

## **Foundations of Robotics Final Project**

**By: Santosh Srinivas (sr6411), Rajasundaram Mathiazhagan (rm6584)**

### **Introduction:**

The objective of our project is to explore and implement key robotics concepts, focusing on the kinematics, visualization and dynamics of David Bowen's Plant Machete robot.

In the initial phase, we dedicated our efforts to understanding the structural and functional aspects of the robot in question by watching the video several times. Our quest to have an intelligent estimate of the dimensions and link lengths led us to a significant realization: the robot closely resembles the Ufactory Xarm series, as suggested by sources on the internet( [reference](#) ). The resemblance, especially in terms of the six degrees of freedom and the physical design of the links, guided our decision to model our project on the **Ufactory Xarm 6** lending realism to our work.

After securing the necessary dimensions of the robot, we implemented our project, utilizing MATLAB as our primary tool for executing the tasks of the project, which included the development of forward kinematics, inverse kinematics, pose visualization, application of the Jacobian method in both inverse kinematics and inverse dynamics, and workspace visualization.

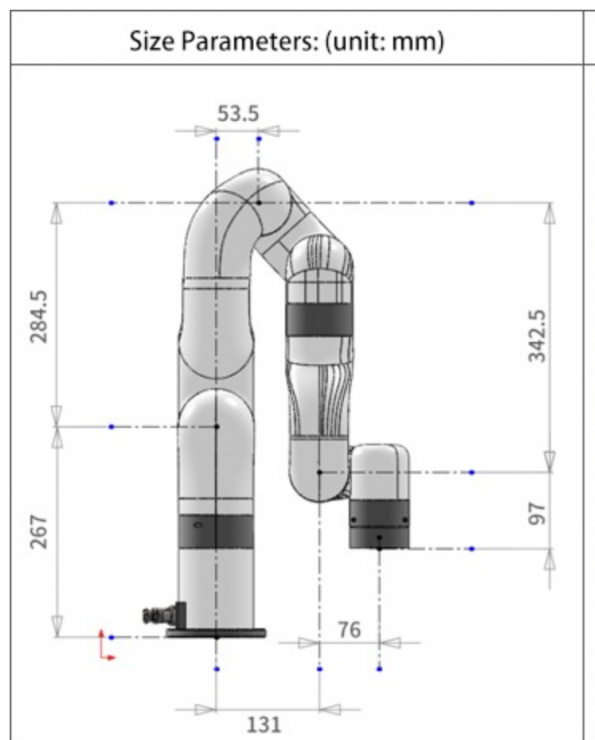
### **Description of the robot:**

The Ufactory Xarm 6 is a versatile robotic arm with six degrees of freedom, designed for precision and flexibility in various industrial and research applications. It features a lightweight yet robust build, supporting a range of programming languages and offering advanced control through intuitive software.

