```
% inital settings
close all; clear; clc;

% --------------------Drawing the 6R robot in 3D (FK)↙
--------------------:

% Cylinder parameters:
h = 100; r = 50;
n = 20;

% Robot DH Parameters (as symbolic variables):
syms d1 a1 a2 a3 d4 d6

% Robot Joint Values (rad):
% Import symbolic toolbox
syms theta1 theta2 theta3 theta4 theta5 theta6

% Assign numerical values
thetas = [0, 0, 0, 0, 0, 0]; % Joint values in degrees
thetas_rad = deg2rad(thetas); % Convert to radians
d1_value = 450;
a1_value = 150;
a2_value = 600;
a3_value = 200;
d4_value = 640;
d6_value = 100;
vals = [ thetas_rad, d1_value a1_value...
    a2_value a3_value d4_value d6_value];
disp('FK vals:');
disp(vals);

% Helping dimension for moment calculations:
% l1,2,3 add to d4
% l4,5 add to d6
```

```
l1_value = d4_value/3;
l2_value = d4_value/3;
l3_value = d4_value/3;
l4_value = 0;
l5_value = d6_value;

% Robot FK:
A_1To0 = Trans_Matrix(d1, a1, theta1, +90);
A_2To1 = Trans_Matrix(0, a2, theta2 + pi/2, +0);
A_3To2 = Trans_Matrix(0, a3, theta3, +90);
A_4To3 = Trans_Matrix(d4, 0, theta4, -90);
A_5To4 = Trans_Matrix(0, 0, theta5, +90);
A_6To5 = Trans_Matrix(d6, 0, theta6, 0);

A_2To0 = A_1To0 * A_2To1;
A_3To0 = A_2To0 * A_3To2;
A_4To0 = A_3To0 * A_4To3;
A_5To0 = A_4To0 * A_5To4;
A_6To0 = A_5To0 * A_6To5;

% Convert each symbolic matrix into a MATLAB function
A_1To0_func = matlabFunction(A_1To0, 'Vars', [theta1, theta2,↙
theta3, theta4, theta5, theta6, d1, a1, a2, a3, d4, d6]);
A_2To1_func = matlabFunction(A_2To1, 'Vars', [theta1, theta2,↙
theta3, theta4, theta5, theta6, d1, a1, a2, a3, d4, d6]);
A_3To2_func = matlabFunction(A_3To2, 'Vars', [theta1, theta2,↙
theta3, theta4, theta5, theta6, d1, a1, a2, a3, d4, d6]);
A_4To3_func = matlabFunction(A_4To3, 'Vars', [theta1, theta2,↙
theta3, theta4, theta5, theta6, d1, a1, a2, a3, d4, d6]);
A_5To4_func = matlabFunction(A_5To4, 'Vars', [theta1, theta2,↙
theta3, theta4, theta5, theta6, d1, a1, a2, a3, d4, d6]);
A_6To5_func = matlabFunction(A_6To5, 'Vars', [theta1, theta2,↙
theta3, theta4, theta5, theta6, d1, a1, a2, a3, d4, d6]);
A_2To0_func = matlabFunction(A_2To0, 'Vars', [theta1, theta2,↙
```

```
theta3, theta4, theta5, theta6, d1, a1, a2, a3, d4, d6]);
A_3To0_func = matlabFunction(A_3To0, 'Vars', [theta1, theta2,↙
theta3, theta4, theta5, theta6, d1, a1, a2, a3, d4, d6]);
A_4To0_func = matlabFunction(A_4To0, 'Vars', [theta1, theta2,↙
theta3, theta4, theta5, theta6, d1, a1, a2, a3, d4, d6]);
A_5To0_func = matlabFunction(A_5To0, 'Vars', [theta1, theta2,↙
theta3, theta4, theta5, theta6, d1, a1, a2, a3, d4, d6]);
A_6To0_func = matlabFunction(A_6To0, 'Vars', [theta1, theta2,↙
theta3, theta4, theta5, theta6, d1, a1, a2, a3, d4, d6]);

% Evaluate each transformation matrix with the specific joint↙
values:
A_1To0_eval = A_1To0_func(vals(1), vals(2), vals(3), vals(4),↙
vals(5), vals(6), vals(7), vals(8), vals(9), vals(10), vals↙
(11), vals(12));
A_2To1_eval = A_2To1_func(vals(1), vals(2), vals(3), vals(4),↙
vals(5), vals(6), vals(7), vals(8), vals(9), vals(10), vals↙
(11), vals(12));
A_3To2_eval = A_3To2_func(vals(1), vals(2), vals(3), vals(4),↙
vals(5), vals(6), vals(7), vals(8), vals(9), vals(10), vals↙
(11), vals(12));
A_4To3_eval = A_4To3_func(vals(1), vals(2), vals(3), vals(4),↙
vals(5), vals(6), vals(7), vals(8), vals(9), vals(10), vals↙
(11), vals(12));
A_5To4_eval = A_5To4_func(vals(1), vals(2), vals(3), vals(4),↙
vals(5), vals(6), vals(7), vals(8), vals(9), vals(10), vals↙
(11), vals(12));
A_6To5_eval = A_6To5_func(vals(1), vals(2), vals(3), vals(4),↙
vals(5), vals(6), vals(7), vals(8), vals(9), vals(10), vals↙
(11), vals(12));


% Evaluate the compound transformation matrices with the↙
specific joint values:
```

```
A_2To0_eval = A_1To0_eval * A_2To1_eval;
A_3To0_eval = A_2To0_eval * A_3To2_eval;
A_4To0_eval = A_3To0_eval * A_4To3_eval;
A_5To0_eval = A_4To0_eval * A_5To4_eval;
A_6To0_eval = A_5To0_eval * A_6To5_eval;

% Get R_6to3:
R_6to3 = A_4To3(1:3,1:3)*A_5To4(1:3,1:3)*A_6To5(1:3,1:3);
R_6to3_func = matlabFunction(R_6to3, 'Vars', [theta1, theta2,↙
theta3, theta4, theta5, theta6, d1, a1, a2, a3, d4, d6]);
R_6to3_eval = R_6to3_func(vals(1), vals(2), vals(3), vals(4),↙
vals(5), vals(6), vals(7), vals(8), vals(9), vals(10), vals↙
(11), vals(12));

% Draw Robot:
%plot3DRobot(A_1To0_func, A_2To1_func, A_3To2_func,↙
A_4To3_func, A_5To4_func, A_6To5_func, vals, h, r, n)

% -----------------------Inverse Kinematics (IK)↙
-----------------------:

% Place of 0 state: (s4 - (790,0,1250) and s6 - (890,0,1250))↙
x = 0; y = -650; z = 450;
% 1st singularity: (d6_value,0,1500), R = [0 0 1;0 -1 0;1 0↙
0]
% and (d6_value*0.65,0,1400+d6_value*0.76), R=[-0.65 0 0.65;0↙
1 0;0.76 0 0.76]
% 2nd singularity: [-0.65 0 0.65;0 1 0;0.76 0 0.76]
x = 0; y = 0; z = 1810;%1250
Pos= [x;y;z];
R = [-1 0 0; 0 -1 0; 0 0 1];%R_6to0 = [1 0 0; 0 1 0; 0 0 1];↙
[0 0 1; 0 -1 0; 1 0 0]

% test:
```

```
t3 = atan(d4_value/a3_value);
vals = [ 0,pi/4,t3,0,t3-pi/2,0, d1_value a1_value...
    a2_value a3_value d4_value d6_value];
%disp('FK vals:');
%disp(vals);
%plot3DRobot(A_1To0_func, A_2To1_func, A_3To2_func, ↙
A_4To3_func, A_5To4_func, A_6To5_func, vals, h, r, n)

% Iterate over each angle using a for loop
d_values = [d1_value d4_value d6_value];
a_values = [a1_value a2_value a3_value];
A_6To3 = A_4To3*A_5To4*A_6To5;

% Singular 1
IK_vals_1 = CalcIK_Plot3DRobot(Pos, R, A_1To0_func, ↙
A_2To1_func, A_3To2_func, A_4To3_func, A_5To4_func, ↙
A_6To5_func,A_3To0_func, h, r, n, A_6To3, d_values, ↙
a_values);

%plot3DRobot(A_1To0_func, A_2To1_func, A_3To2_func, ↙
A_4To3_func, A_5To4_func, A_6To5_func, IK_vals_1, h, r, n)

x = 890; y = -500; z = 1250;
Pos= [x;y;z];
R = [0 0 1; 0 -1 0; 1 0 0];

% Singular 2
t3 = atan(d4_value/a3_value);
thetas_rad = [0,pi/4,t3,0,pi/2-t3,0];
vals = [ thetas_rad, d1_value a1_value...
    a2_value a3_value d4_value d6_value];
%plot3DRobot(A_1To0_func, A_2To1_func, A_3To2_func, ↙
A_4To3_func, A_5To4_func, A_6To5_func, vals, h, r, n)
```

```
x = -350; y = -400; z = 450;
Pos= [x;y;z];
R = [0 -1 0; 0 0 -1; 1 0 0];

% Singular 3
thetas_rad = [0,0,0,0,0,0];
vals = [ thetas_rad, d1_value a1_value...
    a2_value a3_value d4_value d6_value];
%plot3DRobot(A_1To0_func, A_2To1_func, A_3To2_func,
A_4To3_func, A_5To4_func, A_6To5_func, vals, h, r, n)

% -----------------------Jacobian-----------------------:
lengths = [d1_value, a1_value,...
