

Project 2
ME5250 Spring 2024

LAST NAME: FIRST NAME:
GANESH KUMAR CHITHARANJAN

PROJECT TITLE:
**UR5 Robot with End-Effector following a circular trajectory in
Task Space**

Instructor: Dr. Peter Whitney

Table of Contents

1	Introduction	2
2	Forward Kinematics.....	2
3	Trajectory Generation and Inverse Kinematics.....	3
4	Conclusion	5
5	Appendix and References	6

1. Introduction:

In the realm of robotics, the utilization of collaborative robots, commonly known as co-robots, has gained significant traction due to their versatility and adaptability in various industrial and research settings. These robots, equipped with multiple degrees of freedom (DOF), offer enhanced capabilities for performing intricate tasks with precision and efficiency. Among the widely acclaimed co-robots is the Universal Robots UR-5, renowned for its six degrees of freedom, allowing for intricate motion in three-dimensional space.

One of the fundamental aspects of controlling a co-robot like the UR-5 is understanding its kinematics, particularly the forward kinematics, which delineates the relationship between the robot's joint angles and its end-effector's position and orientation. By leveraging Denavit-Hartenberg (DH) parameters, the analytical forward kinematics of the UR-5 can be determined, providing crucial insights into its motion planning and trajectory generation.

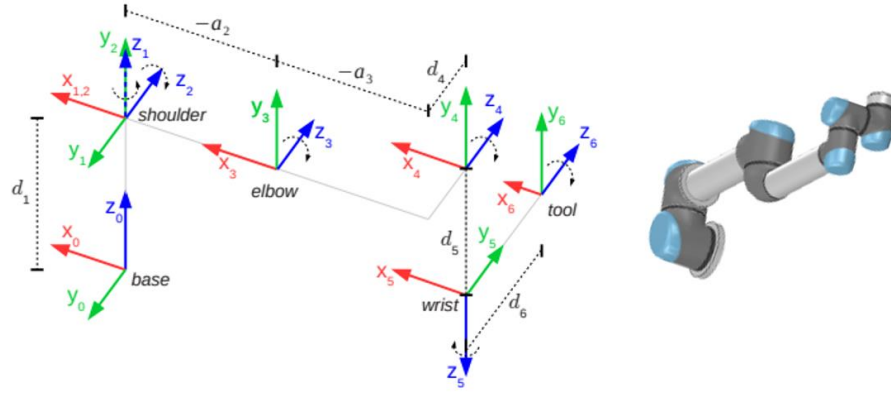
In this project, we embark on a journey to explore the forward kinematics of the UR-5, employing published papers and resources to ascertain the link lengths and DH parameters essential for modeling its motion. Our objective extends beyond mere theoretical understanding, as we aim to devise an intriguing trajectory in the task space. Specifically, we envision the UR-5 following a circular path while maintaining a fixed end-effector frame alignment. To realize this vision, we delve into the development of a basic Newton's method inverse kinematics (IK) solver, a computational approach vital for computing the joint angles required to achieve a desired end-effector configuration. Furthermore, leveraging MATLAB programming languages, we embark on crafting a simplistic yet informative 3D plot animation, depicting the UR-5's motion as it traverses the specified trajectory.

2. Forward Kinematics using D-H Parameters

Forward kinematics using Denavit-Hartenberg (DH) parameters is a fundamental aspect of robot kinematics. It enables the determination of the position and orientation of the robot's end-effector based on the joint angles and link parameters. The DH parameters, including link length a , link twist α , link offset d , and joint angle θ are used to construct the transformation matrices for each link. By sequentially multiplying these transformation matrices, the homogeneous transformation from the base frame to the end-effector frame is obtained. This process allows for the calculation of the end-effector's position and orientation in the robot's workspace. Forward kinematics based on DH parameters form the basis for motion planning, trajectory generation, and control in robotic systems.

