Rohit Raibagkar

ECE/BME 7400
Winter 2017

Report of Individual Project Assignment 2

College of Engineering

Declaration : The project and the report are my own work
Date : 03/19/2017

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.

In the second assignment, I address the processing of images and registration of robot to the images. The data available is in the form of images. Complete processing and making a 3D model of data is challenging task. We here are using Matlab as image processing and registration software. We are using Arduino and potentiometer to scroll through available slices of images. Again, moving EE of robot in the required direction required is problem statement. Inverse kinematics is used to move the robot in required direction.

Table of Contents

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.

- Creating Axial, Coronal and sagittal View

MRI data is already available in Matlab. Using load mri command in Matlab, we can import mri data in variable. Using some transformation on arrays, we must create Axial, Coronal and Sagittal views.

- Inverse kinematics.

We should be able to move EE of robot using Push Buttons provided in MATLAB GUI. Ikine function in Robotics toolbox will help us to move. Application of Ikine is task.

- Interface Arduino and Calibrate potentiometer.

We must interface Arduino to MATLAB and we should calibrate potentiometer to scroll complete set of MRI slices.

- Control of Brightness, Contrast and Range of MRI Slices.

We must be able to make user to control Brightness, Contrast and Range of MRI slices. Dedicated potentiometers are provided on breadboard.

- Create 3D model of Region of Interest selected by the user and apply Delaunay triangulation to make a 3D model.
- 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 :

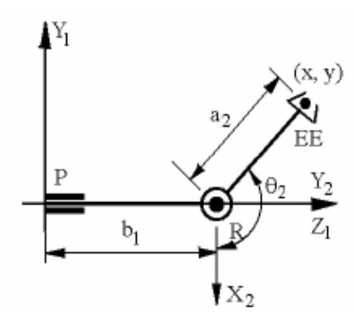
1) Joint angle, Θ : Θ is angle of projection of two common normal X_i and X_{i+1} along z-axis at that joint. For revolute joint, Θ is variable, for prismatic joint, Θ is constant. Rotation will be positive in counter-clockwise direction and negative in clockwise direction.

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 Dpositive, 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 :

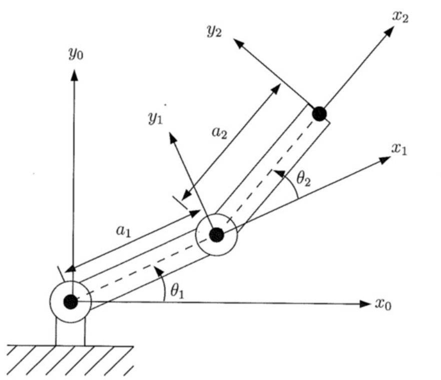
```
>> aesop
>> Aesop

Aesop =

Aesop1000 (6 axis, RRRRRR, stdDH, fastRNE)

+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|     0.3844|          0|          0|
|  2|         q2|          0|    0.05271|      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. Inverse Kinematics

Kinematics is science that deals with motion of the robot arm. The aim is, to determine the end effecter position in 3D space. When we design a robot, it very important to know position and orientation of end effecter 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 effecter 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 Effecter, 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 effecter.

Inverse Kinematics

Inverse kinematics is, finding the corresponding joint angles when we know position and orientation of end effecter. 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.

Input to the Inverse kinematics is transformation matrix, initial joint angle set, mask matrix if you want to mask unavailable DOF or neglect orientation (roll-pitch-yaw) of EE.

Transformation matrix is matrix of position vector, which is required position of EE in 3D space. This can be created using transl command in Matlab.

```
>> T = transl(.725, 0, -.27542)

T =
```

```
    1.0000         0         0    0.7250
         0    1.0000         0         0
         0         0    1.0000   -0.2754
         0         0         0    1.0000
```

Input arguments to transl command are x, y, z coordinates. The result of Ikine is:

```
>> q = handles.MyBot.ikine(T, q, M, 'pinv', 'ilimit', 150000, 'alpha', .5)

q =

   -0.0000   -0.0000   -0.0079   -0.0000    0.0808         0
```

Where, q, initial joint angle set is [0 0 0 0 0 0] and Mask Matrix M is [1 1 1 0 0 0].

5. Creating Axial, Sagittal and Coronal Slices.

In the mri data set provided in MATLAB, axial slices are already available. Montage option will help us to concatenate complete slices.

```
global D; % initiating global parameter for axial view..
axes(handles.axial_axes); % setting axes as axial view...
load mri; % load mri data...
montage(D, map); % concatening data with map...
title('Axial View'); % setting title ...
```

This will create Axial View in MatlabGUI.

For Sagittal and Coronal View, following is the code.

Sagittal

```
global S; % initiating global variable for getting axial slides...
load mri; % load mri data...
axes(handles.sagittal_axes); % setting axes as axial view...
T = maketform('affine',[-2.5 0 0; 0 1 0; 0 0 0.5; 68.5 0 -14]); % making uniformm
array...
R = makeresampler({'cubic','nearest','nearest'},'fill'); % using resampler with cubic
interpolation...
S = tformarray(D,T,R,[4 1 2],[1 2 4],[66 128 35],[],0); % making uniform array...
S = padarray(S,[6 0 0 0],0,'both'); % padding array...
montage(S,map); % making map to show...
title('Sagittal View'); % setting title...
```

Coronal

```
global C; % initiating global variable...
load mri; % load mri...
axes(handles.coronal_axes); % setting axes...
T = maketform('affine',[-2.5 0 0; 0 1 0; 0 0 -0.5; 68.5 0 61]); % making uniform array..
R = makeresampler({'cubic','nearest','nearest'},'fill'); % using cubic interpolation...
C = tformarray(D,T,R,[4 2 1],[1 2 4],[66 128 45],[],0); % using tformarray...
C = padarray(C,[6 0 0 0],0,'both'); % adding array...
montage(C,map); % showing slices...
title('Coronal View'); % setting title...
```

The key commands for transformation are maketform, makeresampler, tformarray and padarray. Makeresampler command provides cubic interpolation.

6. Interfacing Arduino and Calibrating Potentiometer.

Interfacing Arduino

We must download MATLAB Arduino library and interface Arduino. The command used is:

```
a = arduino('COM3', 'Mega2560', 'TraceOn', false); % connecting arduino...
```

Once we connect, we have to set analog and digital input and output pins. This task is achieved by following commands:

```
configurePin(a, 'D22', 'DigitalOutput'); % set digital output for LED...
configurePin(a, 'D23', 'DigitalOutput'); % set digital output for LED...
configurePin(a, 'D24', 'DigitalOutput'); % set digital output for LED...
configurePin(a, 'D25', 'DigitalOutput'); % set digital output for LED...
configurePin(a, 'D26', 'DigitalOutput'); % set digital output for LED...
configurePin(a, 'D27', 'DigitalOutput'); % set digital output for LED...

configurePin(a, 'A0', 'AnalogInput'); % set analog input for pot...
configurePin(a, 'A1', 'AnalogInput'); % set analog input for pot...
configurePin(a, 'A2', 'AnalogInput'); % set analog input for pot...
configurePin(a, 'A3', 'AnalogInput'); % set analog input for pot...
configurePin(a, 'A4', 'AnalogInput'); % set analog input for pot...

configurePin(a, 'A8', 'AnalogInput'); % set analog input for pot...
configurePin(a, 'A9', 'AnalogInput'); % set analog input for pot...
configurePin(a, 'A10', 'AnalogInput'); % set analog input for pot...
configurePin(a, 'A11', 'AnalogInput'); % set analog input for pot...
configurePin(a, 'A12', 'AnalogInput'); % set analog input for pot...
```

We have to set ‘a’ as global variable since it will be used in other functions in GUI.

Calibrate Potentiometer

To calibrate potentiometer, ae have to remove offset voltage read by the potentiometer and nullify it. To nullify the effect, we must subtract offset voltage from read value. Then we must divide the value by the number of variables. The following command gives the calibration of the potentiometer.

```
handles.coron_slicevalue = int64((readVoltage(a, 'A2') - .7)/.075); % reading & calibrating slice value from pot 101...
```

7. Control of Brightness, Contrast and Range of MRI Slices

The Brightness, Contrast and Range of MRI slices can be controlled using options provided in MATLAB Commands.

Control of Range

The range can be controlled by options provided in imshow.