    a2_value, a3_value, d4_value, d6_value];

% Joint 1:
JA1 = [0;0;1];
r0 = A_6To0*[0;0;0;1]; r0 = r0(1:3);
JL1 = cross(JA1,r0);
JL1 = simplify(JL1);

% Joint 2:
JA2 = A_1To0*[0;0;1;0]; JA2 = JA2(1:3);
r1 = -A_1To0*[0;0;0;1]+A_6To0*[0;0;0;1]; r1 = r1(1:3);
JL2 = cross(JA2,r1);
JL2 = simplify(JL2);

% Joint 3:
JA3 = A_2To0*[0;0;1;0]; JA3 = JA3(1:3);
r2 = -A_2To0*[0;0;0;1]+A_6To0*[0;0;0;1]; r2 = r2(1:3);
JL3 = cross(JA3,r2);
JL3 = simplify(JL3);
```

```
% Joint 4:
JA4 = A_3To0*[0;0;1;0]; JA4 = JA4(1:3);
r3 = -A_3To0*[0;0;0;1]+A_6To0*[0;0;0;1]; r3 = r3(1:3);
JL4 = cross(JA4,r3);
JL4 = simplify(JL4);

% Joint 5:
JA5 = A_4To0*[0;0;1;0]; JA5 = JA5(1:3);
r4 = -A_4To0*[0;0;0;1]+A_6To0*[0;0;0;1]; r4 = r4(1:3);
JL5 = cross(JA5,r4);
JL5 = simplify(JL5);

% Joint 6:
JA6 = A_5To0*[0;0;1;0]; JA6 = JA6(1:3);
r5 = -A_5To0*[0;0;0;1]+A_6To0*[0;0;0;1]; r5 = r5(1:3);
JL6 = cross(JA6,r5);
JL6 = simplify(JL6);

% Full Jacobian matrix:
A = [[[1;1];[1;1]] [[1;2];[2;1]]];
J = [[JL1;JA1], [JL2;JA2], [JL3;JA3], [JL4;JA4], [JL5;JA5],↙
[JL6;JA6]];
JL = [JL1, JL2, JL3, JL4, JL5, JL6];
JL_Arm = [JL1, JL2, JL3];JL_Wrist = [JL4, JL5, JL6];
JA = [JA1, JA2, JA3, JA4, JA5, JA6];

A = [JL1, JL2, JL3];
A = subs(A, d6, 0);
det_A = det(A);
det_A = simplify(det_A);
C = J(4:6,4:6);
C = subs(C, d6, 0);
det_C = det(C);
det_C = simplify(det_C);
```

```
JL_sub = double(subs(JL, [theta1, theta2, theta3, theta4,↙
theta5, theta6, d1, a1, a2, a3, d4, d6], vals));
JA_sub = double(subs(JA, [theta1, theta2, theta3, theta4,↙
theta5, theta6, d1, a1, a2, a3, d4, d6], vals));
J_sub = double(subs(J, [theta1, theta2, theta3, theta4,↙
theta5, theta6, d1, a1, a2, a3, d4, d6], vals));
[V_J,D_J] = eig(J_sub.');

disp('J det:');
det(J_sub)
disp('JL det:');
det(JL_sub*JL_sub.')



% -----------------------Trajectory↙
Planning-----------------------:
% Positions of A,B and C:
P0 = [890;0;1250];
R0 = [0 0 1;0 -1 0;1 0 0];
P1 = [500;-400;450];
R1 = [0 -1 0;0 0 -1;1 0 0];
P2 = [-500;-400;450];
distance = sqrt(sum((P0 - P1).^2));
%Speed:
V = 1000;% 1[m/s]->1000[mm/s]
% Scanning sub-positions: 350->...->-350
SP = [300, -300];
% Time intervals:
dt = 0.01;%0.01 is needed at the end
A_B_time = 2.2;C_A_time = 2.65;
Acc_time = 0.5;
T1 = A_B_time;
```

```
T2 = T1+Acc_time;
T3 = T2+(SP(1)-SP(2))/V;
T4 = T3+Acc_time;
% Important time points to mark
t_markers = [0, T1, T2, T3, T4];
% 1,2,3,4,5 -> 1,1.25,1.75,2,3 (0.5 acceleration)
% Physical knowns:
% Cammera weight in [N]
W_cam = 0.7*9.8;
% Link weight in [N]
W = 4*9.8;


% ---------Trajectory from position A to B---------:
[t_AB, lambda_AB,lambda_AB_sym] = quinticBC(0, T1, 0, 1, 0,↙
0, 0, 0, dt);
% Compute the trajectory for each time step
P_t_AB_sym = P0 + lambda_AB_sym .* (P1 - P0);
P_t_AB = P0 + lambda_AB .* (P1 - P0);

% Call the continuous rotation matrix function
R_AB_t_sym = symbolicRotationMatrix(R0, R1, 0, T1);

% Plot the 3D trajectory
%{
figure;
plot3(P_t_AB(1, :), P_t_AB(2, :), P_t_AB(3, :), 'LineWidth',↙
2);
title('3D Trajectory');
xlabel('X');
ylabel('Y');
zlabel('Z');
grid on;
%}
```

```
% ---------Trajectory from position B to C---------:
y_BC = P1(2);z_BC = P1(3);
[t_BC_1, x_t_BC_1,x_t_BC_1_sym] = quinticBC(T1, T2, P1(1), SP↙
(1), 0, -V, 0, 0, dt);
[v_1_vals,a_1_vals] = plot_velocity(t_BC_1, x_t_BC_1_sym);

syms t
t_BC_2 = T2:dt:T3;
x_t_BC_2_sym = (SP(1)+V*T2) - t.*V;
x_t_BC_2 = (SP(1)+V*T2) - t_BC_2.*V;

[t_BC_3, x_t_BC_3,x_t_BC_3_sym] = quinticBC(T3, T4, SP(2), P2↙
(1), -V, 0, 0, 0, dt);
[v_3_vals,a_3_vals] = plot_velocity(t_BC_3, x_t_BC_3_sym);

t_BC_tot = [t_BC_1, t_BC_2, t_BC_3];
x_t_BC_tot = [x_t_BC_1, x_t_BC_2, x_t_BC_3];
x_t_BC_tot_sym = piecewise(T1<=t<=T2, x_t_BC_1_sym,...
    T2<=t<=T3, x_t_BC_2_sym,T3<=t<=T4, x_t_BC_3_sym);
P_t_BC_tot_sym = [x_t_BC_tot_sym;y_BC;z_BC];

% ---------Total Trajectory from position A to B to C to↙
A---------:
P_t_tot_sym = piecewise(0<=t<=T1, P_t_AB_sym,...
    T1<=t<=T4, P_t_BC_tot_sym);
R_t_tot_sym = piecewise(0<=t<=T1, R_AB_t_sym,...
    T1<=t<=T4, R1);

% Define time parameters
t_vals = 0:dt:T4; % Time vector

%{
% Plot rotation matrix in time:
```

```
% Initialize the video writer
video_name = 'rotation_matrix_animation.avi';
v = VideoWriter(video_name);
v.FrameRate = 10; % Adjust this to control video speed
open(v);

% Set up the figure
figure;
axis equal;
grid on;
xlabel('X-axis');
ylabel('Y-axis');
zlabel('Z-axis');
title('Evolution of Rotation Matrix Basis Vectors Over↙
Time');

% Loop through each time step, evaluate and plot
for i = 1:length(t_vals)
    % Evaluate the symbolic rotation matrix at the current↙
time t
    R = double(subs(R_t_tot_sym, t, t_vals(i)));

    % Origin point (for all vectors)
    origin = [0, 0, 0];

    % Extract basis vectors (columns of the rotation matrix)
    vec_x = R(:, 1); % x-direction (first column)
    vec_y = R(:, 2); % y-direction (second column)
    vec_z = R(:, 3); % z-direction (third column)

    % Clear the figure for the new frame
    clf;

    % Plot the basis vectors at the origin as arrows
```

```
    quiver3(origin(1), origin(2), origin(3), vec_x(1), vec_x↙
(2), vec_x(3), 'r', 'LineWidth', 2, 'MaxHeadSize', 0.5);
    hold on;
    quiver3(origin(1), origin(2), origin(3), vec_y(1), vec_y↙
(2), vec_y(3), 'g', 'LineWidth', 2, 'MaxHeadSize', 0.5);
    quiver3(origin(1), origin(2), origin(3), vec_z(1), vec_z↙
(2), vec_z(3), 'b', 'LineWidth', 2, 'MaxHeadSize', 0.5);

    % Set axis limits (adjust based on your rotation scale)
    axis([-1 1 -1 1 -1 1]*1.5); % Adjust limits for better↙
visualization

    % Add time annotation to the plot
    time_str = sprintf('Time: %.2f s', t_vals(i));
    text(0.6, 0.6, 0.6, time_str, 'FontSize', 14, 'Color',↙
'k', 'FontWeight', 'bold');

    % Capture the current frame
    frame = getframe(gcf);

    % Write the frame to the video
    writeVideo(v, frame);

    % Pause for animation effect (optional for live display,↙
but ignored for video)
    pause(0.1);
end

% Close the video file
close(v);

% Notify user that the video has been saved
disp(['Video saved as ', video_name]);
```

```
%}

% Find the index of the closest time value in t_vals to the
target time
target_t = (T2+T3)/2;
[~, idx] = min(abs(t_vals - target_t));

% Evaluate piecewise function
P_t_vals = arrayfun(@(t) double(subs(P_t_tot_sym, t)),
t_vals, 'UniformOutput', false);
P_t_vals = cell2mat(P_t_vals); % Convert to matrix

% Extract components
P1 = P_t_vals(1, :);
P2 = P_t_vals(2, :);
P3 = P_t_vals(3, :);

% Plotting
%{
figure(999);
plot3(P1, P2, P3, 'LineWidth', 2);
xlabel('P1');
ylabel('P2');
zlabel('P3');
title('3D Plot of entire trajectory P(t)');
grid on;
%}

% Initialize arrays for trajectories
x_rel = zeros(size(t_vals));
y_rel = zeros(size(t_vals));
z_rel = zeros(size(t_vals));
x_abs = zeros(size(t_vals));
y_abs = zeros(size(t_vals));
```

```
z_abs = zeros(size(t_vals));

% Generate trajectories
for i = 1:length(t_vals)
    current_t = t_vals(i);

    % Get relative position from symbolic expression
    P_current = double(subs(P_t_tot_sym, 't', current_t));

    % Extract x, y, z components
    x_rel(i) = P_current(1);
    y_rel(i) = P_current(2);
    z_rel(i) = P_current(3);

    % Calculate absolute trajectory by adding platform motion
    x_abs(i) = x_rel(i) + V * current_t;
    y_abs(i) = y_rel(i);
    z_abs(i) = z_rel(i);
end

% ---------Turning Trajectory to Robot values (in discrete↙
time)---------:
% Define time parameters
% Time vector as before: t_vals = 0:dt:T5;
% robot values:
q_t_tot_vals = zeros(6,length(t_vals)); % Adjust based on↙
your robot configuration

% Define box parameters (Barcode)
box_length = 150;  % in x-direction
box_width = 3;   % in y-direction
box_height = 100;  % in z-direction
box_initial_position = [3000, -650, 450];  % Initial center↙
position
```

```
% Loop through each time step for getting robot joint values
for i = 1:length(t_vals)

    curr_t = t_vals(i);

    required_position = double(subs(P_t_tot_sym, curr_t)); %↙
Get desired position from P(t)
    required_rotation = double(subs(R_t_tot_sym, curr_t)); %↙
Get desired position from P(t)

    % Call your IK function to get robot values for the↙
desired position
    val_i = CalcIK_Plot3DRobot(required_position,↙
required_rotation, ...