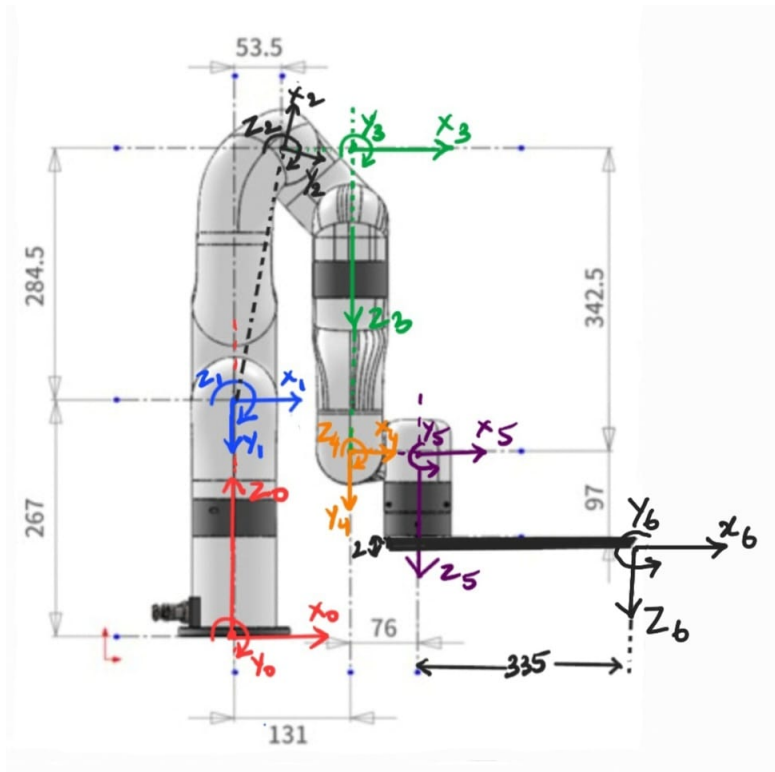
**Ufactory Xarm 6:**



**Home configuration:**



**Frames:**



**DH Table:**

Joint #	$\theta$ (rad)	$d$ (cm)	$a$ (cm)	$\alpha$ (rad)	$q$ (rad)
Joint 1	$\theta_1 = 0 + q_1$	26.7	0	$-\pi/2$	$q_1$
Joint 2	$\theta_2 = -1.384 + q_2$	0	28.9488	0	$q_2$
Joint 3	$\theta_3 = 1.384 + q_3$	0	7.75	$-\pi/2$	$q_3$
Joint 4	$\theta_4 = 0 + q_4$	34.25	0	$+\pi/2$	$q_4$
Joint 5	$\theta_5 = 0 + q_5$	0	7.6	$-\pi/2$	$q_5$
Joint 6	$\theta_6 = 0 + q_6$	9.9	3.35	0	$q_6$

$$a_2 (a \text{ for joint2}) = \sqrt{284.5^2 + 53.5^2} = 289.48866$$

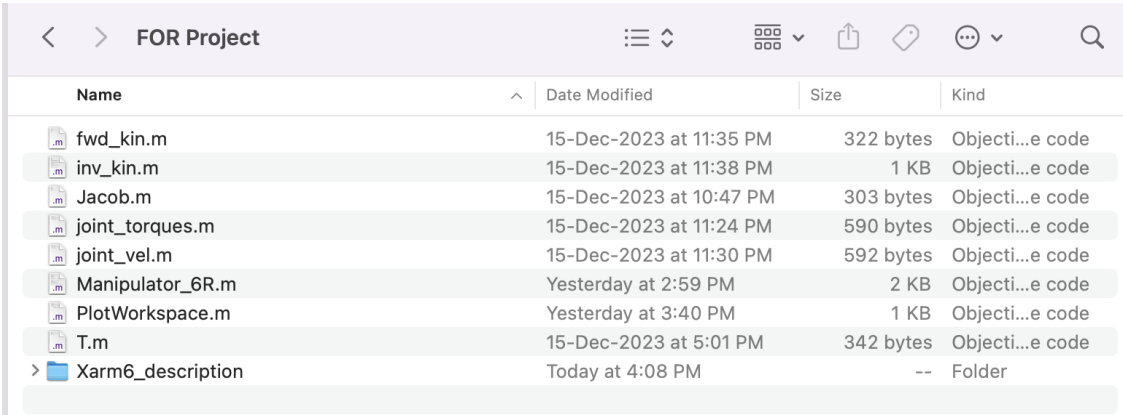
$$\bar{\theta}_2 = -\operatorname{atan}(284.5/53.5) = -1.3849179 (-79.34995^\circ);$$

$$\bar{\theta}_3 = +\operatorname{atan}(342.5/77.5) = +1.3482664 (+79.34995^\circ);$$

$$\text{where } \theta_i = \bar{\theta}_i + q_i$$

### **How to run the code using Matlab:**

1. Download and save the “FOR Project” folder. Make sure Xarm6\_description folder, and all the .m files are in the same folder. You should see something like below.



Name	Date Modified	Size	Kind
fwd_kin.m	15-Dec-2023 at 11:35 PM	322 bytes	Objecti...e code
inv_kin.m	15-Dec-2023 at 11:38 PM	1 KB	Objecti...e code
Jacob.m	15-Dec-2023 at 10:47 PM	303 bytes	Objecti...e code
joint_torques.m	15-Dec-2023 at 11:24 PM	590 bytes	Objecti...e code
joint_vel.m	15-Dec-2023 at 11:30 PM	592 bytes	Objecti...e code
Manipulator_6R.m	Yesterday at 2:59 PM	2 KB	Objecti...e code
PlotWorkspace.m	Yesterday at 3:40 PM	1 KB	Objecti...e code
T.m	15-Dec-2023 at 5:01 PM	342 bytes	Objecti...e code
> Xarm6_description	Today at 4:08 PM	--	Folder

Xarm6\_description contains the files related to the 3D model of Xarm6 manipulator used in the project.