```
% Display a grayscale image, adjust the display range
        h = imshow(I,[0 80]);
```

The matrix [0 80] gives the range of image. The pixels between 0 and 80 pixelvalue are plotted or shown. The other pixels are cropped out.

Control of Brightness and Contrast

Brightness and contrast can be controlled by `imadjust` command in MATLAB. The format is:

```
J = imadjust(I,[LOW_IN; HIGH_IN],[LOW_OUT; HIGH_OUT],GAMMA)
```

Values between low in and high in are mapped to the values between low out and high out. Gamma specifies the shape of curve, describing the relationship between output image and input image. If gamma is less than one, mapping is weighted towards the brighter and if gamma is more than one, mapping is weighted towards darker value.

Low in, High in, Low out, High out and gamma values are varied using the potentiometer provided on the blackboard. If value of Low out > Value of High Out, Image will be completely inverted.

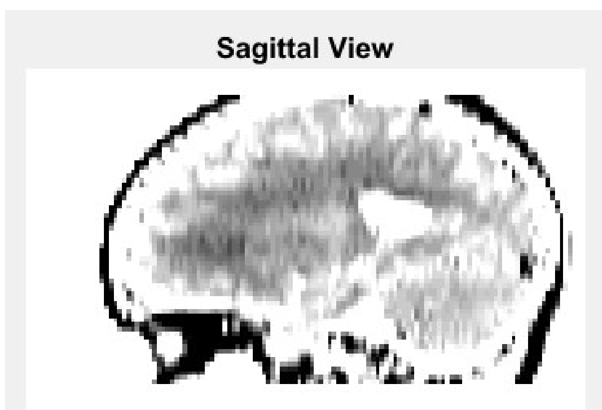


Fig : When Low Out << High Out.

8. 3D Model of MRI Slices

1) Using `plot3` Command

Plot3 command gives 3D plot in Matlab GUI. Syntax is `plot3(x,y,z)`. Following figure gives 3D plot:

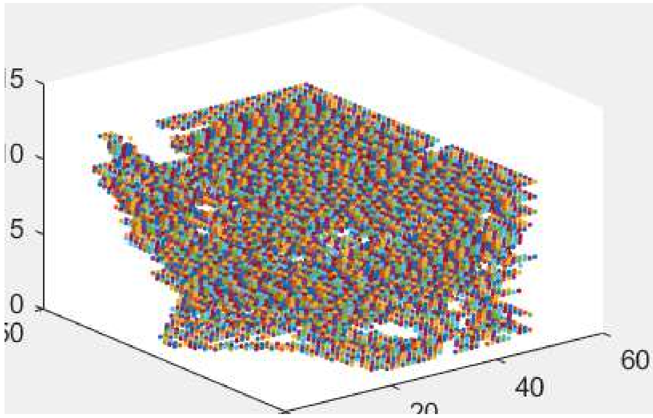


Fig : 3D plot of selected region of slices using plot3

2) Using `surf` Command

Surf Command gives 3D plot of any image with height = pixel depth. Sysntax is `surf(double(image))`. Following figure gives complete 3D plot using surf.

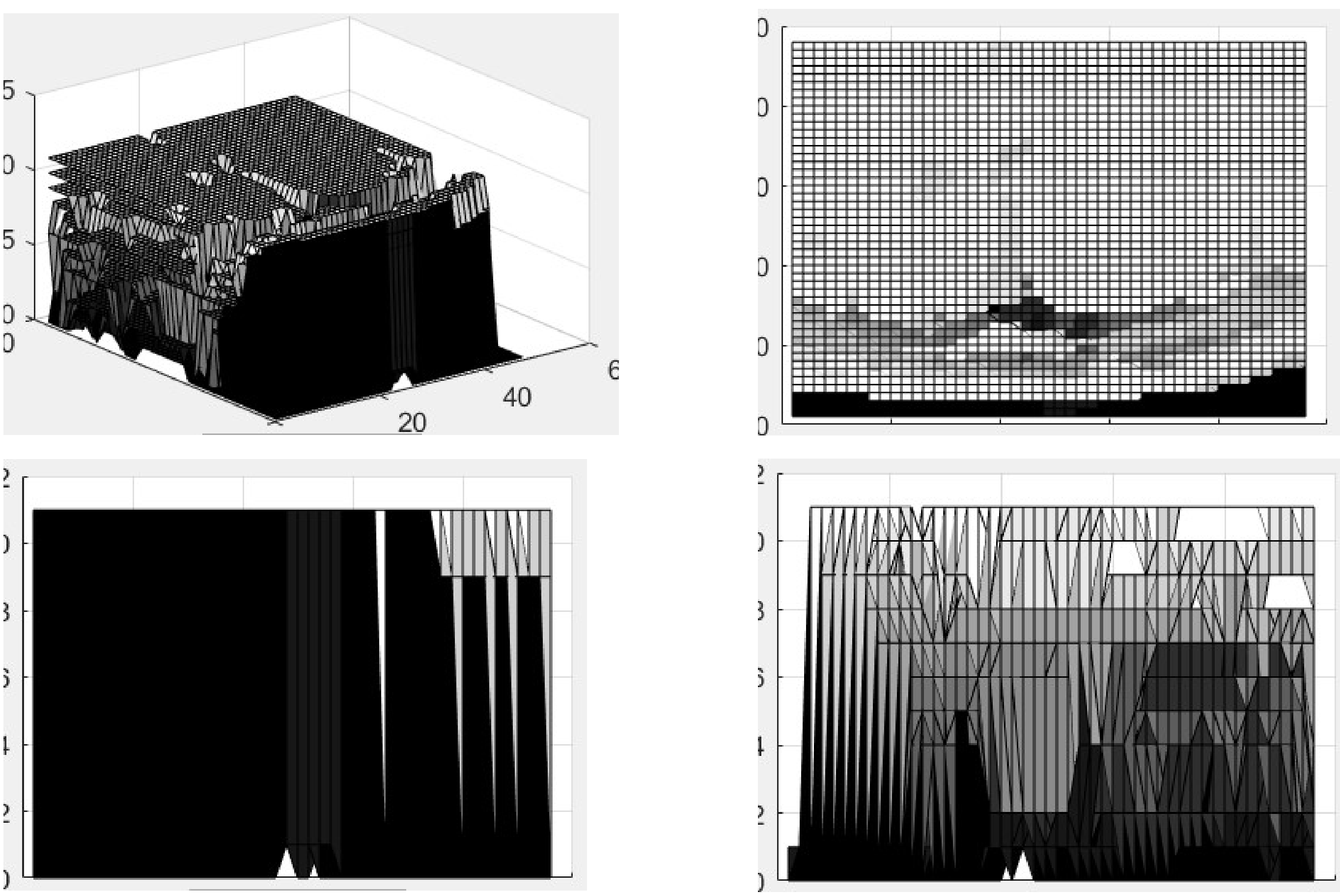


Fig: Plot of region of interest using surf Command

3) Using `delaunay` Triangulation

Delaunay Triangulation is method of triangulation between three points. Delaunay can be applied to 3D array generated by selecting the region of interest. Fig below shows Delaunay triangulation :

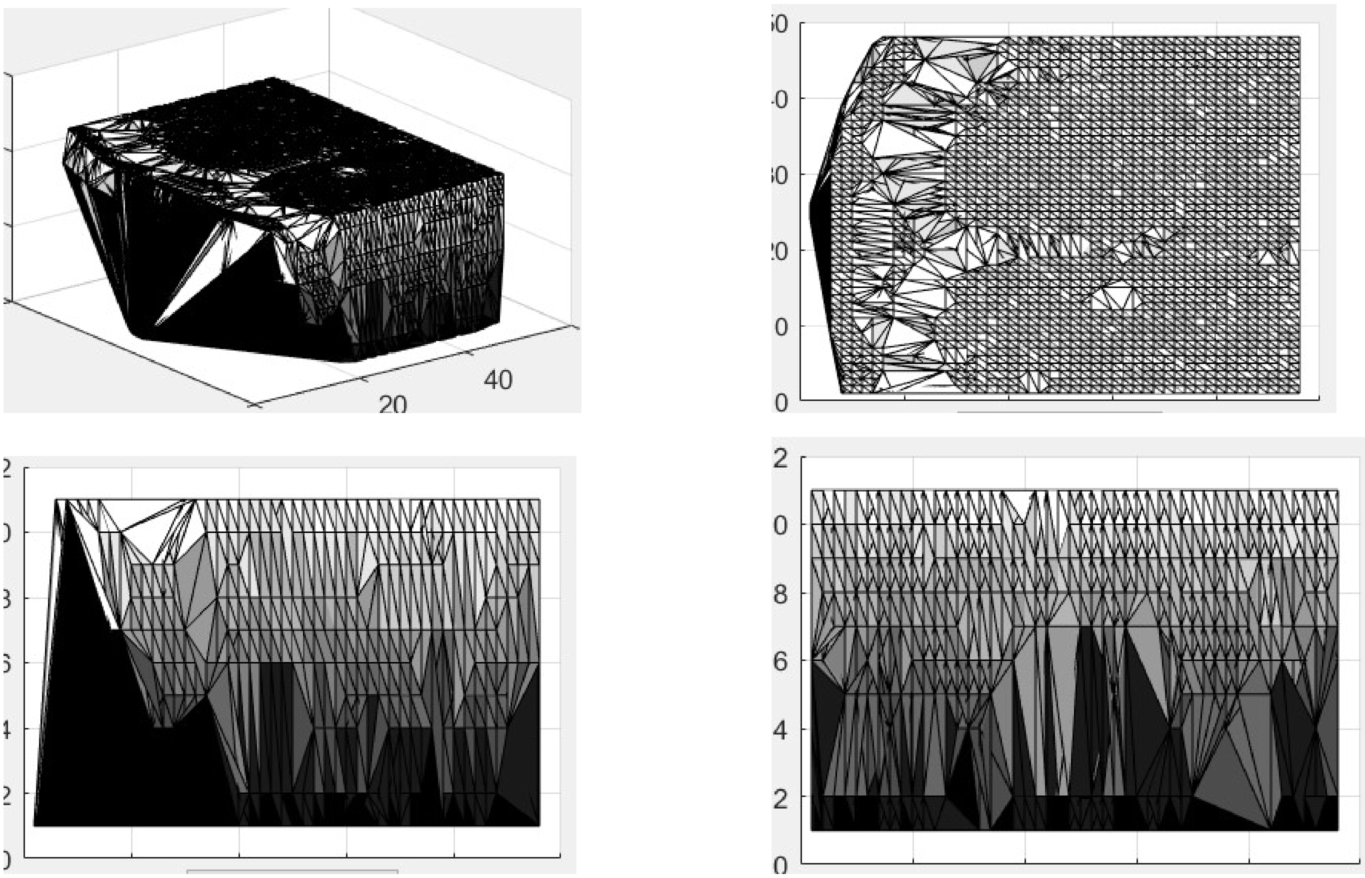


Fig : 3D plot using Delaunay Triangulation

9. Circuit Diagram of Hardware (Arduino) Pin Connections

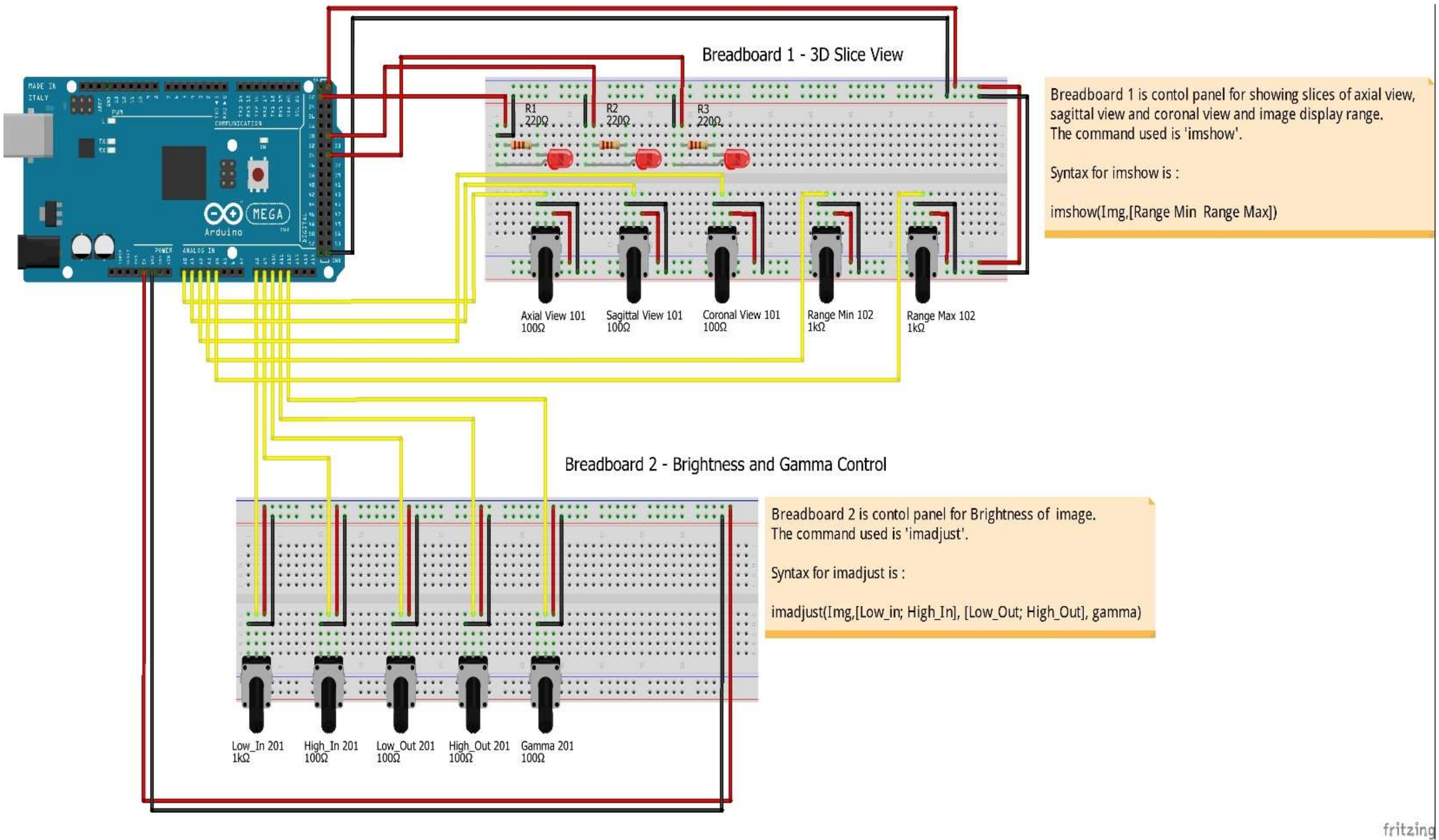


Fig : Connection Circuit Diagram for Project Assignment 2

10. Questions on Image Processing

Q1) Consider the following sub image

10	5	10	5
20	**40**	**30**	20
20	10	10	5

Apply the convolution kernel shown below to the pixels 40, 30. These two pixels' indices are (2, 2) and (2, 3), respectively. Assume the first row and column start from index 1.

-1	-2	-1
0	0	0
1	2	1

Solution :

