

Rohit Raibagkar
ECE/BME 7400
Winter 2017

Report of Individual Project Assignment 1



College of Engineering

Declaration : The project and the report are my own work
Date : 02/07/2017

Abstract

I address the modeling of a robot, its design and implementation, minimizing the errors between virtual model and actual robot (hardware). I present a virtual model of robot AESOP 1000. In this context, my goal is to define the D-H parameters and their importance in modeling and designing a robot. I do the modeling of the robot, AESOP 1000 using MATLAB. AESOP is 7 DOF robot. But as per the problem statement, first link of the AESOP 1000 has been omitted. Further, to understand dynamics, I implemented forward and inverse kinematics of AESOP 1000. The kinematics are very important to get clear idea about reach, limits and dexterousness of robotic arm. I have illustrated Jacobian and its role in kinematics, dynamics. To verify the results, I plotted reach envelop of AESOP, so that I will get clear idea of reach limit and dexterousness of AESOP 1000. Apart from dynamics and kinematics, communication using MATLAB socket by 'tcpip' command is discussed. Communication plays major role in control of robot arm from remote location. Comparison of both the methods of communication along with time delay and results has been discussed in details. The focus of the paper is discussing robot modeling, Communication, forward kinematics, inverse kinematics and errors.

Table of Contents

1.	Introduction	4
2.	Problem Statement	4
3.	D-H Parameters	5
4.	Forward and Inverse Kinematics	6
5.	Modeling in MATLAB Graphical User Interface	7
6.	Modeling of AESOP 1000	8
7.	Dexterousness of Arm	11
8.	Verification of Model with D-H Parameters	13
9.	Questions on Modeling	15
10.	Errors	18
11.	Conclusion	18
12.	Appendix.	
	A. Code of MATLAB GUI	19
	B. Video Link	37

1. Introduction

Robot and robotics have many important applications in human life. Medical robotics and surgeon assistant robot is one of them. Although medical robotics has not evolved that much, robot plays vital role in assisting surgeons in important tasks like image acquisition, image processing, carry out basic tasks like cavity removing, with very high accuracy.

Clinical applications have 5 types, intern replacements, telesurgical systems, navigational aids, precise positioning systems and precise path systems. As mentioned, a medical robot can simply be classified as MRI compatible systems, actually tool insertion assisting systems, surgery assisting systems. History of medical robotics has recorded successful application of robots in neurosurgery, orthopedic surgery, urology, maxillofacial, radiosurgery, ophthalmology and cardiac surgery.

Since these robots are used to operate on the most delicate part of the human or animal body, accuracy of these robots is the most desirable quality. In neuro surgery, if a robot is off by 1 mm, it can cause permanent damage to a particular functional part of brain. Even in maxillofacial surgery, aesthetic appearance is the main concern which can be damaged by offset of a millimeter.

The biggest task is testing the accuracy, errors and results with the standards. The modeling of the robot plays very important role in testing actual behavior of robot with certain standard inputs. We here represent modeling of robot in MATLAB GUI using D-H parameters. D-H parameters are set of constants that is fixed for robot in design. Details are discussed. In this project, Forward and inverse kinematics are also carried out on the designed model AESOP 1000. Results are discussed. Communication plays important role when controlling robot from remote locations. Way of carrying out communication using 'tcpip' command in MATLAB is discussed in separate section.

This report in detail, discusses about problem statement, Systematic approach, kinematics of robotic arm (dynamics), Role of Jacobian matrices, Modeling and verifying results using MATLAB GUI and conclusion.

2. Problem Statement

The approach towards the solution of design and modeling of robot is mentioned below.

- Define D-H parameters.

D-H parameters are constants of robot once it's designed. What is their significance, how they are defined is discussed in this section.

- Explain forward and inverse kinematics.

What is kinematics, what are their types, how they are applied on robotics arm, what are MATLAB commands to apply them on robotic arm is discussed in this section.

- Create MATLAB GUI that allows user to enter D-H parameters and create model of robot up to 6 DOF.

The picture of GUI screen and robots are provided here.

- Verify dexterousness of robot using reach envelop.

When we design a robot, estimation maximum reach of robotic arm is necessary. Also, verifying that modeled robot is behaving like actual D-H parameters is very important. This estimation and verification is carried out in this section.

- Create model of AESOP 100 robot and verify.
- Discuss results, errors and their effects.

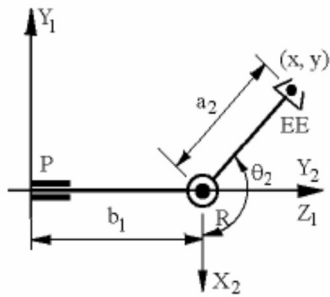
3. D-H Parameters

Denavit-Hartenberg notations or D-H parameters are the constants, of designed robot arm. Once we design robot arm, these parameters become constant for that arm.

There are usually four parameters that are necessary to be specified while defining and modeling a robotic arm. But depending on designer and modeling requirements, one can specify more parameters. Those parameters are :

- 1) Joint angle, Θ : Θ is angle of projection of two common normal X_i and X_{i+1} along z-axis at that joint. For revolute joint, Θ is variable, for prismatic joint, Θ is constant. Rotation will be positive in counter-clockwise direction and negative in clockwise direction.
- 2) Joint offset, d or b : It length of intersection of two common normal along the z-axis of the joint. It is distance between X_i and X_{i+1} along Z_i . If joint is prismatic, d is variable else d is constant.
- 3) Link length a : It is common normal between Z_i and Z_{i+1} along X_i . It is mutually perpendicular distance between two consecutive z-axis.
- 4) Alpha, α : It is angle between two orthogonal z-axis, along x-axis. If counter-clockwise, then alpha is positive, else if clockwise, alpha is negative.

Example below gives clear idea of D-H representation.



Link	b_i	θ_i	a_i	α_i
1	b_2 (JV)	0	0	$\pi/2$
2	0	θ_2 (JV)	a_2	$\pi/2$

If we define a robot in MATLAB using D-H parameters, It will look like below :

```
>> aesop
>> Aesop

Aesop =

Aesop1000 (6 axis, RRRRRR, stdDH, fastRNE)

+-----+-----+-----+-----+-----+
| j |   theta |       d |       a |   alpha |  offset |
+-----+-----+-----+-----+-----+
| 1 |    q1 |       0 |  0.3844 |       0 |       0 |
| 2 |    q2 |       0 |  0.0527 |  1.571 |       0 |
| 3 |    q3 |       0 |       0 |  1.571 |  1.571 |
| 4 |    q4 |  0.2507 |       0 |  1.571 |  3.142 |
| 5 |    q5 |       0 |  0.0172 | -1.571 |  1.571 |
| 6 |    q6 | -0.2754 |       0 |       0 | -1.571 |
+-----+-----+-----+-----+-----+

grav =    0  base = 1  0  0  0  tool = 1  0  0  0
        0          0  1  0  0          0  1  0  0
      9.81        0  0  1  0          0  0  1  0
              0  0  0  1          0  0  0  1

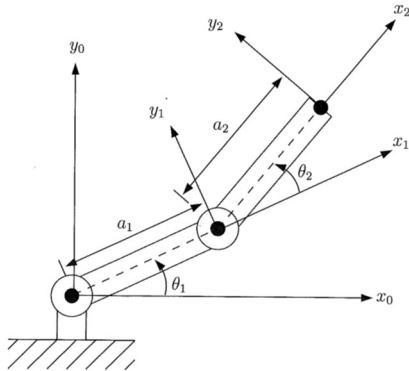
>>
```

4. Forward and Inverse Kinematics

Kinematics is science that deals with motion of the robot arm. The aim is, to determine the end effector position in 3D space. When we design a robot, it very important to know position and orientation of end effector in 3D space. Not only that, if we want to move our EE from one position to another, we must estimate set of joint angles of arm.

Forward Kinematics

When we know joint angles of the robot, we can calculate its end effector position using closed form geometric solutions. This method is called as forward kinematics. Let's take an example of two link arm,



Now to get the position of End Effector, we can simply write geometric equations as :

$$X_1 = a_1 \cos(\theta_1)$$

$$Y_1 = a_1 \sin(\theta_1)$$

$$X_2 = X_1 + a_2 \cos(\theta_1 + \theta_2)$$

$$Y_2 = Y_1 + a_2 \sin(\theta_1 + \theta_2)$$

In robotics mathematics or programming, the x y and z coordinates are represented along with their rotations in transformation Matrix.

```
Aesop.fkine([0 0 0 0 0 0])
```

ans =

0.0000	1.0000	0.0000	0.7050
-1.0000	0.0000	0.0000	-0.0000
0.0000	-0.0000	1.0000	-0.2754
0	0	0	1.0000

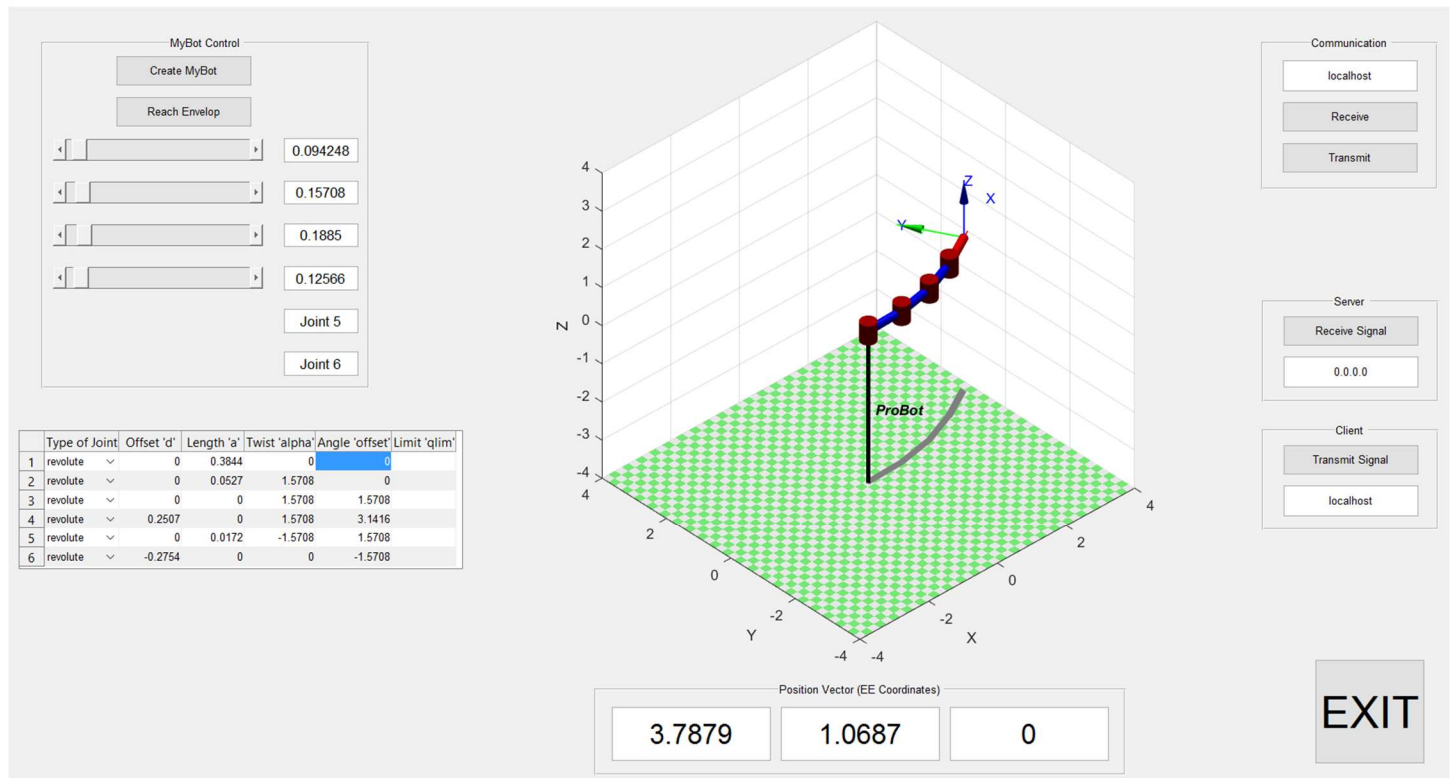
In the transformation matrix above, first three rows three columns represent rotational part and first three elements of 4th column represent co-ordinates of the end effector.

