0,a4,0,3*pi/2,0]);
```

```
H4.qlim = pi/180*[-90 90];
H5 = Link([0,0,0,pi/2,0,3*pi/2]);
H5.qlim = pi/180*[0 90];
H6 = Link([0,a5+a6,0,0,0]);
H6.qlim = pi/180*[-90 90];

% Create SerialLink object
robot = SerialLink([H1 H2 H3 H4 H5 H6],'name','6 DOF arm');

% Define the desired end-effector pose
T_desired = transl(5, 3, 4) * trotx(pi/2);

%% Inverse Kinematics

% Solve Inverse Kinematics
q = robot.ikine(T_desired, 'mask', [1 1 1 0 0 0], 'solve', 'pseudo');

% Display Joint Angles
disp("Inverse Kinematics - Joint Angles (in degrees): ");
disp(rad2deg(q));

%% Forward Kinematics

% Compute Forward Kinematics
T_fk = robot.fkine(q);


disp("Forward Kinematics - End-Effector Pose: ");
disp(T_fk);

figure;
robot.plot(q, 'workspace', [-15 15 -15 15 -15 15]);
title('Forward Kinematics - Robot Configuration');

%  Inverse Kinematics Plot
figure;
robot.plot([0 0 0 0 0 0], 'workspace', [-15 15 -15 15 -15 15]);
title('Inverse Kinematics - Robot Configuration');
% Trajectory points example
start_point = transl(4, 2, 4) * trotx(pi/2);
end_point = transl(6, 7, 14) * trotx(pi/4);

num_steps = 50;

% Generating the trajectory
T_traj = ctraj(start_point, end_point, num_steps);

% Solve inverse kinematics for each point in the trajectory
q_traj = zeros(num_steps, robot.n);
```

```
for i = 1:num_steps
    q_traj(i,:) = robot.ikine(T_traj(:,:,i), 'mask', [1 1 1 0 0 0], 'solve',↙
'pseudo');
end

% Move the robot along the trajectory
for i = 1:num_steps
    robot.plot(q_traj(i,:), 'workspace', [-15 15 -15 15 -15 15]);
    pause(0.1); % pause for a short duration to visualize the motion
end
%% Script3---- Workspace
% Define DH parameters
d1 = 6;      % Link 1 length
a2 = 3;      % Link 2 length
d3 = 3;      % Link 3 displacement (prismatic)
a4 = 0.5;      % Link 4 length
a5 = 1.5;      % Link 5 length
d6 = 1.5;      % Link 6 length

% Define joint limits
theta1_min = -pi/2; % Joint 1 minimum angle
theta1_max = pi/2;  % Joint 1 maximum angle
theta2_min = -pi/2; % Joint 2 minimum angle
theta2_max = pi/2;  % Joint 2 maximum angle
theta4_min = -pi/2; % Joint 4 minimum angle
theta4_max = pi/2;  % Joint 4 maximum angle

% Define workspace boundaries
x_min = -15;  % Minimum x position
x_max = 15;   % Maximum x position
y_min = -15;  % Minimum y position
y_max = 15;   % Maximum y position
z_min = -15;  % Minimum z position
z_max = 15;   % Maximum z position

% Define step size for workspace points
step = 0.2;

% Initialize arrays to store workspace points
workspace_points = [];

% Loop through all possible joint angles and calculate end-effector positions
for theta1 = theta1_min : step : theta1_max
    for theta2 = theta2_min : step : theta2_max
        for theta4 = theta4_min : step : theta4_max
            % Calculate end-effector position using forward kinematics equations
            x = a2 * cos(theta1) * cos(theta2) + a4 * cos(theta1) * cos(theta2 +↙
theta4) + d6 * cos(theta1) * cos(theta2 + theta4);
            y = a2 * sin(theta1) * cos(theta2) + a4 * sin(theta1) * cos(theta2 +↙
```

```
theta4) + d6 * sin(theta1) * cos(theta2 + theta4);
            z = d1 + a2 * sin(theta2) + a4 * sin(theta2 + theta4) + d3 + d6 * sin
(theta2 + theta4);

            % Check if the end-effector position is within the workspace boundaries
            if x >= x_min && x <= x_max && y >= y_min && y <= y_max && z >= z_min &&
z <= z_max
                workspace_points = [workspace_points; x, y, z];  % Store valid
workspace point
            end
        end
    end
end

% Plot workspace points
scatter3(workspace_points(:,1), workspace_points(:,2), workspace_points(:,3), 'r.');
xlabel('X');
ylabel('Y');
zlabel('Z');
title('RRPRR Robot Arm Workspace');
grid on;
axis equal;
```