```
>> img1 = [10 5 10;20 40 30;20 10 10]
img1 =


    10     5    10
    20    40    30
    20    10    10


>> Kernel = [-1 -2 -1;0 0 0;1 2 1]
Kernel =
    -1    -2    -1
     0     0     0
     1     2     1


>> img1.*Kernel
ans =
   -10   -10   -10
     0     0     0
    20    20    10
```

```
New pixel = (-1*10) + (-2*5) + (-1*10) + (0*20) + (0*40) + (0*30) + (1*20) + (2*10)
+ (1*10) = 20

For pixel 30

New pixel = (-1*5) + (-2*10) + (-1*5) + (0*40) + (0*30) + (0*20) + (1*10) + (2*10)
+ (1*5) = 10
```

Q 2) Consider the following sub image

10	5	10	5
20	**40**	**30**	20
20	10	10	5

Apply the convolution kernel shown below to the pixels 40, 30. These two pixels' indices are (2, 2) and (2, 3), respectively. Assume the first row and column start from index 1.

-1	0	1
-2	0	2
-1	0	1

Solution:

- For pix 40

New pix = (-1*10) + (-2*20) + (-1*20) + (0*5) + (0*40) + (0*10) + (1*10) + (2*30) + (1*10) = 10

- For pix 30

New pix = (-1*5) + (-2*40) + (-1*10) + (0*10) + (0*30) + (0*10) + (1*5) + (2*20) + (1*5) = -45

Q3) Consider the following sub image

10	5	10	5
20	**40**	**30**	20
20	10	10	5

Apply the convolution kernel shown below to the pixels 40, 30. These two pixels' indices are (2, 2) and (2, 3), respectively. Assume the first row and column start from index 1.

0	-1	0
-1	4	-1
0	-1	0

Solution:

- For pix 40

New pixel = (0*10) + (-1*5) + (0*10) + (-1*20) + (4*40) + (-1*30) + (0*20) + (-1*10) + (0*10) = 95

- For pix 30

New pixel = (0*5) + (-1*10) + (0*5) + (-1*40) + (4*30) + (-1*20) + (0*10) + (-1*10) + (0*5) = 40

Q 4) Consider the following sub image

10	5	10	5
20	**40**	**30**	20
20	10	10	5

Apply the convolution kernel shown below to the pixels 40, 30. These two pixels' indices are (2, 2) and (2, 3), respectively. Assume the first row and column start from index 1.

-1	-1	-1
-1	8	-1
-1	-1	-1

Solution:

- For pix 40

New pixel = (-1*10) + (-1*5) + (-1*10) + (-1*20) + (8*40) + (-1*30) + (-1 *20) + (-1*10) + (-1*10) = 205

- For pix 30

New pixel = (-1*5) + (-1*10) + (-1*5) + (-1*40) + (8*30) + (-1*20) + (-1 *10) + (-1*10) + (-1*5) = 135

Q 5) If you are given the following subimage. Apply one iteration of the global Thresholding for the subimage using 21 as initial threshold and find the next threshold value.

10	20	15	10	15	45
20	10	15	20	25	35
5	10	20	25	25	25
10	25	30	25	20	20
25	30	30	25	25	25

20	40	45	50	35	30

Solution

When we apply Thresholding using 21 as a threshold, we get two groups:

Group1 contains the pixel values that are less than or equal to 21, which are 16 elements and the average of these pixel values is 15

Group2 contains the pixel values that are greater than 21, which are 20 elements and the average of these values is 31.

So, new T = (15+31)/2 = **23**

Q 6) Compare between the first order derivative and the second order derivative according to:

a. Their definition

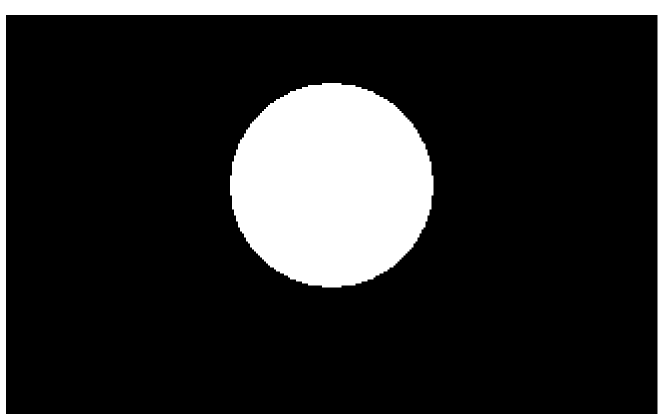
b. Sensitivity for the existence of noise

Q 7) What is the difference between global, adaptive, and local Thresholding? You have to specify the function T that belongs to each one.

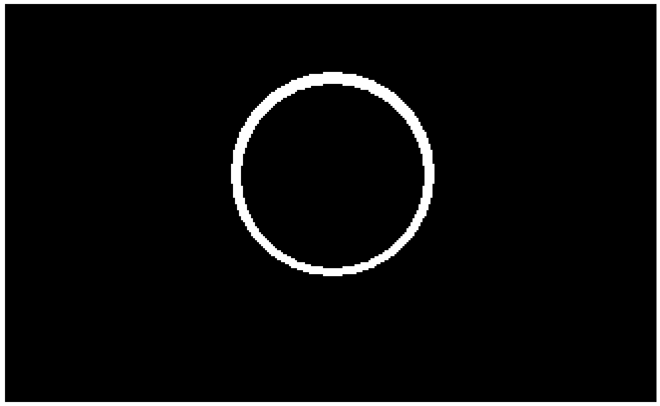
Solution

- Global: is a Thresholding technique that in which we apply a threshold value T to the whole image. T is calculated without taking into consideration spatial information of pixels or any property of these pixels in some neighborhood. T = T [f(x, y)]
- Adaptive: is a Thresholding technique that in which we divide an image to sub images and apply a threshold value T to each subimage. T is calculated inside each subimage, so we take the spatial information as a metric to find T for each subimage. T = [x, y, f(x, y)]
- Local: is a Thresholding technique that in which we apply a threshold value T to some local properties of the image like Thresholding an image after applying edge detection techniques like gradient and laplacian. In this case we take these local properties and threshold them. T = T(f(x, y), p(x, y)]

Q 8) If you are given the following image which is a gray scale image. If we apply the gradient operation what will be the result?



Solution



11. Conclusion

I thus have successfully modeled robot with image processing. Creation of Axial Sagittal and Coronal Slices is done successfully using `maketform, makeresampler, tformarray, padarray`. Connection between Arduino and Matlab has been created. Calibration of potentiometer has been successfully done with highest accuracy, sensitivity and precision. Control of brightness, contrast has been done using `imadjust(I,[LOW_IN; HIGH_IN],[LOW_OUT; HIGH_OUT],GAMMA` Command in Matlab. 3D model successfully created using Delaunay, Plot3 and Surf Command in Matlab.

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 18-Mar-2017 23:13:15

% 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'; % Setting the transmitter IP...
handles.receiver_ip = 'localhost'; % setting receiver IP...
global parameters D S C stop bri_cntrl x y z q M cropped_3D_array; % Defining global
parameters for functions...
bri_cntrl = false; % setting initial condition as false for brightness control...
stop = false; % setting initial condition for stop parameters...
q = [0 0 0 0 0 0]; % setting initial joint angle set for ikine function...
x = 0.70505; y = 0; z = -0.27542; % defining initial position vector ...
M = [1 1 1 0 0 0]; % defining mask matrix neglecting orientation...

% 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...

% Follwing 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, repeatations 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 -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 % setting loop counter...
            type = parameters{n,1}; % getting type of joint to be created from 
parameters...
            type = uint16(type); % converting type to ASCII unsigned integer codes...
            type(:,9) = 0; % setting last parameters to zero...
            if type == [114 101 118 111 108 117 116 101 0] % checking for value...
                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] % checking for value...
                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]}; % defining plot option 
for robot...
            end
            handles.j(n) = 0; % initiating the handles structure with number of elements 
equal to number of degrees of freedom...
        end

    case 'Save me from Robotics..!!!'
        system('shutdown /s /t 60'); % scheduling shutdown for 60 seconds...
        escape = questdlg('System shutdown has been scheduled in 60 seconds. Proceed 
anyway?',...
            'Shutdown..!!!', 'Yes','No', 'No') % another option for cancelling shutdown...
        switch escape % action sequence for cancelling...
            case 'Yes' % if yes is selected...
                system('shutdown /s /t 0') % shutdown immediately...
            case 'No' % if no is selected...
                system('shutdown /a') % abort shutdown...
                return % return...
        end
```

```
end



%for n = 1:1:numDOF % the loop is set to receive the D-H parameters from the user,
repeatedly, repeatations 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 -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...
axes(handles.axes1);
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
tansformation 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...

% --- Executes on button press in del_x_positive.
function del_x_positive_Callback(hObject, eventdata, handles)
% hObject    handle to del_x_positive (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on button press in del_x_negative.
global x y z q M; % initiating global constants...
x = x + .010; % incrementing value of x by 10 mm...
T = transl(x, y, z); % creaating transformation matrix with required position vector...
q = handles.MyBot.ikine(T, q, M, 'pinv', 'ilimit', 150000, 'alpha', .5); % getting joint
angles...
handles.MyBot.plot(q); % plotting robot MyBot with generated joint angles...
set(handles.joint_1_disp, 'String', num2str(q(1))); % converting angle 1 value to string
and setting it in callback variable
% of display for joint 1 to display current angle...
set(handles.joint_2_disp, 'String', num2str(q(2))); % converting angle 2 value to string
and setting it in callback variable
% of display for joint 2 to display current angle...
set(handles.joint_3_disp, 'String', num2str(q(3))); % converting angle 3 value to string
and setting it in callback variable
% of display for joint 3 to display current angle...
set(handles.joint_4_disp, 'String', num2str(q(4))); % converting angle 4 value to string
and setting it in callback variable
% of display for joint 4 to display current angle...
set(handles.joint_5_disp, 'String', num2str(q(5))); % converting angle 5 value to string
and setting it in callback variable
% of display for joint 5 to display current angle...