Inverse Kinematics

Inverse kinematics is, finding the corresponding joint angles when we know position and orientation of end effector. Like forward kinematics, inverse kinematics does not have closed form solution. It has some iterative solution. So solution is not fixed in the case of inverse kinematics. Multiple solutions exist for same problem.

5. Modeling in MATLAB Graphical User Interface

I have built robot in MATLAB. The GUI below shows features to input D-H parameters and control using sliders. End Effector coordinates are displayed below the robot model.



The GUI offers two ways to input D-H parameters : one by using prompt and other by using Table feature provided in GUI. If you want to create a simple robot of any size and parameters, you can use Enter Manually option which will pop-up once you mention DOF of system.

Enter type of joint (revolute or prismatic)
revolute

Enter Joint Angle, theta (in radians)
0

Enter Link Offset, d (in meters)
0

Enter Link Length, a (in meters)
0

Enter alpha (in radians)
0

Enter Offset angle (in radians)
0

Enter Joint Limits (in radians)
0

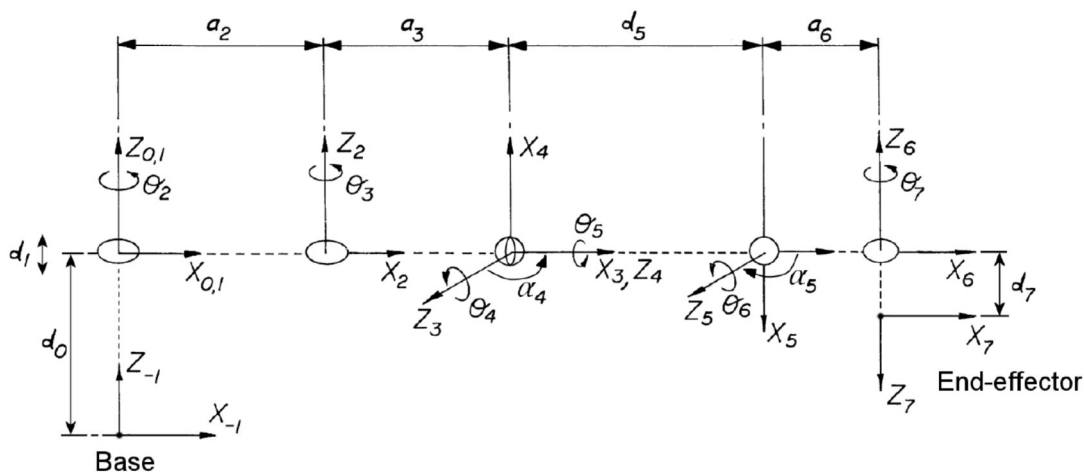
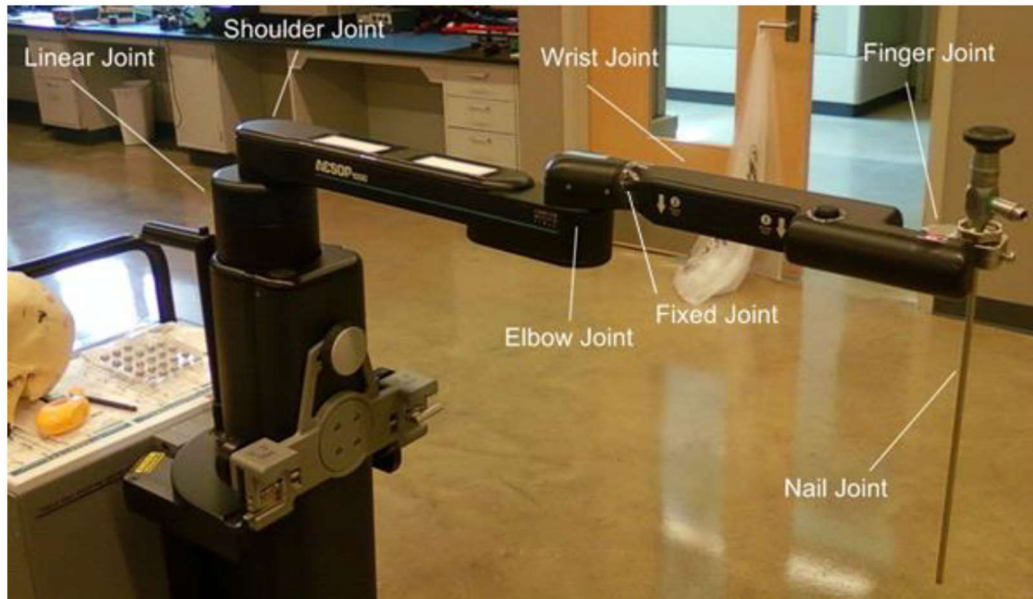
OK Cancel

The prompt that will ask you for D-H parameters is displayed above. The prompt will receive data and store it in array. The data conversion is must once you enter data, since entered data will be in string or cell format.

6. Modeling of AESOP

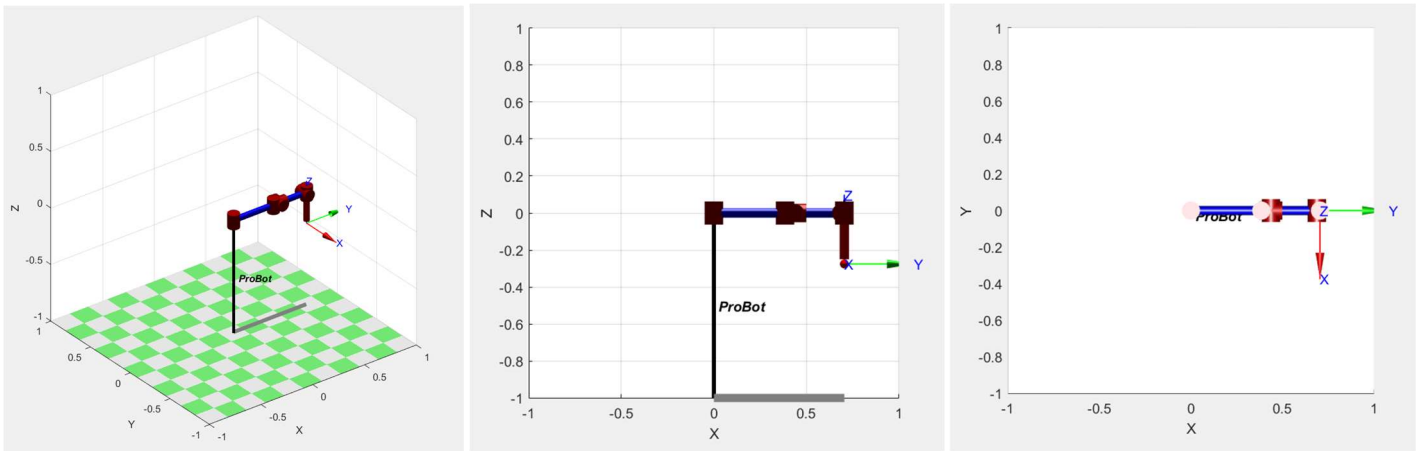
Aesop 1000 is 6 DOF robotic arm. Fig below shows construction, joints description and D-H Parameters of AESOP 1000.

