REFERENCES

[1] O. Allkanjari, and A.Vitalone, “What do we know about phytotherapy of benign prostatic hyperplasia?,” Life Sci., vol. 126, pp. 42–56, 20157

[2] S.A.Kaplan and K.T.Mcvary. Male lower urinary tract system and BPH,2nd ed, Wiley Blackwell, 2014

[3] Yu X, Elliott SP, Wilt TJ, McBean AM. Practice patterns in benign prostatic hyperplasia surgical therapy: the dramatic increase in minimally invasive technologies. J Urol. 2008;180(1):241-5; discussion

[4] Bolt JW, Evans C, Marshall VR. Sexual dysfunction after prostatectomy. Br J Urol. 1987;59(4):319-22.

[5] Chung A, Woo HH. Preservation of sexual function when relieving benign prostatic obstruction surgically: can a trade-off be considered? Curr Opin Urol. 2016;26(1):42-8.

[6] Vecchis DE. Preservation of anterograde ejaculation after transurethral resection of both the prostate and bladder neck. Brit J Urol. 1998;81(6):830-3.

[7] Yu X. , Elliott S.P. ,Wilt T. J., McBean A.M.(2009). Practice patterns in benign prostatic hyperplasia surgical therapy: the dramatic increase in minimally invasive technologies. J Urol, 180: 241-245.

[8] Madersbacher S, Marberger M. (1999). Is transurethral resection of the prostate still justified? Br. J Urol.

[9] Gamage P., Xie S. Q., Delmas P., Xu W. L., (2011). Diagnostic radiograph based 3D bone reconstruction framework: Application to the femur. Comput. Med. Imaging Graph, 35, 427–437.

[10] P. E. Fadero and M. Shah, “Three dimensional (3D) modelling and surgical planning in trauma and orthopaedics.,” Surgeon, vol. 12, no. 6, pp. 6–11, 2014.

[11] D. Cool, D. Downey, J. Izawa, J. Chin, and A. Fenster, “3D prostate model formation from non-parallel 2D ultrasound biopsy images,” Med. Image Anal., vol. 10, pp. 875–887, 2006.

[12] F. a. Cosío, “Automatic initialization of an active shape model of the prostate,” Med. Image Anal., vol. 12, pp. 469–483, 2008.

[13]A. S. Korsager, U. L. Stephansen, J. Carl, and L. R. Østergaard, “The use of an active appearance model for automated

[14] V. Singh, “Introduction” in Digital Image Processing with MatLab and LabView, Bangalore, India, 2013,pp. 2

[15] M. Kwacz, J. Wysocki, and P. Krakowian, “Reconstruction of the 3D Geometry of the Ossicular Chain Based on Micro-CT Imaging,” Biocybern. Biomed. Eng., vol. 32, no. 1, pp. 27–40, 2012.

[16] E. Bermejo, O. Cordón, S. Damas, and J. Santamaría, “A comparative study on the application of advanced bacterial foraging models to image registration,” Inf. Sci. (Ny)., vol. 295, pp. 160–181, 2015.

[17] L. Shen, S. Kim, and A. J. Saykin, “Fourier method for large-scale surface modeling and registration,” Comput. Graph., vol. 33, no. 3, pp. 299–311, 2009.

[18] P. Markelj, D. Tomazevic, B.Likar and F. Pernus. “A review of 3D/2D registration ethods for image-guided interventions,” Journal of Medical Image analysis. 16, 2012.

[19] H. Allioui, M. Sadgal and A Elfazzki., “A cooperative approach for 3D image segmentation”, IEEE, 2016

[20] O. Friman, M. Hindennach, C. Kühnel, and H. O. Peitgen, “Multiple hypothesis template tracking of small 3D vessel structures,” Med. Image Anal., vol. 14, no. 2, pp. 160–171, 2010.

[21] J. O. Pentecost, J. Icardo, and K. L. Thornburg, “3D computer modeling of human cardiogenesis.,” Comput. Med. Imaging Graph., vol. 23, pp. 45–49, 1999.

[22] R. Audigier, R. Lotufo, and A. Falcão, “3D visualization to assist iterative object definition from medical images,” Comput. Med. Imaging Graph., vol. 30, pp. 217–230, 2006.

[23] H.F.Garc´ıa† et al., “3D Brain Atlas Reconstruction Using Deformable Medical Image Registration: Application to Deep Brain Stimulation Surgery”. Harvard publications,

[24] C. Kim, J Yoon and Y.J Lee. “Medical image segmentation by more sensitive adaptive thresholding”, 2016

[25] A. Skalski, T. Dreweniak, and J. Jakubowski. “Kidney Tumor segmentation and detection on computed tomography data”. 2016

[26] L. Zhu, Y Li, Y Yu, B Zhang and L Wang. “3D construction for soft tissue of the human body”, 2016.

[27] Cosio F.A.,”Automatic initialization of an active shape model of prostate, Medical Imgae Analysis”, 2008

[28] A. Fenster, A. Ward, C. Crukley, E. Gibson, G. Bauman, J. Gómez, J. Chin, M. Moussa, M. Gaed, and S. Pautler, “3D prostate histology image reconstruction: Quantifying the impact of tissue deformation and histology section location,” J. Pathol. Inform., vol. 4, p. 31, 2013.

