

https://dspace.org/

Towards automated three-dimensional tracking of nephrons through stacked histological image sets

Bhikha, Charita

2016-03-17
http://hdl.handle.net/10539/20124

Downloaded from a DSpace repository. https://dspace.org/

APPENDICES

Appendix A: Longitudinal reconstructions

Appendix B: Additional Results

Appendix C: Nephron Tracking Spreadsheet

Appendix D: Performance Data

Appendix E: A Review of the Path Comparison Method

Appendix F: Additional Feature Analysis

Appendix G: Proof of Ethics Clearance

Appendix H: MATLAB Code

Appendix I: Research Article published in the journal *Computational and Mathematical Methods in Medicine*

Appendix A: Longitudinal Reconstructions

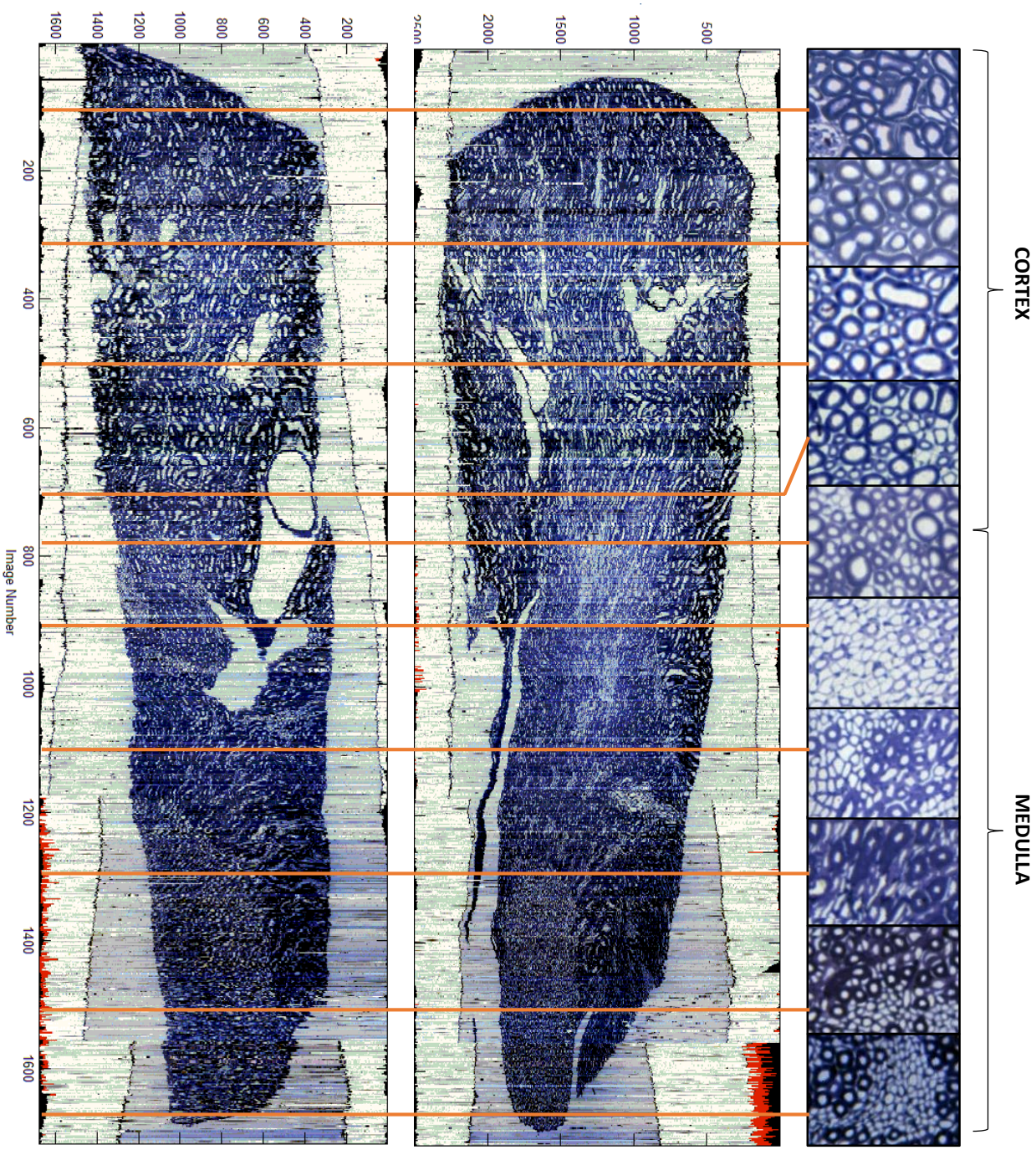


Figure A1: Two longitudinal sections were reconstructed from a mouse image set. The change in morphology can be seen, as well as the zones at which the changes occur. Clips of the transverse images are also shown.

Appendix B: Additional Results

Please refer to the CD to view two videos (of a whole mouse nephron and the PCT-PST of a rat nephron) created from the tracking results.

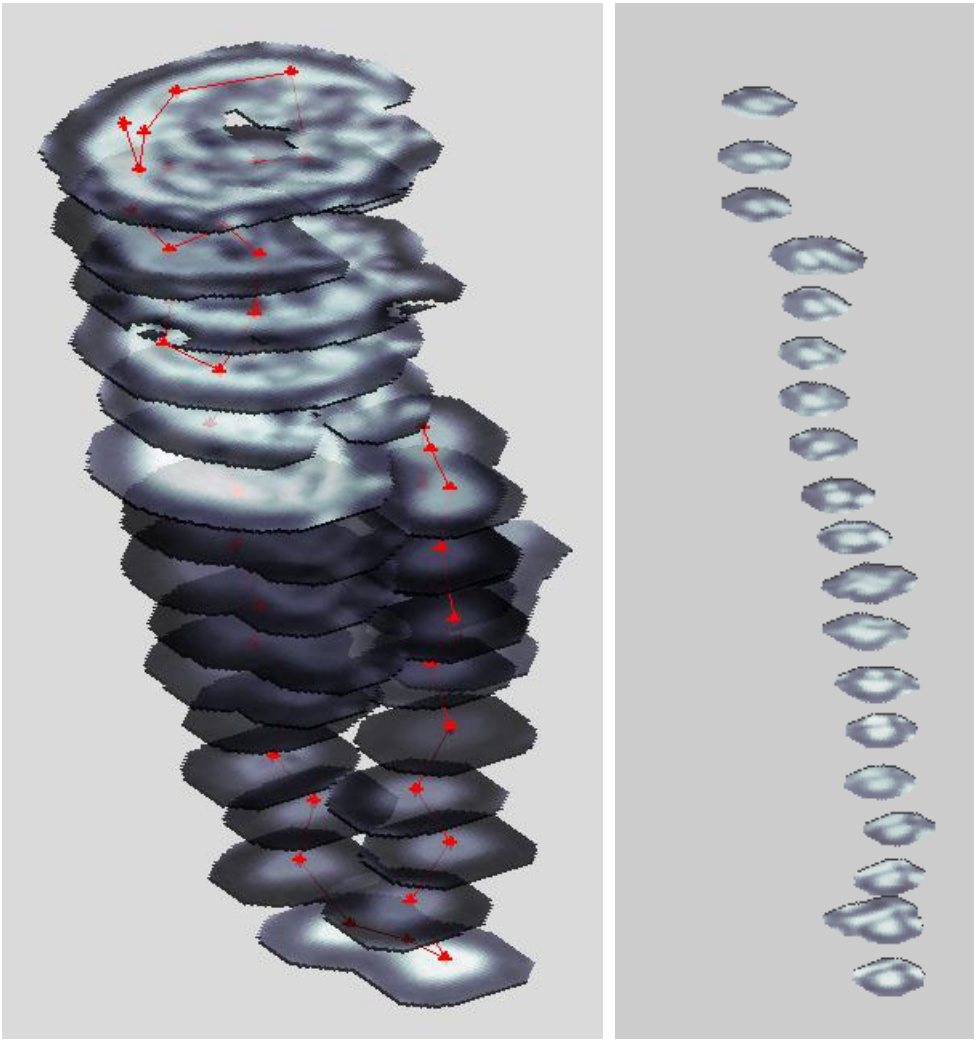


Figure B1: The results of a tracking instance were used to automatically extract the original nephron cross-sections (in the colour image) relating to the tracked nephron. The nodes and their interconnection are shown in red. The slices show an area where the nephron proceeds downwards and then turns upwards. The first elongated cross-section of the bend was automatically chosen during the reconstruction. The nephron then terminates at the glomerulus, as it merges into the urinary pole. The tracking algorithm tracks along a few of the C-shaped cross-sections but eventually terminates the tracking deeper into the glomerulus.

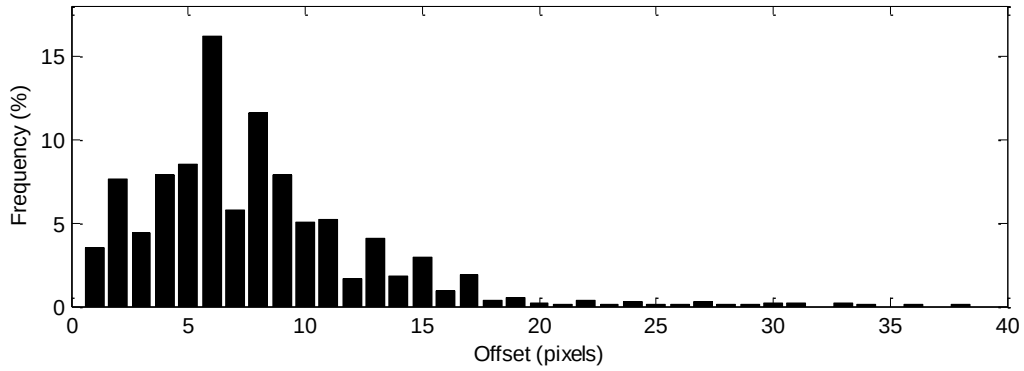


Figure B2: A histogram of the image offsets in the x-y plane between adjacent images during the tracking of 12 mouse nephrons

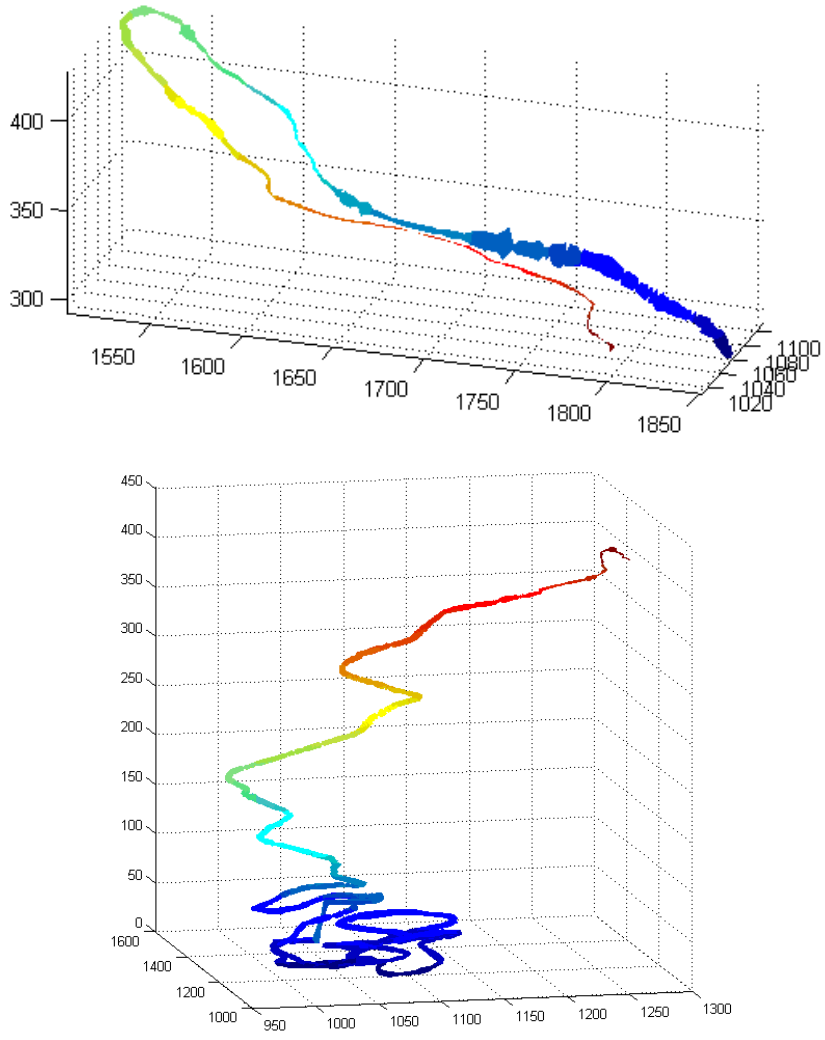


Figure B3: A loop of Henle (top) and the PCT, PST and DTL (bottom) of a mouse nephron is plotted using the extracted minor axis feature. The radius varies wit the size of the cross-sections. The transition between the thick PST and thin DTL can be seen.

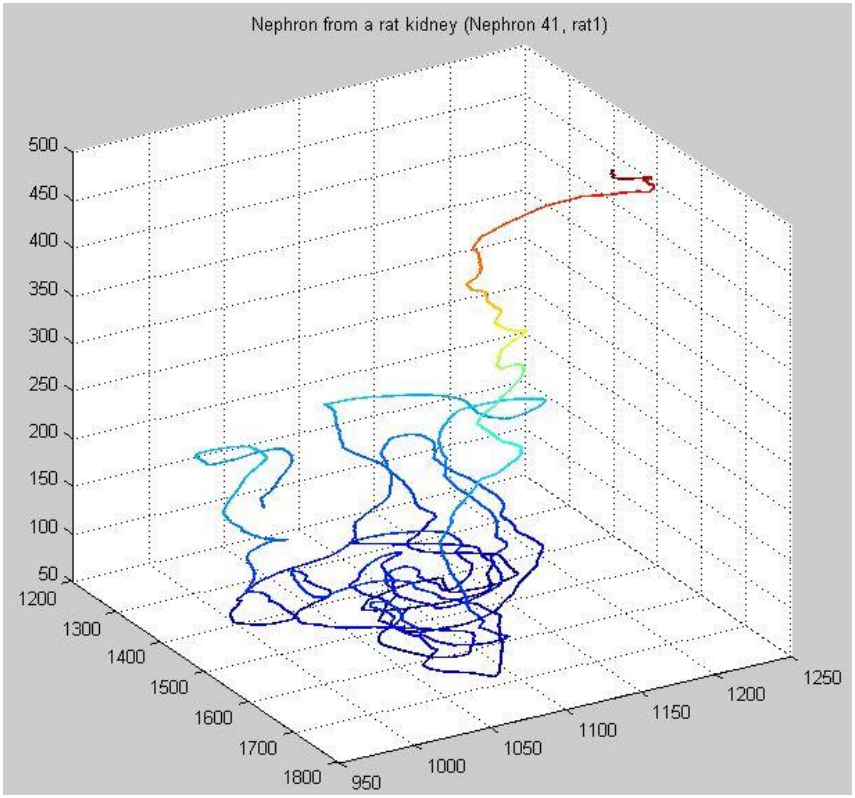


Figure B4: The PCT of a rat nephron is shown to convey the intricacy and complexity of the convolutions present in the PCT.