set(handles.joint_6_disp, 'String', num2str(q(6))); % converting angle 6 value to string
and setting it in callback variable
% of display for joint 6 to display current angle...
set(handles.x_coordinate, 'String', num2str(x)); % Displaying x-coordinate of position
vector to string and setting it in
% callback variable of display for x-coordinate...
set(handles.y_coordinate, 'String', num2str(y)); % Displaying y-coordinate of position
vector to string and setting it in
% callback variable of display for y-coordinate...
set(handles.z_coordinate, 'String', num2str(z)); % Displaying z-coordinate of position
vector to string and setting it in
% callback variable of display for z-coordinate...
guidata(hObject, handles); % Update handles structure...

function del_x_negative_Callback(hObject, eventdata, handles)
% hObject    handle to del_x_negative (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global x y z q M; % initiating global constants...
```

```
x = x - .010; % decrementing value of x by 10 mm...
T = transl(x, y, z); % creaating transformation matrix with required position vector...
q = handles.MyBot.ikine(T, q, M, 'pinv', 'ilimit', 150000, 'alpha', .5); % getting joint
angles...
handles.MyBot.plot(q); % plotting robot MyBot with generated joint angles...
set(handles.joint_1_disp, 'String', num2str(q(1))); % converting angle 1 value to string
and setting it in callback variable
% of display for joint 1 to display current angle...
set(handles.joint_2_disp, 'String', num2str(q(2))); % converting angle 2 value to string
and setting it in callback variable
% of display for joint 2 to display current angle...
set(handles.joint_3_disp, 'String', num2str(q(3))); % converting angle 3 value to string
and setting it in callback variable
% of display for joint 3 to display current angle...
set(handles.joint_4_disp, 'String', num2str(q(4))); % converting angle 4 value to string
and setting it in callback variable
% of display for joint 4 to display current angle...
set(handles.joint_5_disp, 'String', num2str(q(5))); % converting angle 5 value to string
and setting it in callback variable
% of display for joint 5 to display current angle...
set(handles.joint_6_disp, 'String', num2str(q(6))); % converting angle 6 value to string
and setting it in callback variable
% of display for joint 6 to display current angle...
set(handles.x_coordinate, 'String', num2str(x)); % Displaying x-coordinate of position
vector to string and setting it in
% callback variable of display for x-coordinate...
set(handles.y_coordinate, 'String', num2str(y)); % Displaying y-coordinate of position
vector to string and setting it in
% callback variable of display for y-coordinate...
set(handles.z_coordinate, 'String', num2str(z)); % Displaying z-coordinate of position
vector to string and setting it in
% callback variable of display for z-coordinate...
guidata(hObject, handles); % Update handles structure...

% --- Executes on button press in del_y_positive.
function del_y_positive_Callback(hObject, eventdata, handles)
% hObject    handle to del_y_positive (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global x y z q M; % initiating global constants...
y = y + .010; % incrementing value of y by 10 mm...
T = transl(x, y, z); % creating transformation matrix with required position vector...
q = handles.MyBot.ikine(T, q, M, 'pinv', 'ilimit', 150000, 'alpha', .5); % getting joint
angles...
handles.MyBot.plot(q); % plotting robot MyBot with generated joint angles...
set(handles.joint_1_disp, 'String', num2str(q(1))); % converting angle 1 value to string
and setting it in callback variable
% of display for joint 1 to display current angle...
set(handles.joint_2_disp, 'String', num2str(q(2))); % converting angle 2 value to string
and setting it in callback variable
% of display for joint 2 to display current angle...
set(handles.joint_3_disp, 'String', num2str(q(3))); % converting angle 3 value to string
and setting it in callback variable
% of display for joint 3 to display current angle...
set(handles.joint_4_disp, 'String', num2str(q(4))); % converting angle 4 value to string
and setting it in callback variable
% of display for joint 4 to display current angle...
set(handles.joint_5_disp, 'String', num2str(q(5))); % converting angle 5 value to string
and setting it in callback variable
% of display for joint 5 to display current angle...
```

```
set(handles.joint_6_disp, 'String', num2str(q(6))); % converting angle 6 value to string
and setting it in callback variable
% of display for joint 6 to display current angle...
set(handles.x_coordinate, 'String', num2str(x)); % Displaying x-coordinate of position
vector to string and setting it in
% callback variable of display for x-coordinate...
set(handles.y_coordinate, 'String', num2str(y)); % Displaying y-coordinate of position
vector to string and setting it in
% callback variable of display for y-coordinate...
set(handles.z_coordinate, 'String', num2str(z)); % Displaying z-coordinate of position
vector to string and setting it in
% callback variable of display for z-coordinate...
guidata(hObject, handles); % Update handles structure...

% --- Executes on button press in del_y_negative.
function del_y_negative_Callback(hObject, eventdata, handles)
% hObject    handle to del_y_negative (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global x y z q M; % initiating global constants...
y = y - .010; % decrementing value of y by 10 mm...
T = transl(x, y, z); % creating transformation matrix with required position vector...
q = handles.MyBot.ikine(T, q, M, 'pinv', 'ilimit', 150000, 'alpha', .5); % getting joint
angles...
handles.MyBot.plot(q); % plotting robot MyBot with generated joint angles...
set(handles.joint_1_disp, 'String', num2str(q(1))); % converting angle 1 value to string
and setting it in callback variable
% of display for joint 1 to display current angle...
set(handles.joint_2_disp, 'String', num2str(q(2))); % converting angle 2 value to string
and setting it in callback variable
% of display for joint 2 to display current angle...
set(handles.joint_3_disp, 'String', num2str(q(3))); % converting angle 3 value to string
and setting it in callback variable
% of display for joint 3 to display current angle...
set(handles.joint_4_disp, 'String', num2str(q(4))); % converting angle 4 value to string
and setting it in callback variable
% of display for joint 4 to display current angle...
set(handles.joint_5_disp, 'String', num2str(q(5))); % converting angle 5 value to string
and setting it in callback variable
% of display for joint 5 to display current angle...
set(handles.joint_6_disp, 'String', num2str(q(6))); % converting angle 6 value to string
and setting it in callback variable
% of display for joint 6 to display current angle...
set(handles.x_coordinate, 'String', num2str(x)); % Displaying x-coordinate of position
vector to string and setting it in
% callback variable of display for x-coordinate...
set(handles.y_coordinate, 'String', num2str(y)); % Displaying y-coordinate of position
vector to string and setting it in
% callback variable of display for y-coordinate...
set(handles.z_coordinate, 'String', num2str(z)); % Displaying z-coordinate of position
vector to string and setting it in
% callback variable of display for z-coordinate...
guidata(hObject, handles); % Update handles structure...

% --- Executes on button press in del_z_positive.
function del_z_positive_Callback(hObject, eventdata, handles)
% hObject    handle to del_z_positive (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global x y z q M; % initiating global constants...
z = z + .010; % incrementing value of z by 10 mm...
```

```
T = transl(x, y, z); % creating transformation matrix with required position vector...
q = handles.MyBot.ikine(T, q, M, 'pinv', 'ilimit', 150000, 'alpha', .5); % getting joint
angles...
handles.MyBot.plot(q); % plotting robot MyBot with generated joint angles...
set(handles.joint_1_disp, 'String', num2str(q(1))); % converting angle 1 value to string
and setting it in callback variable
% of display for joint 1 to display current angle...
set(handles.joint_2_disp, 'String', num2str(q(2))); % converting angle 2 value to string
and setting it in callback variable
% of display for joint 2 to display current angle...
set(handles.joint_3_disp, 'String', num2str(q(3))); % converting angle 3 value to string
and setting it in callback variable
% of display for joint 3 to display current angle...
set(handles.joint_4_disp, 'String', num2str(q(4))); % converting angle 4 value to string
and setting it in callback variable
% of display for joint 4 to display current angle...
set(handles.joint_5_disp, 'String', num2str(q(5))); % converting angle 5 value to string
and setting it in callback variable
% of display for joint 5 to display current angle...