APPENDIX – A: MATHLAB CODES

A.1 Registration-1st step

```
function varargout = register_new(varargin)
% REGISTER_NEW MATLAB code for register_new.fig
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @register_new_OpeningFcn, ...
                   'gui_OutputFcn',  @register_new_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 register_new is made visible.
function    register_new_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 register_new (see VARARGIN)

% Choose default command line output for register_new
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes register_new wait for user response (see UIRESUME)
% uiwait(handles.figure1);
varargout{1} = handles.output;
% paths           % Running the path setter

global I h w index imdirectory Files currentset impath

index=1;
imdirectory='S1_data/';
Files=dir([imdirectory '/*jpg' ]);
currentset=Files(index).name;
impath=[imdirectory '/' Files(index).name];
I=imread(impath);
```

```
[h, w, ~] = size(I);
axes(handles.axes1)
imshow(I)


set(handles.infobox,'String',[currentset ' Loaded....'])


% --- Outputs from this function are returned to the command line.
function varargout = register_new_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;

global I h w
% paths
hold on
imshow(I)

[h, w, ~] = size(I);
axes(handles.axes1)




% --- Executes on button press in top.
function top_Callback(hObject, eventdata, handles)
% hObject    handle to top (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1)
hold on


for i=1:6
    [x(i),y(i)]=ginput(1);
    plot(x(i),y(i),'r*')
end


global mt ct w
[M]=polyfit(x,y,1);
mt= M(1)
ct= M(2)
xc= 1:w;
yc= mt*xc;
plot(xc, yc+ct, 'b');



% --- Executes on button press in left.
function left_Callback(hObject, eventdata, handles)
% hObject    handle to left (see GCBO)
```

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

axes(handles.axes1)
hold on


for i=1:6
    [x(i),y(i)] = ginput(1)
    plot(x(i),y(i),'r*')
end


global ml cl w
[M] = polyfit(x,y,1);
ml= M(1)
cl= M(2)
xc= 1:w;
yc= ml*xc;
plot(xc, yc+cl, 'b');


% --- Executes on button press in right.
function right_Callback(hObject, eventdata, handles)
% hObject    handle to right (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1)
hold on

for i=1:6
    [x(i),y(i)]=ginput(1);
    plot(x(i),y(i),'r*')
end


global mr cr w
[M]=polyfit(x,y,1);
mr= M(1)
cr= M(2)
xc= 1:w;
yc= mr*xc;
plot(xc, yc+cr, 'b');

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

axes(handles.axes1)
hold on

for i=1:6
    [x(i),y(i)]=ginput(1);
    plot(x(i),y(i),'r*')
end
```

```
global mb cb w I J
[M]=polyfit(x,y,1);
mb= M(1)
cb= M(2)
xc= 1:w;
yc= mb*xc;
plot(xc, yc+cb, 'b');

% J=imrotate(I,rad2deg(atan(poly(mb))));
% axes(handles.axes1)
% imshow(J)



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

global ml mb mr mt cl cb cr ct currentset  Files index corpath ltx lty
rtx rty rbx rby lbx lby
ltx = (cl-ct)/(mt-ml);
lty = ml*ltx+cl;
rtx = (cr-ct)/(mt-mr);
rty = mr*rtx+cr;
rbx = (cr-cb)/(mb-mr);
rby = mr*rbx+cr;
lbx = (cl-cb)/(mb-ml);
lby = ml*lbx+cl;



axes(handles.axes1)
hold on
plot(ltx,lty,'bd','MarkerEdgeColor','k',...
                'MarkerFaceColor',[.49 1 .63],...
                'MarkerSize',10);

plot(rtx,rty,'cd','MarkerEdgeColor','k',...
                'MarkerFaceColor',[.49 1 .63],...
                'MarkerSize',10);

plot(rbx,rby,'yd','MarkerEdgeColor','k',...
                'MarkerFaceColor',[.49 1 .63],...
                'MarkerSize',10);

plot(lbx,lby,'rd','MarkerEdgeColor','k',...
                'MarkerFaceColor',[.49 1 .63],...
                'MarkerSize',10);

rot_angle = rad2deg(atan(mb));
corpath = 'RESULTS\Variables\NewResearch_1mm_S1\set2\';
save([corpath                  '\'                  Files(index).name
'.mat'],'ltx','lty','lbx','lby','rtx','rty','rbx','rby','rot_angle');
```

```
text(ltx-120,lty-100,'P1','BackgroundColor',[.7 .9
.7],'HorizontalAlignment','right');
text(lbx-100,lby+100,'P2','BackgroundColor',[.7 .9
.7],'HorizontalAlignment','right');
text(rbx+100,rby+100,'P3','BackgroundColor',[.7 .9
.7],'HorizontalAlignment','left');
text(rtx+40,rty-40,'P4','BackgroundColor',[.7 .9
.7],'HorizontalAlignment','left');




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

global J I mb

J=imrotate(I,rad2deg(atan(mb)));
axes(handles.axes1)
imshow(J)



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



global index currentset Files impath imdirectory I mb J rotpath corpath


index=index+1;

currentset=Files(index).name;
impath=[imdirectory '/' Files(index).name];
I=imread(impath);
axes(handles.axes1);
imshow(I);

% rotpath='Data\rotimages\'
% imwrite(J,[rotpath '\' Files(index).name]);



set(handles.infobox, 'String',[currentset ' Loaded.....'])

set(handles.infobox, 'String',[currentset ' Saved.....'])
```

A.2 Registration-2nd step

```
%% Initialize
clear all
clc
%% Set Paths
current_sub_folder = 'S1_1mm\';      % Give path
impath = ['raw\' current_sub_folder];
varpath = ['Variables\' current_sub_folder];
savpath = ['Cropped_Output\' current_sub_folder];
addpath('Functions');
% distort = 0;                    % Do you want the output distorted?
1, else 0
% scale_factor = 1;               % Scale using widths? 1, else 0
ImDir = dir(impath);