2. Ensure Robotics system toolbox is installed in Matlab.
3. Open the Manipulator\_6R.m script.

## Manipulator\_6R.m: ( Main script)

### Section 1:

```
1 %Run this section to initialise robot parameters and 3d model
2
3 clc;
4 clear all;
5 global m6r; %Robot object that will be used in some functions in this package
6 global dh; % dh function that can be substituted with
7 global DH; % DH table
8
9 syms q [1 6], % symbolic joint variables
10
11 % the DH table
12 DH=[q(1),26.7,0,-pi/2;
13      q(2)-1.38492,0,28.94866,0;
14      q(3)+1.3849179,0,7.75,-pi/2;
15      q(4),34.25,0,pi/2;
16      q(5),0,7.6,-pi/2;
17      q(6),9.9,33.5,0
18      ];
19
20 % DH table function
21 dh = matlabFunction(DH);
22 dh=@(x)(dh(x(1),x(2),x(3),x(4),x(5),x(6)));
23
24 % For robot visualisation
25 m6r = importrobot("Xarm6_description/urdf/Xarm6.xacro",DataFormat="row"); %importing the 3D model of the manipulator
```

In this script section, the workspace is cleared and global variables for the robotic arm (m6r), Denavit-Hartenberg function (dh), and DH table (DH) are initialized. Symbolic joint variables q are defined, and the DH table is established with specific joint parameters. Finally, the robot's 3D model is imported into the m6r variable for visualization purposes.

**NOTE: Depending on your system Mac or Windows, you might need to change the '/' in file path in line 25.**

For mac; use '/' in file path; ie.

```
m6r = importrobot("Xarm6_description/urdf/Xarm6.xacro",DataFormat="row");
```

For Windows, use '\' in file path ie.

```
m6r = importrobot("Xarm6_description\urdf\Xarm6.xacro",DataFormat="row");
```

## Section 2:

```
%fwd_kin(q) returns the positions of each joint and end effector by taking the array of joint angles as input
fwd_kin([0,0,0,0,0,0])

%inv_kin(pos) returns joint angles for a given input of array containing
%end effector position (px,py,pz), followed by the X-Y-Z moving euler angles (radians) to describe end effector's orientation
Q=inv_kin([0,0,0,0,0,0])

%%Computing joint angular velocities
%Takes end effector position and orientation (moving XYZ Euler angles) (6 x 1 array) and end
%effector's linear and angular velocities (6 x 1 array) as inputs and
%returns joint angular velocities if solution exists
qd = joint_vel([50,10,20,0,pi/2,pi/4],[10,0,0,0,0,0])

%%Computing joint torques
%Takes end effector position and orientation (moving XYZ euler angles) (6 x 1 array) and external force and moments
% on the end effector (6 x 1 array) as inputs and returns joint torques if solution exists
Tau = joint_torques([50,10,20,0,pi/2,pi/4],[-10,0,0,0,0,0])

show(m6r,zeros(1,6)); % displaying the robot with joint angle inputs as the second parameter (home configuration in this example)
%PlotWorkspace(20); % For visualising the workspace of the manipulator. Input is the resolution of the joint angles (in degrees)
```

This section demonstrates the use of forward kinematics, inverse kinematics, inverse kinematics with Jacobian, inverse dynamics with Jacobian, visualization functions. Vary the inputs by changing the parameters inside the function call.

**Forward kinematics** Input for fwd\_kin(): Joint angles ( in radians ) as 1\*6 array

**Inverse Kinematics:** Input for inv\_kin(): 1\*6 array containing end effector position px,py,pz (cm) followed by XYZ moving Euler angles ( in radians)

**Inverse kinematics with Jacobian:** Inputs for joint\_vel(): End effector position (cm) and XYZ moving euler angles ( in radians) as 1\*6 array, followed by end effector's desired linear velocity (cm/s) and angular velocity (rad/s) as 1\*6 array.

**Inverse dynamics with Jacobian:** Inputs for joint\_torques(): End effector position(cm) and XYZ moving euler angles (in radians) as 1\*6 array, followed by external force (N) and moments (N-cm) as 1\*6 array.

**Visualization:** Inputs for show(): Robot model and a 1x6 array containing joint angles

4. Now run the script.
5. You will see a total of 5 outputs - 4 in the command window ( as vectors/matrices ) corresponding to forward kinematics, inverse kinematics, inverse kinematics with Jacobian, inverse dynamics with Jacobian and 1 figure from the last call for visualization.
6. Once the initialization section of the Manipulator\_6R.m script has been executed using the 'Run Section' option in MATLAB, any of the functions demonstrated in section 2 can be called through the command window with appropriate inputs.