                                             A_1To0_func,↙
A_2To1_func, ...
                                             A_3To2_func,↙
A_4To3_func, ...
                                             A_5To4_func,↙
A_6To5_func, ...
                                             A_3To0_func, h,↙
r, n, ...
                                             A_6To3,↙
d_values, a_values);
    q_t_tot_vals(:, i) = val_i(1:6).';

end

% fix beginning, end and discontinous jumps in angles:
q_t_tot_vals(4, 1) = q_t_tot_vals(4, 2);
q_t_tot_vals(6, 1) = q_t_tot_vals(6, 2);
q_t_tot_vals(4, idx+1:end)  = q_t_tot_vals(4, idx+1:end) -↙
2*pi;
```

```
q_t_tot_vals(6, idx+1:end)  = q_t_tot_vals(6, idx+1:end) +↙
2*pi;
q_t_tot_vals(4, end) = q_t_tot_vals(4, end-1);
q_t_tot_vals(6, end) = q_t_tot_vals(6, end-1);

% Get current joint velocities (derivatives of joint↙
positions w.r.t. time)
% Define q_dot with zeros for the velocity matrix (6 x↙
[number of time points])
NoTime_Points = length(t_vals);
q_dot = zeros(6, NoTime_Points);

% Central difference for the interior points
for i = 2:NoTime_Points-1
    q_dot(:, i) = (q_t_tot_vals(:, i+1) - q_t_tot_vals(:, i-↙
1)) / (2 * dt);
end

% Forward difference for the first column
q_dot(:, 1) = (q_t_tot_vals(:, 2) - q_t_tot_vals(:, 1)) / dt;

% Backward difference for the last column
q_dot(:, end) = (q_t_tot_vals(:, end) - q_t_tot_vals(:, end-↙
1)) / dt;

% Initialize q_ddot with the same dimensions as q_t_tot_vals
q_ddot = zeros(6, NoTime_Points);

% Central difference for the interior points
for i = 2:NoTime_Points-1
    q_ddot(:, i) = (q_dot(:, i+1) - q_dot(:, i-1)) / (2 *↙
dt);
end
```

```
% Forward difference for the first column
q_ddot(:, 1) = (q_dot(:, 2) - q_dot(:, 1)) / dt;

% Backward difference for the last column
q_ddot(:, end) = (q_dot(:, end) - q_dot(:, end-1)) / dt;


gripper_velocity = zeros(3,NoTime_Points);

J_sub_det_t = zeros(NoTime_Points);

% Torques calculating:
Motor_Moments = zeros(6,NoTime_Points);

% Loop through each time step for printing robot
for i = 1:length(t_vals)

    t = t_vals(i);

    % Plot the robot:
    vals = [q_t_tot_vals(:,i).', d1_value a1_value...
    a2_value a3_value d4_value d6_value];
    plot3DRobot(A_1To0_func, A_2To1_func, A_3To2_func,↙
A_4To3_func, A_5To4_func, A_6To5_func, vals, h, r, n)


    % Calculate new box position
    box_position_x = box_initial_position(1) - V * t;  %↙
Moving in the -x direction
    box_position = [box_position_x, box_initial_position(2),↙
box_initial_position(3)];

    % Plot the box (using patch for visualization)
    vertices = [box_position(1)-box_length/2, box_position(2)↙
```

```
-box_width/2, box_position(3)-box_height/2;
                box_position(1)-box_length/2, box_position(2)↙
+box_width/2, box_position(3)-box_height/2;
                box_position(1)+box_length/2, box_position(2)↙
+box_width/2, box_position(3)-box_height/2;
                box_position(1)+box_length/2, box_position(2)↙
-box_width/2, box_position(3)-box_height/2;
                box_position(1)-box_length/2, box_position(2)↙
-box_width/2, box_position(3)+box_height/2;
                box_position(1)-box_length/2, box_position(2)↙
+box_width/2, box_position(3)+box_height/2;
                box_position(1)+box_length/2, box_position(2)↙
+box_width/2, box_position(3)+box_height/2;
                box_position(1)+box_length/2, box_position(2)↙
-box_width/2, box_position(3)+box_height/2];

    faces = [1 2 3 4; 5 6 7 8; 1 2 6 5; 2 3 7 6; 3 4 8 7; 4 1↙
5 8];

    patch('Vertices', vertices, 'Faces', faces, 'FaceColor',↙
'blue', 'FaceAlpha', 0.3);

    % --- Calculate the gripper velocity vector ---

    % Calculate the linear Jacobian J_L at this time instance
    JL_sub = double(subs(JL, [theta1, theta2, theta3, theta4,↙
theta5, theta6, d1, a1, a2, a3, d4, d6], vals));  %↙
Substitute current joint values

    % Calculate the Jacobian J at this time instance
    J_sub = double(subs(J, [theta1, theta2, theta3, theta4,↙
theta5, theta6, d1, a1, a2, a3, d4, d6], vals));  %↙
Substitute current joint values
    J_sub_det_t(i) = det(J_sub);
```

```
    % Calculate the end-effector linear velocity in 3D space
    gripper_velocity(:,i) = JL_sub * q_dot(:,i);  % Velocity↙
of the end-effector (3x1 vector)

    % --- Find Torque at this time (depends on joint values↙
only) ---
    % Moments against camera:
    Tou_cam = J_sub.'*[0;0;-W_cam;0;0;0];

    % Moments against link 6-gripper:
    d6_now = l5_value/2;
    vals = [q_t_tot_vals(:,i).', d1_value a1_value...
    a2_value a3_value d4_value d6_now];
    J1_sub = double(subs(J, [theta1, theta2, theta3, theta4,↙
theta5, theta6, d1, a1, a2, a3, d4, d6], vals));  %↙
Substitute current joint values
    Tou_link6 = (J1_sub.')*[0;0;-W;0;0;0];

    % Moments against link 5-6:
    Tou_link5 = zeros(6,1);

    % Moments against link 4-5:
    d4_now = l1_value + l2_value + l3_value/2;
    vals = [q_t_tot_vals(:,i).', d1_value a1_value...
    a2_value a3_value d4_now 0];
    J3_sub = double(subs(J, [theta1, theta2, theta3, theta4,↙
theta5, theta6, d1, a1, a2, a3, d4, d6], vals));  %↙
Substitute current joint values
    J3_sub = J3_sub(1:6,1:4);
    Tou_link4 = zeros(6,1);
    Tou_link4(1:4,1) = (J3_sub.')*[0;0;-W;0;0;0];

    % Moments against link 3-4:
```

```
    a3_now = a3_value/2;
    d4_now = (l1_value + l2_value)/2;
    vals = [q_t_tot_vals(:,i).', d1_value a1_value...
    a2_value a3_now d4_now 0];
    J4_sub = double(subs(J, [theta1, theta2, theta3, theta4,↙
theta5, theta6, d1, a1, a2, a3, d4, d6], vals));  %↙
Substitute current joint values
    J4_sub = J4_sub(1:6,1:3);
    Tou_link3 = zeros(6,1);
    Tou_link3(1:3,1) = (J4_sub.')*[0;0;-W;0;0;0];

    % Moments against link 2-3:
    a2_now = a2_value/2;
    vals = [q_t_tot_vals(:,i).', d1_value a1_value...
    a2_now 0 0 0];
    J5_sub = double(subs(J, [theta1, theta2, theta3, theta4,↙
theta5, theta6, d1, a1, a2, a3, d4, d6], vals));  %↙
Substitute current joint values
    J5_sub = J5_sub(1:6,1:2);
    Tou_link2 = zeros(6,1);
    Tou_link2(1:2,1) = (J5_sub.')*[0;0;-W;0;0;0];

    Motor_Moments(:,i) = Tou_cam + Tou_link6 + Tou_link5 +...
        Tou_link4 + Tou_link3 + Tou_link2;

    % --- Draw the gripper velocity vector ---

    % Reset Vals:
    vals = [q_t_tot_vals(:,i).', d1_value a1_value...
    a2_value a3_value d4_value d6_value];

    % Get FK:
    A_6To0_sub = double(subs(A_6To0, [theta1, theta2, theta3,↙
theta4, theta5, theta6, d1, a1, a2, a3, d4, d6], vals));  %↙
```

Substitute current joint values

```
    % Get the current position of the gripper (end-effector)
    gripper_position = A_6To0_sub(1:3,4);

    smf = 1/2;
    % Plot the gripper velocity vector as an arrow
    quiver3(gripper_position(1), gripper_position(2),↙
gripper_position(3), ...
        gripper_velocity(1,i)*smf, gripper_velocity(2,i)*smf,↙
gripper_velocity(3,i)*smf, ...
        'r', 'LineWidth', 2, 'MaxHeadSize', 0.5);  % Red↙
arrow for velocity

    plot3(x_rel, y_rel, z_rel, 'b-', 'LineWidth', 2);
    grid on;

    % Add markers for important time points
    for q = 1:length(t_markers)
        t_idx = find(abs(t_vals - t_markers(q)) < dt/2, 1);
        if ~isempty(t_idx)
            plot3(x_rel(t_idx), y_rel(t_idx), z_rel(t_idx),↙
'ro', 'MarkerSize', 5);
            text(x_rel(t_idx), y_rel(t_idx), z_rel(t_idx),↙
...
                sprintf('t=%.2fs', t_markers(q)), ...