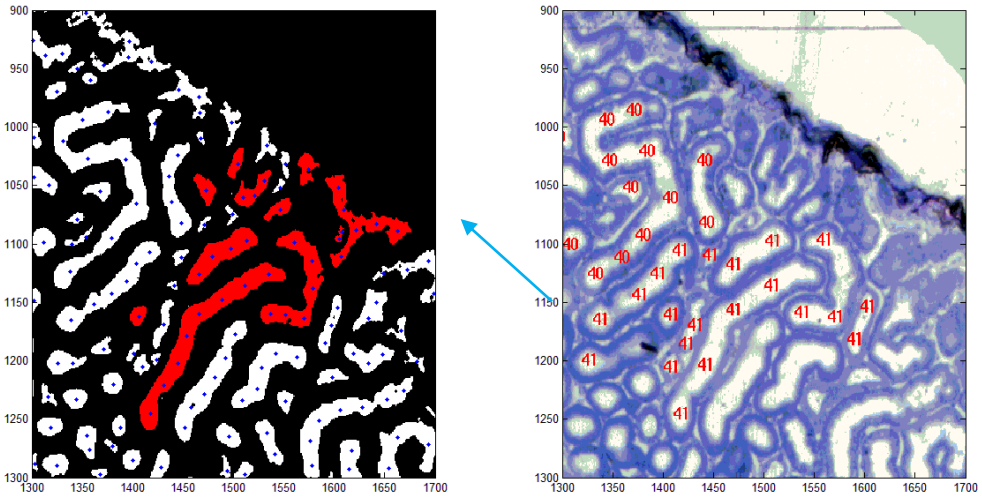


Figure B5: Without machine learning validation, incorrect linkages are often made to interstitial tissue cross-sections, which in turn link to other nephrons. Unlike machine learning validation the other validation rules are not always effective in preventing such cases.

Appendix C: Results of Nephron Tracking

Average Runtime: 5.78 minutes/nephron

MOUSE	Similarity Metrics to Gold Standard		Number of Corrections in each part of the Nephron						TOTALS			
Nephron No.	Alpha (%)	Beta (%)	PCT	PST	DTL +LH	ATL	TAL	DCT	Full Nephron	Up to TAL	Up to ATL	Up to DTL
set1neph3	99,00	77,80	1	2	5	3	2	12	25			
set1neph75	99,13	100,00	0	1	4	0	6	7	18			
set3neph8	98,63	97,03	0	0	6	3	5	2	16			
set3neph38	96,56	100,00	1	0	9	0	1	6	17			
set2neph15	99,51	89,13	3	0	3	4	4	6	20			
set3neph26	99,00	91,14	1	0	5	4	0	5	15			
set3neph4	80,15	97,77	0	0	5	1	2	5	13			
set3neph3	95,81	98,41	0	0	8	11	0	2	21			
set2neph14	93,48	92,01	0	0	3	4	5	3	15			
set1neph70	94,83	84,83	2	1	3	2	4	na		12		
set2neph26	99,10	90,15	2	0	5	0	5	na		12		
set2neph22	92,41	91,26	0	0	7	4	2	na		13		
set1neph38	96,34	82,15	3	1	7	1	4	na		16		
set3neph20	96,15	88,10	1	0	8	6	na	na			15	
set1neph22	93,90	87,81	1	1	5	4	na	na			11	
set1neph133	94,85	100,00	3	0	6	na	na	na				9
set1neph46	98,24	52,14	1	2	5	na	na	na				8
set3neph13	100,00	70,70	0	0	6	na	na	na				6
COUNT			18	18	18	15	13	9	9	4	2	3
Mean	95,95	88,36	1,06	0,44	5,56	3,13	3,08	5,33	**17,78**	13,25	13,00	7,67
									18,60			
Std. Dev.	4,45	11,76	1,08	0,68	1,71	2,75	1,94	2,91	3,49	1,64	2,00	1,25
Average Length			360	230	210	210	180	200	1390			
Average Length (%)			25,90	16,55	15,11	15,11	12,95	14,39				
Weighted Indication			7,59	3,20	39,97	22,54	22,14	38,37				

Appendix C: Results of Nephron Tracking

Average Runtime: 28 minutes/nephron

RAT	Similarity Metrics to Gold Standard		Number of Corrections in each part of the Nephron						TOTALS				
Nephron No.	Alpha (%)	Beta (%)	PCT	PST	DTL +LH	ATL	TAL	DCT	Full Nephron	Up to TAL	Up to ATL	Up to DTL	Other
set5neph10	94,69	98,45	0	2	28	26	6	1	63				
set4neph140	91,75	98,80	6	7	10	17	11	5	56				
set5neph52	94,55	78,88	2	2	33	7	7	na		51			
set5neph31	87,73	94,51	6	6	15	9	5	na		41			
set5neph41	94,51	72,84	0	8	27	8	9	na		52			
set5neph146LL	92,81	40,40	8	7	na	na	na	na			15		
set5neph145	85,51	57,70	2	7	na	na	na	na			9		
set4neph52	93,08	57,81	8	8	na	na	na	na			16		
set5neph47	93,33	36,36	2	na	na	na	na	na				2	
set5neph11	91,92	27,96	16	na	na	na	na	na				16	
set4neph11	94,35	47,04	8	na	na	na	na	na				8	
set5neph147	80,15	48,15	na	na	na	7	1	na					8
set5neph40	99,80	18,28	na	1	16	na	na	na					17
set5neph42	99,59	62,07	na	3	30	16	6	7					62
COUNT			11	10	7	7	7	3	2	3	3	3	
Mean	92,41	59,95	5,27	5,10	22,71	12,86	6,43	4,33	**59,50**	48,00	13,33	8,67	
									56,71				
Projected rat from mouse (x4.7)			4,96	2,09	26,11	14,73	14,46	25,07	83,56				
Std. Dev.	4,97	24,97	4,53	2,62	8,21	6,62	2,92	2,49	3,50	4,97	3,09	5,73	
Average Length			1800	900	780	780	680	980	5920				
Average Length (%)			30,41	15,20	13,18	13,18	11,49	16,55					
Weighted Indication			8,91	8,61	38,37	21,72	10,86	7,32					

Appendix D: Performance Data

MATLAB Profiler results

The MATLAB Profiler measures the execution time of functions, sub-functions and subroutines to aid optimisation of code. The profiler was used to measure an instance of tracking (TrackerFinal.m). The results show that neural network validation, reading images into MATLAB and the fft/ifft used during alignment are the longest subroutines, while the overheads in re-ordering the image matrix and transferring large variables to functions takes up 75% of the self-time of TrackerFinal.m

Profile Summary

Generated 29-Mar-2015 19:59:42 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self tim
TrackerFinal	1	213.834 s	86.303 s	
validationSteps	218	52.349 s	0.200 s	
network.subsref	872	50.680 s	0.250 s	
network.sim	218	50.350 s	8.080 s	
trackStraight	649	37.730 s	0.300 s	
findOffset	431	37.100 s	3.480 s	
setup2	218	27.430 s	0.040 s	
codeHints	218	26.680 s	26.680 s	
imread	173	22.170 s	0.350 s	
imagesci\private\readjpg	173	20.820 s	0.020 s	
imagesci\private\rjpg8c (MEX-file)	173	20.310 s	20.310 s	
MatFile>MatFile.subsref	868	13.402 s	5.860 s	
setup1	218	12.430 s	0.390 s	
fft2	862	12.121 s	12.121 s	
imtransform	431	11.430 s	0.120 s	
poolSize	218	8.750 s	0.070 s	
tformarray	431	8.400 s	0.100 s	
ifft2	431	7.699 s	7.699 s	

distCompAvailable	218	6.870 s	0.030 s	
ver	218	6.840 s	0.010 s	
ver>locGetSingleToolboxInfo	218	6.830 s	1.270 s	
tformarray>resample	431	4.969 s	0.040 s	
images\private\resampsep (MEX-file)	431	4.929 s	4.929 s	
...ile.getVariableInfoIfItExistsInSource	868	4.560 s	0.020 s	
MatFile>MatFile.whos	869	4.540 s	0.100 s	
MatFile>MatFile.genericWho	869	4.440 s	3.440 s	
ver>locGetContentsListFromPath	218	3.490 s	0.830 s	
cell.strcat	436	2.810 s	2.740 s	
maketform	2155	2.070 s	0.270 s	
tforminv	431	1.920 s	0.010 s	
images\private\tform	431	1.910 s	0.140 s	
matlabpool	218	1.810 s	0.280 s	
imtransform>make_composite_tform	431	1.730 s	0.100 s	
MatFile>MatFile.loadEntireVariable	434	1.641 s	1.611 s	
maketform>inv_composite	431	1.550 s	0.120 s	
netHints	218	1.430 s	1.180 s	
...parseInputsAndCheckOutputsForFunction	218	1.400 s	0.060 s	
netHints	218	1.390 s	0.950 s	

***Self time** is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

System Specifications

Development and testing was performed using a system with the following specifications. The developed system should use a system with similar or better performance specifications.

- Processor: Intel Core i5 @ 3.10 GHz
- RAM: 16.00GB
- 512 MB Graphics memory
- MATLAB Version R2011b
- Toolboxes used: Image Processing Toolbox, Statistics and Machine Learning Toolbox, Neural Network Toolbox and the Parallel Computing Toolbox.

Appendix E: A Review of the Path Comparison Method

The MATLAB function used to compare an automatically tracked nephron path to a manually tracked one is displayed below.

```
function [metric, residual] = comparePaths2(x,y, tol)

if nargin==2, tol = 10; end

residual=100.*ones(1,size(y,1));    % Pre-allocate a
vector

%Compare each element in y to coordinates in relevant
image in x
for i=1:1:size(y,1)

    % Get coordinates in images i, i+1 and i-1 in x
    t1 = or(or(x(:,3)==y(i,3)-
1,x(:,3)==y(i,3)+1),x(:,3)==y(i,3));
    xi = x(t1,1:3);

    % Calculate Euclidean distances to those coordinates
from y(i)
    dis = dist(xi,y(i,1:3));

    % Residual is the minimum distance (sum of square
difference)
    if ~isempty(dis)
        residual(i) = min(dis);
    end
end

% The comparative metric is a threshold of the residual
 metric = 100.*sum(residual<tol(y(:,3)))./size(y, 1);
```

The residual is calculated as the sum of square distance to the corresponding points in the manual path. As can be seen in Figure E1, most points are within 15 pixels of the manually tracked path. Whether or not the points actually belong to the nephron of interest or an adjacent one is assessed by comparing the residual to the average cross-sectional radius of a nephron in the respective area. In the cortex, a point is deemed correct if the residual is less than 20 pixels while in the inner medulla, a stricter criterion of 10 pixels is imposed due to the narrow diameter of the thin limbs. The metric is then simply the percentage of points which were deemed correct. High residual values may be indicative of:

- Any correct points which were not included in the manually tracked path, e.g. the glomerulus.
- An image artefact.

- Deviation of the automatically tracked path onto an incorrect path.

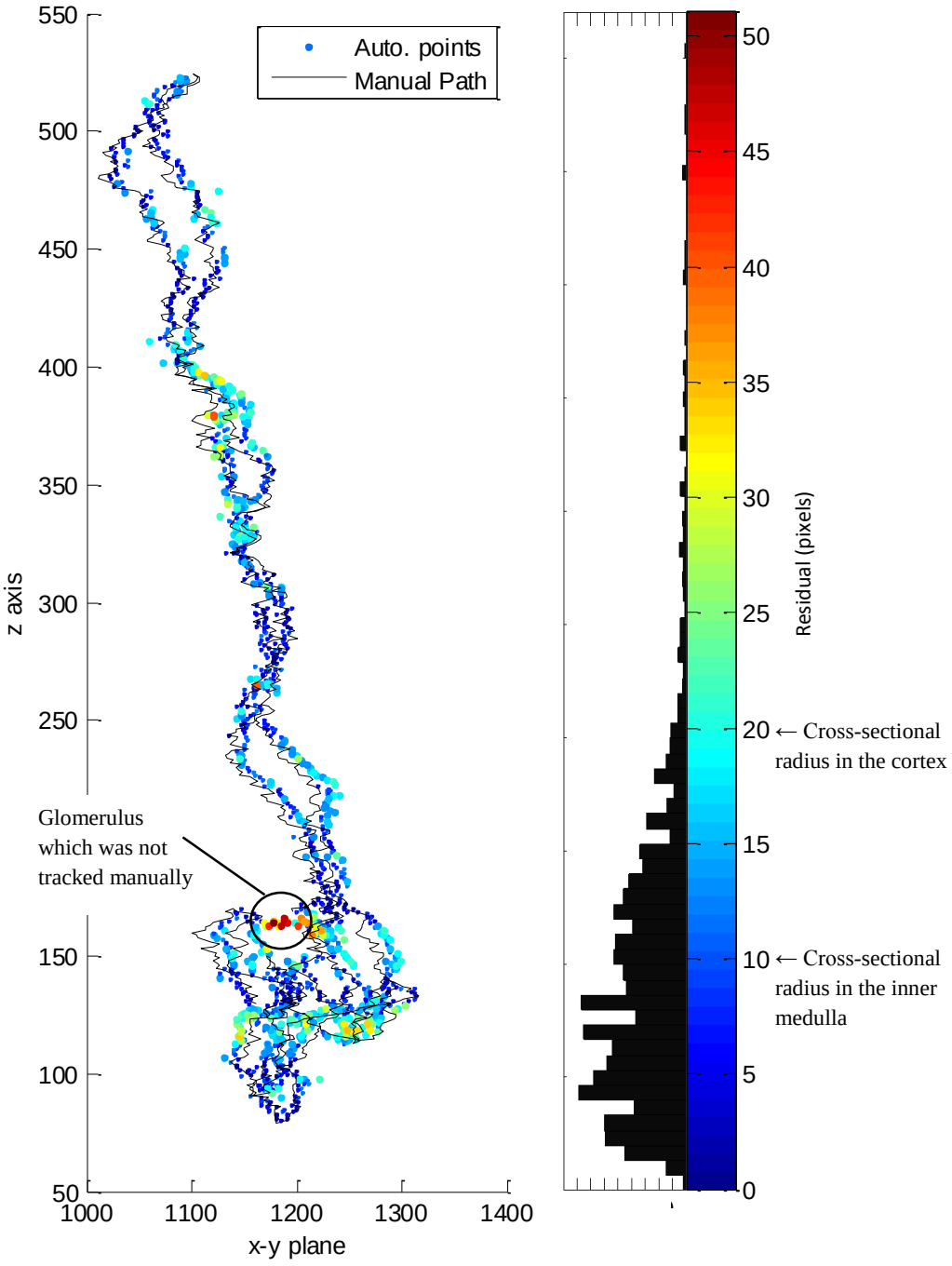


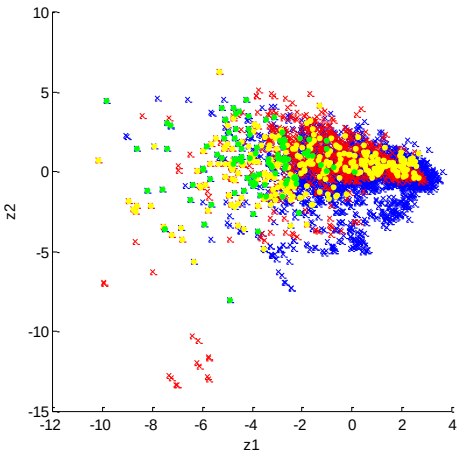
Figure E1: Points from an automatically tracked mouse nephron are compared to the manually tracked path (in black). The similarity measured by the algorithm produces α=97.45% and β=99.84% using a residual threshold of 25 pixels. The points are colour- and size- coded to its corresponding residual value.