Standard Denavit–Hartenberg Model of the AESOP 1000



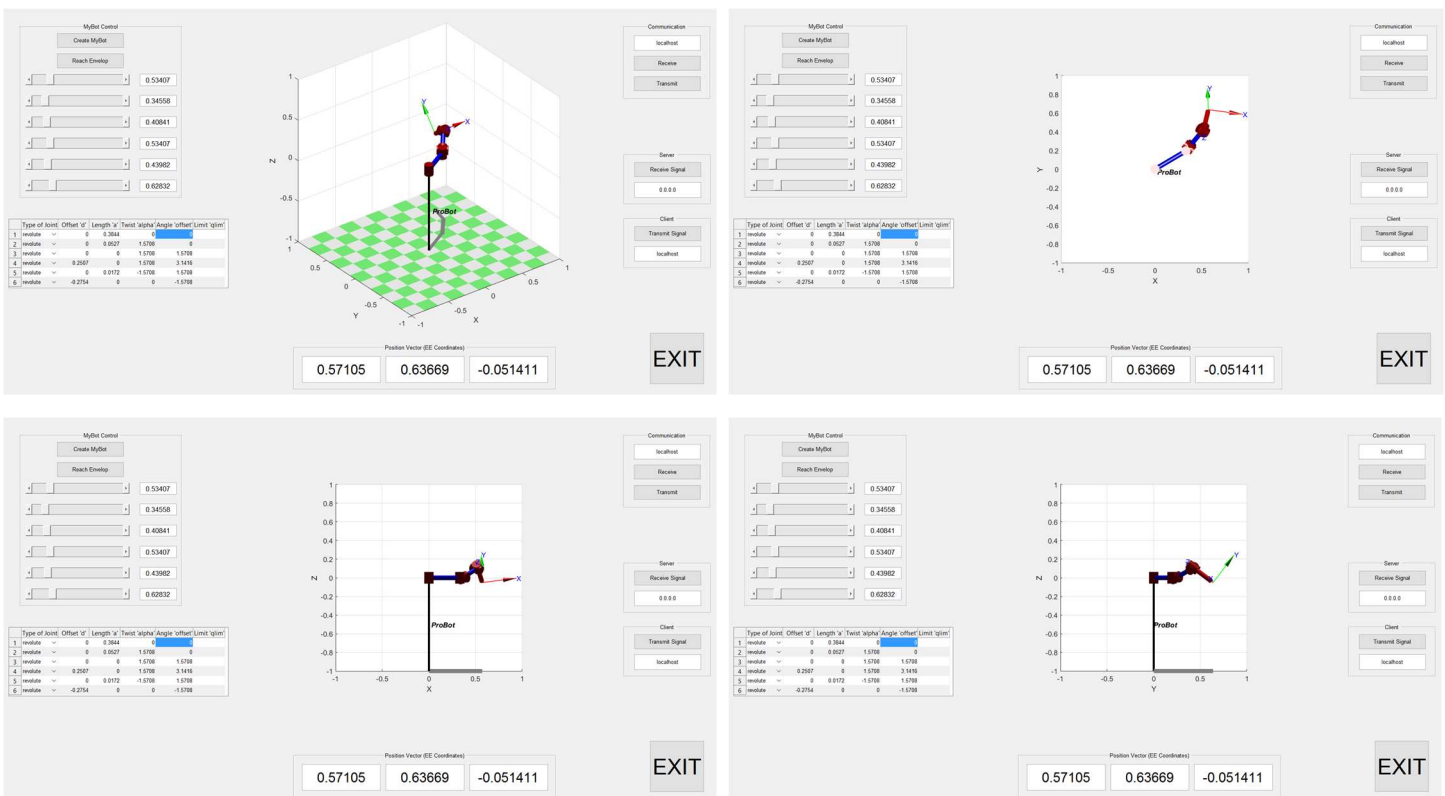
Joint, i	α_i (rad)	a_i (mm)	θ_i (rad)	d_i (mm)
1	0.0000	0.000	0.0000	$d_1^{\text{var}} + 0.000$
2	0.0000	384.440	$\theta_2^{\text{var}} + 0.0000$	0.000
3	1.5708	52.710	$\theta_3^{\text{var}} + 0.0000$	0.000
4	1.5708	0.000	1.5708	0.000
5	1.5708	0.000	$\theta_5^{\text{var}} + 3.1416$	250.700
6	-1.5708	17.200	$\theta_6^{\text{var}} + 1.5708$	0.000
7	0.0000	0.000	$\theta_7^{\text{var}} - 1.5708$	-275.420

It shows that there are 7 joints. But considering the problem statement, we have modeled only 6 joints, joint 2 to joint 6 in MATLAB. Discussed below are the images of AESOP in MATLAB GUI. The D-H parameters mentioned above, we can either input it manually or enter them in table.



The Figure above represent the model of AESOP 100 in MATLAB GUI. Below are the two poses of AESOP 1000 arm.

Position 1.



Note that in the above figure, position of the end effector is varied using the sliders provided in the GUI. Sliders provide joint angles as input to the robot. At the bottom of GUI, position of the end effector is displayed.

Position 2

MyBot Control

Create MyBot

Reach Envelope

1.9321

2.0106

2.3719

2.1677

2.152

2.042

Type of Joint: Offset (r) Length (r) Twist (alpha) Angle (offset) Limit (q/min)

1. revolute 0 0.3844 0 0 0

2. revolute 0 0.8927 1.5708 0 0

3. revolute 0 0 1.5708 1.5708 0

4. revolute 0.2587 0 1.5708 3.1415 0

5. revolute 0 0.0112 -1.5708 1.5708 0

6. revolute -0.2754 0 0 -1.5708 0

Communication

localhost

Receive

Transmit

Server

Receive Signal

0.0.0.0

Client

Transmit Signal

localhost

Position Vector (EE Coordinates)

-0.080386

0.61369

0.39493

EXIT

MyBot Control

Create MyBot

Reach Envelope

1.9321

2.0106

2.3719

2.1677

2.152

2.042

Type of Joint: Offset (r) Length (r) Twist (alpha) Angle (offset) Limit (q/min)

1. revolute 0 0.3844 0 0 0

2. revolute 0 0.8927 1.5708 0 0

3. revolute 0 0 1.5708 1.5708 0

4. revolute 0.2587 0 1.5708 3.1415 0

5. revolute 0 0.0112 -1.5708 1.5708 0

6. revolute -0.2754 0 0 -1.5708 0

Communication

localhost

Receive

Transmit

Server

Receive Signal

0.0.0.0

Client

Transmit Signal

localhost

Position Vector (EE Coordinates)

-0.080386

0.61369

0.39493

EXIT

MyBot Control

Create MyBot

Reach Envelope

1.9321

2.0106

2.3719

2.1677

2.152

2.042

Type of Joint: Offset (r) Length (r) Twist (alpha) Angle (offset) Limit (q/min)

1. revolute 0 0.3844 0 0 0

2. revolute 0 0.8927 1.5708 0 0

3. revolute 0 0 1.5708 1.5708 0

4. revolute 0.2587 0 1.5708 3.1415 0

5. revolute 0 0.0112 -1.5708 1.5708 0

6. revolute -0.2754 0 0 -1.5708 0

Communication

localhost

Receive

Transmit

Server

Receive Signal

0.0.0.0

Client

Transmit Signal

localhost

Position Vector (EE Coordinates)

-0.080386

0.61369

0.39493

EXIT

MyBot Control

Create MyBot

Reach Envelope

1.9321

2.0106

2.3719

2.1677

2.152

2.042

Type of Joint: Offset (r) Length (r) Twist (alpha) Angle (offset) Limit (q/min)

1. revolute 0 0.3844 0 0 0

2. revolute 0 0.8927 1.5708 0 0

3. revolute 0 0 1.5708 1.5708 0

4. revolute 0.2587 0 1.5708 3.1415 0

5. revolute 0 0.0112 -1.5708 1.5708 0

6. revolute -0.2754 0 0 -1.5708 0

Communication

localhost

Receive

Transmit

Server

Receive Signal

0.0.0.0

Client

Transmit Signal

localhost

Position Vector (EE Coordinates)

-0.080386

0.61369

0.39493

EXIT

7. Dexterousness of Arm

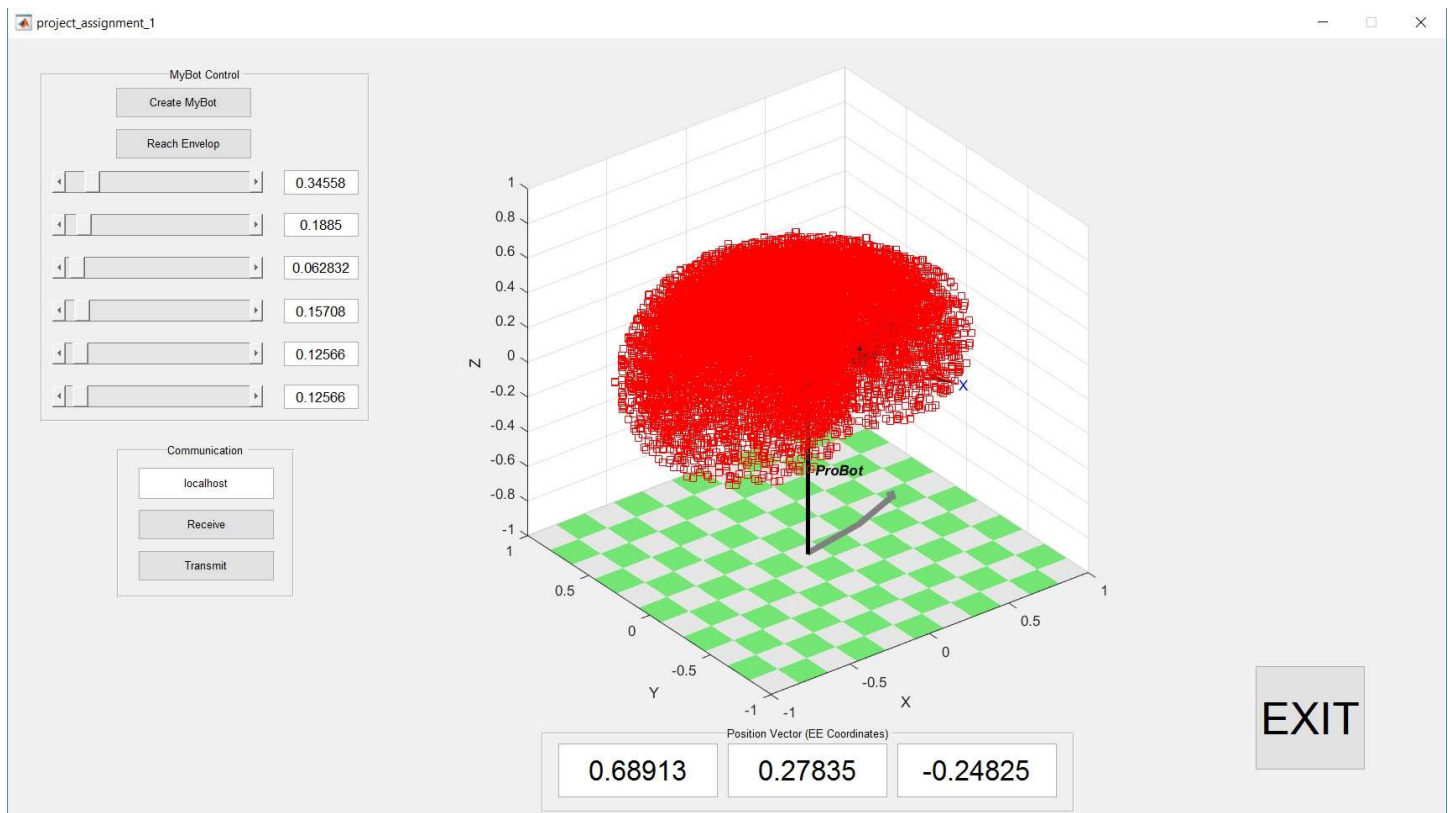
The reach or flexibility of arm can be estimated by using reach envelop. It uses nested for loops to get EE coordinates at slightly increasing joint angles, in steps. A sample nested for loop example for two-link robot is shown below.

```
for j1 = 0:.25:pi % setting loop and step increment of joint angle for joint 1...
    for j2 = 0:.25:pi % setting nested loop and step increment of joint angle for joint 2...
        T = handles.MyBot.fkine([j1, j2]);
        P = T(1:3,4);
        hold on;
        handles.MyBot.plot([j1, j2]);
        plot3(P(1), P(2), P(3), '--rs');
    end % nested loop for joint 2 is ended...
end % end of the loop for reach envelop, for 2 DOF system...
```