set(handles.joint_6_disp, 'String', num2str(q(6))); % converting angle 6 value to string
and setting it in callback variable
% of display for joint 6 to display current angle...
set(handles.x_coordinate, 'String', num2str(x)); % Displaying x-coordinate of position
vector to string and setting it in
% callback variable of display for x-coordinate...
set(handles.y_coordinate, 'String', num2str(y)); % Displaying y-coordinate of position
vector to string and setting it in
% callback variable of display for y-coordinate...
set(handles.z_coordinate, 'String', num2str(z)); % Displaying z-coordinate of position
vector to string and setting it in
% callback variable of display for z-coordinate...
guidata(hObject, handles); % Update handles structure...


% --- Executes on button press in del_z_negative.
function del_z_negative_Callback(hObject, eventdata, handles)
% hObject    handle to del_z_negative (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global x y z q M; % initiating global constants...
z = z - .010; % decrementing value of z by 10 mm...
T = transl(x, y, z); % creating transformation matrix with required position vector...
q = handles.MyBot.ikine(T, q, M, 'pinv', 'ilimit', 150000, 'alpha', .5); % getting joint
angles...
handles.MyBot.plot(q); % plotting robot MyBot with generated joint angles...
set(handles.joint_1_disp, 'String', num2str(q(1))); % converting angle 1 value to string
and setting it in callback variable
% of display for joint 1 to display current angle...
set(handles.joint_2_disp, 'String', num2str(q(2))); % converting angle 2 value to string
and setting it in callback variable
% of display for joint 2 to display current angle...
set(handles.joint_3_disp, 'String', num2str(q(3))); % converting angle 3 value to string
and setting it in callback variable
% of display for joint 3 to display current angle...
set(handles.joint_4_disp, 'String', num2str(q(4))); % converting angle 4 value to string
and setting it in callback variable
% of display for joint 4 to display current angle...
set(handles.joint_5_disp, 'String', num2str(q(5))); % converting angle 5 value to string
and setting it in callback variable
% of display for joint 5 to display current angle...
```

```
set(handles.joint_6_disp, 'String', num2str(q(6))); % converting angle 6 value to string
and setting it in callback variable
% of display for joint 6 to display current angle...
set(handles.x_coordinate, 'String', num2str(x)); % Displaying x-coordinate of position
vector to string and setting it in
% callback variable of display for x-coordinate...
set(handles.y_coordinate, 'String', num2str(y)); % Displaying y-coordinate of position
vector to string and setting it in
% callback variable of display for y-coordinate...
set(handles.z_coordinate, 'String', num2str(z)); % Displaying z-coordinate of position
vector to string and setting it in
% callback variable of display for z-coordinate...
guidata(hObject, handles); % Update handles structure...


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

    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 avaiable.
            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 = 1:1: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.value = 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 curren 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; % storing signal to be transmitted in joint angle structure...
transmitter = tcpip(handles.receiver_ip, 42000, 'NetworkRole', 'client'); % sending
signal using tcpip on communication port...
set(transmitter, 'OutputBufferSize', 4096); % setting output buffer size...
fopen(transmitter); % opening communication port...
fwrite(transmitter, signal, 'float'); % writing signal to port...
fclose(transmitter); % closing port...
```

```
% 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'); % receiving
signal...
set(receiver, 'InputBufferSize', 4096); % setting buffer size of input buffer...
fopen(receiver); % pening receiver port...
while receiver.BytesAvailable == 0 % holding till receiver bytes are zero....
    pause(1) % pause for 1 second...
end
tic; % timer started...
signal = fread(receiver, receiver.BytesAvailable, 'float'); % reading the signal...
signal = signal'; % making transpose of the signal...
numDOF = handles.numDOF; % getting value of number of DOF...
if length(signal) == numDOF % checking whether length of the signal is equal to number of
DOF...
    handles.j = signal; % getting signal bits in joint angles handle...
    handles.MyBot.plot(handles.j); % plotting robot...
end
toc; % timer ended...
disp('Time taken for communication is....'); % displaying time elapsed during
communication...
disp(toc); % % displaying time elapsed during communication...
fclose(receiver); % closing the receiver...
% Update handles structure
guidata(hObject, handles);
```

```matlab
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

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

% --------------------------------------------------------------------
function zoom_Callback(hObject, eventdata, handles)
% hObject    handle to zoom (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
zoom on; % setting zoom...

% updating handles structure with handles and user data (see GUIDATA)
guidata(hObject, handles);

% --- Executes on button press in create_axial_view.
function create_axial_view_Callback(hObject, eventdata, handles)
% hObject    handle to create_axial_view (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global D; % initiating global parameter for axial view..
axes(handles.axial_axes); % setting axes as axial view...
load mri; % load mri data...
montage(D, map); % concatening data with map...
title('Axial View'); % setting title ...
%handles.axial_view = true;
guidata(hObject, handles); % updating handles structure with handles and user data (see
GUIDATA)...
```

```
% --- Executes on slider movement.
function axial_view_Callback(hObject, eventdata, handles)
% hObject    handle to axial_view (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
global D; % initiating global variable for getting axial slides...
axes(handles.axial_axes); % setting axial axes...
handles.ax_slicevalue = int64(get (hObject, 'Value')); % converting slider value to
signed integers...
if handles.ax_slicevalue >= 1 % checking if value is greater than or equal to one...
    imshow(D(:,:,handles.ax_slicevalue)); % showing corresponding slide...
    title('Axial View'); % setting title...
end
guidata(hObject, handles); % updating handles structure with handles and user data (see
GUIDATA)...

% --- Executes during object creation, after setting all properties.
function axial_view_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axial_view (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 manual_for_axial.
function manual_for_axial_Callback(hObject, eventdata, handles)
% hObject    handle to manual_for_axial (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% this function will allow user to scroll through axial slices...
global D a stop bri_cntrl; % initiating global constants...
axes(handles.axial_axes); % setting axis as axial axes...
writeDigitalPin(a, 'D22', 1); % setting blue LED on for axial potentiometer...
handles.axial_view = true; % setting axial view bit ON (true)...
%bri_cntrl = false;
%handles.stop = false;
while handles.device == true && handles.axial_view == true; % checking if hardware is
connected AND axial view is enabled...
    axes(handles.axial_axes); % setting current axes as xial axes..
    if a == 0 || stop == true || handles.sagittal_view == true || handles.coronal_view ==
true ; % checking for necessary conditions...
        handles.axial_view = false; stop = false; writeDigitalPin(a, 'D22', false); %
setting all values to false and turning LED off..
        break % breaking loop...
    end % end...
    handles.range_min = (readVoltage(a, 'A3') - .5)/0.015686274509804; % reading range
min value from pot 101...
    handles.range_max = (readVoltage(a, 'A4') - .5)/0.015686274509804; % reading range
max value from pot 101...
    handles.low_in = (readVoltage(a, 'A8') - .8)/4; % reading low in value from pot
201...
    handles.high_in = (readVoltage(a, 'A9') - .8)/4; % reading high in value from pot
201...
```

```
    handles.low_out = (readVoltage(a, 'A10') - .8)/4; % reading low out value from pot
201...
    handles.high_out = (readVoltage(a, 'A11') - .8)/4; % reading high out value from pot
201...
    handles.gamma = (readVoltage(a, 'A12') - .5)/2; % reading gamma value from pot 201...
    handles.ax_slicevalue = int64((readVoltage(a, 'A0') - .7)/.125); % reading &
calibrating slice value from pot 101...
    if handles.ax_slicevalue >=1 && handles.ax_slicevalue <= 27 && handles.range_min <
handles.range_max;