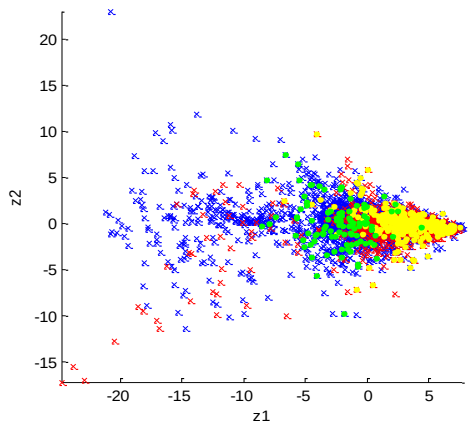
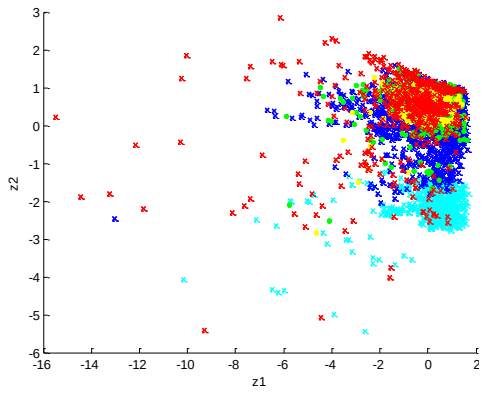
Appendix F: Additional Feature Analysis

Principal Component Analysis (PCA) was performed on data containing various combinations of features in order to visually determine the effect that the features have on classification ability and hence to see which are the most useful features. The generalised method used to perform PCA on data X is as follows:

Algorithm: PCA	
A = (1/m).*(X'*X)	Calculate covariance matrix from data (X)
[U, D, V] = svd(A)	Perform single values decomposition to get
	U = columns of eigenvectors of $A.A^T$
	V = rows of eigenvectors of $A^T.A$
	D = diagonal matrix of eigenvalues
Z = X*U(1:K)	Project data to lower dimension K using first K
eigenvectors	
X_rec = Z*U(1:N)'	Recover original data in higher dimension N
(optional)	

Table F1: The reduced feature plots of different classes of examples with a brief discussion on each.

PCA Feature Plot	Description
	**All features except shape profile features:** When the shape profile features (the actual shape profiles and the similarity metric) are excluded, examples of elongated cross-sections (green) and glomeruli (yellow) cannot be clearly differentiated, validating the need for the shape profiles as part of the feature set. However, a separating boundary can be seen between the normal moves (blue) and those of connective tissue (red), highlighting the role that the shape factors play in the classification.

	**All features except shape factor features:** With the shape profile as the only shape features, a boundary can now be seen between elongated and glomeruli types. However, there is a large overlap between the normal and connective tissue moves, again showing the importance of shape factors.
	**No shape factors or shape profile features:** With no shape features, very little can be said about any of the classes apart from the moves in the inner medulla (cyan), which is most likely classified solely on the z-position feature. This implies that the remaining features (e.g. xy-distance, image difference) can be used to increase confidence of a normal move (blue) but nothing can be said about the other types.

Appendix G: Proof of Ethics Clearance

Clearance from the Animal Ethics Committee was not required due to the research being purely computational, as indicated in the email below. Ethics was obtained by the Danish team when the kidneys were originally processed.

Fwd: Concerning the Animal Ethics Screening Committee (AESC) at the university

Robyn Letts <robyn.letts@gmail.com> Wed, Jan 7, 2015 at 10:31 AM
To: Charita Bhikha <charita.bhikha@gmail.com>

---------- Forwarded message ----------
From: **Arne Andreasen** <aa@biomed.au.dk>
Date: 7 December 2014 at 17:57
Subject: Concerning the Animal Ethics Screening Committee (AESC) at the university
To: Robyn Letts <robyn.letts@gmail.com>

Dear Robyn

Concerning the Animal Ethics Screening Committee (AESC) at the university:

I have managed to find the ethics clearance number that was in use at our Institute of Anatomy when the mouse kidneys were prepared. Since then the Institute of Anatomy has been abolished, some of the functions are now a part of the bigger Institute of Biomedicine.

The license number belonged to Professor Henrik Birn, the number under *The Danish Ministry of Food, Agriculture and Fisheries,* was 2004/561-818.

The Danish name of this ministry is *Ministeriet for Fødevarer*.

I really hope this information will help you.

Kind regards

Arne

PS Please confirm that you got this e-mail

RE: Enquiry on Ethics Clearance

Kennedy Erlwanger <Kennedy.Erlwanger@wits.ac.za>

Fri, Aug 8, 2014 at 12:49 PM

To: Charita Bhikha <charita.bhikha@gmail.com>
Cc: Robyn Letts <Robyn.Letts@wits.ac.za>, David Rubin <David.Rubin@wits.ac.za>, Sidney Engelbrecht <Sidney.Engelbrecht@wits.ac.za>

Dear Charita,

I can confirm that you do not require clearance from the AESC of the University of the Witwatersrand as your study is purely computational and does not involve the direct use of animals or animal tissue.

Kindly note that for any ethical issues around the original animal based study you will have to rely on what the researchers are able to avail to you. Although the AESC of the University of the Witwatersrand would not be able to give retrospective clearance for the study, the nature of the study (as you have described) does not require clearance from the AESC.

Sincerely,

Kennedy

(Chairman, AESC- University of the Witwatersrand)

Assoc Prof K.H. Erlwanger

School of Physiology

Faculty of Health Sciences,

University of the Witwatersrand

7 York Road, Parktown, 2193

SOUTH AFRICA

Private bag 3, Wits, 2050, South Africa.

Tel: +27 (0)11 717 2454

Fax: + 27 (0)11 643 2765

Email: Kennedy.Erlwanger@wits.ac.za

From: Charita Bhikha [mailto:charita.bhikha@gmail.com]
Sent: 08 August 2014 09:40 AM
To: Kennedy Erlwanger
Cc: Robyn Letts; David Rubin
Subject: Enquiry on Ethics Clearance

Dear Prof. Erlwanger

I am a masters student in the School of Electrical and Information Engineering. I would like your advice pertaining to the need for ethics clearance for my research project.

My research involves image processing and analysis on histological image sets of mouse and rat kidneys. These images have been acquired from a group at the University of Aarhus,Denmark. They had originally processed the kidney specimens into digital images.

Since the work is purely computational, and no animals are involved, will I need ethics approval from the AESC? The group in Denmark who had originally done work on these images, have mentioned in their publications that ethics had been obtained. Their paper quotes:

"All animal experiments were carried out in accordance with the provisions for the animal care license provided by the Danish National Animal Experiments Inspectorate."

http://ajprenal.physiology.org/content/306/6/F664

Sidney Engelbrecht had advised that ethics clearance is not needed, so long as the original ethics clearance certificate or number is provided. However, the group in Denmark seems to be having difficulty in locating their clearance certificate/number, as it was done a long while ago. Kindly advise as to what is required from Wits' perspective, and if you require more detailed information.

Kind Regards

Charita Bhikha

Appendix H: MATLAB Code

The systems user interface is by means of two MATLAB scripts – *PreprocessAndFeatureExtractInterface* and *TrackerInterface*. These make use of a number of custom functions, each tasked with a specific function. The functions were written such that they required minimal inputs to provide specific output information, i.e. sub-processes of the tracking were decoupled. They can be used outside the script to analyse the intermediate stages of data. Function encapsulation and abstraction makes the code efficient, and easy to maintain, upgrade and debug. Figure H1 shows an overview of functional dependencies.

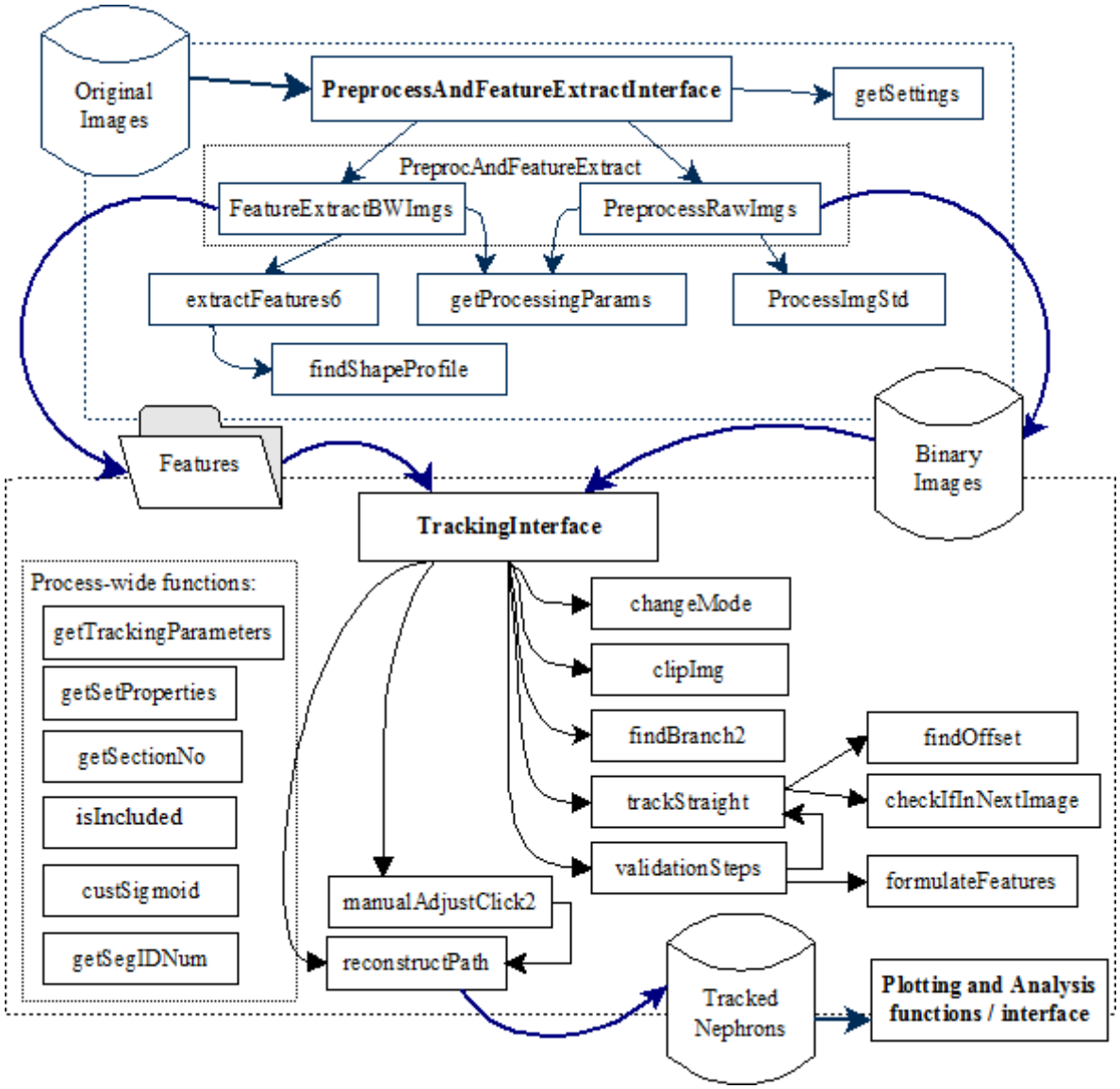


Figure H1: A code dependency graph of the system. The two main scripts of the system make use of various custom-coded functions which are independent of one another. Each function passes information to the function/script above it.

MATLAB Code:

Pre-processing and Feature Extraction

PreprocessAndFeatureExtractInterface.m

```
% PreprocessAndFeatureExtractInterface.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 15-Mar-2015

% Function hierarchy of files required:
%                             getSettings.m
%                                 getSetProperties.m
%                             PreprocessRawImgs.m
%                                 getProcessingParams.m
%                                 ProcessImgStd.m
%                             FeatureExtractBWImgs.m
%                                 getProcessingParams.m
%                                 extractFeatures6.m
%                                     findShapeProfileStretch.m
%                                     litekmeansMod.m
%                             PreprocAndFeatureExtract.m
%                                 getProcessingParams.m
%                                 ProcessImgStd.m
%                                 extractFeatures6.m
%                                     findShapeProfileStretch.m
%                                     litekmeansMod.m

% ---------------------------- START OF CODE -----------------------------

%% Option 1 - STAGE 1: PRE-PROCESSING

% Instructions:
% 1. Set imInPath in getSetProperties.m to the path of the folder
%    containing the raw colour images
% 2. Set imOutPath in getSetProperties.m to the directory where the outputs
%    will be saved. Use the convention setNdataV where N is the image set
%    number and V is the version.
% 3. Set imageSetNo to the image set being used.
% 4. Set version to the output version to be created.
% 5. Open 'getSettings' (right click) and modify the parameters
%    related to the image set no. being used. Save and close.
% 6. Set outputLog to true if a live output is desired, otherwise set it to
%    false.
% 7. Run this section of code. Press CRTL+C to cancel.

% NOTE: A binary image is written to hard disk at each iteration, and so
% one may cancel (in order to pause) and continue by changing the start
% index in settings.

clc, clear

imageSetNo  = 0;
version     = 1;
settings    = getSettings(imageSetNo,version)
outputLog   = true;

PreprocessRawImgs(settings,outputLog);

% ------------------------------------------------------------------------
%% Option 1 - STAGE 2: FEATURE EXTRACTION
```

```
% Instructions:
% 1. Set imOutPath in getSetProperties.m to the path of the folder
%    containing the BINARY images
% 2. Set featFile to the directory where the features will be saved.
% 3. Set SPFile to the directory where the shape profiles will be saved.
%    The shape profiles must be saved separately as they require a large
%    amount of memory. Note that these folders must already exist.
% 4. Set imageSetNo to the image set being used.
% 5. Set version to the output version to be used.
% 6. Open 'getSettings' (right click) and modify the parameters
%    related to the image set no. being used. Save and close.
% 7. Set outputLog to true if a live output is desired, otherwise set it to
%    false.
% 8. Run this section of code. Press CRTL+C to cancel.

% NOTE: The nodes and shape factors are stored in the workspace (RAM)
% during execution of this block of code. Cancellation will result in loss
% of the information already processed. The shape factors may or may not be
% written to hard disk on each iteration, see comments in
% 'getSettings' for more information.

clc, clear

featFile    = 'Test\test0feat1.mat';
SPFile      = 'Test\test0SP1.mat';
imageSetNo  = 0;
version     = 1;
settings    = getSettings(imageSetNo,version)
outputLog   = true;

FeatureExtractBWImgs(featFile, SPFile, settings, outputLog);

% -------------------------------------------------------------------------
%% Option 2: PRE-PROCESSING & FEATURE EXTRACTION

% Instructions:
% 1. Set imInPath in getSetProperties.m to the path of the folder
%    containing the raw colour images
% 2. Set imOutPath in getSetProperties.m to the directory where the output
%    binary images will be
%    saved.
% 3. Set featFile to the directory and name of where the features will be
%    saved (a .mat file).
% 4. Set SPFile to the directory where the shape profiles will be saved.
%    The shape profiles must be saved separately as they require a large
%    amount of memory.
%  Note that these folders must already exist; they will not be created.
%
% 5. Set imageSetNo to the image set being used.
% 6. Set version to the output version to be used.
% 7. Open 'getSettings' (right click) and modify the parameters
%    related to the image set no. being used. Save and close.
% 8. Set outputLog to true if a live output is desired, otherwise set it to
%    false.
% 9. Run this section of code. Press CRTL+C to cancel.

% NOTE: The nodes and shape factors are stored in the workspace (RAM)
% during execution of this block of code. Cancellation will result in loss
% of the information already processed. The shape factors may or may not be
% written to hard disk on each iteration depending on the chosen settings
% (see 'getSettings' for more information).

featFile    = 'Test\test0feat1.mat';
SPFile      = 'Test\test0SP1.mat';
```

```
imageSetNo  = 0;
version     = 1;
settings    = getSettings(imageSetNo,version)
outputLog   = true;

PreprocAndFeatureExtract(featFile,SPFile,settings,outputLog);

% ----------------------------- END OF CODE -------------------------------
```