$${}^{i-1}_iT = \begin{bmatrix} \cos \theta_i & -\sin \theta_i & 0 & a_{i-1} \\ \sin \theta_i \cos(\alpha_{i-1}) & \cos \theta_i \cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -\sin(\alpha_{i-1})d_i \\ \sin \theta_i \sin(\alpha_{i-1}) & \cos \theta_i \sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 1: DH formulation Transformation Matrix



(a) Frames with dh parameters of UR5 robot (b) Physical model of UR5 robot

Figure 2: The UR5 robot in zero position

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	$d_1 = 0.1$	θ_1
2	$\alpha_1 = 90^\circ$	0	0	θ_2
3	0	$a_2 = 0.5$	0	θ_3
4	0	$a_3 = 0.5$	$d_4 = 0.1$	θ_4
5	$\alpha_4 = 90^\circ$	0	$d_5 = 0.1$	θ_5
6	$\alpha_5 = -90^\circ$	0	$d_6 = 0.1$	θ_6

Table 1: DH parameters of UR5 robot

It is straightforward to write the transformation matrix for each link of the UR5 robot by inserting the DH parameters from Table 1 in Equation (4). The complete transformation from base to end-effector can then be derived by multiplication of all 6 transformation matrices.

3. Trajectory Generation and Inverse Kinematics

Circular trajectory generation and inverse kinematics play pivotal roles in the motion planning and control of robotic arms like the UR5. In circular trajectory generation, the goal is to generate a smooth path that describes a circular motion in Cartesian space, which the robot's end-effector follows. This involves determining the desired position, orientation, and velocity of the end-effector along the circular path. Inverse kinematics, on the other hand, deals with calculating the joint angles required for the robot to achieve a desired end-effector pose. For the UR5, which has six degrees of freedom, solving the inverse kinematics involves finding solutions to a set of nonlinear equations relating joint angles to the desired end-effector position and orientation. By combining circular trajectory generation with inverse kinematics, precise and efficient motion planning can be achieved, enabling the UR5 robot to execute complex tasks with accuracy and smoothness.

Waypoints for Circular Trajectory are defined with 20 set points with a radius of 0.2 m. The robot mean position is defined by inputting all theta values to zero and assigning that as the reference point of trajectory (i.e. center of the circle)

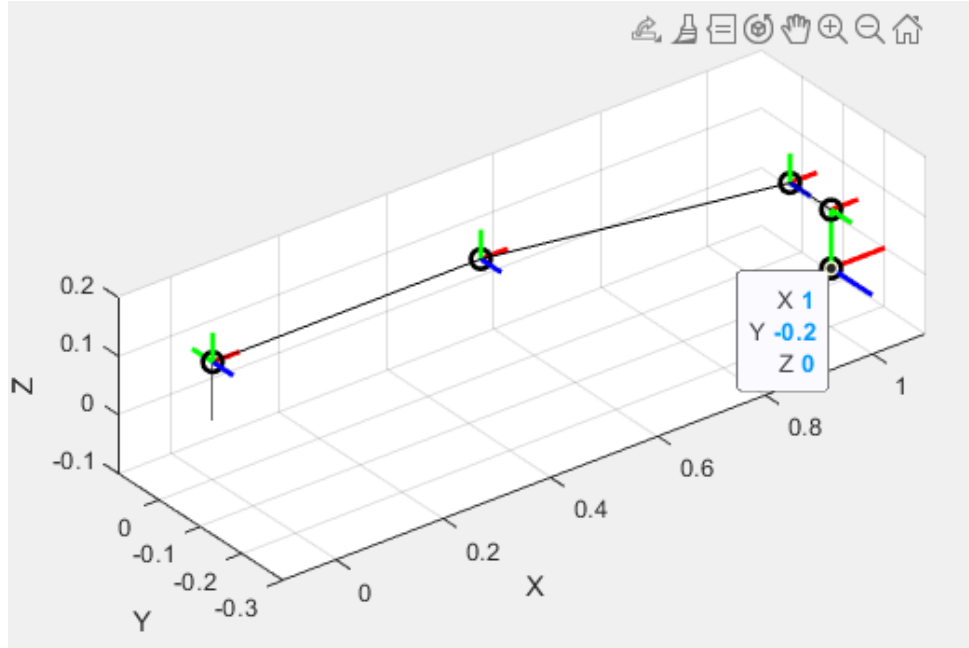


Figure 3: Stick figure of UR5 robot in home position.