## Forward Kinematics

In order to compute the transformation matrix from two successive frames, say  $\{i-1\}$  and  $\{i\}$ , we first rotate  $\{i-1\}$  about its Z axis, so that its X axis align, then shift it along its Z axis so that the X axis is in alignment with  $\{i\}$ , then move it along the new axis so that the origins coincide, then rotate about the new X axis, so that the frames coincide. This series of rotations and translations yields the transformation matrix between  $\{i-1\}$  and  $\{i\}$  as:

$$\therefore {}^{i-1}T_i = \begin{bmatrix} \cos \theta_i & -\cos \alpha_i \sin \theta_i & \sin \alpha_i \sin \theta_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i \cos \theta_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}$$

The transformation matrix for link / frame (k) can be computed by iteratively multiplying the transformation matrices as follows:

$${}^0T_k = {}^0T_1 * {}^1T_2 * \dots * {}^{k-1}T_k$$

The first three entries in the fourth column of the Transformation matrix gives the position of the origin of frame k.

## Function Description:

```
function pos = fwd_kin(q)
    global m6r;
    a=eye(4,4);
    global dh;
    dh_t=dh(q); % DH table with the joint variables substituted
    p=[0;0;0];
    for i=1:length(q)
        a=a*T(dh_t(i,:)); %multiplying (i-1)T(i) at each iteration
        p=[p,a(1:3,4)];
    end

    disp("Positions of Joints:");
    disp(p(:,1:6));
    disp("End effector position:")
    disp(p(:,7));

    %visualisation
    show(m6r,q);
end
```

The fwd\_kin function computes the positions of the origins of link frames and the end-effector frame for given joint angles. Also, it produces a visual representation of the corresponding pose of the robot.

Input: 1\*6 array of joint angles

Output: Position of joints (px, py,pz) and end effector position

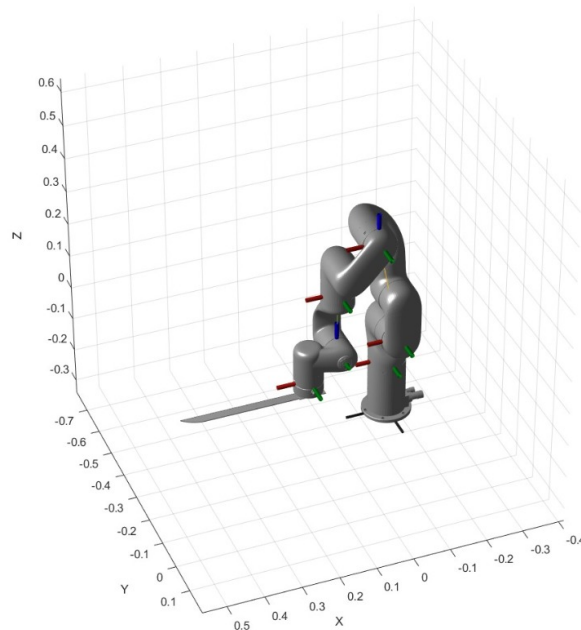
```
>> fwd_kin([0,0,0,0,0,0])
```

Positions of Joints:

0	0	5.3499	13.0999	13.1000	20.7000
0	0	-0.0000	-0.0000	0.0000	0.0000
0	26.7000	55.1500	55.1500	20.9000	20.9000

End effector position:

54.2000
0.0000
11.0001





## Inverse kinematics:

Inverse kinematics calculates the joint parameters needed to achieve a specific position and orientation of the end effector. The joint parameters are computed by solving equations that are obtained by equating the forward kinematics model of the manipulator to the desired end effector position and orientation.

## Function Description:

```
function Q_inv = inv_kin(pos)
n_joints = 6;
px=pos(1);
py=pos(2);
pz=pos(3);
alpha = pos(4);
beta=pos(5);
gama = pos(6);
global m6r;

%Rotation matrices corresponding to the moving euler angles
Rx = [1,0,0;0,cos(alpha),-sin(alpha);0,sin(alpha),cos(alpha)];
Ry = [cos(beta),0,sin(beta);0,1,0;-sin(beta),0,cos(beta)];
Rz= [cos(gama),-sin(gama),0;sin(gama),cos(gama),0;0,0,1];

Goal = [Rx*Ry*Rz,[px;py;pz]]; % First 3 rows of the transformation matrix for given end effector position and orientation
syms q [1 n_joints]
tn = Tn(q); % Transformation matrix (0T6) with joint angles as variables
e=(tn(1:3,:) -Goal); % matrix of equations to be solved

%set of initial estimates to try with (as fsolve is sensitive to initial estimate)
EST=zeros(4^n_joints,n_joints);
for i=1:(4^n_joints)-1
    k=[(num2str(dec2base(i,4))-'0')].*(pi/2);
    EST(i+1,1:length(k))=k;
end

F=(matlabFunction(e));
options=optimoptions("fsolve","Algorithm","levenberg-marquardt","Display","off");

for i= 1:height(EST) %iterating through different set of initial estimates
[sol,fval,exitflag]=fsolve(@(a)F(a(1),a(2),a(3),a(4),a(5),a(6)),EST(i,:),options);
if(exitflag==1)
    Q_inv = sol;
    show(m6r,sol);
    return; % exits the function when a solution is found
end
end

% when no solution is found, message is displayed and returns empty array
disp("Given position and orientation is not attainable");
Q_inv=[];

end
```