getSettings.m

```
function settings = getSettings(imageSetNo,OutVersion)

% getSettings - This function is meant to be modified by the user to
% initialise settings or parameters for pre-processing and feature extraction.
%
% Syntax:  settings = getSettings(imageSetNo)
%
% Inputs:
%    imageSetNo     - The image set identifier number
%    OutVersion     - The preprocessing and feature extraction version
%                      number
% Outputs:
%    settings       - A struct of the parameters are returned with field
%                     names and values.
%
% Other m-files required: getSetProperties.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 12-Mar-2015

%------------- START OF CODE --------------

% MODIFY THE PARAMETERS THAT ARE RELEVANT:

  % >>> Open getSetProperties and set parameters for the image set
  settings = getSetProperties(imageSetNo,OutVersion);

% The start and end index of the images to be processed (referring to the
% binary images that are to be produced).
  settings.startImg = 1;
  settings.endImg = 2;



% SETTINGS FOR SHAPE PROFILE
  settings.angStep = 15;     % Angle increment
  settings.scale = 50;       % Target scale in pixels
  settings.saveMethod = 0;   %'RAM'=0; 'HARDDISK'=1;

%------------- END OF CODE --------------
```

getSetProperties.m

```
function setProperties = getSetProperties(set,OutVersion)

% getSetProperties - Returns a struct containing a number of properties
% related to a specific image set (set) and the version of the
% preprocessing and feature extraction output (OutVersion). These
```

```
% properties are widely used to automatically refer to the images and
% features. The properties for the 6 image sets and a test set are included.

% The properties are:
% id                A unique identifier number for the image set
% offset            The offset between the numeric part of the colour
%                   set and the binary set or The index of the first colour
%                   image (sometimes not '1' due to starting images being
%                   blank)
% latestVersion     The latest version of data that exists for the set
% imOutPath         The directory to which the output binary images will be
%                   saved OR the path to the existing binary image set
% imInPath          Path to the colour image set
% imLabPath         Path to the labelled colour image set
% range             The number of integers in the numerical part of the
%                   colour image's name
% imsize            Image dimensions in pixels [width height]
% setsize           Number of images in the set
% originalSetName   A string describing the origin of the image set

% Syntax:  setProperties = getSetProperties(set,OutVersion)
%          setProperties = getSetProperties(set)
%          setProperties = getSetProperties()
%
% Inputs:
%    set         - The image number of the set (1-6) as a numerical or string
%                  default = 'test'
%    OutVersion  - The preprocessing and feature extraction version number
%                  default = 1
%
% Outputs:
%    setProperties  - A struct of the properties for the image set

% Other m-files required:   none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 12-Mar-2015

%------------- START OF CODE --------------

setProperties = struct;

if nargin==0
    set = 'test';
    OutVersion = '0';
end

if nargin==1
    OutVersion = '1';
end

if isnumeric(OutVersion)
    OutVersion = num2str(OutVersion);
end

switch set
  case {0, 'test', 'Test'}
    setProperties.id = 0;
    setProperties.offset = 0;
    setProperties.latestVersion = 1;
    setProperties.imOutPath = 'Test\out\B_img';
    setProperties.imInPath = 'Test\in\C_img';
    setProperties.imLabPath = 'Test\labelled\L_img';
```

```
    setProperties.range = 2;
    setProperties.imsize1 = [1675 2500];
    setProperties.imsize2 = [2500 2750];
    setProperties.setsize = 100;
    setProperties.originalSetName = 'test';

  case {1, '1','set1','Set1'}
    setProperties.id = 1;
    setProperties.offset = 39;
    setProperties.latestVersion = 5;
    setProperties.imOutPath = ['dataOut\set1data' (OutVersion) '\img'];
    setProperties.imInPath =  'dataIn\imageSet1\morphed-image--';
    setProperties.imLabPath = 'dataIn\imageSet1n\image--';
    setProperties.range = 4;
    setProperties.imsize = [1675 2500];
    setProperties.setsize = 900;
    setProperties.originalSetName = 'mouse 1';

  case {2, '2','set2','Set2'}
    setProperties.id = 2;
    setProperties.offset = 0;
    setProperties.latestVersion = 2;
    setProperties.imOutPath = ['dataOut\set2data' (OutVersion) '\img'];
    setProperties.imInPath =  'dataIn\imageSet2\morphed-image--';
    setProperties.imLabPath = 'dataIn\imageSet2n\image--';
    setProperties.range = 4;
    setProperties.imsize = [1675 2500];
    setProperties.setsize = 990;
    setProperties.originalSetName = 'mouse 3';

  case {3, '3','set3','Set3'}
    setProperties.id = 3;
    setProperties.offset = 0;
    setProperties.latestVersion = 1;
    setProperties.imOutPath = ['dataOut\set3data' (OutVersion) '\img'];
    setProperties.imInPath =  'dataIn\imageSet3\morphed-image--';
    setProperties.imLabPath = 'dataIn\imageSet3n\image--';
    setProperties.range = 4;
    setProperties.imsize = [1675 2500];
    setProperties.setsize = 1000;
    setProperties.originalSetName = 'mouse 4';

  case {4, '4','set4','Set4'}
    setProperties.id = 4;
    setProperties.offset = 0;
    setProperties.latestVersion = 1;
    setProperties.imOutPath = ['dataOut\set4data' (OutVersion) '\img'];
    setProperties.imInPath =  'dataIn\imageSet4\aU';
    setProperties.imLabPath = 'dataIn\imageSet4n\B-';
    setProperties.range = 4;
    setProperties.imsize = [2500 2750];
    setProperties.setsize = 4000;
    setProperties.originalSetName = 'rat 5';

  case {5, '5','set5','Set5'}
    setProperties.id = 5;
    setProperties.offset = 29;
    setProperties.latestVersion = 6;
    setProperties.imOutPath = ['dataOut\set5data' (OutVersion) '\img'];
    setProperties.imInPath = 'dataIn\imageSet5\aU';
    setProperties.imLabPath = 'dataIn\imageSet5n\C-';
    setProperties.range = 4;
    setProperties.imsize = [2500 2750];
    setProperties.setsize = 3500;
    setProperties.originalSetName = 'rat 8';
```

```
  case {6, '6','set6','Set6'}
    setProperties.id = 6;
    setProperties.offset = 0;
    setProperties.latestVersion = 1;
    setProperties.imOutPath = ['dataOut\set6data' (OutVersion) '\img'];
    setProperties.imInPath = 'dataIn\imageSet6\aU';
    setProperties.imLabPath = 'dataIn\imageSet6n\A-';
    setProperties.range = 4;
    setProperties.imsize = [2500 2750];
    setProperties.setsize = 4000;
    setProperties.originalSetName = 'rat 4';

   otherwise
    disp('Invalid set.')

end


%------------- END OF CODE --------------
```

getProcessingParams.m

```
function Parameters = getProcessingParams(dataset, imgNo)

% getProcessingParams - This function returns the processing parameters for
% a specific image in a specific image set. This function is meant to be
% adjusted by the user during once-off calibration/setup of the functions
% for the image sets. Eight processing parameters, each of which varies
% according to a custom sigmoid function (see custSigmoid.m), is calculated
% and returned.
%
% Syntax:  settings = getSettings(imageSetNo)
%
% Inputs:
%    dataset        - The image set identifier number
%    imgNo          - The number of the image in the image set, e.g. 4
%                     or an identifier string, e.g. 'Set4'
% Outputs:
%    Parameters     - An array of the parameters are returned in order.
%                     These are:
%                     1. Background threshold value
%                     2. Equalisation window size
%                     3. Binarisation threshold value
%                     4. Maximum Noise pixel size
%                     5. Maximum Allowed pixel size
%                     6. Number of erase cycles
%                     7. Connective tissue area in pixels (mean)
%                     8. Desired adjacent node distance
%
% Example:
%    P = getProcessingParams(3, 350)
% Gets the parameters for image number 350 in image set 3 in an array P.
%
% Other m-files required:   custSigmoid.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 12-Mar-2015

%------------- START OF CODE --------------

switch dataset
    case {'Set1', 1}     % mouse-1
```

```
        eq_win =             custSigmoid(imgNo,-1, 18, 60, 300, 3);
        noise_pixel_size =   custSigmoid(imgNo, 1, 7, 1, 200, 1);
        allowed_pixel_size = custSigmoid(imgNo, 1, 7000, 5000, 300, 6);
        erase =              custSigmoid(imgNo, 1, 1, 0, 200, 6);
        th =                 custSigmoid(imgNo, 1, 180, 200, 300, 2);
        ctarea =            custSigmoid(imgNo, 1, 20, 7, 200, 1);
        bthr = 10; %low due to normal centre and darker background with lots of
bits
        dist =            custSigmoid(imgNo, 1, 20, 10, 350, 2);

    case {'Set2', 2}    % mouse-3

        eq_win =             custSigmoid(imgNo,-1, 20, 40, 300, 3);
        noise_pixel_size =   custSigmoid(imgNo, 1, 7, 1, 200, 1);
        allowed_pixel_size = custSigmoid(imgNo, 1, 7000, 5000, 350, 6);
        erase =              custSigmoid(imgNo, 1, 1, 0, 200, 6);
        th =                 custSigmoid(imgNo, 1, 180, 200, 350, 2);
        ctarea =            custSigmoid(imgNo, 1, 20, 7, 200, 1);
        bthr = 30; %low due to normal centre and darker background with lots of
bits
        dist =            custSigmoid(imgNo, 1, 20, 10, 350, 2);

    case {'Set3', 3}     % mouse-4

        eq_win =             custSigmoid(imgNo,-1, 20, 40, 350, 3);
        noise_pixel_size =   custSigmoid(imgNo, 1, 7, 1, 300, 1);
        allowed_pixel_size = custSigmoid(imgNo, 1, 7000, 5000, 350, 6);
        erase =              custSigmoid(imgNo, 1, 0, 0, 300, 6);
        th =                 custSigmoid(imgNo, 1, 180, 200, 350, 2);
        ctarea =            custSigmoid(imgNo, 1, 20, 7, 300, 1);
        bthr = -10; %low due to normal centre and darker background with lots of
bits
        dist =            custSigmoid(imgNo, 1, 20, 10, 350, 2);

    case {'Set4', 4}     % rat

        eq_win =             custSigmoid(imgNo,-1, 20, 60, 1300, -10);
        noise_pixel_size =   custSigmoid(imgNo, 1, 10, 0, 1200, 1);
        allowed_pixel_size = custSigmoid(imgNo, 1, 4000, 3000, 1300, 3);
        erase =              custSigmoid(imgNo, 1, 0, 0, 1300, 3);
        th =                custSigmoid(imgNo, 1, 180, 200, 1200, 1);
        ctarea =            custSigmoid(imgNo, 1, 20, 5, 1000, 1);
        bthr = 45; %high due to bright centre with little background variation
        dist =            custSigmoid(imgNo, 1, 20, 10, 1200, 2);

    case {'Set5', 5}     % rat

        eq_win =             custSigmoid(imgNo,-1, 20, 60, 1300, -10);
        noise_pixel_size =   custSigmoid(imgNo, 1, 10, 0, 1200, 1);
        allowed_pixel_size = custSigmoid(imgNo, 1, 4000, 3000, 1300, 3);
        erase =              custSigmoid(imgNo, 1, 0, 0, 1300, 3);
        th =                custSigmoid(imgNo, 1, 180, 200, 1200, 1);
        ctarea =            custSigmoid(imgNo, 1, 20, 5, 1000, 1);
        bthr = 45; %high due to bright centre with little background variation
        dist =            custSigmoid(imgNo, 1, 20, 10, 1200, 2);

    case {'Set6', 6}

        eq_win =             custSigmoid(imgNo,-1, 14, 16, 300, 3);
        noise_pixel_size =   custSigmoid(imgNo, 1, 7, 1, 200, 1);
        allowed_pixel_size = custSigmoid(imgNo, 1, 7000, 5000, 300, 6);
        erase =              custSigmoid(imgNo, 1, 1, 0, 200, 6);
        th =                 custSigmoid(imgNo, 1, 180, 200, 300, 2);
```

```
        ctarea =            custSigmoid(imgNo, 1, 20, 7, 200, 1);
        bthr = 10; %low due to normal centre and darker background with lots of
bits
        dist =            custSigmoid(imgNo, 1, 20, 10, 350, 2);

    case {0, 'test', 'Test'}

        eq_win              = 15;
        noise_pixel_size    = 2;
        allowed_pixel_size  = 6000;
        erase               = 0;
        th                  = 185;
        ctarea              = 10;
        bthr                = 10;
        dist                = 15;

end

    Parameters=[round(bthr),round(eq_win),th,round(noise_pixel_size),...
        round(allowed_pixel_size),round(erase),ctarea,dist];

%------------- END OF CODE --------------
```

PreprocessRawImgs.m

```
function dummy = PreprocessRawImgs(settings,outputLog)

% PreprocessRawImgs - This function performs preprocessing on the raw
% colour kidney images using the settings struct provided. The output
% binary images are saved to disk automatically to the path specified.
%
% Syntax:  [~] = PreprocessRawImgs(imInPath,imOutPath,settings,outputLog)
%
% Inputs:
%    settings       - A struct of the desired settings for pre-processing
%                     as created using the getSettings function
%    outputLog      - Enable (1) or disable (0) live logging/printing to
%                     the command window.
% Outputs:          none
%
% Example:
% PreprocessRawImgs('dataIn\imageSet1col\',...
%                   'dataOut\imageSet1bin\',...
%                   getSettings(1), 1);
%
% Other m-files required: getProcessingParams.m
%                         ProcessImgStd.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 12-Mar-2015

%------------- START OF CODE --------------

ss = settings;

% !!! parallel for loop
parfor i = ss.startImg:ss.endImg

    % Output log if required
```

```
    if outputLog, fprintf(['\n' num2str(i) '    ']), end
    % ================== PREPROCESSING ======================

    % Obtain raw image
    im_num = [];
    for i1 = 1:1:(ss.range)-size(num2str(ss.offset+i),2)
        im_num = [im_num '0'];
    end
    img = rgb2gray(imread([ss.imInPath im_num  ...
                num2str(ss.offset+i) '.jpg'], 'jpg'));

    % Preprocess raw colour image into binary image
    P = getProcessingParams(ss.id, i);
    imset = ProcessImgStd(img,P);
    imin = 255.*uint8((imset(:,:,6))>0);

    % Store binary image
    imwrite(imin, [ss.imOutPath num2str(i) '.jpg'], 'jpg', 'Quality', 50);

end

%------------- END OF CODE --------------
```