Inverse kinematics is defined for waypoints for circular trajectory and joint angles are obtained from Newton's inverse kinematics approximation. Later, the obtained joint angles are used for task space trajectory generation,

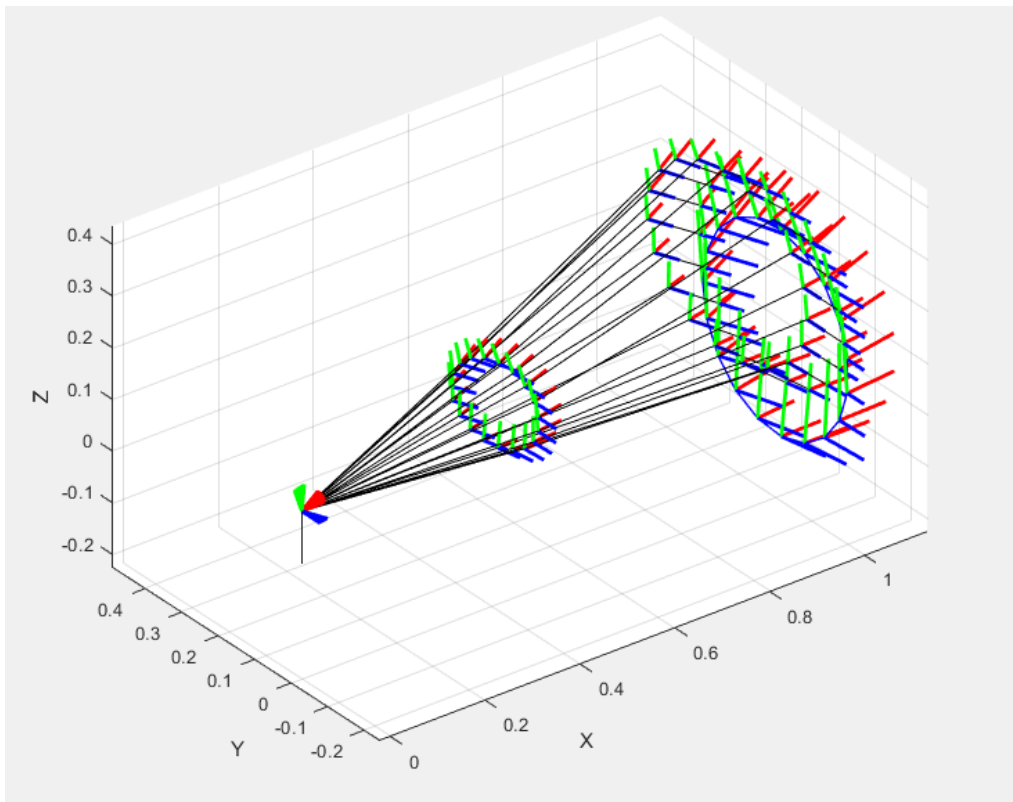


Figure 3: Task Space Circular Trajectory of UR5 robot

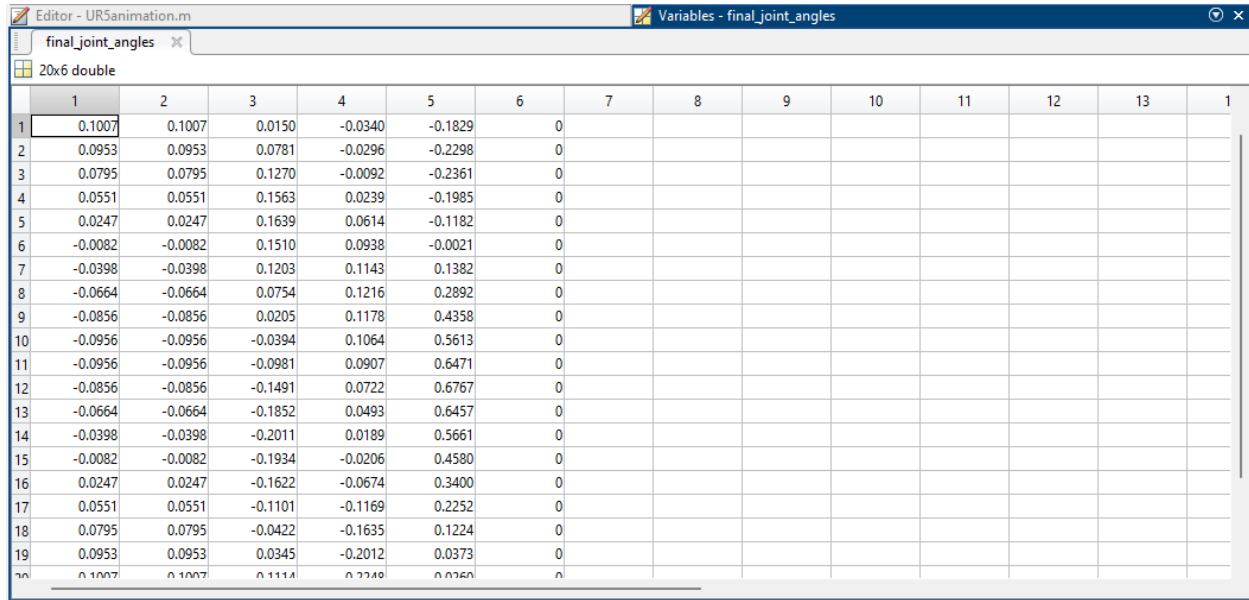
Basic Newtons Inverse Kinematics Solution:

Inputs:

Maximum Iteration: 100

Error: 1e-6

Joint Angles after solving the solver in MATLAB:



The image shows a MATLAB Variables window titled 'Variables - final_joint_angles'. It displays a 20x6 double matrix named 'final_joint_angles'. The matrix contains numerical values for each of the 20 rows and 6 columns. The values are as follows:

	1	2	3	4	5	6
1	0.1007	0.1007	0.0150	-0.0340	-0.1829	0