        if bri_cntrl == true && handles.low_in < handles.high_in && handles.low_in >= 0
&& handles.high_in >= 0 && handles.low_out >= 0 && handles.high_out >=0;
            img = imadjust(D(:,:,handles.ax_slicevalue),[handles.low_in;
handles.high_in],[handles.low_out; handles.high_out], handles.gamma); % adjusting the
image...
            imshow (img, [handles.range_min handles.range_max]); % showing adjusted
image...
        elseif bri_cntrl == false; % checking if brightness control is false...
            imshow(D(:,:,handles.ax_slicevalue),[handles.range_min handles.range_max]); %
showing image ...
        end
        title('Axial View'); % setting title...
    end
    pause(.1); % pause to stabilize graphics...
end
%handles.axial_view = false;
guidata(hObject, handles); % updating handles structure with handles and user data (see
GUIDATA)...

% --- Executes on button press in create_sagittal_view.
function create_sagittal_view_Callback(hObject, eventdata, handles)
% hObject    handle to create_sagittal_view (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global S; % initiating global variable for getting axial slides...
load mri; % load mri data...
axes(handles.sagittal_axes); % setting axes as axial view...
T = maketform('affine',[-2.5 0 0; 0 1 0; 0 0 0.5; 68.5 0 -14]); % making unifornm
array...
R = makeresampler({'cubic','nearest','nearest'},'fill'); % using resampler with cubic
interpolation...
S = tformarray(D,T,R,[4 1 2],[1 2 4],[66 128 35],[],0); % making uniform array...
S = padarray(S,[6 0 0 0],0,'both'); % padding array...
montage(S,map); % making map to show...
title('Sagittal View'); % setting title...
%handles.sagittal_view = true;
guidata(hObject, handles); % updating handles structure with handles and user data (see
GUIDATA)...


% --- Executes on slider movement.
function sagittal_view_Callback(hObject, eventdata, handles)
% hObject    handle to sagittal_view (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
global S; % initiating global variable...
axes(handles.sagittal_axes); % setting axes...
sagittal_slice = int64(get (hObject, 'Value')); % converting slice value to signed
integer...
```

```
if sagittal_slice >= 1
imshow(S(:,:,sagittal_slice)); % showing image corresponding to slice number...
title('Sagittal View'); % setting title...
end
guidata(hObject, handles); % updating handles structure with handles and user data (see
GUIDATA)...


% --- Executes during object creation, after setting all properties.
function sagittal_view_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sagittal_view (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 manual_for_sagittal.
function manual_for_sagittal_Callback(hObject, eventdata, handles)
% hObject    handle to manual_for_sagittal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% this function will allow user to scroll through sagittal slices...
global S a stop bri_cntrl; % initiating global constants...
axes(handles.sagittal_axes); % setting axis as sagittal axes...
writeDigitalPin(a, 'D23', 1); % setting Red LED on for axial potentiometer...
handles.sagittal_view = true; % setting sagittal view bit ON (true)...
while handles.device == true && handles.sagittal_view == true; % checking if hardware is
connected AND sagittal view is enabled...
    axes(handles.sagittal_axes); % setting current axes as sagittal axes..
    if a == 0 || stop == true || handles.axial_view == true || handles.coronal_view ==
true; % checking for necessary conditions...
        handles.sagittal_view = false; stop = false; writeDigitalPin(a, 'D23', false); %
setting all values to false and turning LED off..
        break % breaking loop...
    end
    handles.range_min = (readVoltage(a, 'A3') - .5)/0.015686274509804; % reading range
min value from pot 101...
    handles.range_max = (readVoltage(a, 'A4') - .5)/0.015686274509804; % reading range
max value from pot 101...
    handles.low_in = (readVoltage(a, 'A8') - .8)/4; % reading low in value from pot
201...
    handles.high_in = (readVoltage(a, 'A9') - .8)/4; % reading high in value from pot
201...
    handles.low_out = (readVoltage(a, 'A10') - .8)/4; % reading low out value from pot
201...
    handles.high_out = (readVoltage(a, 'A11') - .8)/4; % reading high out value from pot
201...
    handles.gamma = (readVoltage(a, 'A12') - .5)/2; % reading gamma value from pot 201...
    %handles.sag_slicevalue = readVoltage(a, 'A1') - .75;
    handles.sag_slicevalue = int64((readVoltage(a, 'A1') - .75)/.1); % reading &
calibrating slice value from pot 101...
    if handles.sag_slicevalue >=1 && handles.sag_slicevalue <= 35 && handles.range_min <
handles.range_max;
        if bri_cntrl == true && handles.low_in < handles.high_in && handles.low_in >= 0
&& handles.high_in >= 0 && handles.low_out >= 0 && handles.high_out >=0;
```

```
            img = imadjust(S(:,:,handles.sag_slicevalue),[handles.low_in;
handles.high_in],[handles.low_out; handles.high_out], handles.gamma); % adjusting the
image...
            imshow (img, [handles.range_min handles.range_max]); % showing adjusted
image...
        elseif bri_cntrl == false;
            imshow(S(:,:,handles.sag_slicevalue), [handles.range_min handles.range_max]);
% showing image ...
        end
        title('Sagittal View'); % setting title...
    end
    pause(.1); % pause to stabilize graphics...
end
guidata(hObject, handles); % updating handles structure with handles and user data (see
GUIDATA)...

% --- Executes on button press in create_coronal_view.
function create_coronal_view_Callback(hObject, eventdata, handles)
% hObject    handle to create_coronal_view (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global C; % initiating global variable...
load mri; % load mri...
axes(handles.coronal_axes); % setting axes...
T = maketform('affine',[-2.5 0 0; 0 1 0; 0 0 -0.5; 68.5 0 61]); % making uniform array..
R = makeresampler({'cubic','nearest','nearest'},'fill'); % using cubic interpolation...
C = tformarray(D,T,R,[4 2 1],[1 2 4],[66 128 45],[],0); % using tformarray...
C = padarray(C,[6 0 0 0],0,'both'); % adding array...
montage(C,map); % showing slices...
title('Coronal View'); % setting title...
%handles.coronal_view = true;
guidata(hObject, handles); % updating handles structure with handles and user data (see
GUIDATA)...


% --- Executes on slider movement.
function coronal_view_Callback(hObject, eventdata, handles)
% hObject    handle to coronal_view (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
global C; % initiating global variable...
axes(handles.coronal_axes); % setting axes...
coronal_slice = int64(get (hObject, 'Value')); % getting value from slider and converting
it to signed 64 bit integer...
if coronal_slice >= 1
imshow(C(:,:,coronal_slice)); % showing slice corresponding number..
title('Coronal View'); % setting title...
end
guidata(hObject, handles); % updating handles structure with handles and user data (see
GUIDATA)...


% --- Executes during object creation, after setting all properties.
function coronal_view_CreateFcn(hObject, eventdata, handles)
% hObject    handle to coronal_view (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 manual_for_coronal.
function manual_for_coronal_Callback(hObject, eventdata, handles)
% hObject    handle to manual_for_coronal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% this function will allow user to scroll through coronal slices...
global C a stop bri_cntrl; % initiating global constants...
axes(handles.coronal_axes); % setting axis as coronal axes...
writeDigitalPin(a, 'D24', 1); % setting white LED on for axial potentiometer...
handles.coronal_view = true; % setting coronal view bit ON (true)...
while handles.device == true && handles.coronal_view == true; % checking if hardware is
connected AND sagittal view is enabled...
    axes(handles.coronal_axes); % setting current axes as sagittal axes..
    if a == 0 || stop == true || handles.axial_view == true || handles.sagittal_view ==
true; % checking for necessary conditions...
        handles.coronal_view = false; stop = false; writeDigitalPin(a, 'D24', false); %
setting all values to false and turning LED off...
        break % breaking loop...
    end
    handles.range_min = (readVoltage(a, 'A3') - .5)/0.015686274509804; % reading range
min value from pot 101...
    handles.range_max = (readVoltage(a, 'A4') - .5)/0.015686274509804; % reading range
max value from pot 101...
    handles.low_in = (readVoltage(a, 'A8') - .8)/4; % reading low in value from pot
201...
    handles.high_in = (readVoltage(a, 'A9') - .8)/4; % reading high in value from pot
201...
    handles.low_out = (readVoltage(a, 'A10') - .8)/4; % reading low out value from pot
201...
    handles.high_out = (readVoltage(a, 'A11') - .8)/4; % reading high out value from pot
201...
    handles.gamma = (readVoltage(a, 'A12') - .5)/2; % reading gamma value from pot 201...