ProcessImgStd.m

```
function [imout] = ProcessImgStd(imin,params)

% ProcessImgStd - This function performs a number of image processing steps
% on a raw image of a kidney in order to extract a binary image
% representative of the nephron lumens. The chosen steps are optimised to
% reduce unwanted objects and enhance nephron cross-section contours.
%
% Syntax:  imout = ProcessImgStd(imin,params)
%
% Inputs:
%    imin           - The input image, colour or grayscale
%    params         - An array of 8 parameters as obtained from the
%                     getProcessingParams.m function.
% Outputs:
%    imout          - The mxnx6 array of output images at each stage:
%                       1. Grayscale
%                       2. Grayscale with background removed
%                       3. Equalised image (locally & globally)
%                       4. Thresholded image
%                       5. Thresholded, small segments removed
% FINAL binary image -> 6. Thresholded, small & large segments removed
%
% Example:
%     Out_541 = ProcessImgStd(In_541,getProcessingParams('Set2', 541));
%
% Other m-files required: Image Processing Toolbox
%
% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 12-Mar-2015

%------------- START OF CODE --------------

% Set default parameters
if nargin==1
    bthr = 180;
    eq_win = 64;
    th = 180;
```

```
    noise_pixel_size = 40;
    allowed_pixel_size = 5000;
    erase = 1;
end

if nargin==2
    bthr = params(1);
    eq_win = params(2);
    th = params(3);
    noise_pixel_size = params(4);
    allowed_pixel_size = params(5);
    erase = params(6);
end

% ========== Image Pre-Processing ============ %

% Convert to grayscale if necessary
if size(imin,3)==3
    IM(:,:,1) = rgb2gray(imin);
else
    IM(:,:,1) = imin;
end
IM(:,:,1) = uint8(IM(:,:,1));

% Remove Background
a = IM(:,:,1);
bthr = mean(mean(a))+bthr;
b = 255.*uint8(IM(:,:,1)>bthr);
bg_mask = bwconncomp(b, 4);
numPixels = cellfun(@numel,bg_mask.PixelIdxList);
idx = find(numPixels>allowed_pixel_size*10); %==max(numPixels)
temp = zeros(size(a));
for i=1:size(idx,2)
%     a(bg_mask.PixelIdxList{idx(i)}) = 0;
    temp(bg_mask.PixelIdxList{idx(i)}) = 1;
end
se = strel('disk',20);
temp = imclose(temp,se);
temp = imdilate(temp,strel('disk',2));
a = a.*uint8(~temp);        %imagesc(IM(:,:,2)), colormap gray
IM(:,:,2) = uint8(a);

% Histogram Equalisation
[w,h] = size(IM(:,:,2));
IM(:,:,3) = adapthisteq(IM(:,:,2),'NumTiles', round([w/eq_win h/eq_win]./6));
% IM(:,:,7) = IM(:,:,3);
IM(:,:,3) = adapthisteq(IM(:,:,3),'NumTiles', round([w/eq_win h/eq_win]));

% Erode/dilate equalised image
% kernel = [0 1 0; 1 1 1; 0 1 0];
% bg = imdilate(IM(:,:,3),kernel);
% fg = imerode(IM(:,:,3),kernel);
% IM(:,:,7) = fg-imcomplement(bg);

% Double thresholding
% temp = double(IM(:,:,3));
% temp1 = abs(temp-255);
% imagesc((temp>170)+(temp1>180))
% % hold on
% colormap gray

% Thresholding to form binary image
% High th prevents segments from joining
% Low th makes them join
c = IM(:,:,3);
```

```
thr = th;%graythresh(c).*255.*th
c(c<thr)=0;
c(c>=thr)=255;
IM(:,:,4)=uint8(c);

% ======== Remove unwanted components ======== %

% Erode/Dilate Routine to remove surrounding tissue
d = IM(:,:,4);
for t = 1:erase, d = imerode(d,[0 1 0; 1 1 1; 0 1 0]); end
for t = 1:erase-1, d = imdilate(d,[0 1 0 ; 1 1 1 ;  0 1 0 ]); end

% Remove small segments
d = uint8(bwareaopen(d, noise_pixel_size,8));
d = 255.*uint8(d==1);
IM(:,:,5)=uint8(d);

% Remove large segments
yy = uint8(bwareaopen(d, allowed_pixel_size));
yy(yy==1) = 255;
e = d - yy;
IM(:,:,6)=uint8(e);

imout = IM;

%------------- END OF CODE --------------
```

ProcessImgWS.m

```
function [imout] = ProcessImgWS(imin,params,fs)

% ProcessImgWS - This function performs a number of image processing steps
% on a raw image of a kidney in order to extract a binary image
% representative of the nephron lumens. The steps are similar to those
% implemented in ProcessImgStd.m with the addition of watershed
% segmentation to obtain isolated nephron cross sections.
%
% This function is not used due to oversegmentation of elongated nephron
% sections and merging with interstitial tissue segments.
%
% Syntax:  [imout] = ProcessImgWS(imin,params,fs)
%
% Inputs:
%    imin           - The input image, colour or grayscale
%    params         - An array of 8 parameters as obtained from the
%                     getProcessingParams.m function.
%    fs             - The filter size to use during watershed segmentation.
%
% Outputs:
%    imout          - The mxnx8 array of output images at each stage:
%                         1. Grayscale
%                         2. Grayscale with background removed
%                         3. Equalised image (locally & globally)
%                         4. Thresholded image
%                         5. Thresholded, small segments removed
%                         6. Segmented by Watershed method
%                         7. After merging close-by segments
% FINAL binary image -> 8. Thresholded, small & large segments removed
%
% Other m-files required: Image Processing Toolbox
%                            modWatershed.m
%
% Author: Charita Bhikha
```

```
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 28-Mar-2015

%------------- START OF CODE --------------

% Set default parameters
if nargin==1
    bthr = 180;
    eq_win = 64;
    th = 180;
    noise_pixel_size = 40;
    allowed_pixel_size = 5000;
    erase = 1;
    fs = 3;
end

if nargin==3
    bthr = params(1);
    eq_win = params(2);
    th = params(3);
    noise_pixel_size = params(4);
    allowed_pixel_size = params(5);
    erase = params(6);
    fs = 3;
end

% =========== Image Pre-Processing ============ %

% Convert to grayscale if necessary
if size(imin,3)==3
    IM(:,:,1) = rgb2gray(imin);
else
    IM(:,:,1) = imin;
end
IM(:,:,1) = uint8(IM(:,:,1));

% Remove Background
a = IM(:,:,1);
bthr = mean(mean(a))+bthr;
b = 255.*uint8(IM(:,:,1)>bthr);
bg_mask = bwconncomp(b, 4);
numPixels = cellfun(@numel,bg_mask.PixelIdxList);
idx = find(numPixels>50000); %==max(numPixels)
temp = zeros(size(a));
for i=1:size(idx,2)
%     a(bg_mask.PixelIdxList{idx(i)}) = 0;
    temp(bg_mask.PixelIdxList{idx(i)}) = 1;
end
se = strel('disk',20);
temp = imclose(temp,se);
temp = imdilate(temp,strel('disk',2));
a = a.*uint8(~temp);
a(a==0)=255;
IM(:,:,2) = uint8(a);

% imagesc(IM(:,:,8)), colormap gray

% Histogram Equalisation
% IM(:,:,3) = adapthisteq(IM(:,:,2),'NumTiles', [eq_win eq_win]);
[w,h] = size(IM(:,:,2));
IM(:,:,3) = adapthisteq(IM(:,:,2),'NumTiles', round([w/eq_win h/eq_win]./6));
IM(:,:,3) = adapthisteq(IM(:,:,3),'NumTiles', round([w/eq_win h/eq_win]));

% Thresholding
% High th prevents segments from joining
```

```
% Low th makes them join
c = IM(:,:,3);
thr = th;%graythresh(c).*255.*th
c(c<thr)=0;
c(c>=thr)=255;
IM(:,:,4)=uint8(c);

% ======== Remove unwanted components ======== %

% Erode/Dilate Routine to remove surrounding tissue
d = IM(:,:,4);
for t = 1:erase, d = imerode(d,[0 1 0; 1 1 1; 0 1 0]); end
for t = 1:erase-1, d = imdilate(d,[0 1 0 ; 1 1 1 ;  0 1 0 ]); end

% Remove small segments
d = uint8(bwareaopen(d, noise_pixel_size,8));
d = 255.*uint8(d==1);
IM(:,:,5)=uint8(d);

% Watershed method for segmentation
L = modWatershed(IM(:,:,3),fs);
IM(:,:,6)=uint8(L>0);

% Merge watershed and simple segmented images
L = uint8(L~=0 & L~=1);     % imagesc(L)
yy = uint8(bwareaopen(L, allowed_pixel_size));
yy(yy==1) = 255;
L = L - yy;
L = imclose(L,[0 1 0; 1 1 1; 0 1 0]);
mg = uint8(L|d);
IM(:,:,7)=uint8(mg);

% Remove large segments
yy = uint8(bwareaopen(mg, allowed_pixel_size));
yy(yy==1) = 255;
e = mg - yy;
IM(:,:,8)=uint8(e);

imout = IM;

%------------- END OF CODE --------------
```

modWatershed.m

```
function Iout = modWatershed(I,filtersize)

% Performs watershed segmentation on input image 'I' with a filter size
% specified by 'filtersize'. The method is derived from Matlab's example on
% "Marker-Controlled Watershed Segmentation".
% Link: http://www.mathworks.com/help/images/examples/marker-controlled-watershed-
segmentation.html

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 15-Mar-2015

% ---------------------------- START OF CODE ----------------------------

hy = fspecial('sobel');
hx = hy';
Iy = imfilter(double(I), hy, 'replicate');
```

```
Ix = imfilter(double(I), hx, 'replicate');
gradmag = sqrt(Ix.^2 + Iy.^2);

se = strel('disk', filtersize);
Ie = imerode(I, se);
Iobr = imreconstruct(Ie, I);

Iobrd = imdilate(Iobr, se);
Iobrcbr = imreconstruct(imcomplement(Iobrd), imcomplement(Iobr));
Iobrcbr = imcomplement(Iobrcbr);

fgm = imregionalmax(Iobrcbr);
fgm = bwareaopen(fgm, 5);

bw = im2bw(Iobrcbr, graythresh(Iobrcbr));

D = bwdist(bw);
DL = watershed(D);
bgm = DL == 0;

gradmag2 = imimposemin(gradmag, bgm | fgm);
Iout = watershed(gradmag2);

% ---------------------------- END OF CODE ----------------------------
```

FeatureExtractBWImgs.m

```
function dummy = FeatureExtractBWImgs(featFile, SPFile, settings, outputLog)

% FeatureExtractBWImgs - This function performs feature extraction on the
% binary images using the settings struct provided. The features include
% nodes, six shape factors and a shape profile per binary component in an
% image. The nodes and shape factors are stored together in a .m file
% specified by featFile and the shape profiles are stored seperately in
% SPFile due to large file sizes and saving methods.
%
% Syntax:  [~] = FeatureExtractBWImgs(binImPath, featFile, SPFile, ...
%                                     settings, outputLog)
% Inputs:
%    featFile       - The name and directory to the .m file in which the
%                     features will be saved.
%    SPFile         - The directory to which the shape profile .m files will be
saved.
%    settings       - A struct of the desired settings for pre-processing
%                     as created using the getSettings function
%    outputLog      - Enable (1) or disable (0) live logging/printing to
%                     the command window.
%
% Outputs:          none
%
% Other m-files required: getProcessingParams.m
%                         extractFeatures6.m
%                           findShapeProfileStretch.m
%                           litekmeansMod.m
%
% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 12-Mar-2015

%------------- START OF CODE --------------

% Initialisations
ss = settings;
```

```
shapeprof = matfile(SPFile,'Writable',true);
data = [0 0 0 0];
idx = ones(ss.endImg,1);

for i = ss.startImg:ss.endImg

    % Output log if required
    if outputLog, fprintf(['\n' num2str(i) '    ']), end

    % Read in binary image
    imin = 255.*uint8((imread([ss.imOutPath num2str(i) '.jpg'], 'jpg'))>180);

    % Extract features
    P = getProcessingParams(ss.id, i);
    % !!! 'extractFeatures6' contains the parallel for loop
    [cen_array, shape_fac,shape_prof] = extractFeatures6(imin(:,:,1),...
        P(4), P(8), ss.angStep, ss.scale, 1);

    % Store nodes and shape factors
    ccenters{i} = single([cen_array(:,2) cen_array(:,1) cen_array(:,3)]);
    shapefac{i} = single(shape_fac);

    % Store shape profile
    idx(i+1) = idx(i)+size(shape_prof,1);
    if ss.saveMethod==0   % Concatenate matrix in RAM
      data(idx(i):idx(i+1)-1,1:4) = single(shape_prof);
    elseif ss.saveMethod==1   % Concatenate matrix on hard disk
       shapeprof.data(idx(i):idx(i+1)-1,1:4) = single(shape_prof);
    end

end

if ss.saveMethod==0
    shapeprof.data = data;
end
shapeprof.idx = idx;

% Output features to file
save(featFile,'ccenters','shapefac','shapeprof','-v7.3')

%------------- END OF CODE --------------
```