                'VerticalAlignment', 'bottom');
        end
    end


    % Update the plot
    drawnow;
end
```

```
% find velocity size w.r.t time:
Velocity_Size = sqrt(gripper_velocity(1,:).^2 +↙
gripper_velocity(2,:).^2 + gripper_velocity(3,:).^2);

% Initialize gripper_acceleration with the same dimensions as↙
gripper_velocity
gripper_acceleration = zeros(3, NoTime_Points);

% Central difference for the interior points
for i = 2:NoTime_Points-1
    gripper_acceleration(:, i) = (gripper_velocity(:, i+1) -↙
gripper_velocity(:, i-1)) / (2 * dt);
end

% Forward difference for the first column
gripper_acceleration(:, 1) = (gripper_velocity(:, 2) -↙
gripper_velocity(:, 1)) / dt;

% Backward difference for the last column
gripper_acceleration(:, end) = (gripper_velocity(:, end) -↙
gripper_velocity(:, end-1)) / dt;

% Calculate the magnitude of the acceleration at each time↙
point
Acceleration_Size = sqrt(gripper_acceleration(1,:).^2 + ...
                         gripper_acceleration(2,:).^2 + ...
                         gripper_acceleration(3,:).^2);



% ---- Create Video -----:
numFrames = (T4/dt+1); % Total number of frames

filePrefix = 'figure'; % Assuming saved figures are named↙
```

```
like 'figure1.png'


% Create a VideoWriter object
videoFile = 'my_video_rel.avi'; % Video filename
v = VideoWriter(videoFile);
% Set the frame rate - Frames = NoF/T4, NoF = T4/dt+1
v.FrameRate = floor( numFrames/T4 );
open(v); % Open the video file

% Loop through the existing figure windows in reverse order
for i = 1:1:numFrames
    fig = figure(i);  % Access the i-th figure (pre-existing)

    % Temporarily hide the figure while processing
    set(fig, 'Visible', 'off');

    % Make the figure full-screen
    set(gcf, 'Position', get(0, 'Screensize'));  % Set figure↙
to full screen

    % Display time on the figure
    current_time = t_vals(i);
    time_text = sprintf('Time = %.2f [s]', current_time);  %↙
Create the time string
    annotation('textbox', [0.1, 0.85, 0.2, 0.1], 'String',↙
time_text, 'FontSize', 14, ...
               'Color', 'black', 'EdgeColor', 'none',↙
'BackgroundColor', 'white');  % Position and style


    % Capture the plot as an image
    frame = getframe(gcf);  % Get the frame from the current↙
figure
```

```
    writeVideo(v, frame);   % Write the frame to the video
end

% Close the video file
close(v);
disp(['Video saved as ' videoFile]);


% ---- take specific figures of relative motion -----:
for i = 1:1:numFrames
    if ismember(t_vals(i),t_markers)
        fig = figure(i);  % Access the i-th figure (pre-↙
existing)

        % Temporarily hide the figure while processing
        set(fig, 'Visible', 'on');

        % Make the figure full-screen
        set(gcf, 'Position', get(0, 'Screensize'));  % Set↙
figure to full screen

        % Display time on the figure
        current_time = t_vals(i);
        time_text = sprintf('Time = %.2f [s]', current_time);↙
% Create the time string
        annotation('textbox', [0.1, 0.85, 0.2, 0.1],↙
'String', time_text, 'FontSize', 14, ...
               'Color', 'black', 'EdgeColor', 'none',↙
'BackgroundColor', 'white');  % Position and style

        %New title:
        title(sprintf('Manipulator Position at t = %.2f s (%s↙
view)', t_vals(i)));
```

```
    end
end

% ---- figures of relative-to-ground motion and taking↙
markers figures -----:
close all;  % Close all open figures

gripper_velocity_abs = zeros(3,NoTime_Points);

% Loop through each time step for printing robot
for i = 1:length(t_vals)

    t = t_vals(i);

    % Plot the robot:
    vals = [q_t_tot_vals(:,i).', d1_value a1_value...
    a2_value a3_value d4_value d6_value];
    plot3DRobotAbs(A_1To0_func, A_2To1_func, A_3To2_func,↙
A_4To3_func, A_5To4_func, A_6To5_func, vals, h, r, n,V,t)


    % Calculate new box position
    box_position_x = box_initial_position(1);  % Moving in↙
the -x direction
    box_position = [box_position_x, box_initial_position(2),↙
box_initial_position(3)];

    % Plot the box (using patch for visualization)
    vertices = [box_position(1)-box_length/2, box_position(2)↙
-box_width/2, box_position(3)-box_height/2;
                box_position(1)-box_length/2, box_position(2)↙
+box_width/2, box_position(3)-box_height/2;
                box_position(1)+box_length/2, box_position(2)↙
+box_width/2, box_position(3)-box_height/2;
```

```
                box_position(1)+box_length/2, box_position(2)↙
-box_width/2, box_position(3)-box_height/2;
                box_position(1)-box_length/2, box_position(2)↙
-box_width/2, box_position(3)+box_height/2;
                box_position(1)-box_length/2, box_position(2)↙
+box_width/2, box_position(3)+box_height/2;
                box_position(1)+box_length/2, box_position(2)↙
+box_width/2, box_position(3)+box_height/2;
                box_position(1)+box_length/2, box_position(2)↙
-box_width/2, box_position(3)+box_height/2];

    faces = [1 2 3 4; 5 6 7 8; 1 2 6 5; 2 3 7 6; 3 4 8 7; 4 1↙
5 8];

    patch('Vertices', vertices, 'Faces', faces, 'FaceColor',↙
'blue', 'FaceAlpha', 0.3);

    % --- Calculate the gripper velocity vector ---

    % Calculate the linear Jacobian J_L at this time instance
    JL_sub = double(subs(JL, [theta1, theta2, theta3, theta4,↙
theta5, theta6, d1, a1, a2, a3, d4, d6], vals));  %↙
Substitute current joint values

    % Calculate the Jacobian J at this time instance
    J_sub = double(subs(J, [theta1, theta2, theta3, theta4,↙
theta5, theta6, d1, a1, a2, a3, d4, d6], vals));  %↙
Substitute current joint values
    J_sub_det_t(i) = det(J_sub);

    % Calculate the end-effector linear velocity in 3D space
    gripper_velocity_abs(:,i) = JL_sub * q_dot(:,i) + [V;0;↙
0];  % Velocity of the end-effector (3x1 vector)
```

```
    % --- Draw the gripper velocity vector ---

    % Get FK:
    A_6To0_sub = double(subs(A_6To0, [theta1, theta2, theta3,↙
theta4, theta5, theta6, d1, a1, a2, a3, d4, d6], vals));  %↙
Substitute current joint values

    % Get the current position of the gripper (end-effector)
    gripper_position = A_6To0_sub(1:3,4);

    smf = 1/2;
    % Plot the gripper velocity vector as an arrow
    quiver3(gripper_position(1)+V*t, gripper_position(2),↙
gripper_position(3), ...
        gripper_velocity_abs(1,i)*smf, gripper_velocity_abs↙
(2,i)*smf, gripper_velocity_abs(3,i)*smf, ...
        'r', 'LineWidth', 2, 'MaxHeadSize', 0.5);  % Red↙
arrow for velocity

    plot3(x_abs, y_abs, z_abs, 'b-', 'LineWidth', 2);
    grid on;

    % Add markers for important time points
    for q = 1:length(t_markers)
        t_idx = find(abs(t_vals - t_markers(q)) < dt/2, 1);
        if ~isempty(t_idx)
            plot3(x_abs(t_idx), y_abs(t_idx), z_abs(t_idx),↙
'ro', 'MarkerSize', 5);
            text(x_abs(t_idx), y_abs(t_idx), z_abs(t_idx),↙
...
                sprintf('t=%.2fs', t_markers(q)), ...
                'VerticalAlignment', 'bottom');
        end
    end
```

```
    % Update the plot
    drawnow;
end

% ---- Create Video for absolute path-----:

filePrefix = 'figure'; % Assuming saved figures are named↙
like 'figure1.png'


% Create a VideoWriter object
videoFile = 'my_video_abs.avi'; % Video filename
v = VideoWriter(videoFile);
% Set the frame rate - Frames = NoF/T4, NoF = T4/dt+1
v.FrameRate = floor( numFrames/T4 );
open(v); % Open the video file

% Loop through the existing figure windows
for i = 1:1:numFrames
    fig = figure(i);  % Access the i-th figure (pre-existing)

    % Temporarily hide the figure while processing
    set(fig, 'Visible', 'off');

    % Make the figure full-screen
    set(gcf, 'Position', get(0, 'Screensize'));  % Set figure↙
to full screen

    % Display time on the figure
    current_time = t_vals(i);
    time_text = sprintf('Time = %.2f [s]', current_time);  %↙
Create the time string
```

```
    annotation('textbox', [0.1, 0.85, 0.2, 0.1], 'String',↙
time_text, 'FontSize', 14, ...
               'Color', 'black', 'EdgeColor', 'none',↙
'BackgroundColor', 'white');  % Position and style


    % Capture the plot as an image
    frame = getframe(gcf);  % Get the frame from the current↙
figure
    writeVideo(v, frame);   % Write the frame to the video
end

% Close the video file
close(v);
disp(['Video saved as ' videoFile]);



% ---- take specific figures of relative motion -----:
for i = 1:1:numFrames
    if ismember(t_vals(i),t_markers)
        fig = figure(i);  % Access the i-th figure (pre-↙
existing)

        % Temporarily hide the figure while processing
        set(fig, 'Visible', 'on');

        % Make the figure full-screen
        set(gcf, 'Position', get(0, 'Screensize'));  % Set↙
figure to full screen

        % Display time on the figure
        current_time = t_vals(i);
        time_text = sprintf('Time = %.2f [s]', current_time);↙
```

```
% Create the time string
        annotation('textbox', [0.1, 0.85, 0.2, 0.1],↙
'String', time_text, 'FontSize', 14, ...