%% General functions for all images
% if distort == 1
%     savpath = [savpath 'distorted\'];
% else
%     if scale_factor == 1
%         savpath = [savpath 'original\width\'];
%     else
%         savpath = [savpath 'original\height\'];
%     end
% end

% if unavailable, find and save min_blk_height and blk_height params
at
% variables path

if   ~exist([varpath   'min_blk_height.mat'],'file')||~exist([varpath
'blk_heights.mat'],'file')||~exist([varpath
'blk_widths.mat'],'file')||~exist([varpath
'min_blk_width.mat'],'file')
    [blk_heights,blk_widths,min_blk_height,min_blk_width]           =
findMinBlockHeight(varpath);
else
    load([varpath 'min_blk_height.mat'])
    load([varpath 'blk_heights.mat'])
    load([varpath 'blk_widths.mat'])
    load([varpath 'min_blk_width.mat'])
end


%% Image Explorer
for i = 3:length(ImDir)           % Correct code
    % for i = 3:3                         % Comment after finding all
coordinates
    current_image = imread([impath ImDir(i).name]);

    %% Rotation of Image and tracking intersect points...
    [im_height, im_width, ~] = size(current_image);
    cx = round(im_width/2);
    cy = round(im_height/2);

    load([varpath ImDir(i).name '.mat']);
```

```
    rotated_image = imrotate(current_image,rot_angle,'crop');        %
Rotate the image
    inv_rot_angle = -rot_angle;                                      %
Invert the angle to facilitate coord rot.
    old_coord    =    [lbx-cx,lby-cy,ltx-cx,lty-cy,rbx-cx,rby-cy,rtx-
cx,rty-cy];  % Old_Coord @origin
    rot_mat    =    [cosd(inv_rot_angle),    sind(inv_rot_angle);    -
sind(inv_rot_angle), cosd(inv_rot_angle)];

    for j = 1:2:length(old_coord)-1
        new_coord(j:j+1)  =  old_coord(j:j+1)*rot_mat;               %
Coord(x,y)*rotation_matrix
    end
    new_coord(1:2:end) = new_coord(1:2:end)+cx;             % Bias the
coordinates back to center
    new_coord(2:2:end) = new_coord(2:2:end)+cy;

    %       Visualize  all  outputs  -  Warning!  Comment  before  running
bulk!!!
    %     figure
    %     subplot 121
    %     imshow(current_image)
    %     hold on
    %     plot([lbx ltx rbx rtx],[lby lty rby rty],'g*');
    %     subplot 122
    %     imshow(rotated_image)
    %     hold on
    %     plot([lbx ltx rbx rtx],[lby lty rby rty],'g*');
    %     plot(new_coord(1:2:end),new_coord(2:2:end),'r*');
    %     title('Old-coord - green, New-coord - red');

    %% Rescaling all images and tracking intercept points...

%     if distort==1              % Scale by both height and width
%           height_scale_factor = min_blk_height/blk_heights(i-2); %
Down scaling only! (height)
%          width_scale_factor = min_blk_width/blk_widths(i-2);      %
Down scaling only! (width)
%                                                 rescaled_image     =
imresize(rotated_image,[height_scale_factor*im_height
width_scale_factor*im_width]);
%         rescaled_coord = new_coord;
%                                        rescaled_coord(2:2:end)     =
height_scale_factor*rescaled_coord(2:2:end);
%                                        rescaled_coord(1:2:end)     =
width_scale_factor*rescaled_coord(1:2:end);
%
%     else                       % Preserve original aspect ratio
%         if scale_factor == 1    % Scale using widths
%            width_scale_factor = min_blk_width/blk_widths(i-2);     %
Down scaling only! (width)
%                                                   rescaled_image    =
imresize(rotated_image,width_scale_factor);
%             rescaled_coord = width_scale_factor*new_coord;
%         else                   % Scale using heights
%              height_scale_factor = min_blk_height/blk_heights(i-2);
% Down scaling only! (height)
%                                                   rescaled_image    =
imresize(rotated_image,height_scale_factor);
%             rescaled_coord = height_scale_factor*new_coord;
```

```
%         end
%     end


%     figure;
%     imshow(rescaled_image)

 height_scale_factor = min_blk_height/blk_heights(i-2); % Down scaling
only! (height)
    width_scale_factor  =  min_blk_width/blk_widths(i-2);       %  Down
scaling only! (width)


    % These two lines are for a non-distorted image!
        rescaled_image = imresize(rotated_image,height_scale_factor);
        rescaled_coord = height_scale_factor*new_coord;

    % These  four  lines  are  to  resize  with  distortion  to  get  higher
accuracy!

    rescaled_image                                                        =
imresize(rotated_image,[height_scale_factor*im_height
width_scale_factor*im_width]);
    rescaled_coord = new_coord;
    rescaled_coord(2:2:end)                                               =
height_scale_factor*rescaled_coord(2:2:end);
    rescaled_coord(1:2:end)                                               =
width_scale_factor*rescaled_coord(1:2:end);

    %         Visualize all outputs - Warning! Comment before running
bulk!!!
    %         figure
    %         subplot 121
    %         imshow(rotated_image)
    %         hold on
    %         plot([lbx ltx rbx rtx],[lby lty rby rty],'g*');
    %         plot(new_coord(1:2:end),new_coord(2:2:end),'r*');
    %         title('Old-coord - green, New-coord - red');
    %         subplot 122
    %         imshow(rescaled_image)
    %         hold on
    %         plot(new_coord(1:2:end),new_coord(2:2:end),'g*')
    %
plot(rescaled_coord(1:2:end),rescaled_coord(2:2:end),'r*');
    %         title('Old-coord - green, New-coord - red');