extractFeatures6.m

```
function [nodes, shape_factors,shape_prof] = ...
    extractFeatures6(imin,minArea,dist,angStep,scale,clus_method)

% extractFeatures6 - Extracts nodes, shape factors and shape profiles for
% each component in a binary image. This function is custom-coded for binary
% images of kidney cross-sections.
%
% Syntax:  [nodes, shape_factors,shape_prof] =
%    extractFeatures6(imin,minArea,dist,angStep,scale,clus_method)
%
% Inputs:
%    imin        - The input binary image.
%    minArea     - The smallest binary component size to be processed.
%                  Components with an area (in pixels) smaller than this
%                  will be ignored. This is so that noise pixels are not
%                  allocated any features.
%    dist        - The desired minimum distance between adjacent nodes on a
%                  single binary component.
```

```
%    angStep     - The angle increment to use for the shape profiles.
%    scale       - The target scale of a binary component to use during
%                  shape profile extraction. Each component will be scaled
%                  to this size.
%    clus_method - Must an integer 1, 2 or 3 for:
%                       1: Matlab's built-in K-means
%                       2: litekmeans (faster, less accurate)
%                       3: Fuzzy c-means
% Outputs:
%    nodes          - An (mx3) array of the m nodes allocated on the image.
%                     Each row is a node. The first two columns are the x
%                     and y coordinates of the nodes respectively. The 3rd
%                     column is a binary component ID to be able to link
%                     nodes that belong to a common component.
%    shape_factors  - An (mx6) array of the m sets of shape factors. 6
%                     shape factors are extracted per node:
%                      1. circularity, 2. area,        3. eccentricity,
%                      4. solidity,    5. aspectRatio, 6. minorAxisLength
%    shape_prof     - An array of the m sets of shape profiles calculated
%                     for the m binary components in on the image.
%
% Other m-files required: findShapeProfileStretch.m
%                         litekmeansMod.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 12-Mar-2015

%------------- START OF CODE --------------

% Input parsing
if nargin==1
    minArea = 10;
    dist = 20;
    angStep = 15;
    scale = 50;
    clus_method = 1;
end

% Segment binay image and obatin required properties
% imlab = bwlabel(imin,4);
imin = bwconncomp(imin,4);

% ======================== Shape Factors ========================

stats = regionprops(imin, 'Area', ...
    'Eccentricity', 'EquivDiameter', 'Perimeter', 'BoundingBox',...
    'MinorAxisLength','MajorAxisLength','Image');



area = zeros(size(stats,1),1);
boundingBox = zeros(size(stats,1),4);
perimeter = area; equivDiameter = area; eccentricity = area;
solidity = area; majorAxisLength = area; minorAxisLength = area;
boxSize = area;
for i=1:1:size(stats,1)
    boundingBox(i,:) = stats(i).BoundingBox;
    area(i) = stats(i).Area;
    perimeter(i) = stats(i).Perimeter;
    equivDiameter(i) = stats(i).EquivDiameter;
    eccentricity(i) = stats(i).Eccentricity;
    majorAxisLength(i) = stats(i).MajorAxisLength;
    minorAxisLength(i) = stats(i).MinorAxisLength;
    %     solidity(i) = stats(i).Solidity;
```

```
    % Alternate solidity measure to increase speed
    tm = round(stats(i).BoundingBox);
    boxSize(i) = tm(3)*tm(4);
    solidity(i) = (stats(i).Area)./boxSize(i) + 0.21;
    if solidity(i)>1, solidity(i)=1; end

end
circularity = 1./((perimeter.^2)./(4.*pi.*area));
aspectRatio = majorAxisLength./minorAxisLength;

% ========================== Nodes ===================================

nodes = cell(size(stats,1),1);
shape_factors = cell(size(stats,1),1);
shape_prof = cell(size(stats,1),1);

parfor k=1:size(stats,1)

    % Obtain an image of the kth segment using the bounding box
    % Bound indices incase near image ends
    t = round(boundingBox(k,:));
%     if t(3)>t(4), w = t(3);
%     else w = t(4); end
%     t(4) = bound(1, size(imin,1),t(2)+w-1);
%     t(3) = bound(1, size(imin,2),t(1)+w-1);
% %     imseg = uint8(imlab(t(2):t(4),t(1):t(3))==k);
%     tx = bound(1,size(imin,1),[t(2)-10 t(4)+10]);
%     ty = bound(1,size(imin,2),[t(1)-10 t(3)+10]);
%     imseg = uint8(imlab(tx(1):tx(2),ty(1):ty(2))==k);

    imseg = zeros(t(4)+20, t(3)+20);
    imseg(11:11+t(4)-1,11:11+t(3)-1) = stats(k).Image;

    %imagesc(imseg)  imagesc(imlab)

    % Normalise to 1
    if area(k)>20*minArea
        kern = strel('disk',round(custSigmoid(area(k), -1, 4, 1, 300, 2)));
        imseg = imerode(imdilate(imseg,kern),kern); %imclose
    end
    imseg = uint8(imseg);
    imseg = imseg./max(max(imseg));
    imseg = uint8(imseg(10+1:end-10,10+1:end-10));

    % Find coordinates in the image ==1 to cluster
    [v,u] = ind2sub(size(imseg), find(imseg==1));
    v = reshape(v,numel(v),1);
    u = reshape(u,numel(u),1);
    step = 1;
    x = [v(1:step:end),u(1:step:end)];
    x = double(x');

    % Dont label connective tissue
    C=[];
    if area(k)<minArea %%|| (area(k)<ctarea && eccentricity(k)>0.9  )
        K=0; C=[];

    % If a segment is very small or round, K=1
    elseif area(k)<(40*minArea) || circularity(k)>0.9
        K=1; C = round([mean(x(1,:)) mean(x(2,:))]);

    % If a segment is elongated
    else
        K=2;
```

```
        terminate=false;
        while terminate==false && K<15   % Maximum centroids per segment

            if clus_method==1
                warning('off','stats:kmeans:FailedToConverge');
                warning('off','stats:kmeans:EmptyCluster');
                [~, C] = kmeans(x', K, 'EmptyAction','drop',...
                    'Start', 'sample');
            elseif clus_method==2
                C = litekmeansMod(x, K);
            else%if clus_method==3
                [C,~,~] = fcm(x', K,[2 100 1e-5 0]);

            end
            %   K = size(C,1);
                cond = pdist(C);
                cond = sort(cond);
                cond = cond(1:size(C,1)-1);

%                 cond=[];
%                 for i=1:1:size(C,1)
%                     d = sqrt(sum((bsxfun(@minus, C(i,:), C )).^2,2));
%                     d(d==0)=[];
%                     cond(i) = min(d);
%                 end

                if mean(cond)>dist
                    K=K+1;
                    if K>numel(x)
                        terminate=true;
                    end
%                  elseif std(cond)>5
                else
                    terminate=true;
                end
        end
    end
    %=========================================================================

    if K~=0

        if size(C,2)==2 && sum(sum(isnan(C)))==0

            C = round(C);
            % Remove centroids that are on empty space
%             rem=[];
%             for i=1:size(C,1)
%                 if imseg(C(i,1),C(i,2))==0, rem=[rem i]; end
%             end
%             C(rem,:)=[];

            % ============== Shape Profile ============
            spc=[];
            if ~isempty(size(C,1))

%                 [ang,dis] = findShapeProfile(imseg,angStep,C,scale);
%                     ang=ang';
              [ang,dis] = findShapeProfileStretch(imseg,angStep,C,scale);
                for ii=1:1:size(dis,2)
                    spc = [spc; [ang dis(:,ii) ii.*ones(numel(ang),1)]];
                end
            end
            shape_prof{k} = [spc k.*ones(size(spc,1),1)];
```

```
            %====================================

            % Translate to large image axis
            C(:,1) = C(:,1) + t(2) -1;
            C(:,2) = C(:,2) + t(1) -1;

            % Store centroids in array form and cell structure
            s = ones(size(C,1),1);
            nodes{k} = [C k.*s];
            shape_factors{k} = [circularity(k).*s area(k).*s ...
                  eccentricity(k).*s solidity(k).*s aspectRatio(k).*s...
                  minorAxisLength(k).*s];

        end
    end

end

nodes = cell2mat(nodes);
shape_factors = cell2mat(shape_factors);
shape_prof = cell2mat(shape_prof);

%------------- END OF CODE --------------
```

findShapeProfile.m

```
function [ang,dis] = findShapeProfile(imseg,angStep,C,scaling)

% findShapeProfile - This function calculates the shape profile (a
% radial plot) of a given binary segment at the specified angle increment
% around the given centre points. A scaling factor can be used to decrease
% the error associated with pixel discretisation on small segments.
%
% Syntax:  [angles,radii] = findShapeProfile(imseg,angStep,centers,scaling)
%
% Inputs:
%    imseg          - An image of the binary segment
%    angStep        - The desired angular increment of the profile
%    C              - The (x,y) location of the reference point/s about
%                     which the shape profile is desired.
%    scaling        - The target size to which the image is scaled prior to
%                     calculation (in pixels)
%
% Outputs:
%    ang              - The angles starting at -180 up to 179 in steps of angStep
%    dis              - The associated radii for ang.
%
% Example:
%  [angles,radii] = findShapeProfile(seg1,10,[15 20],50);
% Will calculate the shape profile for seg1 at 10 degree intervals around
% the point (x,y)=(15,20). The segment will be scaled to 50x50 pixels prior
% to calculation and the results are de-scaled after calculation.
%
% Other m-files required: none
%
% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 12-Mar-2015

%------------- START OF CODE --------------

% if nargin==0
```

```
%     imseg = zeros(6,6);
%     imseg(2:5,2:5) = ones(4,4);
%     imagesc(imseg)
%     angStep = 15;
%     C = [3 4];
%     scaling=50;
% end

ang=[];
dis=[];
if angStep>0

    ang = 0:angStep:359;
    tan_ang = tand(ang);

    scale=1;
    if scaling~=0 && (scaling/size(imseg,1))>1
        scale = scaling./size(imseg,1);
        imseg = imresize(imseg, scale, 'nearest','Colormap','original')>0.5;
        C = C.*scale;
    end

    % Find edge image and edge coordinates
    temp = zeros(size(imseg,1)+4,size(imseg,2)+4);
    temp(3:end-2,3:end-2)=imseg;
    C = C +2;
    b = imdilate(temp,[0 1 0;1 1 1;0 1 0]);
    ee=logical(b-temp);
%     ee = edge(temp,'log',0.5,0.4);
    [v1,u1] = ind2sub(size(ee), find(ee==1));
    order = double([v1(1:1:end),u1(1:1:end)]);

%    %Order edge coordinates
%     [xs,ys] = find(ee==1,1);
%     order=[xs ys];
%     while 1
%         ee(xs,ys) = 0;
%         nei = ee(xs-1:xs+1,ys-1:ys+1);
%         [xs,ys] = find(nei==1,1);
%         if isempty(xs), break, end
%         xs = xs + order(end,1) -2;
%         ys = ys + order(end,2) -2;
%         order = [order; xs ys];
%     end

    % Find shape profile
    dis=[];
    for ii=1:1:size(C,1)

        cen = C(ii,:);
        de= (bsxfun(@minus,order,cen));
        xx=[];

        for i=1:numel(ang)

            mx = order(:,1);
            my = order(:,2);
            theta = ang(i);

            if theta>0 && theta<90
                mask = and(de(:,1)>=0,de(:,2)>=0);
            elseif theta>90 && theta<180
                mask = and(de(:,1)<0,de(:,2)>=0);
            elseif theta>180 && theta<270
```

```
                mask = and(de(:,1)<=0,de(:,2)<=0);
            elseif theta>270 && theta<360
                mask = and(de(:,1)>=0,de(:,2)<=0);
            elseif theta==90
                mask = and(de(:,1)<1000,de(:,2)>0);
            elseif theta==270
                mask = and(de(:,1)<1000,de(:,2)<0);
            elseif theta==0
                mask = and(de(:,1)>0,de(:,2)<1000);
            elseif theta==180
                mask = and(de(:,1)<0,de(:,2)<1000);
            end

            mx = mx.*mask;
            my = my.*mask;
            mx(mx==0)=inf;
            my(my==0)=nan;

            %cand is candidate c=y-intercept which we want to be close to 0
            if (theta==90 || theta==270), cand = mx-cen(1);
            elseif (theta==0 || theta==180), cand = my-cen(2);
            else
                cand = -tan_ang(i)*(mx-cen(1))+(my-cen(2));
            end

            cand = abs(cand);
            % cand(cand<0)=nan;

            % Choose candidate with closest angle OR
            %  id = find(cand==min(cand),1);

             % Find top candidates at the desired angle
             [~,I] = sort(cand);
             if numel(I)>5
                id1 = (I(1:5));
             else
                 id1 = (I(1:end));
             end
            % Choose the one with smallest distance
            distt = sum((bsxfun(@minus,order(id1,:),cen)).^2,2);
            [~,id2] = min(distt);
            id = id1(id2);

            xx(i,:) = order(id,:);

        end

        dis(:,ii) = sqrt(sum((bsxfun(@minus,xx,cen)).^2,2));
        dis(:,ii) = (dis(:,ii)-2)./scale;

    end
end

%------------- END OF CODE --------------
```

findShapeProfileStretch.m

```
function [ang,Dis,fang,fdis,a3,d3] =
findShapeProfileStretch(imseg,angStep,centers,scaling)

% findShapeProfileStretch - This function calculates the shape profile (a
% radial plot) of a given binary segment at the specified angle increment
```

```
% around the given centre points. A scaling factor can be used to decrease
% the error associated with pixel discretisation on small segments.
%
% Syntax:  [angles,radii] = findShapeProfileStretch(imseg,angStep,centers,scaling)
%
% Inputs:
%    imseg          - An image of the binary segment
%    angStep        - The desired angular increment of the profile
%    centers        - The (x,y) location of the reference point/s about
%                     which the shape profile is desired.
%    scaling        - The target size to which the image is scaled prior to
%                     calculation (in pixels)
%
% Outputs:
%    ang              - The angles starting at -180 up to 179 in steps of angStep
%    Dis              - The associated radii for ang.
%    fang (optional)  - All angles present along the boundry of the segment.
%    fdis (optional)  - The associated radii for fang.
%    a3   (optional)  - The unwinded version of fang (redundant angles removed)
%    d3   (optional)  - The associated radii for a3.
%
% Example:
%  [angles,radii] = findShapeProfileStretch(seg1,10,[15 20],50);
% Will calculate the shape profile for seg1 at 10 degree intervals around
% the point (x,y)=(15,20). The segment will be scaled to 50x50 pixels prior
% to calculation and the results are de-scaled after calculation.
%
% Other m-files required: none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 12-Mar-2015

%------------- START OF CODE --------------

ang=[];
dis=[];

if angStep>0

    scale=1;
    if scaling~=0 && (scaling/size(imseg,1))>1
        scale = scaling./size(imseg,1);
        imseg = imresize(imseg, scale, 'nearest','Colormap','original')>0.5;
        centers = centers.*scale;
    end

    % Find edge image  and edge coordinates using dilation (faster than filter)
    temp = zeros(size(imseg,1)+4,size(imseg,2)+4);
    temp(3:end-2,3:end-2)=imseg;
    centers = centers +2;
    b = imdilate(temp,[0 1 0;1 1 1;0 1 0]);
    ee=logical(b-temp);
%     ee = edge(temp,'log',0.5,0.4);
    [v1,u1] = ind2sub(size(ee), find(ee==1));
    edg_pix = double([v1(1:1:end),u1(1:1:end)]);

    for c=1:size(centers,1)

        C = centers(c,1:2);
        ang=[];
        dis = [];

        % Find radius and angle of each edge pixel wrt C
        dis = sqrt(sum((bsxfun(@minus,edg_pix,C)).^2,2));
```

```
        delta= (bsxfun(@minus,edg_pix,C));
        ang = atan2d(delta(:,2),delta(:,1));
        ang = round(ang);

        % Store full radius and angle profiles
        fang = ang;
        fdis = dis;

        % Unwind and stretch to closest edges at desired angle increments
        des_ang = (-180:angStep:179);
        d=zeros(1,numel(des_ang)); %a=d;
        for i=1:numel(des_ang)
        %   t1 = ang(abs(ang-tempang(i))-min(abs(ang-tempang(i)))<4)
            t2 = fdis(abs(fang-des_ang(i))-min(abs(fang-des_ang(i)))<4);
            if isempty(t2)
                t2 = fdis(abs(fang-des_ang(i))==min(abs(fang-des_ang(i))));
            end
            t2 = t2(t2==min(t2),:);
            d(i) = t2(1);
        %   a(i) = t1(t2==min(t2),:);
        end

        ang = des_ang;
        dis = d;

        % Unwind and stretch to eliminate redundant angles
        if nargout>2

            [fang,sID] = sort(fang);
            fdis = fdis(sID);

            d=zeros(size(fang,1),1);
            a=d;
            idx=d;
            for i=1:size(fang,1)
               idx = find(des_ang-fang(i)<2) ;
               d(idx(1)) = fdis(i);
               a(idx(1)) = fang(i);
            end
            a(d==0)=[];
            d(d==0)=[];

            a3 = a(1:angStep:end);
            d3 = d(1:angStep:end);
            d3 = d3./scale;

        end
        %=====================================


        dis = dis./scale;
        fdis = fdis./scale;

        Dis(:,c) = dis;

    end
    ang = ang';
end

%------------- END OF CODE --------------
```

litekmeansMod.m

```
function m = litekmeansMod(X, k)

% litekmeansMod - Fast implementation of k-means clustering. The clustering
% process is repeated up to 20 times if the desired number of centroids is
% not obtained. This is a custom requirement for node allocation on nephron
% cross-sections. This function is a modification of the open-source
% function litekmeans.m written by Michael Chen.
%
% Syntax:  C = litekmeansMod(X, k)
%
% Inputs:
%    X           - d x n data matrix
%    k           - Number of seeds or centroids required.
%
% Outputs:
%   m            - the centroids of the clusters formed
%
% Other m-files required: none
%
% Original (litekmeans.m) Written by Michael Chen (sth4nth@gmail.com).
% http://www.mathworks.com/matlabcentral/fileexchange/24616-kmeans-clustering

% Modified by: Charita Bhikha (charita.bhikha@gmail.com)
% March 2015; Last revision: 12-Mar-2015

%------------- START OF CODE --------------

k_goal = k;
count = 0;
% reset = false;
success = false;
n = size(X,2);
last = 0;
label = ceil(k_goal*rand(1,n));  % random initialization
m=[];
u=[];

while success==false

    while any(label ~= last') %&& reset == false
        % remove empty clusters
        [u,~,label] = unique(label);
        k = length(u);
        % transform label into indicator matrix
        E = sparse(1:n,label,1,n,k,n);
        % compute m of each cluster
        m = X*(E*spdiags(1./sum(E,1)',0,k,k));
        last = label;
        % assign samples to the nearest centers
        [~,label] = max(bsxfun(@minus,m'*X,dot(m,m,1)'/2),[],1);
    end

%     if length(u)~=k_goal
%       reset = true;
%     end

    if length(u)==k_goal %reset==false
        success = true;
%         break
    else
        count = count +1;
        if count>20
```

```
            success = true;
%             break;
        else
%             reset = false;
            n = size(X,2);
            last = 0;
            label = ceil(k_goal*rand(1,n));  % random initialization
            m=[];
        end
    end


end
% [~,~,label] = unique(label);
if ~isempty(m)
    m=m';
end

%------------- END OF CODE --------------
```

litekmeans.m

```
function m = litekmeans(X, k)
% Perform k-means clustering.
%   X: d x n data matrix
%   k: number of seeds
% Written by Michael Chen (sth4nth@gmail.com).

n = size(X,2);
last = 0;
label = ceil(k*rand(1,n));  % random initialization
while any(label ~= last')
    [u,~,label] = unique(label);   % remove empty clusters
    k = length(u);
    E = sparse(1:n,label,1,n,k,n);  % transform label into indicator matrix
    m = X*(E*spdiags(1./sum(E,1)',0,k,k));    % compute m of each cluster
    last = label;
    [~,label] = max(bsxfun(@minus,m'*X,dot(m,m,1)'/2),[],1); % assign samples to
the nearest centers
end
% [~,~,label] = unique(label);
m=m';
```