                'Color', 'black', 'EdgeColor', 'none',↙
'BackgroundColor', 'white');  % Position and style

        %New title:
        title(sprintf('Manipulator Position at t = %.2f s',↙
t_vals(i)));

    end
end


% -----------------Plot the result-----------------

figure;
plot(t_BC_tot, x_t_BC_tot);
title('B to C totall x(t)');
xlabel('Time');
ylabel('Position');
grid on;

% Create figure for joint positions
figure('Name', 'Joint Positions', 'NumberTitle', 'off');

% Plot joint values
figure;
hold on;
for i = 1:6
    plot(t_vals, q_t_tot_vals(i, :), 'DisplayName', ['Joint↙
', num2str(i)]);
end
hold off;
```

```
% Add labels and title
xlabel('Time (s)');
ylabel('Joint values (rad)');
title('Robot Joint Values Over Time');
legend show;
grid on;

% Plot joint speeds
figure;
hold on;
for i = 1:6
    plot(t_vals, q_dot(i, :), 'DisplayName', ['Joint ',↙
num2str(i)]);
end
hold off;

% Add labels and title
xlabel('Time (s)');
ylabel('Joint Speeds (rad/s)');
title('Robot Joint Speeds Over Time');
legend show;
grid on;

% Plot joint Accelerations
figure;
hold on;
for i = 1:6
    plot(t_vals, q_ddot(i, :), 'DisplayName', ['Joint ',↙
num2str(i)]);
end
hold off;

% Add labels and title
```

```
xlabel('Time (s)');
ylabel('Joint accelerations (rad/s^2)');
title('Robot Joint Accelerations Over Time');
legend show;
grid on;

% Plot joint Moments
figure;
hold on;
for i = 1:6
    plot(t_vals, Motor_Moments(i, :), 'DisplayName', ['Joint↙
', num2str(i)]);
end
hold off;

% Add labels and title
xlabel('Time (s)');
ylabel('Moments (Nmm)');
title('Robot Joint Moments Over Time');
legend show;
grid on;

% Plot the position (relative to ground) components:
% Plot rx, ry, and rz in separate subplots
figure;  % Create a new figure

% Plot rx (gripper_velocity(1,:))
subplot(3, 1, 1);  % 3 rows, 1 column, first plot
plot(t_vals, x_abs, 'r', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('r_x (mm)');
title('Velocity in X direction');
grid on;  % Add grid to the plot
```

```
% Plot ry (gripper_velocity(2,:))
subplot(3, 1, 2);  % 3 rows, 1 column, second plot
plot(t_vals, y_abs, 'g', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('r_y (mm)');
title('Velocity in Y direction');
grid on;  % Add grid to the plot

% Plot rz (gripper_velocity(3,:))
subplot(3, 1, 3);  % 3 rows, 1 column, third plot
plot(t_vals, z_abs, 'b', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('r_z (mm)');
title('Velocity in Z direction');
grid on;  % Add grid to the plot

% Adjust the layout
sgtitle('Gripper Displacement (r_x, r_y, r_z)');  % Super↙
title for the figure

% Plot the position (relative to robot base) components:
% Plot rx, ry, and rz in separate subplots
figure;  % Create a new figure

% Plot rx (gripper_velocity(1,:))
subplot(3, 1, 1);  % 3 rows, 1 column, first plot
plot(t_vals, x_rel, 'r', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('r_x (mm)');
title('Velocity in X direction');
grid on;  % Add grid to the plot

% Plot ry (gripper_velocity(2,:))
subplot(3, 1, 2);  % 3 rows, 1 column, second plot
```

```
plot(t_vals, y_rel, 'g', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('r_y (mm)');
title('Velocity in Y direction');
grid on;  % Add grid to the plot

% Plot rz (gripper_velocity(3,:))
subplot(3, 1, 3);  % 3 rows, 1 column, third plot
plot(t_vals, z_rel, 'b', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('r_z (mm)');
title('Velocity in Z direction');
grid on;  % Add grid to the plot

% Adjust the layout
sgtitle('Gripper Displacement (relative to robot base) (r_x,↙
r_y, r_z)');  % Super title for the figure

% Plot speed vs time:
figure();plot(t_vals,Velocity_Size(:));
% Add labels and title
xlabel('Time (s)');
ylabel('Speed of gripper (mm/s)');
title('Robot gripper Speeds Over Time');
grid on;

% Plot the speeds components:
% Plot vx, vy, and vz in separate subplots
figure;  % Create a new figure

% Plot vx (gripper_velocity(1,:))
subplot(3, 1, 1);  % 3 rows, 1 column, first plot
plot(t_vals, gripper_velocity(1,:), 'r', 'LineWidth', 2);
xlabel('Time (s)');
```

```
ylabel('v_x (mm/s)');
title('Velocity in X direction');
grid on;  % Add grid to the plot

% Plot vy (gripper_velocity(2,:))
subplot(3, 1, 2);  % 3 rows, 1 column, second plot
plot(t_vals, gripper_velocity(2,:), 'g', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('v_y (mm/s)');
title('Velocity in Y direction');
grid on;  % Add grid to the plot

% Plot vz (gripper_velocity(3,:))
subplot(3, 1, 3);  % 3 rows, 1 column, third plot
plot(t_vals, gripper_velocity(3,:), 'b', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('v_z (mm/s)');
title('Velocity in Z direction');
grid on;  % Add grid to the plot

% Adjust the layout
sgtitle('Gripper Velocities (v_x, v_y, v_z)');  % Super title↙
for the figure

% Plot speed vs time:
figure();plot(t_vals,Acceleration_Size(:));
% Add labels and title
xlabel('Time (s)');
ylabel('Acceleration of gripper (mm/s^2)');
title('Robot gripper Speeds Over Time');
grid on;

% Plot the speeds components:
% Plot ax, ay, and az in separate subplots
```

```
figure;  % Create a new figure

% Plot vx (gripper_velocity(1,:))
subplot(3, 1, 1);  % 3 rows, 1 column, first plot
plot(t_vals, gripper_acceleration(1,:), 'r', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('a_x (mm/s^2)');
title('Acceleration in X direction');
grid on;  % Add grid to the plot

% Plot vy (gripper_velocity(2,:))
subplot(3, 1, 2);  % 3 rows, 1 column, second plot
plot(t_vals, gripper_acceleration(2,:), 'g', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('a_y (mm/s^2)');
title('Acceleration in Y direction');
grid on;  % Add grid to the plot

% Plot vz (gripper_velocity(3,:))
subplot(3, 1, 3);  % 3 rows, 1 column, third plot
plot(t_vals, gripper_acceleration(3,:), 'b', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('a_z (mm/s^2)');
title('Acceleration in Z direction');
grid on;  % Add grid to the plot

% Adjust the layout
sgtitle('Gripper Acceleration (a_x, a_y, a_z)');  % Super↙
title for the figure


% -------------FUNCTIONS-------------:
% Find A matrix:
function A = Trans_Matrix(d,a,theta,alpha)
```

```
    Ct = cos(theta); St =sin(theta);
    Ca = cos(alpha); Sa =sin(alpha);
    % Check conditions
    if alpha == 0
        Ca = 1; Sa = 0;
    elseif alpha == 90
        Ca = 0; Sa = 1;
    elseif alpha == -90
        Ca = 0; Sa = -1;
    else
        disp('None of the specified conditions are met.');
    end
    A = [Ct -St*Ca St*Sa a*Ct; St Ct*Ca -Ct*Sa a*St; 0 Sa Ca↙
d; 0 0 0 1];
end

% Plot Cylinder: pos (x,y,z), R - Rotation Matrix, h=height↙
of cylinder,
% r=radius, n=nodes
function plotCylinder(pos,R,h,r,n)

    [X,Y,Z] = cylinder(r,n);
    Z = Z*h;
    Z = Z-h/2;

    % Rotation:
    for k=1:2
        for l=1:(n+1)
            Pos_Rot = R*[X(k,l);Y(k,l);Z(k,l)];
            X(k,l)=Pos_Rot(1);Y(k,l)=Pos_Rot(2);Z(k,l)↙
=Pos_Rot(3);
        end
    end
```

```
    % Translation:
    X = X + pos(1); Y = Y + pos(2); Z = Z + pos(3);
    save('XYZVALID.mat','X','Y','Z')

    surf(X,Y,Z)
    hold on

end



function plotBox(pos, a, b, c, R)
    % pos: [x, y, z] is the position to translate the box
    % a, b, c: are the dimensions of the box
    % R - Rotation Matrix

    % Define the vertices of the box (centered at (0,0,0))
    vertices = [
        -a/2, -b/2, 0;  % Bottom-left-back
        a/2, -b/2, 0;   % Bottom-right-back
        a/2, b/2, 0;    % Bottom-right-front
        -a/2, b/2, 0;   % Bottom-left-front
        -a/2, -b/2, c;  % Top-left-back
        a/2, -b/2, c;   % Top-right-back
        a/2, b/2, c;    % Top-right-front
        -a/2, b/2, c    % Top-left-front
    ];

    % Apply rotation to each vertex
    rotated_vertices = (R * vertices')'; % Rotate vertices

    % Check sizes before translation
    disp('Size of rotated_vertices:');
    disp(size(rotated_vertices)); % Should be Nx3
```

```
    % Ensure pos is a row vector of size 1x3
    if size(pos, 1) ~= 1 || size(pos, 2) ~= 3
        error('Position vector "pos" must be a 1x3 vector.');
    end

    % Translate the box to the desired position
    rotated_vertices = rotated_vertices + pos; % Add position↙
vector directly

    % Define the faces of the box using vertex indices
    faces = [
        1, 2, 3, 4;   % Bottom face
        5, 6, 7, 8;   % Top face
        1, 2, 6, 5;   % Left face
        2, 3, 7, 6;   % Back face
        3, 4, 8, 7;   % Right face
        4, 1, 5, 8    % Front face
    ];

    % Plot the box using patch for better control
    hold on;
    for i = 1:size(faces, 1)
        % Get the vertices for the current face
        v = rotated_vertices(faces(i, :), :);
        patch(v(:, 1), v(:, 2), v(:, 3), 'cyan', 'FaceAlpha',↙
0.5, 'EdgeColor', 'k');
    end
    hold on;
end

function drawSphere(pos, radius)
    % drawSphere - Draws a sphere at the specified position.