    %% Cropping the image to the size of the block
    %    old_coord  =  [lbx-cx,lby-cy,ltx-cx,lty-cy,rbx-cx,rby-cy,rtx-
cx,rty-cy];  % Old_Coord @origin
%          cropped_image  =  imcrop(rescaled_image,[rescaled_coord(3)
rescaled_coord(4)            abs(rescaled_coord(7)-rescaled_coord(3))
abs(rescaled_coord(4)-rescaled_coord(2))]);
%     imwrite(cropped_image,[savpath ImDir(i).name]);



   cropped_image      =      imcrop(rescaled_image,[rescaled_coord(3)
rescaled_coord(4)            abs(rescaled_coord(7)-rescaled_coord(3))
abs(rescaled_coord(4)-rescaled_coord(2))]);
    [crop_height,crop_width,~] = size(cropped_image);
```

```
    %       Visualize all outputs - Warning! Comment before running bulk!!!
    %     [crop_height,crop_width,~] = size(cropped_image);
    %
    %     figure
    %     imshow(cropped_image)
    %      title([num2str(crop_height) '\times' num2str(crop_width) ' = ' num2str(crop_height*crop_width)]);

     figure
    imshow(cropped_image)
    title([num2str(crop_height) '\times' num2str(crop_width) ' = ' num2str(crop_height*crop_width)]);
    J=cropped_image;
   imwrite(J,[savpath '\' ImDir(i).name ]);
end
```

A.3 Registration Analysis

```
clear all
clc

%% Load data for analysis
current_sub_folder = 'S1\new\';      % Give path
varpath = ['Variables\' current_sub_folder];
% crop_path = ['Cropped_Output\' current_sub_folder];
crop_path=['Cropped_Output\' current_sub_folder];
impath = ['raw\' current_sub_folder];
if exist([varpath 'Analysis_output.mat'],'file')
    load([varpath 'Analysis_output.mat'])
else

    load([varpath 'blk_heights.mat'])
    load([varpath 'blk_widths.mat'])

    %% Raw Variation in ratio and area
    Tau_raw = blk_heights./blk_widths;
    Rho_raw = blk_heights.*blk_widths;

    %% Variation in crop with no distortion - height param
    Crop_no_distort_path = [crop_path 'original\height\'];
    CropNoDistDir = dir(Crop_no_distort_path);
    crop_orig_height  =  zeros(1,length(CropNoDistDir)-2);         % Definition
    crop_orig_width = crop_orig_height;
    for i = 3:length(CropNoDistDir)
        [crop_orig_height(i-2),crop_orig_width(i-2),~]            = size(imread([Crop_no_distort_path CropNoDistDir(i).name]));
    end
    Tau_rch = crop_orig_height./crop_orig_width;
    Rho_rch = crop_orig_height.*crop_orig_width;

    %% Variation in crop with no distortion (Tau_rc) - width param
    Crop_no_distort_path = [crop_path 'original\width\'];
```

```
    CropNoDistDir = dir(Crop_no_distort_path);
    crop_orig_height  =  zeros(1,length(CropNoDistDir)-2);         %
Definition
    crop_orig_width = crop_orig_height;
    for i = 3:length(CropNoDistDir)
        [crop_orig_height(i-2),crop_orig_width(i-2),~]              =
size(imread([Crop_no_distort_path CropNoDistDir(i).name]));
    end
    Tau_rcw = crop_orig_height./crop_orig_width;
    Rho_rcw = crop_orig_height.*crop_orig_width;

    %% Variation in after crop - with distorted
    Crop_distort_path = [crop_path 'distorted\'];
    CropDistDir = dir(Crop_no_distort_path);
    crop_dist_height  =  zeros(1,length(CropNoDistDir)-2);         %
Definition
    crop_dist_width = crop_orig_height;
    for i = 3:length(CropNoDistDir)
        [crop_dist_height(i-2),crop_dist_width(i-2),~]              =
size(imread([Crop_distort_path CropDistDir(i).name]));
    end
    Tau_dc = crop_dist_height./crop_dist_width;
    Rho_dc = crop_dist_height.*crop_dist_width;

end

%% View ratio outputs
figure
set(gcf, 'Position', get(0,'Screensize')); % Maximize figure.
plot(Tau_raw,'r^')
hold on
plot(Tau_rch,'b.')
plot(Tau_rcw,'m.')

plot(Tau_dc,'ko')
grid minor
title('Measured block height-width ratio comparison');
xlabel('Image Number - subfolder-S3');
ylabel('Variants of \tau = Height:Width');
legend('\tau_{raw}','\tau_{rch}','\tau_{rcw}','\tau_{dc}','Location',
'northwest')


%% View area outputs
figure
set(gcf, 'Position', get(0,'Screensize')); % Maximize figure.
plot(Rho_raw,'r^')
hold on
plot(Rho_rch,'b.')
plot(Rho_rcw,'m.')
plot(Rho_dc,'ko')
grid minor
title('Measured block area comparison');
xlabel('Image Number - subfolder-S3');
ylabel('Variants of \rho = Height\times Width');
legend('\rho_{raw}','\rho_{rch}','\rho_{rcw}','\rho_{dc}','Location',
'southeast')
```