PreprocAndFeatureExtract.m

```
function dummy = PreprocAndFeatureExtract(featFile,SPFile,settings,outputLog)

% PreprocAndFeatureExtract - This function performs pre-processing and
% feature extraction on the raw colour images using the settings struct
% provided. The output binary images are saved to disk automatically to the
% path specified. The features include nodes, six shape factors and a shape
% profile per binary component per image. The nodes and shape factors are
% stored together in a mat file specified by featFile and the shape
% profiles are stored seperately as specified in SPFile due to its large
% file size and required saving method.
%
% Syntax:  [~] = PreprocAndFeatureExtract(imInPath,imOutPath,...
%                         featFile,SPFile,settings,outputLog)
%
% Inputs:
%    featFile       - The name and directory to the .m file in which the
```

```
%                       features will be saved.
%    SPFile          - The directory to which the shape profile .m files will be
saved.
%    settings        - A struct of the desired settings for pre-processing
%                      as created using the getSettings function
%    outputLog       - Enable (1) or disable (0) live logging/printing to
%                      the command window.
%
% Outputs:           none
%
% Other m-files required: getProcessingParams.m
%                         ProcessImgStd.m
%                         extractFeatures6.m
%                           findShapeProfileStretch.m
%                           litekmeansMod.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 12-Mar-2015

%------------- START OF CODE --------------

% Initialisations
ss = settings;
shapeprof = matfile(SPFile,'Writable',true);
data = [0 0 0 0];
idx = ones(ss.endImg,1);

for i = ss.startImg:ss.endImg

      % Output log if required
    if outputLog, fprintf(['\n' num2str(i) '    ']), end
    % ==================== PREPROCESSING ======================

    % Obtain raw image
    im_num = [];
    for i1 = 1:1:(ss.range)-size(num2str(ss.offset+i),2)
        im_num = [im_num '0'];
    end
    img = rgb2gray(imread([ss.imInPath im_num  ...
                num2str(ss.offset+i) '.jpg'], 'jpg'));

    % Preprocess raw colour image into binary image
    P = getProcessingParams(ss.id, i);
    imset = ProcessImgStd(img,P);
    imin = 255.*uint8((imset(:,:,6))>0);

    % Store binary image
    imwrite(imin, [ss.imOutPath num2str(i) '.jpg'], 'jpg', 'Quality', 50);

    %=========================================================================
    % Extract features
    [cen_array, shape_fac,shape_prof] = extractFeatures6(imin(:,:,1),...
        P(4), P(8), ss.angStep, ss.scale, 1);

    % Store nodes and shape factors
    ccenters{i} = single([cen_array(:,2) cen_array(:,1) cen_array(:,3)]);
    shapefac{i} = single(shape_fac);

    % Store shape profile
    idx(i+1) = idx(i)+size(shape_prof,1);
    if ss.saveMethod==0   % Concatenate matrix in RAM
       data(idx(i):idx(i+1)-1,1:4) = single(shape_prof);
    elseif ss.saveMethod==1   % Concatenate matrix on hard disk
```

```
        shapeprof.data(idx(i):idx(i+1)-1,1:4) = single(shape_prof);
    end

end

if ss.saveMethod==0
    shapeprof.data = data;
end
shapeprof.idx = idx;

% Output features to file
save(featFile,'ccenters','shapefac','shapeprof','-v7.3')


%------------- END OF CODE --------------
```

Utility Functions

isIncluded.m

```
function [stat, row] = isIncluded(matrix, entry)

% isIncluded - Checks if a given row entry R is present in some matrix
% A. The number of columns in A must be equal to the number of elements in
% R. A status flag and row number is returned.

% Syntax:  [stat, row] = isIncluded(matrix, entry)
%
% Inputs:
%    matrix         - The input matrix (mxn)
%    entry          - The entry being queried (1xn)
%
% Outputs:
%    stat           - A flag indicating if the entry was found (1) or not (0)
%    row            - The row in which the entry was found if applicable
%
% Example:
%   [flag, row] = isIncluded([1 2 5; 4 5 6; 7 5 3], [4 5 6])
%   Returns flag = 1 and row = 2
%
% Other m-files required: none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

% Find entry in matrix if columns of matrix!=columns of entry
% matrix = single(full(matrix));
% entry = single(full(entry));

if isempty(entry) || isempty(matrix)
    stat = 0;    row = [];
else
    temp = and(((matrix(:,3)-entry(3))==0),and(((matrix(:,1)-
entry(1))==0),((matrix(:,2)-entry(2))==0)));
    % [row,col] = ind2sub(size(matrix),find(tempo));
  % More effecient version on ind2sub
    nrows = size(matrix,1);
    % ncols = size(matrix,2);
```

```
    idx = find(temp);
    row = rem(idx-1,nrows)+1;
    % col = (idx-row)/nrows + 1;
    stat = ~isempty(idx);

end

%------------- END OF CODE --------------
```

isIncluded2.m

```
function [stat, row] = isIncluded2(matrix, entry)

% isIncluded2 - Checks if a given row entry R is present in some matrix
% A. The number of columns in A must be equal to the number of elements in
% R. A status flag and row number is returned.
% Shorter code but slower than isIncluded.m

% Syntax:  [stat, row] = isIncluded2(matrix, entry)
%
% Inputs:
%    matrix         - The input matrix (mxn)
%    entry          - The entry being queried (1xn)
%
% Outputs:
%    stat           - A flag indicating if the entry was found (1) or not (0)
%    row            - The row in which the entry was found if applicable
%
% Example:
%   [flag, row] = isIncluded2([1 2 5; 4 5 6; 7 5 3], [4 5 6])
%   Returns flag = 1 and row = 2
%
% Other m-files required: none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

% Find entry in matrix if columns of matrix!=columns of entry
% matrix = single(full(matrix));
% entry = single(full(entry));

stat = 0;
row = [];
if ~isempty(entry) && ~isempty(matrix)
    temp = sum(abs(bsxfun(@minus,matrix,entry)),2);
    idx = find(temp==0);
    if ~isempty(idx)
        row = idx(1);
        stat = 1;
    end
end

%------------- END OF CODE --------------
```

bound.m

```
function out = bound(min, max, exp)
```

```
% bound - Takes in a vector or matrix of numeric values (exp), and a
% minimum and maximum value. All values below MIN is made equal to MIN; all
% values above MAX is made equal to MAX and others are left as is.
%
% Syntax:  y = bound(min, max, x)
%
% Example:
%   y = bound(5, 15, [0 6 -2 8 10.5 15 21]);
%   Returns y = [5 6 5 8 10.5 15 15]
%
% Other m-files required: none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

    min = min.*ones(size(exp));
    max = max.*ones(size(exp));

    mask = exp > max;
    exp = ~mask.*exp + mask.*max;
    mask = exp < min;
    out = ~mask.*exp + mask.*min;

%------------- END OF CODE --------------
```

getSegIDNum.m

```
function [seg_no, row_idx] = getSegIDNum(ccenters, coord)

% Finds the given coordinate (coord) in the node matrix (ccenters) and
% hence the respective segment ID number through the row index.

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

% Find the row of the coordinate
row_idx = find( and( abs(ccenters(:,1) - coord(1))<1 ,...
                abs(ccenters(:,2) - coord(2))<1 ));
% Obtain the segments ID number
seg_no = full(ccenters(row_idx,3));
```

custSigmoid.m

```
function out = custSigmoid(in, mode, FL, SL, TP, steepness)

% custSigmoid - This function models a typical sigmoid function with custom
% transition point, saturation levels and steepness. Mode (+1 or -1) can be
% used to simply flip the function about the turning point.
%
% Syntax:  f(x) = custSigmoid(x, mode, UL, LL, TP, steepness)
%
%   f(x)                    .
%    |                      . ,--------------- Second Limit (SL)
%    |                      ./
%    |                      /                      (mode = 1)
%    |                     /.
%    |                    /  .
%    |   ---------------'   .                  First Limit (FL)
%    |_____________________.____________________x
```

```
%                           TP
% Inputs:
%   in              - The input x value
%   mode            - Set to 1 to have the sigmoid go from the FL to the SL
%                     or set to -1 to make it go from the SL to the FL.
%   FL              - The first saturation limit
%   SL              - The second saturation limit
%   TP              - The turning or transition point on the x-axis
%   steepness       - A steepness coefficient of the sigmoid function.
%
% Outputs:
%    out            - The output f(x) value
%
% Other m-files required: none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

if mode==1
    out = -mode.*(FL-SL)./(1+exp(-(in-TP)/(10.*abs(10-steepness))))+FL;
elseif mode==-1
    out = -mode.*(FL-SL)./(1+exp(-(in-TP)/(10.*abs(10-steepness))))+SL;
end

%------------- END OF CODE --------------
```

dist.m

```
function d = dist(p1,p2)

% Calculates the Euclidean distances between a Mx3 coordinate matrix p1 and
% the coordinate p2.

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

d = sqrt(sum((bsxfun(@minus,p1,p2)).^2,2));
```

eucdist.m

```
function dis = eucdist(x,y)

% Given a list of x and y coordinates, this function calculates a
% progressive, cumulative Euclidean distance between adjacent pairs.

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------
dis = 0;
for i=1:1:numel(x)-1
   dis = dis + abs(sqrt((x(i)-x(i+1)).^2+(y(i)-y(i+1)).^2));
end

if numel(x)>2
dis = dis + abs(sqrt((x(1)-x(end)).^2+(y(1)-y(end)).^2));
end
```

```
% dis = round(dis);
%------------- END OF CODE --------------
```