    %
    % Syntax: drawSphere(pos, radius)
```

```
    %
    % Inputs:
    %    pos    - A 1x3 vector specifying the position [x, y,↙
z].
    %    radius - The radius of the sphere.

     % Create the sphere data
    [X, Y, Z] = sphere(30); % Create a sphere with 30x30 grid

    % Scale the sphere to the desired radius
    X = X * radius;
    Y = Y * radius;
    Z = Z * radius;

    % Translate the sphere to the specified position
    X = X + pos(1);
    Y = Y + pos(2);
    Z = Z + pos(3);

    % Plot the sphere with green color
    surf(X, Y, Z, 'FaceColor', 'g', 'EdgeColor', 'none',↙
'FaceAlpha', 1); % Draw the sphere in green
    hold on; % Keep the current plot

    % Create a slightly larger sphere for the outline
    outline_radius = radius + 0.05; % Increase the radius↙
slightly for outline
    [X_outline, Y_outline, Z_outline] = sphere(30);
    X_outline = X_outline * outline_radius + pos(1);
    Y_outline = Y_outline * outline_radius + pos(2);
    Z_outline = Z_outline * outline_radius + pos(3);

    % Plot the outline
    surf(X_outline, Y_outline, Z_outline, 'FaceColor',↙
```

```
'none', 'EdgeColor', 'k', 'LineWidth', 1.5); % Black outline
    hold on;


end

% FK plotting of robot by q-values:
function plot3DRobot(A_1To0_func, A_2To1_func, A_3To2_func,↙
A_4To3_func, A_5To4_func, A_6To5_func, vals, h, r, n)
    % Evaluate each transformation matrix with the specific↙
joint values:
    A_1To0_eval = A_1To0_func(vals(1), vals(2), vals(3), vals↙
(4), vals(5), vals(6), vals(7), vals(8), vals(9), vals(10),↙
vals(11), vals(12));
    A_2To1_eval = A_2To1_func(vals(1), vals(2), vals(3), vals↙
(4), vals(5), vals(6), vals(7), vals(8), vals(9), vals(10),↙
vals(11), vals(12));
    A_3To2_eval = A_3To2_func(vals(1), vals(2), vals(3), vals↙
(4), vals(5), vals(6), vals(7), vals(8), vals(9), vals(10),↙
vals(11), vals(12));
    A_4To3_eval = A_4To3_func(vals(1), vals(2), vals(3), vals↙
(4), vals(5), vals(6), vals(7), vals(8), vals(9), vals(10),↙
vals(11), vals(12));
    A_5To4_eval = A_5To4_func(vals(1), vals(2), vals(3), vals↙
(4), vals(5), vals(6), vals(7), vals(8), vals(9), vals(10),↙
vals(11), vals(12));
    A_6To5_eval = A_6To5_func(vals(1), vals(2), vals(3), vals↙
(4), vals(5), vals(6), vals(7), vals(8), vals(9), vals(10),↙
vals(11), vals(12));

    % Evaluate the compound transformation matrices with the↙
specific joint values:
    A_2To0_eval = A_1To0_eval * A_2To1_eval;
    A_3To0_eval = A_2To0_eval * A_3To2_eval;
```

```
    A_4To0_eval = A_3To0_eval * A_4To3_eval;
    A_5To0_eval = A_4To0_eval * A_5To4_eval;
    A_6To0_eval = A_5To0_eval * A_6To5_eval;

    %Check sys 4 pos:
    disp('Place of gripper:');
    A_6To0_eval*[0;0;0;1]
    disp('Rotation of gripper:');
    A_6To0_eval(1:3,1:3)
    %disp('Rotation of sys5:');
    %A_5To0_eval(1:3,1:3)

    % Draw Robot:
    % Sys 1->2:
    q1 = A_2To0_eval * [0; 0; 0; 1] + ...
        (227) * A_2To0_eval * [0; 0; 1; 0] - vals(9) *↙
A_2To0_eval * [1; 0; 0; 0];
    q2 = A_2To0_eval * [0; 0; 0; 1] + (227) * A_2To0_eval *↙
[0; 0; 1; 0];

    % Sys 2->3 + d4 slices:
    d41 = 120; d42 = 100; d43 = vals(11) - d41 - d42;
    q3 = A_2To0_eval * [0; 0; 0; 1] + (d41) * A_3To0_eval *↙
[0; 0; 1; 0];
    q4 = q3 + (vals(10)) * A_3To0_eval * [1; 0; 0; 0];
    q5 = q4 + (d42) * A_3To0_eval * [0; 0; 1; 0];

    % Collect positions for plotting
    Pos = [[0; 0; 0], [0; 0; vals(7)], A_1To0_eval(1:3, 4),↙
q1(1:3), ...
        q2(1:3), A_2To0_eval(1:3, 4), q3(1:3), q4(1:3), q5(1:↙
3), ...
        A_4To0_eval(1:3, 4), A_5To0_eval(1:3, 4), A_6To0_eval↙
(1:3, 4)];
```

```
    % Plot robot arm at this↙
instance-------------------------------------
    %figure();
    % Temporarily hide the figure while processing
    figure('Visible', 'off');
    plot3(Pos(1, :), Pos(2, :), Pos(3, :))
    hold on;

    % Draw Joints using cylinders:
    % base:
    pos = [0, 0, 0 + h/2];
    R = [1 0 0; 0 1 0; 0 0 1];
    plotCylinder(pos, R, h, r, n)
    % theta2 joint
    pos = A_1To0_eval(1:3, 4);
    R = A_1To0_eval(1:3, 1:3);
    plotCylinder(pos, R, h, r, n)

    % theta3 joint
    pos = A_2To0_eval(1:3, 4);
    R = A_2To0_eval(1:3, 1:3);
    plotCylinder(pos, R, h, r, n)

    % theta4 joint
    pos = q5(1:3);
    R = A_3To0_eval(1:3, 1:3);
    plotCylinder(pos, R, h, r, n)

    % theta 5+6 joints
    pos = A_4To0_eval(1:3, 4);
    R = A_4To0_eval(1:3, 1:3);
    plotCylinder(pos, R, h, r, n)
```

```
    pos = A_5To0_eval(1:3, 4);
    R = A_5To0_eval(1:3, 1:3);
    plotCylinder(pos, R, h, r, n)

    % Gripper
    pos = A_6To0_eval(1:3, 4).';
    a = 60; b = 90; c = 20;
    R = A_6To0_eval(1:3, 1:3);
    plotBox(pos, a, b, c, R)

    % Set labels and grid
    xlabel('X-axis');
    ylabel('Y-axis');
    zlabel('Z-axis');
    grid on;
    view(3); % 3D view
    axis equal; % Keep aspect ratio
    title('6R Robot');
    camlight left; % Add a light source on the left side
    lighting gouraud; % Set the lighting to Gouraud shading
    material shiny; % Set the material to shiny for better↙
reflection

    %xlim([-500 1200])%-1420 1420 (small -500 500)
    %ylim([-800 800])%-1420 1420
    %zlim([0 2000])%1500

    xlim([-900 3500])%-1420 1420 (small -500 500)
    ylim([-700 500])%-1420 1420
    zlim([0 1700])

end

% IK plotting of robot by Position and Orientation:
```

```
function vals = CalcIK_Plot3DRobot(Pos, R, A_1To0_func,↙
A_2To1_func, A_3To2_func, A_4To3_func, A_5To4_func,↙
A_6To5_func,A_3To0_func, h, r, n, A_6To3, d_values, a_values)
    % Extract d and a values from the input arguments
    d1_value = d_values(1);
    d4_value = d_values(2);
    d6_value = d_values(3);

    a1_value = a_values(1);
    a2_value = a_values(2);
    a3_value = a_values(3);

    % Desired position
    x = Pos(1);
    y = Pos(2);
    z = Pos(3);

    % Rotation matrix
    R_6to0 = R; % Use the provided rotation matrix R_6to0
    Pc = Pos - d6_value * R_6to0 * [0; 0; 1];
    x0 = Pc(1);
    y0 = Pc(2);
    z0 = Pc(3);

    % Placement Problem:↙
-------------------------------------------------------------
    K = +sqrt(x0^2 + y0^2); % 2 options: +-sqrt(...)

    if K ==0
        disp("SINGULARITY! Multiple theta1-s are Solution!")
        theta1_IK=0; % choose arbitrary theta1
    else
        % theta 1 solution
        S1 = y0/K;
```

```
        C1 = x0/K;
        theta1_IK = atan2(S1, C1);
    end


    % FIND A1to0:
    A_1To0_eval_IK = A_1To0_func(theta1_IK, 0, 0, 0, 0, 0,↙
d1_value, a1_value,...
    a2_value, a3_value, d4_value, d6_value);

    % theta 3 solution
    R_0To1 = A_1To0_eval_IK(1:3, 1:3);
    w = [-a1_value; -d1_value; 0] + (R_0To1.') * Pc;
    w_x1_y1_SIZE = sqrt(w(1)^2 + w(2)^2);
    A = 2 * a2_value * a3_value;
    B = 2 * a2_value * d4_value;
    C = w_x1_y1_SIZE^2 - a2_value^2 - a3_value^2 -↙
d4_value^2;
    Y = +sqrt(A^2 + B^2 - C^2);
    theta3_IK = atan2(B, A) + atan2(-Y, C); % 2 options: +-Y

    C3 = cos(theta3_IK);
    S3 = sin(theta3_IK);
    if K == 0
        disp("SINGULARITY!")