The `inv_kin` function generates equations by equating the final transformation matrix  ${}^0T_6$  with the joint angles as variables to the transformation matrix corresponding to the end effector position and moving XYZ Euler angles given as inputs. This forms a set of 12 equations that is then solved by the `fsolve()` function in MATLAB. Since the `fsolve()` method is highly sensitive to the initial estimates of the solution, multiple trials with different initial estimates is necessary to conclude the absence of a solution with higher confidence. For best shot at finding the solution, while ensuring lower computation time, the `inv_kin` function iterates through different combinations of joint angles as initial estimates, with each joint angle taking the values of 0,  $\pi/2$ ,  $\pi$ ,  $3\pi/2$ .

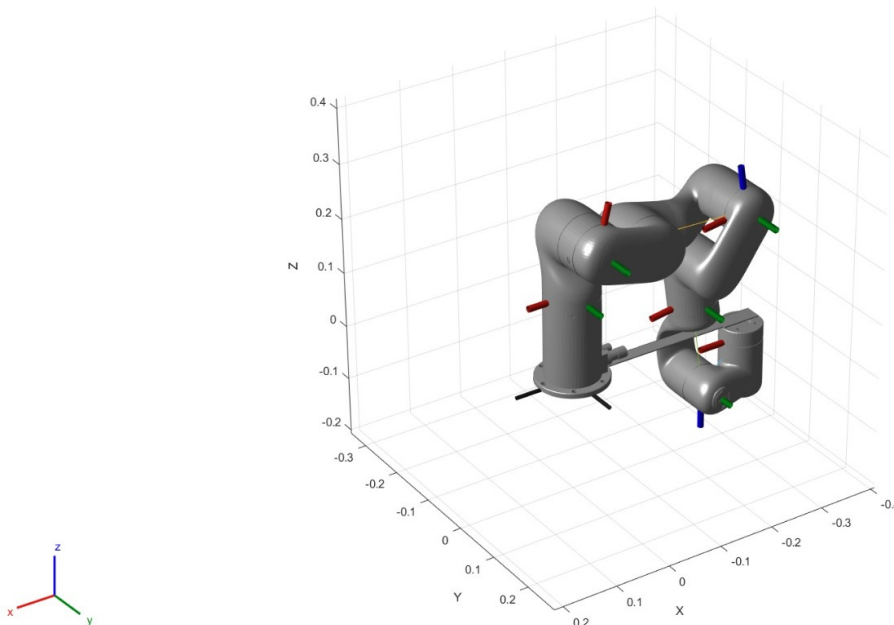
Input: A 1x6 array containing end effector position in cm and moving XYZ Euler angles in radians

Output: Joint angles if solution is found, else the message "Given position and orientation is not attainable". The corresponding robot pose is also displayed if a solution is found.

```
>> Q=inv_kin([0,0,0,0,0,0])
```

Q =

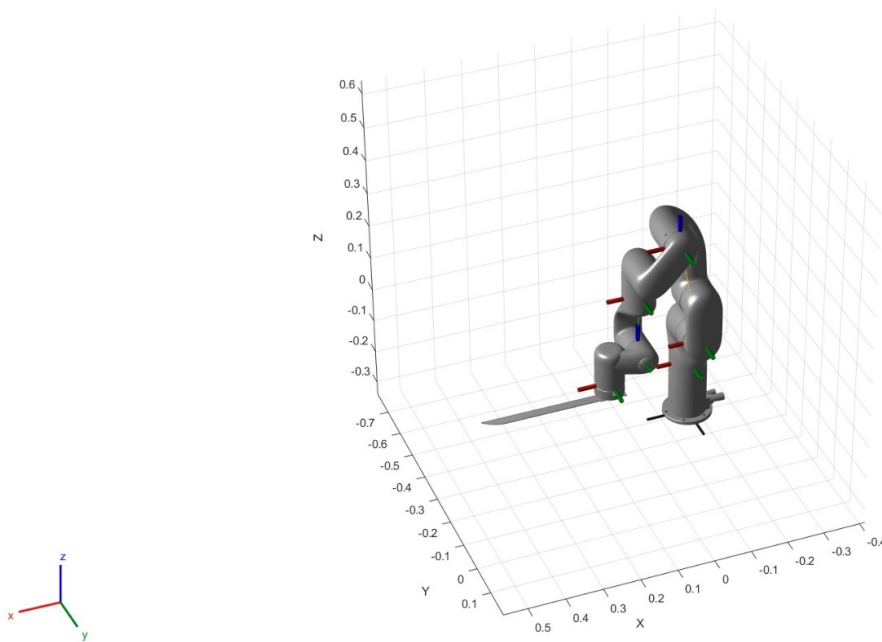
```
0.0000    -1.8124    1.9493    0.0000   -3.2785   -3.1416
```