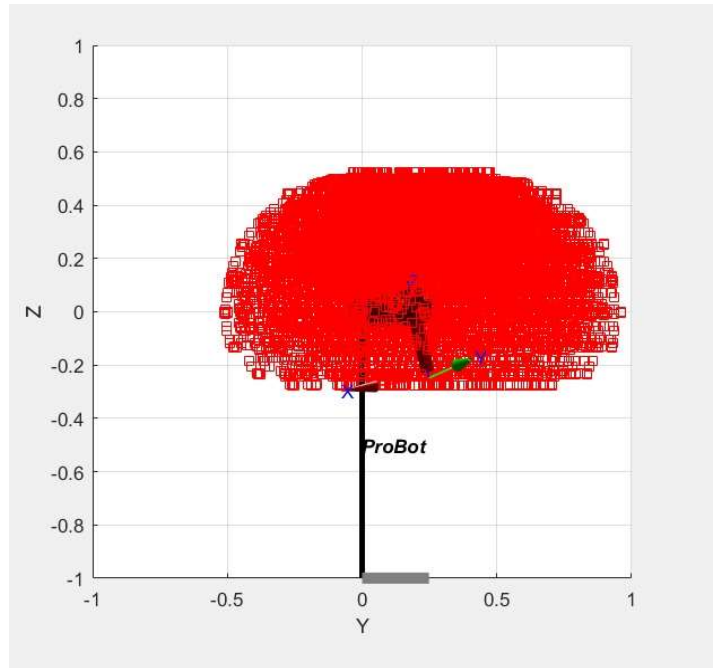
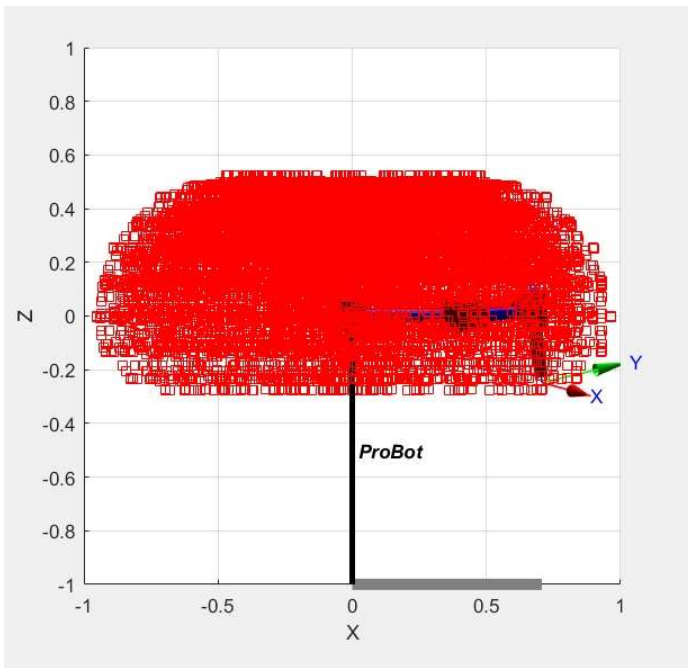
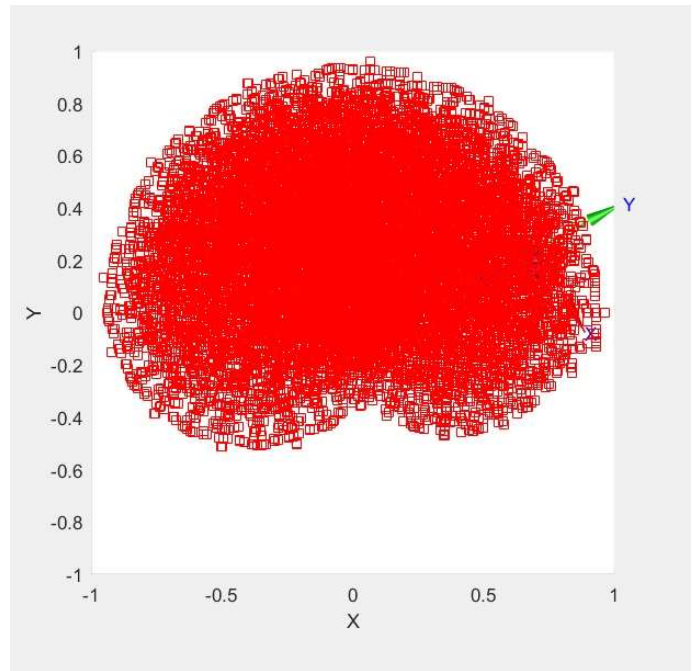
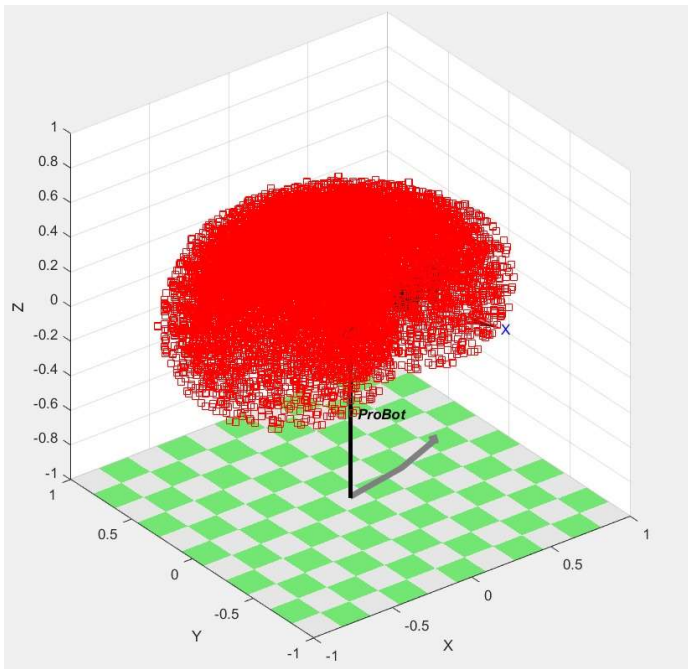
Since AESOP we modeled is 6 DOF arm, we should provide 6 nested for loops to get reach envelop. It uses fkine function of robotic toolbox, which gives transformation matrix with end effector coordinates. We hold them on graph using hold command option in MATLAB. Then we plot them on the graph with red square.

Below are the reach envelop images of AESOP 1000.

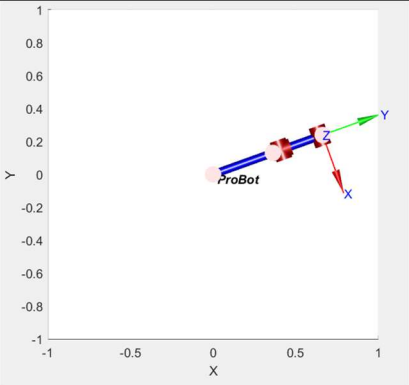
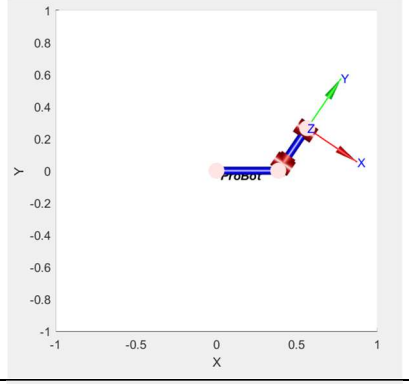
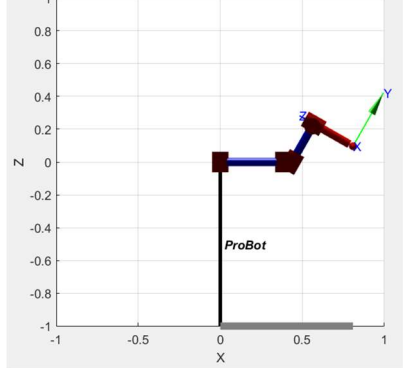
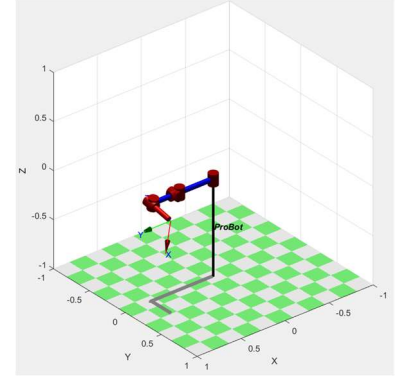
Reach Envelop of AESOP 1000

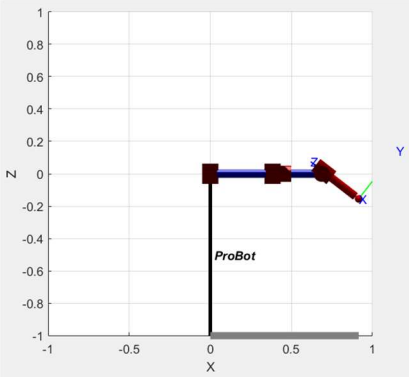
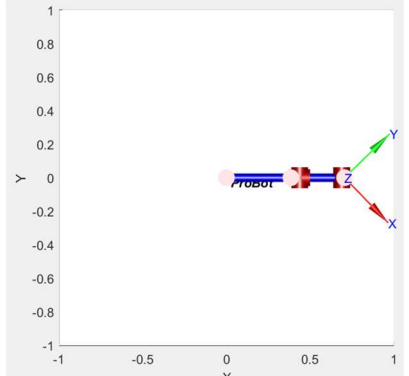


Figures below show xy view, xz view and yz view of



8. Verification of Model with D-H Parameters

Joint Type	Input Variable	Image	Working as defined in D-H Parameters?
Joint 1 : Revolute	Joint angle, Θ Using Slider 1 and keeping all other joint angles (sliders) at zero.		Yes
Joint 2 : Revolute	Joint angle, Θ Using Slider 2 and keeping all other joint angles (sliders) at zero.		Yes
Joint 3 : Revolute	Joint angle, Θ Using Slider 3 and keeping all other joint angles (sliders) at zero.		Yes
Joint 4 : Revolute	Joint angle, Θ Using Slider 4 and keeping all other joint angles (sliders) at zero.		Yes

Joint 5 : Revolute	Joint angle, Θ Using Slider 5 and keeping all other joint angles (sliders) at zero.		Yes
Joint 6 : Revolute	Joint angle, Θ Using Slider 6 and keeping all other joint angles (sliders) at zero.		Yes

9. Questions on Modeling

1) How Jacobian matrices are used in computation of kinematics and evaluation of robotic designs?