        % theta 2 solution
        A = [a2_value + a3_value * C3 + d4_value * S3,↙
d4_value * C3 - a3_value * S3;...
         a3_value * S3 - d4_value * C3, a2_value + a3_value *↙
C3 + d4_value * S3];
        b = [-a1_value; z0 - d1_value];
        SOL = A \ b;
        C2 = SOL(1);
        S2 = SOL(2);
```

```
        theta2_IK = atan2(S2, C2);
    elseif C1 == 0
        disp("C1 = 0")
        % theta 2 solution
        A = [a2_value + a3_value * C3 + d4_value * S3,↙
d4_value * C3 - a3_value * S3;...
        a3_value * S3 - d4_value * C3, a2_value + a3_value *↙
C3 + d4_value * S3];
        b = [((y0) / S1) - a1_value; z0 - d1_value];
        SOL = A \ b;
        C2 = SOL(1);
        S2 = SOL(2);
        theta2_IK = atan2(S2, C2);
    else
        % theta 2 solution
        A = [a2_value + a3_value * C3 + d4_value * S3,↙
d4_value * C3 - a3_value * S3;...
         a3_value * S3 - d4_value * C3, a2_value + a3_value *↙
C3 + d4_value * S3];
        b = [((x0) / C1) - a1_value; z0 - d1_value];
        SOL = A \ b;
        C2 = SOL(1);
        S2 = SOL(2);
        theta2_IK = atan2(S2, C2);
    end


    % Orientation Problem:
    % Get the R parametricly:
    R_6To3 = A_6To3(1:3, 1:3);

    % Find R_3To0 from previous IK:
    A_3To0_eval = A_3To0_func(theta1_IK, theta2_IK- pi/2,↙
theta3_IK, 0, 0, 0, d1_value, a1_value,...
```

```
    a2_value, a3_value, d4_value, d6_value);
    R_3To0_IK = A_3To0_eval(1:3, 1:3);


    % Calculate orientation
    R_Orientation_Prob = R_3To0_IK.' * R_6to0;
    a = R_Orientation_Prob(1, 1); b = R_Orientation_Prob(1,↙
2); c = R_Orientation_Prob(1, 3);
    d = R_Orientation_Prob(2, 1); E = R_Orientation_Prob(2,↙
2); f = R_Orientation_Prob(2, 3);
    g = R_Orientation_Prob(3, 1); H = R_Orientation_Prob(3,↙
2); I = R_Orientation_Prob(3, 3);

    S5 = (-1)*sqrt(1 - I^2);% -
    C5 = I;
    theta5_IK = atan2(S5, C5);

    % Check if I is not equal to 1 AND I is not equal to -1
    if I ~= 1 && I ~= -1
        S4 = f / S5; C4 = c / S5;
        S6 = H / S5; C6 = -g / S5;

        theta4_IK = atan2(S4, C4);
        theta6_IK = atan2(S6, C6);
    else
        disp("S5 = 0!")
        % Convert solutions to regular numbers
        sum = atan2(d,a);
        theta4_IK = sum/2;
        theta6_IK = sum/2;
    end

    % Normalize angles:
    %theta1_IK = mod(theta1_IK + pi, 2 * pi) - pi;
```

```
    %theta2_IK = mod((theta2_IK - pi / 2) + pi, 2 * pi) - pi;
    %theta3_IK = mod(theta3_IK + pi, 2 * pi) - pi;
    %theta4_IK = mod(theta4_IK + pi, 2 * pi) - pi;
    %theta5_IK = mod(theta5_IK + pi, 2 * pi) - pi;
    %theta6_IK = mod(theta6_IK + pi, 2 * pi) - pi;

    % Assign numerical values
    thetas = [theta1_IK, theta2_IK - pi / 2, theta3_IK,...
        theta4_IK, theta5_IK, theta6_IK]; % Joint values in↙
rad

    vals = [thetas, d1_value, a1_value, a2_value, a3_value,↙
d4_value, d6_value];

    % Display results
    disp('IK vals:');
    disp(vals);

    % Draw Robot:
    %plot3DRobot(A_1To0_func, A_2To1_func, A_3To2_func,↙
A_4To3_func, A_5To4_func, A_6To5_func, vals, h, r, n);
end

% Solves for x(t) given BC and dt size:
function [t_vals, x_vals,x_t_solved] = quinticBC(t1, t2, x1,↙
x2, v1, v2, ac1, ac2, dt)
    % QUINTICBC_DT solves for x(t) given boundary conditions↙
on position,
    % velocity, and acceleration at t1 and t2, and returns↙
evaluated values
    % for x(t) at time steps defined by dt.
    %
    % INPUTS:
    %   t1, t2        - Boundary times
```

```
    %   x1, x2          - Positions at t1 and t2
    %   v1, v2          - Velocities at t1 and t2
    %   a1, a2          - Accelerations at t1 and t2
    %   dt              - Time step for the evaluation of x(t)
    %
    % OUTPUTS:
    %   t_vals          - Time values where the polynomial is
evaluated
    %   x_vals          - Position values of x(t) at the
corresponding t_vals

    % Define symbolic variable
    syms t real

    % Quintic polynomial form
    syms a0 a1 a2 a3 a4 a5
    x_t = a0 + a1*t + a2*t^2 + a3*t^3 + a4*t^4 + a5*t^5;

    % Velocity and acceleration by differentiating x(t)
    v_t = diff(x_t, t);  % Velocity
    a_t = diff(v_t, t);  % Acceleration

    % Set up boundary condition equations
    eqs = [
        subs(x_t, t, t1) == x1,    % Position at t1
        subs(x_t, t, t2) == x2,    % Position at t2
        subs(v_t, t, t1) == v1,    % Velocity at t1
        subs(v_t, t, t2) == v2,    % Velocity at t2
        subs(a_t, t, t1) == ac1,    % Acceleration at t1
        subs(a_t, t, t2) == ac2     % Acceleration at t2
    ];

    % Solve for the coefficients a0, a1, a2, a3, a4, a5
    coeffs = solve(eqs, [a0, a1, a2, a3, a4, a5]);
```

```
    % Substitute coefficients into the polynomial
    x_t_solved = subs(x_t, [a0, a1, a2, a3, a4, a5], ...
        [coeffs.a0, coeffs.a1, coeffs.a2, coeffs.a3, coeffs.↙
a4, coeffs.a5]);

    % Generate time values based on dt
    t_vals = t1:dt:t2;

    % Evaluate the polynomial at the time values
    x_vals = double(subs(x_t_solved, t, t_vals));

    % Plot the result
    %figure;
    %plot(t_vals, x_vals);
    %title('Quintic Polynomial with Boundary Conditions on↙
Position, Velocity, and Acceleration');
    %xlabel('Time');
    %ylabel('Position');
    %grid on;
end

% Solves for v(t) and a(t) frome x(t):
function [v_vals,a_vals] = plot_velocity(t_vals, x_t_solved)
    % PLOT_VELOCITY computes the velocity v(t) and plots it.
    %
    % INPUTS:
    %   t_vals      - Time values where x(t) is evaluated
    %   x_t_solved  - The solved quintic polynomial x(t)
    %
    % OUTPUT:
    %   v_vals      - Velocity values of v(t) at the↙
corresponding t_vals
```

```
    % Define symbolic variable for time
    syms t real

    % Differentiate x(t) to get v(t)
    v_t = diff(x_t_solved, t);

    % Evaluate velocity at the given time points
    v_vals = double(subs(v_t, t, t_vals));

    % Differentiate v(t) to get a(t)
    a_t = diff(v_t, t);

    % Evaluate velocity at the given time points
    a_vals = double(subs(a_t, t, t_vals));

    % Plot the velocity over time
    %{
    figure;
    plot(t_vals, v_vals, 'LineWidth', 2);
    title('Velocity over Time');
    xlabel('Time');
    ylabel('Velocity');
    grid on;

    % Plot a(t) over time
    figure;
    plot(t_vals, a_vals, 'LineWidth', 2);
    title('Acceleration over Time');
    xlabel('Time');
    ylabel('Acceleration');
    grid on;
    %}
end
```

```
% Solves for q(t) given BC and dt size:
function [t_vals, q_vals, q_t_solved] = quinticBC_6x1(t1, t2,↙
q1, q2, v1, v2, ac1, ac2, dt)
    % QUINTICBC_6X1 solves for a 6x1 vector q(t) given↙
boundary conditions
    % on position, velocity, and acceleration at t1 and t2,↙
and returns
    % evaluated values for q(t) at time steps defined by dt.
    %
    % INPUTS:
    %   t1, t2          - Boundary times
    %   q1, q2          - 6x1 vectors of positions at t1 and t2
    %   v1, v2          - 6x1 vectors of velocities at t1 and↙
t2
    %   ac1, ac2        - 6x1 vectors of accelerations at t1↙
and t2
    %   dt              - Time step for the evaluation of q(t)
    %
    % OUTPUTS:
    %   t_vals          - Time values where the polynomial is↙
evaluated
    %   q_vals          - 6xN matrix of position values of q(t)↙
at the corresponding t_vals
    %   q_t_solved      - Symbolic solutions for q(t)

    % Initialize outputs
    q_vals = [];
    q_t_solved = sym(zeros(6, 1));  % Symbolic solution for↙
each element of q(t)

    % Generate time values based on dt
    t_vals = t1:dt:t2;

    % Loop over each of the 6 components of q(t)
```

```
    for i = 1:6
        % Define symbolic variable
        syms t real

        % Quintic polynomial form
        syms a0 a1 a2 a3 a4 a5
        q_t = a0 + a1*t + a2*t^2 + a3*t^3 + a4*t^4 + a5*t^5;

        % Velocity and acceleration by differentiating q(t)
        v_t = diff(q_t, t);  % Velocity
        ac_t = diff(v_t, t); % Acceleration

        % Set up boundary condition equations for the i-th↙
component
        eqs = [
            subs(q_t, t, t1) == q1(i),     % Position at t1
            subs(q_t, t, t2) == q2(i),     % Position at t2
            subs(v_t, t, t1) == v1(i),     % Velocity at t1
            subs(v_t, t, t2) == v2(i),     % Velocity at t2
            subs(ac_t, t, t1) == ac1(i),   % Acceleration at↙
t1
            subs(ac_t, t, t2) == ac2(i)    % Acceleration at↙
t2
        ];

        % Solve for the coefficients a0, a1, a2, a3, a4, a5
        coeffs = solve(eqs, [a0, a1, a2, a3, a4, a5]);

        % Substitute coefficients into the polynomial
        q_t_solved(i) = subs(q_t, [a0, a1, a2, a3, a4, a5],↙
...