## **Visualization:**

Robotic Systems Toolbox is used in this code for visualizing the robot poses. The 3D model of the robot is stored in the Xarm6\_description folder, which also contains other information such as the location and types of joints of the robot. The robot model is imported to MATLAB using the `importrobot()` function with the path to the .xacro file ( "Xarm6\_description/urdf/Xarm6.xacro") as the input. DataFormat parameter to the `importrobot` function is set to "row" for easier handling of the visualization code. The imported model, stored in the variable 'm6r', is then passed as a parameter to the `show()` function along with the joint angles vector for displaying the robot pose.

`show(m6r,[0,0,0,0,0,0] )` produces the following visualization of the home configuration of the robot.



## **Jacobian:**

The Jacobian of the system with respect to the base frame is computed using the geometric method. The  $Z_{i-1}$  and  $P_{i-1}$  required for the column of the jacobian matrix corresponding to each joint can be obtained from the Transformation matrix  ${}^0T_{i-1}$ .

$$J_i(\mathbf{q})_{(6 \times 1)} = \begin{bmatrix} J_{P,i}(\mathbf{q})_{(3 \times 1)} \\ J_{O,i}(\mathbf{q})_{(3 \times 1)} \end{bmatrix} = \begin{cases} \begin{bmatrix} \hat{Z}_{i-1} \\ \mathbf{0} \end{bmatrix} & \leftarrow \text{Prismatic joint } i \\ \begin{bmatrix} \hat{Z}_{i-1} \times (P_n - P_{i-1}) \\ \hat{Z}_{i-1} \end{bmatrix} & \leftarrow \text{Revolute joint } i \end{cases}$$

### Function Description:

/MATLAB Drive/FOR Project 2/Jacob.m

```

1 function jac = Jacob()
2 global DH;
3 Z=[0;0;1];
4 P=[0;0;0];
5 t=eye(4,4);
6 for i = 1:(height(DH)-1)
7     t = t*T(DH(i,:));
8     Z=[Z,t(1:3,3)];
9     P=[P,t(1:3,4)];
10 end
11 t=t*T(DH(height(DH),:));
12 Pn = t(1:3,4);
13 jac=[];
14 for i = 1:height(DH)
15     jac=[jac,[cross(Z(:,i),Pn - P(:,i));Z(:,i)]];
16 end
17 end
18

```

The Jacob() function calculates the Jacobian matrix for the robot using its Denavit-Hartenberg (DH) parameters. It initializes vectors for rotational (Z) and positional (P) elements and constructs transformation matrices for each joint. The matrices update Z and P with the robot's joint axes and positions. After processing all joints, it determines the end effector's position (Pn). The Jacobian is then formed by combining cross products of Z vectors and positional differences (for linear velocity) with Z vectors (for angular velocity) for each joint. This matrix crucially links joint angular velocities to the end effector's linear and angular velocities.

## Inverse Kinematics with Jacobian:

Once the Jacobian is computed, the joint angle velocity is computed by multiplying the jacobian inverse with the end effector velocity.