2	0.0953	0.0953	0.0781	-0.0296	-0.2298	0
3	0.0795	0.0795	0.1270	-0.0092	-0.2361	0
4	0.0551	0.0551	0.1563	0.0239	-0.1985	0
5	0.0247	0.0247	0.1639	0.0614	-0.1182	0
6	-0.0082	-0.0082	0.1510	0.0938	-0.0021	0
7	-0.0398	-0.0398	0.1203	0.1143	0.1382	0
8	-0.0664	-0.0664	0.0754	0.1216	0.2892	0
9	-0.0856	-0.0856	0.0205	0.1178	0.4358	0
10	-0.0956	-0.0956	-0.0394	0.1064	0.5613	0
11	-0.0956	-0.0956	-0.0981	0.0907	0.6471	0
12	-0.0856	-0.0856	-0.1491	0.0722	0.6767	0
13	-0.0664	-0.0664	-0.1852	0.0493	0.6457	0
14	-0.0398	-0.0398	-0.2011	0.0189	0.5661	0
15	-0.0082	-0.0082	-0.1934	-0.0206	0.4580	0
16	0.0247	0.0247	-0.1622	-0.0674	0.3400	0
17	0.0551	0.0551	-0.1101	-0.1169	0.2252	0
18	0.0795	0.0795	-0.0422	-0.1635	0.1224	0
19	0.0953	0.0953	0.0345	-0.2012	0.0373	0
20	0.1007	0.1007	0.1114	0.2240	0.0760	0