```
save([varpath                                'Analysis_output.mat'],
'Tau_raw','Tau_rch','Tau_rcw','Tau_dc','Rho_raw','Rho_rch','Rho_rcw',
'Rho_dc');

%% Showing the difference in the form of images
% Finding the worst affected image
[~,max_diff_idx] = max(abs(Tau_raw-Tau_dc));
Im_dir = dir(im_path);
figure
set(gcf, 'Position', get(0,'Screensize')); % Maximize figure.
subplot 221
imshow([im_path Im_dir(max_diff_idx-3).name]);  % Show the raw image
title('Original Image');
subplot 222
imshow([crop_path 'distorted\' Im_dir(max_diff_idx-3).name]);  % Show
the distorted crop
title('Distorted');
subplot 223
imshow([crop_path 'original\height\' Im_dir(max_diff_idx-3).name]);  %
Show the height scaled crop
title('Non-Distorted Height Scaled');
subplot 224
imshow([crop_path 'original\width\' Im_dir(max_diff_idx-3).name]);  %
Show the widht scaled crop
title('Non-Distorted Width Scaled');
```

A.4 Segmentation

```
function varargout = MAIN(varargin)
% MAIN MATLAB code for MAIN.fig
%      MAIN, by itself, creates a new MAIN or raises the existing
%      singleton*.
%
%      H = MAIN returns the handle to a new MAIN or the handle to
%      the existing singleton*.
%
%        MAIN('CALLBACK',hObject,eventData,handles,...) calls the
local
%        function named CALLBACK in MAIN.M with the given input
arguments.
%
%       MAIN('Property','Value',...) creates a new MAIN or raises
the
%     existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before MAIN_OpeningFcn gets called.  An
%       unrecognized property name or invalid value makes property
application
%     stop.  All inputs are passed to MAIN_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 MAIN

% Last Modified by GUIDE v2.5 11-Mar-2016 10:56:13

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @MAIN_OpeningFcn, ...
    'gui_OutputFcn',  @MAIN_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 MAIN is made visible.
function MAIN_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 MAIN (see VARARGIN)

% Choose default command line output for MAIN
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

addpath('Functions')
% UIWAIT makes MAIN wait for user response (see UIRESUME)
% uiwait(handles.figure1);
%
global impath I
global currentSet  Files Imdirectory index
index = 1;
Imdirectory='Data\S3\';
Files = dir([Imdirectory '/*.jpg']);
currentSet = Files(index).name;
impath = [Imdirectory '/' Files(index).name];
I = imread(impath);


% set(handles.infoBox,'String',['Image ' impath ' loaded. Select
Left and Right masks...'])

%%

% global fname
% % fname = 'P3290177';
```

```
% I = imread(['Data\S3\' fname '.jpg']);

% axes(handles.axes1);
% imshow(I);

% I = rgb2gray(imread('P3290224.jpg'));
% imshow(I)
%  str = 'Click to select initial contour location. Double-click to
confirm and proceed.';
% title(str,'Color','b','FontSize',12);
% disp(sprintf('\nNote: Click close to object boundaries for more
accurate result.'))
% mask = roipoly;
%
% figure, imshow(mask)
% title('Initial MASK');
% maxIterations = 200;
% bw = activecontour(I, mask, maxIterations, 'Chan-Vese');
%
% % Display segmented image
% figure, imshow(bw)
% title('Segmented Image');
%
global I impath
axes(handles.axes1);
imshow(I);

global radiusR
global radiusL
radiusR = 30;
radiusL = 30;
global xR
global yR
global xL
global yL ix iy

[ix, iy, ~]=size(I);

xL = 100;
yL = 100;
xR = 200;
yR = 200;


global hcirc
if radiusL > 0
    hcirc = viscircles([xL yL],radiusL,'EdgeColor','m');
end
global hcircR
if radiusR > 0
    hcircR = viscircles([xR yR],radiusR,'EdgeColor','r');
end



% --- Outputs from this function are returned to the command line.
function varargout = MAIN_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 mouse press over figure background, over a
disabled or
% --- inactive control, or over an axes background.
function figure1_WindowButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


global xL xR yL yR radiusR radiusL
global hcirc
global hcircR
global flag

if strcmp(get(handles.figure1,'selectionType') , 'normal')
    flag = 1;                       % Left click = 1
    axes(handles.axes1);
    [xL,yL] = ginput(1);

    if radiusL > 0
        delete(hcirc)
    end
    if radiusL > 0
        hcirc = viscircles([xL yL],radiusL,'EdgeColor','m');
    end
end
if strcmp( get(handles.figure1,'selectionType') , 'alt')
    flag = 0;               % Right click = 0
    axes(handles.axes1);
    [xR,yR] = ginput(1);
    if radiusR > 0
        delete(hcircR)
    end
    if radiusR > 0
        hcircR = viscircles([xR yR],radiusR,'EdgeColor','r');
    end
end


% --- Executes on scroll wheel click while the figure is in focus.
function figure1_WindowScrollWheelFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  structure with the following fields (see FIGURE)
%    VerticalScrollCount: signed integer indicating direction and
number of clicks
%   VerticalScrollAmount: number of lines scrolled for each click
% handles    structure with handles and user data (see GUIDATA)
set(gcf, 'WindowScrollWheelFcn', @figScroll);

function figScroll(src,evnt)
direction = evnt.VerticalScrollCount;
```

```
global radiusR radiusL
global hcirc
global xL xR
global yL yR
global hcircR
global flag

if flag
    if radiusL - 5*direction < 0
        radiusL = 5;
    else
        radiusL = radiusL - 5*direction;
    end
    if radiusL > 0
        delete(hcirc)
    end

    if radiusL > 0
        hcirc = viscircles([xL yL],radiusL,'EdgeColor','m');
      end

else
    if radiusR - 5*direction < 0
        radiusR = 5;
    else
        radiusR = radiusR - 5*direction;
    end
    if radiusR > 0
        delete(hcircR)
    end

    if radiusR > 0
        hcircR = viscircles([xR yR],radiusR,'EdgeColor','r');
    end
end