    %handles.coron_slicevalue = readVoltage(a, 'A2') - .7;
    handles.coron_slicevalue = int64((readVoltage(a, 'A2') - .7)/.075); % reading &
calibrating slice value from pot 101...
    if handles.coron_slicevalue >=1 && handles.coron_slicevalue <= 45 &&
handles.range_min < handles.range_max;
        if bri_cntrl == true && handles.low_in < handles.high_in && handles.low_in >= 0
&& handles.high_in >= 0 && handles.low_out >= 0 && handles.high_out >=0;
            img = imadjust(C(:,:,handles.coron_slicevalue),[handles.low_in;
handles.high_in],[handles.low_out; handles.high_out], handles.gamma); % adjusting the
image...
            imshow (img, [handles.range_min handles.range_max]); % showing adjusted
image...
        elseif bri_cntrl == false;
            imshow(C(:,:,handles.coron_slicevalue), [handles.range_min
handles.range_max]); % showing image ...
        end
        title('Coronal View'); % setting title...
    end
    pause(.1); % pause to stabilize graphics...
end
guidata(hObject, handles); % updating handles structure with handles and user data (see
GUIDATA)...
```

```
% --- Executes on button press in cropped_three_D_array.
function cropped_three_D_array_Callback(hObject, eventdata, handles)
% hObject    handle to cropped_three_D_array (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% this function will allow user to select the region of interest...
global cropped_3D_array; % initiating global variable...
z = int64(handles.ax_slicevalue); % setting depth of aray...
try % setting try...
    cropped = imcrop; % storing cropped image into array...
    cropped = imresize(cropped, [48 48]); % resizing image to 48 X 48 uniform array...
catch % catching error...
    errordlg('Failed to detect image..!! Please update Image.', 'GUI update..!!',
'modal'); % showing error message...
    return; % exiting loop...
end

cropped_3D_array(:,:,z) = cropped; % setting 3D array of image...
guidata(hObject, handles); % updating handles structure with handles and user data (see
GUIDATA)...


% --- Executes on button press in three_dimensional_view.
function three_dimensional_view_Callback(hObject, eventdata, handles)
% hObject    handle to three_dimensional_view (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% this function will allow user to create 3D array...

global cropped_3D_array; % initiating global 3D array...
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...
pixelValue = inputdlg({'Enter value of pixel you want to crop..'}, 'Value', [1 50],
{'0'}, options);
pixelValue = str2num(pixelValue{:}); % converting data to number...
dim = size(cropped_3D_array); % getting dimensions of array...
axes(handles.three_D_view); % setting axes...
rotate3d(handles.three_D_view, 'on'); % setting rotate option on...
z = int64(handles.ax_slicevalue); % setting z as number...

plot_option = questdlg('How would you like to create 3D model of portion??',...
                  'Select option to plot 3D', 'Use plot3', 'Use surf', 'Use Delaunay',
'Use plot3');

switch plot_option; % setting plot_option as switch...
    case 'Use plot3';
        [X,Y,Z] = ind2sub(size(cropped_3D_array),find(cropped_3D_array > pixelValue)); %
getting x, y, z co-ordinates...
        %[x,y] = find(cropped_3D_array > pixelValue);
        %plot3(x,y,z,'.');
        %hold on;
        for n = 1:1:numrows(X); %n = numrows(y)
            plot3(X(n,:), Y(n,:), Z(n,:), '.'); % plotting xyz...
            hold on; % hold is on...
        end
```

```
    case 'Use surf';
        output_image = zeros(dim(:,1), dim(:,2)); % creating a blank array...
        %F = griddedInterpolant(double(cropped));  %[xq,yq] = ndgrid(-5:.1:5); %vq = 
F(xq,yq); %surf(xq,yq,vq);
        %blank_array = zeros(dim(:,1),dim(:,2));
        for z = 1:1:dim(:,3); % setting nested loops...
            for x = 1:1:dim(:,1);
                for y = 1:1:dim(:,2);
                    pixelVal = cropped_3D_array(x, y, z); % getting pixelvalue...
                    if pixelVal > pixelValue; % checking for value...
                        output_image(x, y) = z; % setting pixel as slice value...
                    end
                end
            end
            surf(double(output_image)); % plotting image...
            hold on;    hidden off; % hold is on and hidden  offf....
        end
        %surf(double(output_image));
        %hold on;    hidden off;

    case 'Use Delaunay'; % using delaunay...
        [X,Y,Z] = ind2sub(size(cropped_3D_array),find(cropped_3D_array > pixelValue)); % 
getting arrays of x y and z co-ordinates...
        tri = delaunay(X,Y,Z); % applying delaunay transformation...
        trisurf(tri, X, Y, Z); %plotting x y and z...
        hold on;    hidden off; % hold on and hidden is offf....

    otherwise
        disp('Input parameter not selected...!!!'); % otherwise display...
end
guidata(hObject, handles); % updating handles structure with handles and user data (see 
GUIDATA)...


% --- Executes on button press in delaunay_triangulation.
function delaunay_triangulation_Callback(hObject, eventdata, handles)
% hObject    handle to delaunay_triangulation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% this function will create complete delaunay triangulation for axial,
% sagittal and coronal view...
axes(handles.three_D_view); % setting axes...
rotate3d(handles.three_D_view, 'on'); % rotete on...
global D S C; % initiating global variable...
view_choice = questdlg('Which view would you like to plot in 3-D ??',...
                  'Select view to plot 3D', 'Axial', 'Sagittal', 'Coronal', 'Axial'); % 
will ask which view to  plot...
switch view_choice; % switching choice...
    case 'Axial'; % when axial is selected...
        dim = size(D); % getting dimensions...
        array3d = zeros(dim(:,1), dim(:,2), dim(:,4)); % creating 3D array...
        for Z = 1:1:dim(:,4); % counter for Z...
            ax_slice = D(:,:,Z); % getting axial slice...
            for X = 1:1:dim(:,1); % counter for X...
                for Y = 1:1:dim(:,2); % counter for Y...
                    pixel_value = ax_slice(X, Y); % getting pixel value...
                    if pixel_value > 0
                        array3d(X, Y, Z) = pixel_value; % setting that pixel to blank 
array...
```

```
                    end
                end
            end
        end
        [x,y,z] = ind2sub(size(array3d),find(array3d)); % finding x, y, z locations of
pixels...
        tri = delaunay(x,y,z); % applying delaunay triangulation to x y z co-
ordinaties...
        trisurf(tri, x, y, z); % plotting x y z with delaunay...
        hold on;    hidden off; % hold on and hidden off...

    case 'Sagittal'; % when sagittal is selected...
        dim = size(S); % getting dimensions...
        array3d = zeros(dim(:,1), dim(:,2), dim(:,4)); % creating 3D array...
        for Z = 1:1:dim(:,4); % counter for Z...
            sag_slice = S(:,:,Z); % getting axial slice...
            for X = 1:1:dim(:,1); % counter for X...
                for Y = 1:1:dim(:,2); % counter for Y...
                    pixel_value = sag_slice(X, Y); % getting pixel value...
                    if pixel_value > 0
                        array3d(X, Y, Z) = pixel_value; % setting that pixel to blank
array...
                    end
                end
            end
        end
        [x,y,z] = ind2sub(size(array3d),find(array3d)); % finding x, y, z locations of
pixels...
        tri = delaunay(x,y,z); % applying delaunay triangulation to x y z co-
ordinaties...
        trisurf(tri, x, y, z); % plotting x y z with delaunay...
        hold on;    hidden off; % hold on and hidden off...

    case 'Coronal'; % when coronal is selected...
        dim = size(C); % getting dimensions...
        array3d = zeros(dim(:,1), dim(:,2), dim(:,4)); % creating 3D array...
        for Z = 1:1:dim(:,4); % counter for Z...
            coron_slice = C(:,:,Z); % getting axial slice...
            for X = 1:1:dim(:,1); % counter for X...
                for Y = 1:1:dim(:,2); % counter for Y...
                    pixel_value = coron_slice(X, Y); % getting pixel value...
                    if pixel_value > 0
                        array3d(X, Y, Z) = pixel_value; % setting that pixel to blank
array...
                    end
                end
            end
        end
        [x,y,z] = ind2sub(size(array3d),find(array3d)); % finding x, y, z locations of
pixels...
        tri = delaunay(x,y,z); % applying delaunay triangulation to x y z co-
ordinaties...
        trisurf(tri, x, y, z); % plotting x y z with delaunay...
        hold on;    hidden off; % hold on and hidden off...

    otherwise
        disp('Input parameter not selected...!!!');
end

% --------------------------------------------------------------------
function connect_arduino_Callback(hObject, eventdata, handles)
```