Jacobian is matrix of partial derivatives. When we want to estimate force or the velocity of end effector by giving only one joint angle as input and keeping other constant, we must take partial derivative of Matrix.

$$J = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \dots & \frac{\partial F_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1} & \dots & \frac{\partial F_m}{\partial x_n} \end{bmatrix}$$

The Jacobian Matrix is shown above. As we move from left to right, EE displacement is varied while as we move from top to bottom, force exerted by end effector is varied.

In robotics, with the number of degrees of freedom, the Jacobian can give us the estimate of x, y, z co-ordinates and rotation matrix ie. Roll, pitch and yaw. If we have 150 DOF system, the Jacobian will be 150 X 6.

In Inverse kinematics, general iterative solutions involves some math iterations.

$v = J(Q) * Q'$ where Q' is derivative. The approach is based on Jacobian transpose. Jacobian Transpose can be computed for any joint orientation.

Jacobian and manipulability for AESOP 1000 can be computed as

```
>> J = Aesop.jacob0([pi/2 pi/4 pi/3 pi/4 pi/6 pi/3])
```

J =

-0.5489	-0.1645	0.1895	-0.0154	0.0611	0
-0.3944	-0.3944	-0.1895	-0.2145	0.0915	0
0	0.0000	0.3424	0.0813	0.2531	0
0.0000	0.0000	0.7071	-0.3536	0.0670	0.9848
-0.0000	-0.0000	0.7071	0.3536	0.9330	-0.1188
1.0000	1.0000	0	0.8660	-0.3536	-0.1268

```
>> Aesop.manipilty([0 0 0 0 0 0])
```

ans =

0.0568

2) What is the Difference between forward and inverse kinematics?

Forward Kinematics

When we know joint angles of the robot, we can calculate its end effector position using closed form geometric solutions. This method is called as forward kinematics. Forward kinematics use multiplication of transformation matrices. Forward kinematics is closed form solution. An fkine command in MATLAB will give you Transformation matrix as:

```
>> Aesop.fkine([0 0 0 0 0 0])
ans =
    0.0000    1.0000    0.0000    0.7050
   -1.0000    0.0000    0.0000   -0.0000
    0.0000   -0.0000    1.0000   -0.2754
         0         0         0    1.0000
```

Inverse Kinematics

Inverse kinematics is, finding the corresponding joint angles when we know position and orientation of end effector. Like forward kinematics, inverse kinematics does not have closed form solution. It has some iterative solution. So solution is not fixed in the case of inverse kinematics. Multiple solutions exist for same problem. Jacobian is used to compute iterative solution. An ikine option in MATLAB will give joint angles as:

```
>> T = transl(1,1,1)
```

```
T =
     1     0     0     1
     0     1     0     1
     0     0     1     1
     0     0     0     1
```

```
>> Aesop.ikine(T)
```

```
Warning: Initial joint configuration results in a (near-)singular configuration, this may slow convergence
```

```
> In SerialLink/ikine (line 156)
```

```
Warning: solution diverging at step 523, try reducing alpha
```

```
> In SerialLink/ikine (line 260)
```

```
Warning: ikine: iteration limit 1000 exceeded (row 1), final err 3.170588
```

```
> In SerialLink/ikine (line 179)
```

```
ans =
```

```
1.0e+05 *
    0.0790   -0.3173    0.0534   -1.5419   -0.2259   -0.1877
```

```
>> Aesop.ikine(T, [0 0 0 0 0 0])
```

```
Warning: Initial joint configuration results in a (near-)singular configuration, this may slow convergence
```

```
> In SerialLink/ikine (line 156)
```

```
Warning: solution diverging at step 523, try reducing alpha
```

```
> In SerialLink/ikine (line 260)
```

```
Warning: ikine: iteration limit 1000 exceeded (row 1), final err 3.170588
```

```
> In SerialLink/ikine (line 179)
```

```
ans =
```

```
1.0e+05 *
```


0.0790 -0.3173 0.0534 -1.5419 -0.2259 -0.1877

3) How a model of robot used will be useful?

A model of robot can be useful to improve the accuracy of the surgery. Like paper 2 represented by group 1, Robot can be used for the needle insertion and cement injection purposes. Like paper 1, Robot can be used for minimal invasive single port surgery.

4) How to verify accuracy of robot?

Accuracy of the Robot can be given by ratio of Error with required result.

% error = (magnitude of deflection of EE / Results obtained from MATLAB) X 100.

10. Errors

Joint No.	'a' varied by 1 mm	'd' varied by 1 mm	Error in EE co-ordinates
1	.38444 to .38544	NA	Error in X by 1 mm
2	.05271 to .05371	NA	Error in X by 1 mm
3	NA	NA	NA
4	NA	.2507 to .2517	Error in Z by 1 mm
5	.0172 to .0182	NA	Error in X by 1 mm
6	NA	-.2754 to -.2764	Error in Z by 1 mm

11. Conclusion

We thus have successfully modeled AESOP 1000 and managed to model custom robot defined by user. We successfully tested the results. We successfully verified fkine and ikine results with MATLAB. We calculated Jacobian using MATLAB for AESOP 1000. The results accurately match with Model of robot. Also, Communication with robot using MATLAB carried out successfully.

12. Appendix

A. Code for MATLAB GUI.

```
function varargout = project_assignment_1(varargin)
% PROJECT_ASSIGNMENT_1 MATLAB code for project_assignment_1.fig
% PROJECT_ASSIGNMENT_1, by itself, creates a new PROJECT_ASSIGNMENT_1 or raises the existing
% singleton*.
%
% H = PROJECT_ASSIGNMENT_1 returns the handle to a new PROJECT_ASSIGNMENT_1 or the handle to
% the existing singleton*.
%
% PROJECT_ASSIGNMENT_1('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in PROJECT_ASSIGNMENT_1.M with the given input arguments.
%
% PROJECT_ASSIGNMENT_1('Property','Value',...) creates a new PROJECT_ASSIGNMENT_1 or raises
the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before project_assignment_1_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to project_assignment_1_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help project_assignment_1

% Last Modified by GUIDE v2.5 11-Feb-2017 00:01:53

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn', @project_assignment_1_OpeningFcn, ...
                  'gui_OutputFcn',  @project_assignment_1_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before project_assignment_1 is made visible.
function project_assignment_1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin    command line arguments to project_assignment_1 (see VARARGIN)

% Choose default command line output for project_assignment_1
handles.output = hObject;
handles.ip = 'localhost'; % setting the default value of ip address to be connect to...
handles.transmitter_ip = '0.0.0.0';
handles.receiver_ip = 'localhost';
global parameters;
% Update handles structure
guidata(hObject, handles);
```

```

% UIWAIT makes project_assignment_1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = project_assignment_1_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in create_MyBot.
function create_MyBot_Callback(hObject, eventdata, handles)
% hObject handle to create_MyBot (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% function to get value for setting loop...

% Following program will prompt user to enter required degrees of freedom.
% As soon as user enters number, the robot having DOF entered by the user
% will be created....

prompt = {'Enter desired number of DOF system, less than or equal to 6..'}; % Setting the dialog
for dialogbox suggesting the user to input number of DOF..
title = 'DOF'; % setting the title of the dialogbox
num_lines = [1 50]; % an option to set line and columns of input line...
default_ans = {'0'}; % setting default answer of input line...
options.Resize = 'on'; % setting 'resize' parameter as 'on' of option to resize dialog box...
options.WindowStyle = 'normal'; % setting 'window style' parameter as 'normal' of option to set the
windowstyle...
options.Interpreter='tex'; % setting 'interpreter' parameter as 'tex' of option to prompt strings
as rendered...
numDOF = inputdlg(prompt, title, num_lines, default_ans, options); % Now getting input entered by
user in the variable 'numDOF' as 'string'....
numDOF = str2num(numDOF{:}); % since input by user is in string format, converting user input to
number & updating the variable 'numDOF' as number...
handles.numDOF = numDOF; % storing number of DOF in handles structure so that it can be used
anywhere in the program, especially in
% reach envelop callback function...

% following code is to check whether user has entered correct and valid
% number of DOF and generate error message...

if numDOF == 0 % checking if number input by user is zero...
    errordlg('Please enter number of DOF less than or equal to 6 ...', 'DOF not entered !!!',
'modal');
    % an error message of 'not entered' will be generated..
    return % and MATLAB will exit the loop, stop the script and return control to the invoking
function or command prompt..

elseif numDOF > 6 % checking if number input by user is greater than 6...
    errordlg('Please enter number of DOF less than or equal to 6 ...', 'DOF exceeded max limit
!!!', 'modal');
    % an error message of 'number of DOF Exceeded' will be generated...
    return % and MATLAB will exit the loop, stop the script and return control to the invoking
function or command prompt..

elseif numDOF < 0 % checking if number input by user is less than 0...
    errordlg('Please enter a valid positive integer between 1 and 6 ...', 'Invalid DOF !!!',
'modal');
    % an error message of 'invalid number of DOF' will be generated...
    return % and MATLAB will exit the loop, stop the script and return control to the invoking
function or command prompt..
end % loop to check the input value is ended...

```

```

% Following program checks the number of DOF entered by the user and
% disables remaining sliders, leaving number of sliders equal to number of
% DOF active...

if numDOF == 5 % check whether number of DOF entered by User is 5...
    set(handles.joint_6, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    6...
elseif numDOF == 4 % check whether number of DOF entered by User is 4...
    set(handles.joint_6, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    6...
    set(handles.joint_5, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    5...
elseif numDOF == 3 % check whether number of DOF entered by User is 3...
    set(handles.joint_6, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    6...
    set(handles.joint_5, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    5...
    set(handles.joint_4, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    4...
elseif numDOF == 2 % check whether number of DOF entered by User is 2...
    set(handles.joint_6, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    6...
    set(handles.joint_5, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    5...
    set(handles.joint_4, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    4...
    set(handles.joint_3, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    3...
elseif numDOF == 1 % check whether number of DOF entered by User is 1...
    set(handles.joint_6, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    6...
    set(handles.joint_5, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    5...
    set(handles.joint_4, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    4...
    set(handles.joint_3, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    3...
    set(handles.joint_2, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    2...
elseif numDOF == 0 % check whether number of DOF entered by User is 0...
    set(handles.joint_6, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    6...
    set(handles.joint_5, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    5...
    set(handles.joint_4, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    4...
    set(handles.joint_3, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    3...
    set(handles.joint_2, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    2...
    set(handles.joint_1, 'Enable', 'off', 'Visible', 'off'); % disable slider controlling joint
    1...
end % End of program to check DOF and disable sliders...

% Program to get value from the user and creating robot...
% The following loop will receive D-H parameter values from the user and
% will store in variable LinkParameters...

choice = questdlg('How would you like to enter D-H parameters of your model??',...
    'Select option to enter D-H Parameters', 'Enter Manually',...
    'Use Table', 'Save me from Robotics..!!!', 'Enter Manually')

switch choice
    case 'Enter Manually'
        for n = 1:1:numDOF % the loop is set to receive the D-H parameters from the user,
            repeatedly, repetitions equal to number of DOF...
                prompt = {'Enter type of joint (revolute or prismatic)', % The prompt option to get
                    type of joint...