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}(\mathbf{q})\mathbf{V}$$

### Function Description:

```
function qd = joint_vel(pos,vel)

    %Obtaining jacobian matrix as a function
    j=Jacob();
    j=matlabFunction(j);
    j=@(x)(j(x(1),x(2),x(3),x(4),x(5),x(6)));

    q = inv_kin(pos); %obtaining joint angles correspondng to the given end effector position and orientation
    if isempty(q)
        return; %exits if the given position is not reachable
    else
        J = j(q);
        if det(J)==0 %singular point
            disp("Given position is a singular point");
        else
            qd = inv(J)*vel'; %computing joint rates
        end
    end
end
```

The joint\_vel function calculates the joint angular velocities (qd) of a robotic arm for a given end effector position (pos) and velocity (vel). It first obtains the Jacobian matrix as a function using Jacob(), and then determines the joint angles corresponding to the specified end effector position using inv\_kin(pos). If the position is reachable and not at a singular point (where the Jacobian determinant is not zero), the function computes the joint velocities by inverting the Jacobian and multiplying it with the end effector velocities (vel). If the position is unreachable or a singular point, the function displays an appropriate message and exits.

Input: 1\*6 array containing end effector position and XYZ Moving Euler angles and 1\*6 array containing end effector linear and angular velocity

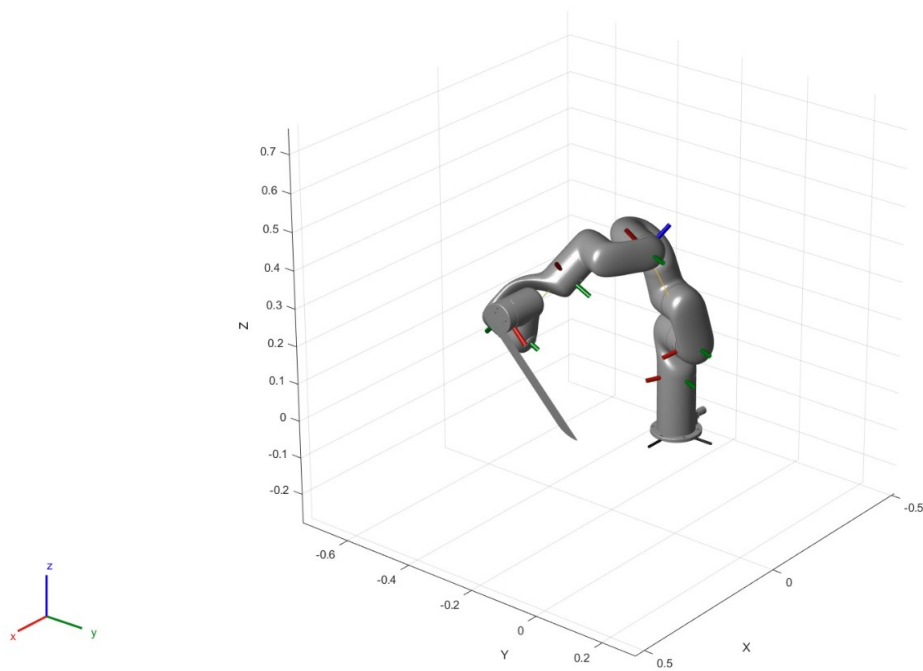
Output: 1\*6 array containing Joint angular velocities

```
>> qd = joint_vel([50,10,20,0,pi/2,pi/4],[10,0,0,0,0,0])
```

```
qd =
```

```
0.0881  
0.2833  
-0.3958  
0.0516  
0.1310  
0.0072
```

The function also produces a visual representation of the pose of the manipulator using the solution from the `inv_kin` function.



### **Inverse Dynamics with Jacobian:**

Once the Jacobian is computed, the joint torque is computed by multiplying the jacobian transpose with the force at the end effector.

$$\tau = -J^T F$$

## Function Description:

```
function F = joint_torques(pos,fext)

    %obtaining jacobian matrix as a function
    j=Jacob();
    j=matlabFunction(j);
    j=@(x)(j(x(1),x(2),x(3),x(4),x(5),x(6)));

    q = inv_kin(pos); %obtaining joint angles correspodng to the given end effector position and orientation
    if isempty(q)
        return; %exits if the given position is not reachable
    else
        J = j(q);
        if det(J)==0 %singular point
            disp("Given position is a singular point");
        else
            F = -J'*fext'; %computing joint torques
        end
    end
end
```

The function `joint_torques` computes the joint torques ( $F$ ) for a given end effector position , orientation and external forces/moments acting on it. It first derives the joint angles for the specified position and orientation using `inv_kin(pos)` and then obtains the Jacobian matrix using the `Jacob()` function. The function checks for reachability of the position and singularity (where the Jacobian determinant is zero). If the position is reachable and non-singular, it calculates the joint torques by multiplying the transpose of the Jacobian with the negative of the external forces/moments.