% --- Executes on button press in addmask.
function addmask_Callback(hObject, eventdata, handles)
% hObject    handle to addmask (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global xL yL xR yR ix iy
global I

global radiusL radiusR currentSet meshL meshR

meshL = mesh_grid(xL, yL, ix, iy, radiusL);
meshR = mesh_grid(xR, yR, ix, iy, radiusR);
% axes(handles.axes1);
% I = rgb2gray(imread('P3290262.jpg'));
% imshow(I)
% BW = roipoly;
%
%
% maxIterations = 200;
% seg = activecontour(I, mask, maxIterations, 'Chan-Vese');
```

```
%
% % Display segmented image
% figure, imshow(seg)
% title('Segmented Image');



save(['Masks\' currentSet(1:end-4) '.mat'],'meshL','meshR')
set(handles.infoBox,'String',['Mask for ' currentSet ' saved...'])




% --- Executes on button press in optimize.
function optimize_Callback(hObject, eventdata, handles)
% hObject    handle to optimize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
addpath('Functions\Optimizer');
global I meshL meshR hOvm
%                            optiMaskL                            =
deploy_snake(rgb2gray(I),meshL,maxIterations,algorithm,smoothness)
;
%                            optiMaskR                            =
deploy_snake(rgb2gray(I),meshR,maxIterations,algorithm,smoothness)
;
size(I)
size(meshL)
algorithm = get(handles.chanvase,'Value');
iterations = str2double(get(handles.iterations,'String'));
smoothness = 2*get(handles.smooth,'Value');


optiMaskL                                                         =
deploy_snake(rgb2gray(I),meshL,iterations,algorithm,smoothness);
optiMaskR                                                         =
deploy_snake(rgb2gray(I),meshR,iterations,algorithm,smoothness);
axes(handles.axes1)
if exist('hOvm')
    delete(hOvm)
end
hOvm = alphamask(optiMaskL|optiMaskR);


  h = msgbox({'Operation' 'Completed'});




% --- Executes on button press in next.
function next_Callback(hObject, eventdata, handles)
% hObject    handle to next (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Files Imdirectory currentSet index meshL meshR I
index = index+1;
currentSet = Files(index).name;
impath = [Imdirectory '/' Files(index).name];
I = imread(impath);
axes(handles.axes1);
imshow(I);
```

```
global radiusR
global radiusL
radiusR = 30;
radiusL = 30;
global xR
global yR
global xL
global yL ix iy

[ix, iy, ~]=size(I);

xL = 100;
yL = 100;
xR = 200;
yR = 200;


global hcirc
if radiusL > 0
    hcirc = viscircles([xL yL],radiusL,'EdgeColor','m');
end
global hcircR
if radiusR > 0
    hcircR = viscircles([xR yR],radiusR,'EdgeColor','r');
end



% save(['OptiMasks\' currentSet(1:end-4) '.mat'],'meshL','meshR')
% save(['OptiMasks\' currentSet(1:end-4) '.raw'],'meshL','meshR')
save(['OptiMasks\' currentSet(1:end-4) '.mat'],'meshL','meshR')


set(handles.infoBox,'String',['Mask for ' currentSet ' saved...'])


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



function iterations_Callback(hObject, eventdata, handles)
% hObject    handle to iterations (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 iterations as
text
%        str2double(get(hObject,'String')) returns contents of
iterations as a double


% --- Executes on slider movement.
```

```
function smooth_Callback(hObject, eventdata, handles)
% hObject    handle to smooth (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


% --- Executes during object creation, after setting all properties.
function smooth_CreateFcn(hObject, eventdata, handles)
% hObject    handle to smooth (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
```

A.5 Preparation for modelling

```
function align_urethra
fname='S7';                          % call images from subset:S7
A=dir(['prostate_mask\' fname]);      % list the folder contains:
'prostate mask'
mkdir('binMatrices');             % make a new folder: 'binMatrices'
n=length(A);                            % run all the images inside
variable:'A'
count=1;                             % 1st count is taken as 1

%convert to 1 layer
for i=3:n
    load(['prostate_mask\' fname '\' A(i).name]);       % load the:
'prostate mask'
    maskedRgbImage=(maskedRgbImage(:,:,1)>0);             % convert
to 1D binary
    save(['binMatrices\' A(i).name],'maskedRgbImage');   % save it
inside: 'binMatrices'
end


% Combine 3 layers
mkdir('CombinedMatrices')                                    % make
a new folder: 'combinedMatrices'
for i=3:n
    load(['binMatrices\' A(i).name]);                        % load
the: 'binMatrices'
    load(['ducts_n_urethra\' fname '\' A(i).name]);          % load
the: 'ducts_n_urethra'
    FinalMesh=meshL + meshM + meshR;                        % Final
mesh comprise of meshL(~Left Duct),meshM(~Urethra)& meshR(~Right
Duct)
```

```
    maskedRgbImage=maskedRgbImage-FinalMesh;                      %
Determine the: maskedRgbImage(~Prostate mask)
    save(['CombinedMatrices\' A(i).name],'maskedRgbImage'); % save
maskedRgbImage in the: 'combinedMatrices'
end