4. Conclusion

Implementing task space trajectory generation using Newton's Inverse Kinematics presents a robust approach for controlling robotic manipulators with precision and flexibility. By leveraging Newton's method, the algorithm efficiently computes joint configurations that correspond to desired end-effector poses, allowing the robot to follow complex trajectories in Cartesian space. This approach offers advantages such as improved accuracy, smoother motion, and the ability to handle nonlinearities and constraints inherent in robotic systems. Furthermore, the integration of task space trajectory generation with Newton's Inverse Kinematics opens doors to a wide range of applications, from manufacturing to healthcare, where precise motion control is crucial. Overall, this project demonstrates the effectiveness of combining theoretical principles with practical implementation to achieve advanced motion planning capabilities in robotic systems.

5. Appendix and References:

[Link to video: UR5 Circular Task Space Trajectory](#)

References:

Textbook: <https://northeastern.instructure.com/courses/171917/files/25571934?wrap=1>

[Singularity Analysis and Complete Methods to Compute the Inverse Kinematics for a 6-DOF UR/TM-Type Robot](#)

[Statistical comparison of Denavit-Hartenberg based inverse kinematic solutions of the UR5 robotic manipulator](#)

Code:

```
clc;
clear;
a = [0 0 0.5 0.5 0 0];
alpha = [0 pi/2 0 0 pi/2 -pi/2];
d = [0.1 0 0 0.1 0.1 0.1];
theta = [0, 0, 0, 0, 0, 0];
max_iterations = 100;
epsilon = 1e-6;

theta_guess = zeros(1, 6);
final_joint_angles = zeros(20, 6);
center = [1.0 -0.2 0]; % Center point of the circle
radius = 0.2;
num_points = 20;
t = linspace(0, 2*pi, num_points);
x_points = repmat(center(1), 1, num_points); % All points have the same x-coordinate
y_points = center(2) + radius * cos(t);
z_points = center(3) + radius * sin(t);

for j=1:num_points
    target_pos = [x_points(j), y_points(j), z_points(j)]';
    for iter = 1:max_iterations
        current_pos = forward_kinematics(theta_guess, a, alpha, d, theta);
        error = target_pos - current_pos;
        if norm(error) < epsilon
            disp('Converged!');
            break;
        end
        J = compute_jacobian(theta_guess, a, alpha, d, theta);
        delta_theta = pinv(J) * error;
        theta_guess = theta_guess + delta_theta';
    end
    final_joint_angles(j,:) = theta_guess;
end
disp('Final joint angles:');
disp(final_joint_angles);

figure;
hold on;
```

```

grid on;
xlabel('X');
ylabel('Y');
zlabel('Z');
axis equal;
plot3(x_points, y_points, z_points, 'b','LineWidth', 1); % Plot circle
view(3);

for k= 1: num_points
    theta_f = final_joint_angles(k,:);
    draw_robot(a, alpha, d, theta_f);
end

% Function to compute forward kinematics
function pos = forward_kinematics(theta, a, alpha, d, theta_offset)
    num_links = length(theta);
    pos = eye(4);
    for i = 1:num_links
        A_i = [
            cos(theta(i) + theta_offset(i)), -sin(theta(i) + theta_offset(i)) *
            cos(alpha(i)), sin(theta(i) + theta_offset(i)) * sin(alpha(i)), a(i) * cos(theta(i) +
            theta_offset(i));
            sin(theta(i) + theta_offset(i)), cos(theta(i) + theta_offset(i)) *
            cos(alpha(i)), -cos(theta(i) + theta_offset(i)) * sin(alpha(i)), a(i) * sin(theta(i)
            + theta_offset(i));
            0, sin(alpha(i)), cos(alpha(i)), d(i);
            0, 0, 0, 1
        ];
        pos = pos * A_i;
    end
    pos = pos(1:3, 4);
end

% Function to compute Jacobian matrix
function J = compute_jacobian(theta, a, alpha, d, theta_offset)
    num_links = length(theta);
    J = zeros(3, num_links);
    for i = 1:num_links
        theta_perturbed = theta;
        theta_perturbed(i) = theta_perturbed(i) + 1e-6;
        pos_perturbed = forward_kinematics(theta_perturbed, a, alpha, d,
        theta_offset);
        pos_current = forward_kinematics(theta, a, alpha, d, theta_offset);
        J(:, i) = (pos_perturbed - pos_current) / 1e-6;
    end
end

% Function to draw robot
function draw_robot(a, alpha, d, theta)
    T = eye(4);
    positions = zeros(3, length(a)+1);
    for i = 1:length(a)
        A = [cos(theta(i)), -sin(theta(i))*cos(alpha(i)),
            sin(theta(i))*sin(alpha(i)), a(i)*cos(theta(i));
            sin(theta(i)), cos(theta(i))*cos(alpha(i)), -
            cos(theta(i))*sin(alpha(i)), a(i)*sin(theta(i));