            [coeffs.a0, coeffs.a1, coeffs.a2, coeffs.a3,↙
coeffs.a4, coeffs.a5]);
```

```
        % Evaluate the polynomial at the time values and↙
store in q_vals
        q_vals(i, :) = double(subs(q_t_solved(i), t,↙
t_vals));
    end

    %{
    % Plot the results for each component
    figure;
    for i = 1:6
        subplot(3, 2, i);
        plot(t_vals, q_vals(i, :), '-o');
        title(['Component ', num2str(i), ' of q(t)']);
        xlabel('Time');
        ylabel(['q_', num2str(i), '(t)']);
        grid on;
    end
    %}
end

% functions for rotating in countinous time (zero BC):
function R_t = symbolicRotationMatrix(R_i, R_f, t_i, t_f)
    % Define symbolic variables
    syms t a b c d e f real

    % Compute the relative rotation matrix R
    R_ftoi = R_i'*R_f; % Relative rotation from R_i to R_f

    % Extract the rotation axis and angle from the relative↙
rotation matrix
    [axis, theta] = rotationMatrixToAxisAngle(R_ftoi);

    % Define the polynomial coefficients based on boundary↙
conditions
```

```
    % The polynomial for angle theta(t) is of the form:
    % theta(t) = a*t^5 + b*t^4 + c*t^3 + d*t^2 + e*t + f

    % Set up the system of equations for the coefficients
    eqs = [
        a*t_i^5 + b*t_i^4 + c*t_i^3 + d*t_i^2 + e*t_i + f ==↙
0, % Start from initial angle (0 at t_i)
        a*t_f^5 + b*t_f^4 + c*t_f^3 + d*t_f^2 + e*t_f + f ==↙
theta, % End at final angle
        5*a*t_i^4 + 4*b*t_i^3 + 3*c*t_i^2 + 2*d*t_i + e == 0,↙
% Zero velocity at t_i
        5*a*t_f^4 + 4*b*t_f^3 + 3*c*t_f^2 + 2*d*t_f + e == 0,↙
% Zero velocity at t_f
        20*a*t_i^3 + 12*b*t_i^2 + 6*c*t_i + 2*d == 0, % Zero↙
acceleration at t_i
        20*a*t_f^3 + 12*b*t_f^2 + 6*c*t_f + 2*d == 0  % Zero↙
acceleration at t_f
    ];

    % Solve the system of equations for coefficients a, b, c,↙
d, e, f
    coeffs = solve(eqs, [a, b, c, d, e, f]);

    % Create the polynomial theta(t)
    theta_t = coeffs.a*t^5 + coeffs.b*t^4 + coeffs.c*t^3 +↙
coeffs.d*t^2 + coeffs.e*t + coeffs.f;

    % Construct the rotation matrix R(t) using axis-angle↙
representation
    R_t = R_i * rotationMatrixFromAxisAngle(axis, theta_t);
end

function [axis, angle] = rotationMatrixToAxisAngle(R)
    % Function to extract axis and angle from a rotation↙
```

```
matrix
    % Compute the angle
    angle = acos((trace(R) - 1) / 2);

    % Compute the rotation axis
    if angle == 0
        axis = [0; 0; 0]; % No rotation
    else
        % Extract the axis of rotation using the rotation↙
matrix elements
        axis = [
            R(3, 2) - R(2, 3);
            R(1, 3) - R(3, 1);
            R(2, 1) - R(1, 2)
        ];
        axis = axis / (2*sin(angle)); % Normalize the axis
    end
end

function R = rotationMatrixFromAxisAngle(axis, theta)
    % Function to create a rotation matrix from an axis and↙
angle (axis-angle representation)
    axis = axis / norm(axis); % Normalize the axis
    u = axis(1);
    v = axis(2);
    w = axis(3);

    % Rotation matrix using the axis-angle formula
    R = [
        cos(theta) + u^2 * (1 - cos(theta)),      u*v*(1 -↙
cos(theta)) - w*sin(theta), u*w*(1 - cos(theta)) + v*sin↙
(theta);
        v*u*(1 - cos(theta)) + w*sin(theta),     cos(theta) +↙
v^2 * (1 - cos(theta)), v*w*(1 - cos(theta)) - u*sin(theta);
```

```
        w*u*(1 - cos(theta)) - v*sin(theta),      w*v*(1 - cos↙
(theta)) + u*sin(theta), cos(theta) + w^2 * (1 - cos(theta))
    ];
end

% FK plotting of robot by q-values, with x(t) movement:
function plot3DRobotAbs(A_1To0_func, A_2To1_func,↙
A_3To2_func, A_4To3_func, A_5To4_func, A_6To5_func, vals, h,↙
r, n,V,t_current)
    % Evaluate each transformation matrix with the specific↙
joint values:
    A_1To0_eval = A_1To0_func(vals(1), vals(2), vals(3), vals↙
(4), vals(5), vals(6), vals(7), vals(8), vals(9), vals(10),↙
vals(11), vals(12));
    A_2To1_eval = A_2To1_func(vals(1), vals(2), vals(3), vals↙
(4), vals(5), vals(6), vals(7), vals(8), vals(9), vals(10),↙
vals(11), vals(12));
    A_3To2_eval = A_3To2_func(vals(1), vals(2), vals(3), vals↙
(4), vals(5), vals(6), vals(7), vals(8), vals(9), vals(10),↙
vals(11), vals(12));
    A_4To3_eval = A_4To3_func(vals(1), vals(2), vals(3), vals↙
(4), vals(5), vals(6), vals(7), vals(8), vals(9), vals(10),↙
vals(11), vals(12));
    A_5To4_eval = A_5To4_func(vals(1), vals(2), vals(3), vals↙
(4), vals(5), vals(6), vals(7), vals(8), vals(9), vals(10),↙
vals(11), vals(12));
    A_6To5_eval = A_6To5_func(vals(1), vals(2), vals(3), vals↙
(4), vals(5), vals(6), vals(7), vals(8), vals(9), vals(10),↙
vals(11), vals(12));

    % Evaluate the compound transformation matrices with the↙
specific joint values:
    A_2To0_eval = A_1To0_eval * A_2To1_eval;
    A_3To0_eval = A_2To0_eval * A_3To2_eval;
```

```
    A_4To0_eval = A_3To0_eval * A_4To3_eval;
    A_5To0_eval = A_4To0_eval * A_5To4_eval;
    A_6To0_eval = A_5To0_eval * A_6To5_eval;

    %Check sys 4 pos:
    disp('Place of gripper:');
    A_6To0_eval*[0;0;0;1]
    disp('Rotation of gripper:');
    A_6To0_eval(1:3,1:3)
    %disp('Rotation of sys5:');
    %A_5To0_eval(1:3,1:3)

    % Draw Robot:
    % Sys 1->2:
    q1 = A_2To0_eval * [0; 0; 0; 1] + ...
        (227) * A_2To0_eval * [0; 0; 1; 0] - vals(9) *↙
A_2To0_eval * [1; 0; 0; 0];
    q2 = A_2To0_eval * [0; 0; 0; 1] + (227) * A_2To0_eval *↙
[0; 0; 1; 0];

    % Sys 2->3 + d4 slices:
    d41 = 120; d42 = 100; d43 = vals(11) - d41 - d42;
    q3 = A_2To0_eval * [0; 0; 0; 1] + (d41) * A_3To0_eval *↙
[0; 0; 1; 0];
    q4 = q3 + (vals(10)) * A_3To0_eval * [1; 0; 0; 0];
    q5 = q4 + (d42) * A_3To0_eval * [0; 0; 1; 0];

    % Collect positions for plotting
    Pos = [[0; 0; 0], [0; 0; vals(7)], A_1To0_eval(1:3, 4),↙
q1(1:3), ...
        q2(1:3), A_2To0_eval(1:3, 4), q3(1:3), q4(1:3), q5(1:↙
3), ...
        A_4To0_eval(1:3, 4), A_5To0_eval(1:3, 4), A_6To0_eval↙
(1:3, 4)];
```

```
    Pos = Pos + t_current * [V;0;0];

    % Plot robot arm at this↙
instance-------------------------------------
    %figure();
    % Temporarily hide the figure while processing
    figure('Visible', 'off');
    plot3(Pos(1, :), Pos(2, :), Pos(3, :))
    hold on;

    % Draw Joints using cylinders:
    % base:
    pos = [0, 0, 0 + h/2] + t_current * [V,0,0];
    R = [1 0 0; 0 1 0; 0 0 1];

    plotCylinder(pos, R, h, r, n)
    % theta2 joint
    pos = A_1To0_eval(1:3, 4) + t_current * [V;0;0];

    R = A_1To0_eval(1:3, 1:3);
    plotCylinder(pos, R, h, r, n)

    % theta3 joint
    pos = A_2To0_eval(1:3, 4) + t_current * [V;0;0];
    R = A_2To0_eval(1:3, 1:3);
    plotCylinder(pos, R, h, r, n)

    % theta4 joint
    pos = q5(1:3) + t_current * [V;0;0];
    R = A_3To0_eval(1:3, 1:3);
    plotCylinder(pos, R, h, r, n)

    % theta 5+6 joints
```

```
    pos = A_4To0_eval(1:3, 4) + t_current * [V;0;0];
    R = A_4To0_eval(1:3, 1:3);
    plotCylinder(pos, R, h, r, n)

    pos = A_5To0_eval(1:3, 4) + t_current * [V;0;0];
    R = A_5To0_eval(1:3, 1:3);
    plotCylinder(pos, R, h, r, n)

    % Gripper
    pos = A_6To0_eval(1:3, 4).' + t_current * [V 0 0];
    a = 60; b = 90; c = 20;
    R = A_6To0_eval(1:3, 1:3);
    plotBox(pos, a, b, c, R)

    % Set labels and grid
    xlabel('X-axis');
    ylabel('Y-axis');
    zlabel('Z-axis');
    grid on;
    view(3); % 3D view
    axis equal; % Keep aspect ratio
    title('6R Robot');
    camlight left; % Add a light source on the left side
    lighting gouraud; % Set the lighting to Gouraud shading
    material shiny; % Set the material to shiny for better↙
reflection

    %xlim([-500 1200])%-1420 1420 (small -500 500)
    %ylim([-800 800])%-1420 1420
    %zlim([0 2000])%1500

    xlim([-500 4000])%-1420 1420 (small -500 500)
    ylim([-700 500])%-1420 1420
    zlim([0 1700])
```

```
end
```