```

```

'theta'...      'Enter Joint Angle, theta (in radians)', % The Prompt option to get Joint Angle,
'd'...          'Enter Link Offset, d ( in meters )', % The Prompt option to get Link offset,
'a'...          'Enter Link Length, a ( in meters )', % The Prompt option to get Link length,
                'Enter alpha ( in radians )', % The Prompt option to get Link twist, 'alpha'...
                'Enter Offset angle ( in radians )', % The Prompt option to get offset angle,
'offset'...     'Enter Joint Limits ( in radians )'; % The Prompt option to get Link joint limit,
'qlim'...
    title = 'D-H Parameters'; % setting the title of the dialog box...
    num_lines = [1 50]; % setting the number of lines and columns of each prompt...
    default_ans = {'revolute', '0', '0', '0', '0', '0', '0'}; % setting the default answer
of each answer...
    LinkParameters = inputdlg(prompt,title,num_lines,default_ans); % storing the received
values in variable called 'LinkParameters'...

    % Since the input values are string format, we have to convert them in number format or
character format to use them...

    type = char(LinkParameters{1,:}); % converting the type of the joint entered by the
user into 'character' format...
    type = uint16(type); % converting the characters stored in variable 'type' to 16-bit
unsigned integers
    % and updating variable 'type'...
    type(:,9) = 0; % setting the 9th element of matrix to zero to avoid index mismatch
error...
    angle = str2num(LinkParameters{2,:}); % converting 'angle' parameter to number format
from string format...
    d = str2num(LinkParameters{3,:}); % converting link offset, 'd' parameter to number
format from string format...
    a = str2num(LinkParameters{4,:}); % converting link length, 'a' parameter to number
format from string format...
    alpha = str2num(LinkParameters{5,:}); % converting link twist, 'alpha' parameter to
number format from string format...
    offset = str2num(LinkParameters{6,:}); % converting offset, 'offset' parameter to
number format from string format...
    qlim = str2num(LinkParameters{7,:}); % converting joint limit, 'qlim' parameter to
number format from string format...
    if type == [114 101 118 111 108 117 116 101 0] % checking whether user has entered
'revolute' as link type...
        L(n) = Link('revolute', 'd', d, 'a', a, 'alpha', alpha, 'offset', offset, 'qlim',
qlim); % creating a revolute link...
    elseif type == [112 114 105 115 109 97 116 105 0] % checking whether user has entered
'prismatic' as link type...
        L(n) = Link('prismatic', 'd', d, 'a', a, 'alpha', alpha, 'offset', offset);%,
creating a prismatic link...'qlim', qlim, 'theta', angle,
        MyBot.plotopt = {'workspace', [-2 2 -2 2 -2 2]};
    end % condition to check the type of link has ended here...
    handles.j(n) = 0; % initiating the handles structure with number of elements equal to
number of degrees of freedom...
    end % the loop set to receive the D-H parameters from the user and creating robot has
ended...

case 'Use Table'
    global parameters;
    for n = 1:1:numDOF
        type = parameters{n,1};
        type = uint16(type);
        type(:,9) = 0;
        if type == [114 101 118 111 108 117 116 101 0]
            L(n) = Link('revolute', 'd', parameters{n,2}, 'a', parameters{n,3}, 'alpha',
parameters{n,4}, 'offset', parameters{n,5}, 'qlim', parameters{n,6});
        elseif type == [112 114 105 115 109 97 116 105 0]
            L(n) = Link('prismatic', 'd', parameters{n,2}, 'a', parameters{n,3}, 'alpha',
parameters{n,4}, 'offset', parameters{n,5});
            MyBot.plotopt = {'workspace', [-2 2 -2 2 -2 2]};

```

```

        end
        handles.j(n) = 0;
    end

    case 'Save me from Robotics..!!!'
        return
    end

%for n = 1:1:numDOF % the loop is set to receive the D-H parameters from the user, repeatedly,
repeations equal to number of DOF...
%    prompt =    {'Enter type of joint (revolute or prismatic)', % The prompt option to get type of
joint...
%                'Enter Joint Angle, theta (in radians)', % The Prompt option to get Joint Angle,
'theta'...
%                'Enter Link Offset, d ( in meters )', % The Prompt option to get Link offset,
'd'...
%                'Enter Link Length, a ( in meters )', % The Prompt option to get Link length,
'a'...
%                'Enter alpha ( in radians )', % The Prompt option to get Link twist, 'alpha'...
%                'Enter Offset angle ( in radians )', % The Prompt option to get offset angle,
'offset'...
%                'Enter Joint Limits ( in radians )'}; % The Prompt option to get Link joint limit,
'qlim'...
%    title = 'D-H Parameters'; % setting the title of the dialog box...
%    num_lines = [1 50]; % setting the number of lines and columns of each prompt...
%    default_ans = {'revolute', '0', '0', '0', '0', '0', '0'}; % setting the default answer of each
answer...
%    LinkParameters = inputdlg(prompt,title,num_lines,default_ans); % storing the received values
in variable called 'LinkParameters'...
%
%    % Since the input values are string format, we have to convert them in number format or
character format to use them...
%
%    type = char(LinkParameters{1,:}); % converting the type of the joint entered by the user into
'character' format...
%    type = uint16(type); % converting the characters stored in variable 'type' to 16-bit unsigned
integers
%    % and updating variable 'type'...
%    type(:,9) = 0; % setting the 9th element of matrix to zero to avoid index mismatch error...
%    angle = str2num(LinkParameters{2,:}); % converting 'angle' parameter to number format from
string format...
%    d = str2num(LinkParameters{3,:}); % converting link offset, 'd' parameter to number format
from string format...
%    a = str2num(LinkParameters{4,:}); % converting link length, 'a' parameter to number format
from string format...
%    alpha = str2num(LinkParameters{5,:}); % converting link twist, 'alpha' parameter to number
format from string format...
%    offset = str2num(LinkParameters{6,:}); % converting offset, 'offset' parameter to number
format from string format...
%    qlim = str2num(LinkParameters{7,:}); % converting joint limit, 'qlim' parameter to number
format from string format...
%    if type == [114 101 118 111 108 117 116 101 0] % checking whether user has entered 'revolute'
as link type...
%        L(n) = Link('revolute', 'd', d, 'a', a, 'alpha', alpha, 'offset', offset, 'qlim', qlim); %
creating a revolute link...
%    elseif type == [112 114 105 115 109 97 116 105 0] % checking whether user has entered
'prismatic' as link type...
%        L(n) = Link('prismatic', 'd', d, 'a', a, 'alpha', alpha, 'offset', offset);%, creating a
prismatic link...'qlim', qlim, 'theta', angle,
%        MyBot.plotopt = {'workspace', [-2 2 -2 2]};
%    end % condition to check the type of link has ended here...
%    handles.j(n) = 0; % initiating the handles structure with number of elements equal to number
of degrees of freedom...
%end % the loop set to receive the D-H parameters from the user and creating robot has ended...

```

```

handles.MyBot = SerialLink(L, 'name', 'ProBot'); % creating robot with links created and storing it
in handles structure...
handles.MyBot.plot(handles.j); % plotting the robot with initial joint angles as input...
zoom on; % setting zoom option to magnify the diagram...

% Updating the handles structure...
guidata(hObject,handles);

% --- Executes on slider movement.
function joint_1_Callback(hObject, eventdata, handles)
% hObject      handle to joint_1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
handles.j(1) = get(hObject, 'Value'); % getting slider value and storing in first element of joint
angle set array...
set(handles.joint_1_disp, 'String', num2str(handles.j(1))); % converting slider 1 value to string
and setting it in callback variable
% of display for joint 1 to display current angle...
handles.MyBot.plot(handles.j); % plotting robot using new joint angles stored in handles variable..
handles.T = handles.MyBot.fkine([handles.j]); % making forward kinematics of joint angles and
storing that transformation matrix
% into T in handles...
set(handles.x_coordinate, 'String', num2str(handles.T(1,4))); % Converting x-coordinate of position
vector to string and setting it in
% callback variable of display for x-coordinate...
set(handles.y_coordinate, 'String', num2str(handles.T(2,4))); % Converting y-coordinate of position
vector to string and setting it in
% callback variable of display for y-coordinate...
set(handles.z_coordinate, 'String', num2str(handles.T(3,4))); % Converting z-coordinate of position
vector to string and setting it in
% callback variable of display for z-coordinate...
% update handles structure
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function joint_1_CreateFcn(hObject, eventdata, handles)
% hObject      handle to joint_1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function joint_2_Callback(hObject, eventdata, handles)
% hObject      handle to joint_2 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
handles.j(2) = get(hObject, 'Value'); % getting slider value and storing in second element of joint
angle set array...
set(handles.joint_2_disp, 'String', num2str(handles.j(2))); % converting slider 2 value to string
and setting it in callback variable
% of display for joint 2 to display current angle...
handles.MyBot.plot(handles.j); % plotting robot using new joint angles stored in handles
variable...
handles.T = handles.MyBot.fkine([handles.j]); % making forward kinematics of joint angles and
storing that transformation matrix
% into T in handles...