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

% --------------------------------------------------------------------
function connect_Callback(hObject, eventdata, handles)
% hObject    handle to connect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a;
try
    a = arduino('COM3', 'Mega2560', 'TraceOn', false); % connecting arduino...
    configurePin(a, 'D22', 'DigitalOutput'); % set digital output for LED...
    configurePin(a, 'D23', 'DigitalOutput'); % set digital output for LED...
    configurePin(a, 'D24', 'DigitalOutput'); % set digital output for LED...
    configurePin(a, 'D25', 'DigitalOutput'); % set digital output for LED...
    configurePin(a, 'D26', 'DigitalOutput'); % set digital output for LED...
    configurePin(a, 'D27', 'DigitalOutput'); % set digital output for LED...

    configurePin(a, 'A0', 'AnalogInput'); % set analog input for pot...
    configurePin(a, 'A1', 'AnalogInput'); % set analog input for pot...
    configurePin(a, 'A2', 'AnalogInput'); % set analog input for pot...
    configurePin(a, 'A3', 'AnalogInput'); % set analog input for pot...
    configurePin(a, 'A4', 'AnalogInput'); % set analog input for pot...

    configurePin(a, 'A8', 'AnalogInput'); % set analog input for pot...
    configurePin(a, 'A9', 'AnalogInput'); % set analog input for pot...
    configurePin(a, 'A10', 'AnalogInput'); % set analog input for pot...
    configurePin(a, 'A11', 'AnalogInput'); % set analog input for pot...
    configurePin(a, 'A12', 'AnalogInput'); % set analog input for pot...

    handles.device = true; % setting device condition bit as true...
catch
    errordlg('Failed to connect to Arduino. Please check port number, board name.',
'Connection failed..!!'); % errordlg...
end

guidata(hObject, handles); % Update handles structure...

% --------------------------------------------------------------------
function disconnect_Callback(hObject, eventdata, handles)
% hObject    handle to disconnect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a;
a = 0;
clear a;
guidata(hObject, handles);

% --- Executes on button press in shut_down.
function shut_down_Callback(hObject, eventdata, handles)
% hObject    handle to shut_down (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

system_termination = questdlg('System will shut down. You will lose any unsaved data.
Proceed anyway?',...
    'System Shutdown', 'Yes', 'No', 'No');
switch system_termination;
    case 'Yes';
        system('shutdown /s /t 0');
```

```
    case 'No';
        return;
    otherwise
        disp('You just cancelled shutdown...!!');
end
guidata(hObject, handles);

% --- Executes on button press in brightness_control.
function brightness_control_Callback(hObject, eventdata, handles)
% hObject    handle to brightness_control (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global bri_cntrl;   bri_cntrl = true;
guidata(hObject, handles);

% --- Executes on button press in stop_all_view.
function stop_all_view_Callback(hObject, eventdata, handles)
% hObject    handle to stop_all_view (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global stop bri_cntrl;    stop = true;  bri_cntrl = false;
guidata(hObject, handles);


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

% --------------------------------------------------------------------
function clear_robot_model_Callback(hObject, eventdata, handles)
% hObject    handle to clear_robot_model (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cla(handles.axes1,'reset');
guidata(hObject, handles);

% --------------------------------------------------------------------
function clear_axial_view_Callback(hObject, eventdata, handles)
% hObject    handle to clear_axial_view (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cla(handles.axial_axes,'reset');
guidata(hObject, handles);

% --------------------------------------------------------------------
function clear_sagittal_view_Callback(hObject, eventdata, handles)
% hObject    handle to clear_sagittal_view (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cla(handles.sagittal_axes,'reset');
guidata(hObject, handles);

% --------------------------------------------------------------------
function clear_coronal_view_Callback(hObject, eventdata, handles)
% hObject    handle to clear_coronal_view (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cla(handles.coronal_axes,'reset');
guidata(hObject, handles);
```

```
% --------------------------------------------------------------------
function clear_three_D_view_Callback(hObject, eventdata, handles)
% hObject    handle to clear_three_D_view (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cla(handles.three_D_view,'reset');
guidata(hObject, handles);
```

B. Video Link

Please check video tutorial of Image Processing in Robotics.

https://youtu.be/xCtRKyN0DXQ