Input: 1\*6 array containing end effector position and XYZ Moving Euler angles and 1\*6 array containing force and moment at the end effector

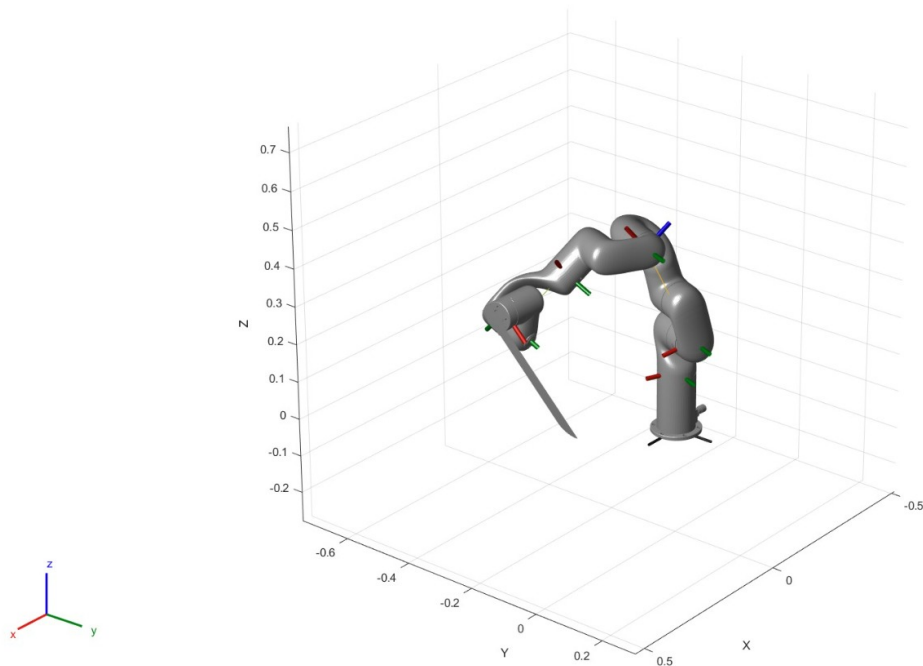
Output: 1\*6 array containing Joint Torques

```
>> Tau = joint_torques([50,10,20,0,pi/2,pi/4],[-10,0,0,0,0,0])
```

```
Tau =
```

```
-100.0000
-61.7575
-303.2866
222.7806
-39.9569
-0.0000
```

The function also produces a visual representation of the pose of the manipulator corresponding to the solution returned by `inv_kin` function.



### **Workspace:**

The workspace boundary of a robot is identified using the Jacobian matrix by:

1. Calculating the Jacobian for various joint configurations based on the robot's kinematics.
2. Evaluating the determinant of the Jacobian at each configuration; a non-zero determinant indicates feasible movement.
3. Identifying singularities where the determinant is zero, indicating restrictions in movement.
4. Mapping the end-effector positions for configurations with a viable Jacobian determinant.
5. The collection of these end-effector positions defines the robot's reachable workspace.



```

function ws = PlotWorkspace(res)
    syms q [1 6];
    j=(Jacob());
    j=matlabFunction(j);
    j=@(x)(j(x(1),x(2),x(3),x(4),x(5),x(6))); % det(j) was found to be independent of q6
    % j is now a function that computes det(Jacobian) for given joint angles

    pos = Tn(q);
    pos = matlabFunction(pos(1:3,4));
    pos = @(x)(pos(x(1),x(2),x(3),x(4),x(5),x(6))) ; % pos is now a function that computes [px,py,pz] for given joint angle inputs

    disp("Workspace computation will take long time!");

    % joint variables are varied within the joint limits provided by the manufacturer
    for q1=0:res:360
        for q2=-118:res:120
            for q3 = -225:res:11
                for q4 = 0:res:360
                    for q5=-97:res:180
                        for q6 = 0:res:360
                            if det(j([q1,q2,q3,q4,q5,q6])) <= 1e-5
                                p=pos([q1,q2,q3,q4,q5,q6].*(pi/180));
                                scatter3(p(1),p(2),p(3),'red');
                                hold on
                            end
                        end
                    end
                end
            end
        end
    end
    return;
end

```

### Function Description:

The PlotWorkspace function computes and visualizes the workspace of a robotic arm by iterating over possible joint angles within their limits as provided by the manufacturer. It uses a symbolic Jacobian matrix to determine the position reachability, checking if the determinant of the Jacobian is below a threshold, indicating singularity. For each combination of joint angles that passes this test, the function calculates the corresponding end-effector position and plots it in a 3D scatter plot. The workspace representation helps in understanding the areas in space that the robot can reach.

Input: Joint angle Increment size in degrees

Output: 3D Scatter plot of computed workspace