```



```

set(handles.x_coordinate, 'String', num2str(handles.T(1,4))); % Converting x-coordinate of position
vector to string and setting it in
% callback variable of display for x-coordinate...
set(handles.y_coordinate, 'String', num2str(handles.T(2,4))); % Converting y-coordinate of position
vector to string and setting it in
% callback variable of display for y-coordinate...
set(handles.z_coordinate, 'String', num2str(handles.T(3,4))); % Converting z-coordinate of position
vector to string and setting it in
% callback variable of display for z-coordinate...
% update handles structure
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function joint_2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to joint_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function joint_3_Callback(hObject, eventdata, handles)
% hObject    handle to joint_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
handles.j(3) = get(hObject, 'Value'); % getting slider value and storing in third element of joint
angle set array...
set(handles.joint_3_disp, 'String', num2str(handles.j(3))); % converting slider 3 value to string
and setting it in callback variable
% of display for joint 3 to display current angle...
handles.MyBot.plot(handles.j); % plotting robot using new joint angles stored in handles
variable...
handles.T = handles.MyBot.fkine([handles.j]); % making forward kinematics of joint angles and
storing that transformation matrix
% into T in handles...
set(handles.x_coordinate, 'String', num2str(handles.T(1,4))); % Converting x-coordinate of position
vector to string and setting it in
% callback variable of display for x-coordinate...
set(handles.y_coordinate, 'String', num2str(handles.T(2,4))); % Converting y-coordinate of position
vector to string and setting it in
% callback variable of display for y-coordinate...
set(handles.z_coordinate, 'String', num2str(handles.T(3,4))); % Converting z-coordinate of position
vector to string and setting it in
% callback variable of display for z-coordinate...
% update handles structure
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function joint_3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to joint_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.

```

```

function joint_4_Callback(hObject, eventdata, handles)
% hObject      handle to joint_4 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%          get(hObject,'Min') and get(hObject,'Max') to determine range of slider
handles.j(4) = get(hObject, 'Value'); % getting slider value and storing in fourth element of joint
angle set array...
set(handles.joint_4_disp, 'String', num2str(handles.j(4))); % converting slider 4 value to string
and setting it in callback variable
% of display for joint 4 to display current angle...
handles.MyBot.plot(handles.j); % plotting robot using new joint angles stored in handles
variable...
handles.T = handles.MyBot.fkine([handles.j]); % making forward kinematics of joint angles and
storing that transformation matrix
% into T in handles...
set(handles.x_coordinate, 'String', num2str(handles.T(1,4))); % Converting x-coordinate of position
vector to string and setting it in
% callback variable of display for x-coordinate...
set(handles.y_coordinate, 'String', num2str(handles.T(2,4))); % Converting y-coordinate of position
vector to string and setting it in
% callback variable of display for y-coordinate...
set(handles.z_coordinate, 'String', num2str(handles.T(3,4))); % Converting z-coordinate of position
vector to string and setting it in
% callback variable of display for z-coordinate...
% update handles structure
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function joint_4_CreateFcn(hObject, eventdata, handles)
% hObject      handle to joint_4 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function joint_5_Callback(hObject, eventdata, handles)
% hObject      handle to joint_5 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%          get(hObject,'Min') and get(hObject,'Max') to determine range of slider
handles.j(5) = get(hObject, 'Value'); % getting slider value and storing in fifth element of joint
angle set array...
set(handles.joint_5_disp, 'String', num2str(handles.j(5))); % converting slider 5 value to string
and setting it in callback variable
% of display for joint 5 to display current angle...
handles.MyBot.plot(handles.j); % plotting robot using new joint angles stored in handles
variable...
handles.T = handles.MyBot.fkine([handles.j]); % making forward kinematics of joint angles and
storing that transformation matrix
% into T in handles...
set(handles.x_coordinate, 'String', num2str(handles.T(1,4))); % Converting x-coordinate of position
vector to string and setting it in
% callback variable of display for x-coordinate...
set(handles.y_coordinate, 'String', num2str(handles.T(2,4))); % Converting y-coordinate of position
vector to string and setting it in
% callback variable of display for y-coordinate...
set(handles.z_coordinate, 'String', num2str(handles.T(3,4))); % Converting z-coordinate of position
vector to string and setting it in

```

```

% callback variable of display for z-coordinate...
% update handles structure
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function joint_5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to joint_5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function joint_6_Callback(hObject, eventdata, handles)
% hObject    handle to joint_6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider
handles.j(6) = get(hObject, 'Value'); % getting slider value and storing in fifth element of joint
angle set array...
set(handles.joint_6_disp, 'String', num2str(handles.j(6))); % converting slider 6 value to string
and setting it in callback variable
% of display for joint 6 to display current angle...
handles.MyBot.plot(handles.j); % plotting robot using new joint angles stored in handles
variable...
handles.T = handles.MyBot.fkine([handles.j]); % making forward kinematics of joint angles and
storing that transformation matrix
% into T in handles...
set(handles.x_coordinate, 'String', num2str(handles.T(1,4))); % Converting x-coordinate of position
vector to string and setting it in
% callback variable of display for x-coordinate...
set(handles.y_coordinate, 'String', num2str(handles.T(2,4))); % Converting y-coordinate of position
vector to string and setting it in
% callback variable of display for y-coordinate...
set(handles.z_coordinate, 'String', num2str(handles.T(3,4))); % Converting z-coordinate of position
vector to string and setting it in
% callback variable of display for z-coordinate...
% update handles structure
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function joint_6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to joint_6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on button press in reach_envelop.
function reach_envelop_Callback(hObject, eventdata, handles)
% hObject    handle to reach_envelop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
tic; % turning the timer on...
if handles.numDOF == 1 % checking whether number of DOF is 1...
    for j1 = 0:.25:pi % setting loop and step increment of joint angle for joint 1...

```

```

        T = handles.MyBot.fkine([j1]); % making fkine for joint angle and storin transformation
matrix in T...
        P = T(1:3,4); % getting position vector...
        hold on; % holding the current graph value...
        handles.MyBot.plot([j1]); % plotting robot with the generated joint angle...
        plot3(P(1), P(2), P(3), '--rs'); % plotting a red square on the graph and holding it...
    end % end of the loop for reach envelop, for 1 DOF system...
elseif handles.numDOF == 2 % if number of DOF is not 1, checking whether number of DOF is 2...
    for j1 = 0:.25:pi % setting loop and step increment of joint angle for joint 1...
        for j2 = 0:.25:pi % setting nested loop and step increment of joint angle for joint 2...
            T = handles.MyBot.fkine([j1, j2]);
            P = T(1:3,4);
            hold on;
            handles.MyBot.plot([j1, j2]);
            plot3(P(1), P(2), P(3), '--rs');
        end % nested loop for joint 2 is ended...
    end % end of the loop for reach envelop, for 2 DOF system...
elseif handles.numDOF == 3 % if number of DOF is not 1,2 checking whether number of DOF is 3...
    for j1 = 0:.25:pi % setting loop and step increment of joint angle for joint 1...
        for j2 = 0:.25:pi % setting nested loop and step increment of joint angle for joint 2...
            for j3 = 0:.25:pi % setting nested loop and step increment of joint angle for joint
3...
                T = handles.MyBot.fkine([j1, j2, j3]);
                P = T(1:3, 4);
                hold on;
                handles.MyBot.plot([j1, j2, j3]);
                plot3(P(1), P(2), P(3), '--rs');
            end % nested loop for joint 3 is ended...
        end % nested loop for joint 2 is ended...
    end % end of the loop for reach envelop, for 3 DOF system...
elseif handles.numDOF == 4 % if number of DOF is not 1,2,3 checking whether number of DOF is 4...
    for j1 = 0:.25:pi % setting loop and step increment of joint angle for joint 1...
        for j2 = 0:.25:pi % setting nested loop and step increment of joint angle for joint 2...
            for j3 = 0:.25:pi % setting nested loop and step increment of joint angle for joint
3...
                for j4 = 0:.25:pi % setting nested loop and step increment of joint angle for joint
4...
                    T = handles.MyBot.fkine([j1, j2, j3, j4]);
                    P = T(1:3,4);
                    hold on;
                    handles.MyBot.plot([j1, j2, j3, j4]);
                    plot3(P(1), P(2), P(3), '--rs');
                end % nested loop for joint 4 is ended...
            end % nested loop for joint 3 is ended...
        end % nested loop for joint 2 is ended...
    end % end of the loop for reach envelop, for 4 DOF system...
elseif handles.numDOF == 5 % if number of DOF is not 1,2,3,4 checking whether number of DOF is 5...
    for j1 = 0:.25:pi % setting loop and step increment of joint angle for joint 1...
        for j2 = 0:.25:pi % setting nested loop and step increment of joint angle for joint 2...
            for j3 = 0:.25:pi % setting nested loop and step increment of joint angle for joint
3...
                for j4 = 0:.25:pi % setting nested loop and step increment of joint angle for joint
4...
                    for j5 = 0:.25:pi % setting nested loop and step increment of joint angle for
joint 5...
                        T = handles.MyBot.fkine([j1, j2, j3, j4, j5]);
                        P = T(1:3,4);
                        hold on;
                        handles.MyBot.plot([j1, j2, j3, j4, j5]);
                        plot3(P(1), P(2), P(3), '--rs');
                    end % nested loop for joint 5 is ended...
                end % nested loop for joint 4 is ended...
            end % nested loop for joint 3 is ended...
        end % nested loop for joint 2 is ended...
    end % end of the loop for reach envelop, for 5 DOF system...
elseif handles.numDOF == 6 % if number of DOF is not 1,2,3,4,5 checking whether number of DOF is
6...

```

```

    for j1 = 0:.25:pi % setting loop and step increment of joint angle for joint 1...
        for j2 = 0:.25:pi % setting nested loop and step increment of joint angle for joint 2...
            for j3 = 0:.25:pi % setting nested loop and step increment of joint angle for joint
3...
                for j4 = 0:.25:pi % setting nested loop and step increment of joint angle for joint
4...
                    for j5 = 0:.25:pi % setting nested loop and step increment of joint angle for
joint 5...
                        for j6 = 0:.25:pi % setting nested loop and step increment of joint angle
for joint 6...
                            T = handles.MyBot.fkine([j1, j2, j3, j4, j5, j6]);
                            P = T(1:3,4);
                            hold on;
                            handles.MyBot.plot([j1, j2, j3, j4, j5, j6]);
                            plot3(P(1), P(2), P(3), '--rs');
                        end % nested loop for joint 6 is ended...
                    end % nested loop for joint 5 is ended...
                end % nested loop for joint 4 is ended...
            end % nested loop for joint 3 is ended...
        end % nested loop for joint 2 is ended...
    end % end of the loop for reach envelop, for 6 DOF system...
end % Loop to check number of DOF and taking actions accordingly is ended...
hold off; % hold is made off...
toc; % turning the timer off...
disp(toc); % displaying the time elapsed...