```

```

        0,                sin(alpha(i)),
cos(alpha(i)),          d(i);
        0,                0,                0,
1];

    T = T * A;
    positions(:, i+1) = T(1:3, 4);
end

hold on;
grid on;
xlabel('X');
ylabel('Y');
zlabel('Z');
axis equal;

for i = 1:length(a)
    if i == length(a)
        frame_length = 0.1;
        x_axis = T(1:3, 1) * frame_length + positions(:, i+1);
        y_axis = T(1:3, 2) * frame_length + positions(:, i+1);
        z_axis = T(1:3, 3) * frame_length + positions(:, i+1);
        plot3([positions(1, i+1) x_axis(1)], [positions(2, i+1) x_axis(2)],
[positions(3, i+1) x_axis(3)], 'r', 'LineWidth', 2);
        plot3([positions(1, i+1) y_axis(1)], [positions(2, i+1) y_axis(2)],
[positions(3, i+1) y_axis(3)], 'g', 'LineWidth', 2);
        plot3([positions(1, i+1) z_axis(1)], [positions(2, i+1) z_axis(2)],
[positions(3, i+1) z_axis(3)], 'b', 'LineWidth', 2);
    else
        frame_length = 0.05;
        x_axis = T(1:3, 1) * frame_length + positions(:, i+1);
        y_axis = T(1:3, 2) * frame_length + positions(:, i+1);
        z_axis = T(1:3, 3) * frame_length + positions(:, i+1);
        plot3([positions(1, i+1) x_axis(1)], [positions(2, i+1) x_axis(2)],
[positions(3, i+1) x_axis(3)], 'r', 'LineWidth', 2);
        plot3([positions(1, i+1) y_axis(1)], [positions(2, i+1) y_axis(2)],
[positions(3, i+1) y_axis(3)], 'g', 'LineWidth', 2);
        plot3([positions(1, i+1) z_axis(1)], [positions(2, i+1) z_axis(2)],
[positions(3, i+1) z_axis(3)], 'b', 'LineWidth', 2);

        plot3([positions(1, i) positions(1, i+1)], [positions(2, i) positions(2,
i+1)], [positions(3, i) positions(3, i+1)], 'k');
    end
end

max_lim = max(positions, [], 2);
min_lim = min(positions, [], 2);
xlim([min_lim(1)-0.1, max_lim(1)+0.3]);
ylim([min_lim(2)-0.3, max_lim(2)+0.3]);
zlim([min_lim(3)-0.3, max_lim(3)+0.3]);

pause(1);
hold off;
end

```