% Overlap Urethra
mkdir('overlappedMatrices')                                    % make
a new folder:'overlappedMatrices'

% Take out the reference
load(['ducts_n_urethra\' fname '\' A(3).name]);              % load:
'ducts_n_urethra'
ure_center=bwmorph(meshM,'shrink','inf');                    % shrink
objects to points for meshM (~Urethra)
[urefy,urefx]=find(ure_center);                                % find
the   center(x,y   coordinates)   of   the   1st   image   of   the
Urethra(REFERENCE)


for i=3:n
    load(['combinedMatrices\'  A(i).name]);                    % load
the: 'combinedMatrices'
   load(['ducts_n_urethra\' fname '\' A(i).name]);             % load
: 'ducts_n_urethra'
   ure_center=bwmorph(meshM,'shrink','inf');                 % shrink
objects to points for meshM (~Urethra)for the rest of the  images
[uy,ux]=find(ure_center);                                      % find
the center(x,y) of the urethra for rest of the images within the
same set(set:S7)
dy=urefy-uy;                                                   % find
translated y distance (dy)
dx=urefx-ux;                                                   % find
translated x distance (dx)
FinalMesh=meshL + meshM + meshR;                              % final
mesh  consist  of  meshL(~Left  Duct),meshM(~Urethra)&  meshR(~Right
Duct)
    maskedRgbImage=maskedRgbImage-FinalMesh;                   % find
the maskedRgbImage (~prostate mask)

maskedRgbImage=imtranslate(maskedRgbImage,[dy,dx],'fillvalues',255
,'outputview','full'); % translated all the objects by the value of
[dy,dx]
save(['overlappedMatrices\' A(i).name],'maskedRgbImage');   % save
it as : 'overlappedMatrices'
end


% % testing purposes
% imshow(maskedRgbImage)
% hold on
% plot(urefx,urefy,'rx');
% pause(5)
%
%
% imshow(maskedRgbImage)
%
% figure;
% plot(ux,uy,'gx');
% hold off
```

```
%
%    stackedMat(:,:,i-2)=maskedRgbImage(urefy-682:urefy+350,urefx-
443:urefx+550);
% end

% find the largest dim
A=dir('combinedMatrices');
a=0;
b=0;
n=length(A)-2;                                  % nnumber of matrices
in the stack
for i=3:n+2
    load(['combinedMatrices\'  (A(i).name)]);        %  load  the:
'combinedMatrices'
    [p,q]=size(maskedRgbImage);                   % find the rows and
columns (size) of the maskedRgbImage
    if p>a                                         % Find the largest
height
       a=p;
    end
    if q>b                                         % find the largest
width
        b=q;
    end
end

mkdir('PaddedMatrices')                          % make a new folder:
'PaddedMatrices'
for i=3:n+2
    load(['combinedMatrices\'   (A(i).name)]);           %   load:
'combinedMatrices'
    [p,q]=size(maskedRgbImage);                    % find the size of
the maskedRgbImage
    maskedRgbImage=padarray(maskedRgbImage,[round((a-
p)/2)],[round((b-q)/2)]);  %  perform  padding  to  the  largest
dimension
    save(['PaddedMatrices\'          A(i).name],'maskedRgbImage');
% save maskedRgbImage as: 'PaddedMatrices'
end



% Perform Cropping
mkdir('croppedMatrices')                                      % make
a new folder: 'croppedMatrices'

for i = 3:n+2
    load(['paddedMatrices\' (A(i).name)]);                  % load:
'paddedMatrices'
    maskedRgbImage = imcrop(maskedRgbImage,[0  0  b  a]);      % Not
happy with 0 0? (crop the image). now a and b contains the largest
dimensions
    save(['croppedMatrices\' A(i).name],'maskedRgbImage');  % save
the maskedRgbImage as: 'croppedMatrices'
end


% % Perform Stacking
stackedMat = zeros(a,b,n);                         % create a matrix
of all zeros
```

```
for i = 3:n+2
    load(['croppedMatrices\'   (A(i).name)]);             %   load:
'croppedMatrices'
    stackedMat(:,:,i-2) = maskedRgbImage;      % stack the matrices
end

stackedMat(stackedMat == 0) = -100;
stackedMat(stackedMat == 1) = 100;

save('stackedMatrix.mat','stackedMat');        % save the matrices
as: 'stackedMatrix.mat'
end




% % Employed overlapped urethras
% function boundaryurethra
% fname='S7';
% A=dir(['prostate_mask\' fname]);
% n=length(A);
% points=[];
%
% for i=3:n
%      load(['overlappedMatrices\'  A(i).name]);
%     load(['ducts_n_urethra\' fname '\' A(i).name]);
%     [u_shell]=makeshell(meshM);
%     u_p=sorter(u_shell,i);
%     points=[points;u_p];
%    end
%
% assignin('base','points',points);
% X=points(:,1);
% Y=points(:,2);
% Z=points(:,3);
% save(['overlappedMatrices\' A(i).name],'maskedRgbImage');
% end
%
%
% % Obtain Urethra points
%
% function obtainurethrapoints
%
% fname='S7';
% A=dir(['prostate_mask\' fname]);
% n=length(A);
% points=[];
%
% for i=3:n
%     load(['ducts_n_urethra\' fname '\' A(i).name]);
% %     load(['prostate_mask\' fname '\' A(i).name]);
%      load(['PaddedMatrices\' (A(i).name)]);
%     [u_shell]=makeshell(meshM);
%     u_p=sorter(u_shell,i);
%     points=[points;u_p];
% end
%
% assignin('base','points',points);
% X=points(:,1);
% Y=points(:,2);
% Z=points(:,3);
```