```

```

function joint_1_disp_Callback(hObject, eventdata, handles)
% hObject    handle to joint_1_disp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of joint_1_disp as text
%        str2double(get(hObject,'String')) returns contents of joint_1_disp as a double

```

```

% --- Executes during object creation, after setting all properties.
function joint_1_disp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to joint_1_disp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

function joint_2_disp_Callback(hObject, eventdata, handles)
% hObject    handle to joint_2_disp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of joint_2_disp as text
%        str2double(get(hObject,'String')) returns contents of joint_2_disp as a double

```

```

% --- Executes during object creation, after setting all properties.
function joint_2_disp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to joint_2_disp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

function joint_3_disp_Callback(hObject, eventdata, handles)
% hObject      handle to joint_3_disp (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of joint_3_disp as text
%          str2double(get(hObject,'String')) returns contents of joint_3_disp as a double

% --- Executes during object creation, after setting all properties.
function joint_3_disp_CreateFcn(hObject, eventdata, handles)
% hObject      handle to joint_3_disp (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%          See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function joint_4_disp_Callback(hObject, eventdata, handles)
% hObject      handle to joint_4_disp (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of joint_4_disp as text
%          str2double(get(hObject,'String')) returns contents of joint_4_disp as a double

% --- Executes during object creation, after setting all properties.
function joint_4_disp_CreateFcn(hObject, eventdata, handles)
% hObject      handle to joint_4_disp (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%          See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function joint_5_disp_Callback(hObject, eventdata, handles)
% hObject      handle to joint_5_disp (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of joint_5_disp as text
%          str2double(get(hObject,'String')) returns contents of joint_5_disp as a double

% --- Executes during object creation, after setting all properties.
function joint_5_disp_CreateFcn(hObject, eventdata, handles)
% hObject      handle to joint_5_disp (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%          See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function joint_6_disp_Callback(hObject, eventdata, handles)
% hObject      handle to joint_6_disp (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of joint_6_disp as text
% str2double(get(hObject,'String')) returns contents of joint_6_disp as a double

% --- Executes during object creation, after setting all properties.
function joint_6_disp_CreateFcn(hObject, eventdata, handles)
% hObject handle to joint_6_disp (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function x_coordinate_Callback(hObject, eventdata, handles)
% hObject handle to x_coordinate (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of x_coordinate as text
% str2double(get(hObject,'String')) returns contents of x_coordinate as a double

% --- Executes during object creation, after setting all properties.
function x_coordinate_CreateFcn(hObject, eventdata, handles)
% hObject handle to x_coordinate (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function y_coordinate_Callback(hObject, eventdata, handles)
% hObject handle to y_coordinate (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of y_coordinate as text
% str2double(get(hObject,'String')) returns contents of y_coordinate as a double

% --- Executes during object creation, after setting all properties.
function y_coordinate_CreateFcn(hObject, eventdata, handles)
% hObject handle to y_coordinate (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function z_coordinate_Callback(hObject, eventdata, handles)
% hObject handle to z_coordinate (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB

```



```

% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of z_coordinate as text
%        str2double(get(hObject,'String')) returns contents of z_coordinate as a double

% --- Executes during object creation, after setting all properties.
function z_coordinate_CreateFcn(hObject, eventdata, handles)
% hObject      handle to z_coordinate (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in transmit_data.
function transmit_data_Callback(hObject, eventdata, handles)
% hObject      handle to transmit_data (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
server(handles.j, 3000, -1);

% --- Executes on button press in receive_data.
function receive_data_Callback(hObject, eventdata, handles)
% hObject      handle to receive_data (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
%This code connects to the given local host and waits on the
%given port for signals from the remote server.

import java.net.Socket % importing java.net library for socket communication...
import java.io.* % Importing java input output library for input-output  bits/bytes after
communication...

%if you are running the test from the same machine, use the hostname as
%local host as follows:
%host= 'localhost';

%if you have a specific ip address, you will use it here....You may
%have to ask the IT folks to diaable certain checks they may have to
%allow you to do this.

host = handles.ip; % storing the ip address in variable 'host'...
% if communication is on the same machine, then ip address will be
% 'localhost'.... else ip address will be entered by user for
% communication...
port = 3000; % setting the port number...
number_of_retries = -1; % setting the number of retries, max retries will be 20...
% setting retries -1 for infinite retry...

retry      = 0; % setting retry as zero for start of loop...
input_socket = []; % setting empty input socket variable array...
message     = []; % seting empty message variable array for receiving bytes...

%keep trying to connect
while true % setting loop to check if true...

    retry = retry + 1; % increasing retry value by 1...
    if ((number_of_retries > 0) && (retry > number_of_retries)) % checking whether retry prompt
        % is set positive AND number of retries exceeded maximum limits....
        fprintf(1, 'Too many retries\n'); % printing the message that too many retries but failed
        to connect...
        break; % break the loop...
    end
end

```



```

try % setting try to execute set of operations...
    fprintf(1, 'Retry %d connecting to %s:%d\n', ...
        retry, host, port); % if connection is failed then print as retry
    % status message...

    % throws if unable to connect
    input_socket = Socket(host, port); % getting host address and port number
    % in the variable, using Socket function of java library...

    % get a buffered data input stream from the socket
    fprintf(1, 'Connected to server, waiting for inputs\n'); % print the successful connection
message...
    input_stream = input_socket.getInputStream; % getting input stream in
    % input_socket variable...
    d_input_stream = DataInputStream(input_stream); % getting data in variable
    % using function of input stream...

    % read data from the socket - wait a short time first
    pause(0.5); % inserting a delay...

    %once connected, wait for inputs from the server.
    while (1)
        pause (0.5); % inserting a delay...
        bytes_available = input_stream.available ; % checking the bytes available...

        %check to see if any bytes are available.
        if (bytes_available > 0) % checking whether bytes available are greater than 0...
            % get the message that has been sent...
            message = zeros(1, bytes_available, 'uint8'); % creating unsigned 8-bit empty array
'message'
            % equal to length of received bytes...
            for i = 1:bytes_available % setting loop equal to available bytes to store bytes in
variable...
                message(i) = d_input_stream.readByte; % reading byte and storing in empty
message array...
            end

            message = char(message) % updating variable by converting received string to
character string...

            %convert the message to two number...It is assumed
            %that the output from the server is two numbers in one
            %line...
            [value] = sscanf (message, '%f %f %f %f %f %f') % converting value to float...
            %[value] = uint8(message) %its joints
            %[value] = uint16(message) %its joints

            %check the values of the sent messages.
            if (length (value) == 6) % checking whether value array length is equal to number
of joints...
                for n = 1:length(value) % setting loop equal to length of array...
                    handles.j(n) = value(n); % setting handles array by values of received
data...
                end
                %handles.a = value(1);
                %handles.b = value(2);
                %if all is okay, set the value in the plot of the
                %robot
                % print the robot.
                handles.MyBot

                %plot the robot.
                handles.MyBot.plot(handles.j); % plotting the robot with received values...
            end
        end
    end

    % cleanup
    input_socket.close;

```

```

        break;

        %if there are any errors on the port, retry the connection.
    catch
        if ~isempty(input_socket)
            input_socket.close;
        end

        % pause before retrying
        pause(1);
    end
end
%received_data = client('localhost', 3000, -1);
%received_data = uint8(received_data);
%handles.j = received_data;
%handles.MyBot.plot(handles.j);
%guidata(hObject, handles);

% --- Executes on button press in exit.
function exit_Callback(hObject, eventdata, handles)
% hObject    handle to exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(gcf); % closing current figure handle...

function ip_address_input_Callback(hObject, eventdata, handles)
% hObject    handle to ip_address_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ip_address_input as text
%        str2double(get(hObject,'String')) returns contents of ip_address_input as a double

handles.ip = get(hObject,'String'); % getting user entered ip address...
% Update handles structure
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function ip_address_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ip_address_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes when entered data in editable cell(s) in dh_parameters.
function dh_parameters_CellEditCallback(hObject, eventdata, handles)
% hObject    handle to dh_parameters (see GCBO)
% eventdata  structure with the following fields (see MATLAB.UI.CONTROL.TABLE)
%   Indices: row and column indices of the cell(s) edited
%   PreviousData: previous data for the cell(s) edited
%   EditData: string(s) entered by the user
%   NewData: EditData or its converted form set on the Data property. Empty if Data was not changed
%   Error: error string when failed to convert EditData to appropriate value for Data
% handles    structure with handles and user data (see GUIDATA)
global parameters;
parameters = get(handles.dh_parameters, 'data');
guidata(hObject, handles);

% Code for Socket : CLIENT (Transmitter) initialized...
% --- Executes on button press in transmit_signal.

```

```

function transmit_signal_Callback(hObject, eventdata, handles)
% hObject    handle to transmit_signal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

signal = handles.j;
transmitter = tcpip(handles.receiver_ip, 42000, 'NetworkRole', 'client');
set(transmitter, 'OutputBufferSize', 4096);
fopen(transmitter);
fwrite(transmitter, signal, 'float');
fclose(transmitter);
% Update handles structure
guidata(hObject, handles);

function server_receiver_ip_Callback(hObject, eventdata, handles)
% hObject    handle to server_receiver_ip (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of server_receiver_ip as text
%        str2double(get(hObject,'String')) returns contents of server_receiver_ip as a double

handles.receiver_ip = get(hObject,'String'); % getting user entered transmitter ip address...
% Update handles structure
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function server_receiver_ip_CreateFcn(hObject, eventdata, handles)
% hObject    handle to server_receiver_ip (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% Code for Socket : SERVER (Receiver) initialized...
% --- Executes on button press in receive_signal.
function receive_signal_Callback(hObject, eventdata, handles)
% hObject    handle to receive_signal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
receiver = tcpip(handles.transmitter_ip, 42000, 'NetworkRole', 'server');
set(receiver, 'InputBufferSize', 4096);
fopen(receiver);
while receiver.BytesAvailable == 0
    pause(1)
end
tic;
signal = fread(receiver, receiver.BytesAvailable, 'float');
signal = signal';
numDOF = handles.numDOF;
if length(signal) == numDOF
    handles.j = signal;
    handles.MyBot.plot(handles.j);
end
toc;
disp('Time taken for communication is....');
disp(toc);
fclose(receiver);
% Update handles structure
guidata(hObject, handles);

function client_transmitter_ip_Callback(hObject, eventdata, handles)

```

```

% hObject      handle to client_transmitter_ip (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of client_transmitter_ip as text
%         str2double(get(hObject,'String')) returns contents of client_transmitter_ip as a double

handles.transmitter_ip = get(hObject,'String'); % getting user entered transmitter ip address...
% Update handles structure
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function client_transmitter_ip_CreateFcn(hObject, eventdata, handles)
% hObject      handle to client_transmitter_ip (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

B. Video Link

Please check video tutorial of Modeling of Robot.

<https://youtu.be/bYzpb4QtChE>