```
% vtkwrite('UrethraOverlapOnly.vtk','polydata','lines',X, Y, Z)
% end
%
%
%
%
% % % % Make the shell
% function varargout=makeshell(varargin)
% for i=1:nargin
%     maskedRgbImage=maskedRgbImage(:,:,1)>0;
%     bw_im=maskedRgbImage;
%     bw_im=varargin{i};
%     bw_im=bwmorph(bw_im,'remove');
%     varargout{i}=bw_im;
% end
% end
%
%
% % Sort the points
% function points=sorter(bw_im,i)
%    bw_im=bwmorph(meshM,'remove');
%     [idy,idx]=find(bw_im);
%     refx=idx(1);
%     refy=idy(1);
%     sortedx=refx;
%     sortedy=refy;
%     idx(1)=[];
%     idy(1)=[];
%
%     while~isempty(idx)
%         Pointidx=findEucDist(refx,refy,idx,idy);
%         refx=idx(Pointidx);
%         refy=idy(Pointidx);



%         sortedx=[sortedx refx];
%         sortedy=[sortedy refy];
%         idx(Pointidx)=[];
%         idy(Pointidx)=[];
%     end
%
%     sortedx = 0.0292*sortedx;
%     sortedy = 0.0292*sortedy;
%     sortedz = 2*(i-2)*ones(1,length(sortedx));
%     points = [points; sortedx' sortedy' sortedz'];
% end
%
```

1. Stacking

```
% fname='S7'
A = dir(['prostate_mask\S7\']);
load(['prostate_mask\' (A(3).name)]);        % temp loaded
mkdir('binMatrices')
for i = 3:length(A)
    load(['prostate_mask\' (A(i).name)]);
    maskedRgbImage = (maskedRgbImage(:,:,1)>0);
    save(['binMatrices\' A(i).name],'maskedRgbImage');
end

% Combine the three layers now...
```

```
mkdir('combinedMatrices')
for i = 3:length(A)
    load(['binMatrices\' (A(i).name)]);
    load(['ducts_n_urethra\' (A(i).name)]);
%    maskedRgbImage = bwmorph(maskedRgbImage,'remove');
    FinalMesh = meshL + meshM + meshR;
%    maskedRgbImage = maskedRgbImage + FinalMesh;
        maskedRgbImage = maskedRgbImage -FinalMesh;

    save(['combinedMatrices\' A(i).name],'maskedRgbImage');
end

% overlap urethras

mkdir('overlappedMatrices')

% Take out the reference
load(['ducts_n_urethra\' (A(3).name)]);
ure_center = bwmorph(meshM,'shrink','inf');
[urefy, urefx] = find(ure_center);

for i = 4:length(A)
    load(['combinedMatrices\' (A(i).name)]);
    load(['ducts_n_urethra\' (A(i).name)]);
    ure_center = bwmorph(meshM,'shrink','inf');
    [uy, ux] = find(ure_center);
    FinalMesh = meshL + meshM + meshR;
    maskedRgbImage = maskedRgbImage -FinalMesh;
    maskedRgbImage  =  imtranslate(maskedRgbImage,[urefx-ux,urefy-
uy],'FillValues',255);

    save(['overlappedMatrices\' A(i).name],'maskedRgbImage');
end


% find the largest dim

A = dir('combinedMatrices');
a = 0;
b = 0;
n = length(A)-2;    % number of matrices in the stack

for i = 3:n+2
    load(['combinedMatrices\' (A(i).name)]);
    [p,q] = size(maskedRgbImage);
    if p > a
        a = p;
    end
    if q > b
        b = q;
    end
% now a and b contains the smallest dimensions
end

% Perform Padding
mkdir('PaddedMatrices')

for i = 3:n+2
```

```
    load(['combinedMatrices\' (A(i).name)]);
    [p,q] = size(maskedRgbImage);
    maskedRgbImage   =   padarray(maskedRgbImage,[round((a-p)/2),
round((b-q)/2)]);     % Not happy with 0 0?
    save(['paddedMatrices\' A(i).name],'maskedRgbImage');
% now a and b contains the largest dimensions
end

% Find the smallest dimensions
A = dir('overlappedMatrices');
a = 10000;
b = 10000;
n = length(A)-2;    % number of matrices in the stack

for i = 3:n+2
    load(['overlappedMatrices\' (A(i).name)]);
    [p,q] = size(maskedRgbImage);
    if p < a
        a = p;
    end
    if q < b
        b = q;
    end
% now a and b contains the smallest dimensions
end

% Perform Cropping
mkdir('croppedMatrices')

for i = 3:n+2
    load(['paddedMatrices\' (A(i).name)]);
    maskedRgbImage = imcrop(maskedRgbImage,[0 0 b a]);      % Not
happy with 0 0?
    save(['croppedMatrices\' A(i).name],'maskedRgbImage');
% now a and b contains the largest dimensions
end

% % Perform Stacking
stackedMat = zeros(a,b,n);
for i = 3:n+2
    load(['croppedMatrices\' (A(i).name)]);
    stackedMat(:,:,i-2) = maskedRgbImage;
end

stackedMat(stackedMat == 0) = -100;
stackedMat(stackedMat == 1) = 100;

save('stackedMatrix.mat','stackedMat');


%% write to vtk file
WriteToVTK(stackedMat,'stacked.vtk')
```