Tracking

TrackerFinal.m

```
% TrackerFinal - This is the main script used to automatically track
% nephrons using the binary images and features from the previous stages.
% It is used as the user interface for tracking and manual correction.

% The following is a list of the m-files required by this script. The
% tabbed files indicate functions that are used exlusively within a
% respective function. The terms 'node', 'point' and 'coordinate' are used
% interchangeably within the function comments.

% ----- M-files required for tracking:
% changeMode.m
% clipImg.m
% findBranch2.m         - Implements horizontal tracking
% trackStraight.m       - Implements vertical tracking
%   findOffset.m        - Performs local image alignment
%   checkIfInNextImage.m
% reconstructPath.m     - Orders closed list into path coordinates
% validationSteps.m     - Performs the 5 validations for each move
%   formulateFeatures.m
% combineFeatures.m
% manualAdjustClick2.m
%   getEndPoints.m
%
% ----- ImageSet-specific functions
% These must be modified to ensure the image set being used is accoomodated for
% getSetProperties.m   - Get properties of the image set being used
% getSectionNo.m       - Get an identifier for the area of the image set being used
% getShapeProfileCells.m
% getTrackingParams.m  - Get tracking parameters for the current image
%
% ----- Function-wide utility functions:
% isIncluded.m
% isIncluded2.m
% bound.m
% displayCoord.m
% displayMove.m
% custSigmoid.m
% dist.m
% sortCell.m
% getSegIDNum.m        - Get the ID number for a nephron cross-section
%
% ----- M-files required for display & analysis:
% PlottingTools.m*
% plotTrackingResults.m*
% viewManualNephrons.m*
% array2struct_trackingData.m
% comparePaths2.m       - The function used to compare automatically &
%                         manually tracked nephrons
% getShapeProfile.m     - Used to view the shape profile of at a node
% tubeplot.m
%   frenet.m
%   frame.m
% tubeplot1.m
% saveobjtube.m - Exports the tube as a .obj file for use in a CAD environment
```

```
% >>>>>>>>>>>>>>>>>>>>>>> INSTRUCTIONS <<<<<<<<<<<<<<<<<<<<<<<<<

% Settings have been selected and parameters have been tuned for the 3
% mouse and 3 rat datasets. If a new data set is to be used, prior to
% proceeding, the following functions must be modified/checked to ensure
% that parameters and settings for the image set is set up:
% getSetProperties.m        - Set up relevant fields for the image set
% getShapeProfileCells.m    - Set up the path/s to the shape profile data
% getSectionNo.m            - Check if image set is included
% getTrackingParams.m       - Set up parameter variation through the image
%                             set using sigmoid functions
% changeMode.m              - Set up tracking settings per area of the
%                             nephron

% Thereafter, in this script:
% 1. Under section '0. LOAD DATA', set the image set number (imset) to the
%    set being used and version to the relevant data version.
% 2. Also choose which machine learning algorithm to load.
% 3. Run the section '0. LOAD DATA'. Variables ccenters, shapefac,
%    shapeprof and predictor must be loaded into the workspace

% 4. Under section '1. INITIALISE', set the initial seed. A seed can be
%    obtained by using the plotting tools to view the binary image with nodes,
%    and then selecting a node with the data cursor to get its coordinate.
% 5. Also set the desired option for capturing moves, liveplot and parallel
%    computing.
% 6. Run the section '1. INITIALISE'.

% 7. Run the section '2. RUN TRACKING'. The tracking process will now
%    proceed, and a live log will be seen in the command window. Once tracking
%    has proceeded as far as possible, the function ManualAdjustClick2.m will
%    be called to prompt the user to manually correct the path at the
%    end-points. This is done by viewing the 7 images shown, and linking the
%    central node in image 4 to the correct one that could not be
%    automatically tracked. Once clicked upon, enter the numerical number of
%    the image selected and press ENTER. Do this repeatedly until the requests
%    (usually 2-3) are done.
% 8. If a mode change has occured during the manual intervention, change
%    the parameters accordingly in the section 'A. MANUALLY CHANGE MODE'.
% 9. If manual correction have been done, run section '2. RUN TRACKING'
%    again. Repeat this process until the whole path, or desired length of the
%    path, has been tracked.

% 10.Finally, run the section '3. RECONSTRUCT PATH'. The final path will be
%    contained in the cell array fpath, with fpath{1} being the longest, most
%    complete path formed.
% 11.Use PlottingTools.m to view and analyse the data. A number of matrices
%    will be present in the workspace:
%     closed        - The pairs of child-parent nodes found during
%                     tracking.
%     manualCorrec  - The pairs of manual corrections made.
%     capMoves      - The pairs of all moves made during tracking along
%                     with the alignment and validation results for each
%                     move.
%     MOVES         - A struct version of capMoves.
%     distMeas      - The list of moves that did not pass distance
%                     validation.
%     skipBlock     - The list of moves where an image skip was attempted
%                     but blocked.
%     skipAllow     - The list of moves where an image skip was allowed.
%     biDirInv      - The list of moves that did not pass bidirectional
%                     movement validation.
%     misAligned    - The list of moves that were blocked due to high image
%                     misalignment.
%     mismatch      - The list of moves that were blocked by machine
%                     learning validation (neural net or svm)
```

```
%     ignored       - The list of nodes that were ignored due to being
%                     below the minimum area threshold.
%     fpath         - The cell array of unique paths formed, arranged from
%                     longest to shortest in the array.
%     mpath         - The cell array of the ambiguous/multiple paths formed.

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

% clc, clear all
% open('plotTrackingResults.m')

%% 0. LOAD DATA
clc
clear all

% Choose data set
imset = 3;
version = 1;

% Choose one, comment the other
load('dataOut\TrainedML\net_67features.mat'), predictor = net;
% load('dataOut\TrainedML\svm_67features.mat'), predictor = svm;

% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
s = num2str(imset); v = num2str(version);
set = getSetProperties(s,v);
load(['dataOut\set' s 'data' v '\set' s 'feat' v '.mat'])
shapeprof = getShapeProfileCells(s,v);

%% 1. INITIALISE

clearvars -EXCEPT ccenters shapefac shapeprof predictor s v set
clc

% Initial seed coordinate
init_seed    = [1176 1053 424];

captureMoves = true;
useParallel  = false;

% Live plotting options
liveplot = false;
w        = 200; %zoom in pixels

% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

numImgs      = size(ccenters,2);
minImgIdx    = 1;

curr_coord = [init_seed 0 0 0];

ss = changeMode('PCTPST');

log = 1;%fopen('log.doc','w');

if useParallel
    matlabpool open % use parallel processing if available
end
```

```
ii = curr_coord(3);    % ii = variable to track position through image set

terminate = false;
up = false;     down = false;       branch = false;
up_coord = [];  down_coord = [];    branch_coord = [];

% Skipping variables
sk_up = 0;   prvs_sk_up = [0 0 0 0];     buff_up = ones(1,8);
sk_dn = 0;   prvs_sk_dn = [0 0 0 0];     buff_dn = ones(1,8);

mism_up = 0; mism_dn = 0;
err_up = 0;  err_dn = 0;
off_up = 0; off_dn = 0;

im_have = [0 0 0];
img = zeros(set.imsize(1),set.imsize(2),3);
imtemp = img;

% Lists
open = curr_coord;
closed = [0 0 0 0 0 0];
deadend = [];
capMoves = [];
misMatch = [];
skipBlock = [];
skipAllow = [];
biDirInv = [];
distMeas = [];
ignored=[];
misAligned=[];
manualCorrec = [];
runtime = [];

modes.PCTPST = 0;
modes.DTL = 0;
modes.ATL = 0;
modes.TALDCT = 0;

loopHenle = false;

% load 'dataOut/Manual Data/mouse1.mat'
% manPath = nef.num87;
% manPath(:,3) = manPath(:,3).*4.3;


%% A. MANUALLY CHANGE MODE

% loopHenle = true;

% ss = changeMode('PCTPST');
% modes.PCTPST  = 1;
% modes.DTL     = 0;
% modes.ATL     = 0;
% modes.TALDCT  = 0;



%% 2. RUN TRACKING

looptimer=[];
tic
```

```
fprintf(['\nImg:' num2str(ii) ' start'])
while terminate==false

% Control mode of tracking
if size(capMoves,1)>10 && std([open(:,3); ii])<20%size(open,1)<1

    IMsig = smooth(capMoves(bound(1,size(capMoves,1),size(capMoves,1)-
100):size(capMoves,1),13),10);
    IMsig_eval = mean(IMsig(end-10:end));

    if (IMsig_eval > ss.IM_sens && modes.PCTPST==true)
        modes.PCTPST = false;
        modes.DTL = true;
        ss = changeMode('DTL');
        fprintf(log,'\n !!!!!!Inner Medulla!!!!!!!')

    elseif (loopHenle==true && modes.DTL==true)
        modes.DTL = false;
        modes.ATL = true;
        ss = changeMode('ATL');
        fprintf(log,'\n !!!!Loop of Henle!!!!!')

    elseif (IMsig_eval < (ss.IM_sens+0.2) && modes.ATL==true)
        modes.ATL = false;
        modes.TALDCT = true;
        ss = changeMode('TALDCT');
        fprintf(log,'\n !!!!DCT - Cortex!!!!!')
    end

end

% Evaluate manual path during tracking (slow)
% [m1, ~] = comparePaths2(manPath,[closed(:,1:2) (closed(:,3)+set.offset)],30);
% [m2, ~] = comparePaths2([closed(:,1:2) (closed(:,3)+set.offset)],manPath,30);
% fprintf(log,['\n' num2str(size(open,1)) '  ' num2str(round(m1)) ...
%     '  ' num2str(round(m2)) ' Img:' num2str(ii)]);

% ====================== Track horizontally ========================

if ss.brEnable==1
    [branch, tips, interim] = findBranch2(ccenters{ii}, curr_coord, open, closed);
else
    branch = false; interim = []; tips = [];
end

% If the current node is an 'entering' branch node, dont track vertically
if branch && size(tips,1)==2
    up=false; up_coord=[];
    upflags=[0 0 0 0 0]; upMLpred=[0 0 0 0 0]; upvals = [0 0 0 0 0];
    down=false; down_coord=[];
    dnflags=[0 0 0 0 0]; dnMLpred=[0 0 0 0 0]; dnvals = [0 0 0 0 0];
else

    % ========================= Obtain Images ============================
    im_want = [ii ii+sk_up+1 ii-sk_dn-1];

    [~,IA,IB] = intersect(im_have,im_want);
    [~,IC] = setdiff(im_want,im_have);
    imtemp(:,:,IB) = img(:,:,IA);
    for k=1:numel(IC)
        imtemp(:,:,IC(k)) = imread([set.imOutPath num2str(im_want(IC(k))) '.jpg']);
        % imtemp(:,:,IC(k)) = rgb2gray(imread([set.imInPath '0'
num2str(im_want(IC(k))) '.jpg']));
    end
```

```
% assert(size(imtemp,3)==3)

img = imtemp;
im_have = im_want;

img_clip = clipImg(img, curr_coord(:,1:2), ss.clip, set.imsize)>180;

% ==================== Track vertically ============================

% Get tracking parameters for image ii
tr = getTrackingParams(ii, set.id);

% Get shape profiles for current image
[sectn,iioff] = getSectionNo(ii,set.id);
SPidx = shapeprof{sectn}.idx;
spcurr = shapeprof{sectn}.data(SPidx(iioff):SPidx(iioff+1)-1,1:4);

% >>> UP

% Attempt to track upwards
if ss.upEnable==1
    s1=shapefac{ii+sk_up+1};
    [up, up_coord, err_up,off_up,mism_up,~] = trackStraight(...
                cat(3, img_clip(:,:,1),img_clip(:,:,2)), ...
                curr_coord, ii+sk_up+1, ...
                ccenters{ii+sk_up+1}(s1(:,2)>tr.areaLim,:), ...
                tr.trackRad, closed,[tr.maxOffset ss.max_match]);
else
    up=false; up_coord=[];
end

% Validate potential coordinate if found
if up
    % Get shape profiles for image above
     [sectn,iioff] = getSectionNo(ii+sk_up+1,set.id);
     SPidx = shapeprof{sectn}.idx;
     spup = shapeprof{sectn}.data(SPidx(iioff):SPidx(iioff+1)-1,1:4);

    % Validate the upward edge
    [up, upflags, upMLpred, upvals] = validationSteps(...
                off_up, predictor, tr.trackRad, ...
                curr_coord, ccenters{ii}, shapefac{ii}, spcurr,...
                up_coord, ccenters{ii+sk_up+1}, shapefac{ii+sk_up+1}, spup, ...
                ss, set, mism_up);
else
    upflags=[0 0 0 0 0]; upMLpred=[0 0 0 0 0]; upvals = [0 0 0 0 0];
end

% >>> DOWN

% Attempt to track downwards
if ss.dnEnable==1
    s1=shapefac{ii-sk_dn-1};
    [down, down_coord,err_dn,off_dn,mism_dn,~] = trackStraight(...
                cat(3, img_clip(:,:,1),img_clip(:,:,3)), ...
                curr_coord, ii-sk_dn-1, ...
                ccenters{ii-sk_dn-1}(s1(:,2)>tr.areaLim,:), ...
                tr.trackRad, closed, [tr.maxOffset ss.max_match]);
else
    down=false; down_coord=[];
end

% Validate potential coordinate if found
if down
```

```
        % Get shape profiles for image below
        [sectn,iioff] = getSectionNo(ii-sk_dn-1,set.id);
         SPidx = shapeprof{sectn}.idx;
         spdn = shapeprof{sectn}.data(SPidx(iioff):SPidx(iioff+1)-1,1:4);

         % Validate the upward edge
        [down, dnflags, dnMLpred, dnvals] = validationSteps(...
                    off_dn, predictor, tr.trackRad, ...
                    curr_coord, ccenters{ii}, shapefac{ii}, spcurr,...
                    down_coord, ccenters{ii-sk_dn-1}, shapefac{ii-sk_dn-1},
spdn,...
                    ss, set, mism_dn);
    else
        dnflags=[0 0 0 0 0]; dnMLpred=[0 0 0 0 0];dnvals = [0 0 0 0 0];
    end

end

% ================== Store Iteration Results ======================
while 1

if captureMoves==true
    if up,   capMoves(end+1,:) = [curr_coord(1:3) up_coord(1:3)   off_up upMLpred'
mism_up upvals]; end
    if down, capMoves(end+1,:) = [curr_coord(1:3) down_coord(1:3) off_dn dnMLpred'
mism_dn dnvals]; end
    if ~isempty(interim), capMoves =  [capMoves; [interim
zeros(size(interim,1),13)]]; end
    if ~isempty(tips), capMoves =  [capMoves; [tips zeros(size(tips,1),13)]]; end
end

% ======================== Output live log ========================
if up, fprintf(log,'\t up'); end
if upflags(1)~=0, ignored(end+1,:)= [up_coord upvals(1)];
        fprintf('\t sizeValUp '), end
if upflags(2)~=0, distMeas(end+1,:) = [curr_coord(1:3) up_coord(1:3) upvals(2)];
        fprintf('\t distValUp'), end
if upflags(3)~=0, skipBlock(end+1,:) = [curr_coord(1:3) up_coord(1:3) upvals(3)];
        fprintf('\t skipBlockUp'), end
if upflags(4)~=0, biDirInv(end+1,:) = [curr_coord(1:3) up_coord(1:3) upvals(4)];
        fprintf('\t BidirValUp'), end
if upflags(5)~=0, misMatch(end+1,:) = [curr_coord(1:3) up_coord(1:3) upMLpred'
upvals(5)];
        fprintf('\t shapeValUp '), end
if up && sk_up>0
    skipAllow(end+1,:) = [curr_coord(1:3) up_coord(1:3)];
    fprintf(log,'\t skipAllowUp')
end
if err_up, misAligned(end+1,:) = [curr_coord(1:3) curr_coord(1:2) ii+sk_up+1
mism_up];
    fprintf(log,'\t misAlignUp')
end

if down, fprintf(log,'\t down'); end
if dnflags(1)~=0, ignored(end+1,:)= [down_coord dnvals(1)];
        fprintf('\t sizeValDn '), end
if dnflags(2)~=0, distMeas(end+1,:) = [curr_coord(1:3) down_coord(1:3) dnvals(2)];
        fprintf('\t distValDn'), end
if dnflags(3)~=0, skipBlock(end+1,:) = [curr_coord(1:3) down_coord(1:3) dnvals(3)];
        fprintf('\t skipBlockDn'), end
if dnflags(4)~=0, biDirInv(end+1,:) = [curr_coord(1:3) down_coord(1:3) dnvals(4)];
        fprintf('\t BidirValDn'), end
if dnflags(5)~=0, misMatch(end+1,:) = [curr_coord(1:3) down_coord(1:3) dnMLpred'
dnvals(5)];
        fprintf('\t shapeValDn '), end
```

```
if down && sk_dn>0
    skipAllow(end+1,:) = [curr_coord(1:3) down_coord(1:3)];
    fprintf(log,'\t skipAllowDn')
end
if err_dn, misAligned(end+1,:) = [curr_coord(1:3) curr_coord(1:2) ii-sk_dn-1
mism_dn];
    fprintf(log,'\t misAlignDn')
end

if ~isempty(interim), fprintf(log,'\t <-=->');
elseif branch, fprintf(log,'\t <-> '); end

%========================= Live Display ============================

if liveplot
    I = imread([set.imOutPath num2str(ii) '.jpg'])>100;
    ind = find(closed(:,3)==ii);
    fill = double(round(sub2ind(size(I),open(1,2),open(1,1))));
    imagesc(~imfill(~I,fill)+0.6.*I);
    hold on, colormap hot
    scatter(closed(ind,1),closed(ind,2),'.','b')
    % scatter(ccenters{im}(:,1),ccenters{im}(:,2),'.','b')
    % title([num2str(ii) '    ' num2str(prediction)])
    title(num2str(ii))
    hold off, axis([init_seed(1)-w init_seed(1)+w init_seed(2)-w init_seed(2)+w])
    pause(0.5)
end

break
end


 % ==================== Skipping Control ============================%

sk_up = 0; sk_dn = 0;

% Update skip buffers
buff_up(end+1) = up; buff_up(1) = [];
buff_dn(end+1) = down; buff_dn(1) = [];

% If images were highly mismatched, increase chances of a skip
if err_up, buff_up(1:5)=1; end
if err_dn, buff_dn(1:5)=1; end

% Up
if (~branch && ~up && sum(buff_up)>=3 && ...
     sum(abs(prvs_sk_up-ii)<ss.skipRefrac)<ss.consecSkips )

    [~, ida] = getSegIDNum(ccenters{ii},curr_coord(1:3));
    if shapefac{ii}(ida,2)>tr.skipArea
        if any(prvs_sk_up(end-2:end)==ii), sk_up=2;
        else sk_up=1; end
        prvs_sk_up(end+1) = ii;
        prvs_sk_up(1)=[];
        fprintf([' skipup:' num2str(sk_up)])
    end

end

% Down
if (~branch && ~down && sum(buff_dn)>=3 && ...
     sum(abs(prvs_sk_dn-ii)<ss.skipRefrac)<ss.consecSkips )
```

```
    [~, ida] = getSegIDNum(ccenters{ii},curr_coord(1:3));
    if shapefac{ii}(ida,2)>tr.skipArea
        if any(prvs_sk_dn(end-2:end)==ii), sk_dn=2;
        else sk_dn=1; end
        prvs_sk_dn(end+1) = ii;
        prvs_sk_dn(1)=[];
        fprintf([' skipdw:' num2str(sk_dn)])
    end

end

% ==================== Update Lists ==========================%

% Dont update if a skip is to be attempted because tracking from curr_coord
% must be attempted again
if  sk_up==0 && sk_dn==0

    % If nothing is found from the current node, term it a dead-end
    if (~branch && ~up && ~down)
       deadend(end+1,:) = curr_coord(1,1:3);
       fprintf(log,' deadend ');
    end

    % Done exploring current node, so move it from open to closed list
    if ~isIncluded(closed(:,1:3), curr_coord(1:3))
        closed(end+1,:) = curr_coord;
    end

    % Add found coordinates to the open list
   if up
        if ~isIncluded(open(:,1:3),   up_coord) && ...
           ~isIncluded(closed(:,4:6), up_coord) && ...
           ~isIncluded(closed(:,1:3), up_coord)
            open(end+1,:) = [up_coord curr_coord(1,1:3)];
        end
   end

    if down
        if ~isIncluded(open(:,1:3),   down_coord) && ...
           ~isIncluded(closed(:,4:6), down_coord) && ...
           ~isIncluded(closed(:,1:3), down_coord)
            open(end+1,:) = [down_coord curr_coord(1,1:3)];
        end
    end

    if branch
        for k=1:1:size(tips,1)
            if ~isIncluded(open(:,1:3),   tips(k,1:3)) && ...
               ~isIncluded(closed(:,1:3), tips(k,1:3)) && ...
               ~isIncluded(closed(:,4:6), tips(k,1:3))
                open(end+1,:) = tips(k,:);
            end
        end
        % Only interim nodes get added to the closed list
        for k=1:1:size(interim,1)
            if ~isIncluded(closed(:,1:3), interim(k,1:3))
                closed(end+1,:) = interim(k,:);
            end
        end
    end

    % Terminate if all nodes have been explored
    if isempty(open)
```

```
        fprintf(log,' terminate ');
        break
    end

    % Define a new current node from the open list
    % 1. Choose a node which is in the lowest image
    newCoord = find(open(:,3)==min(open(:,3)),1);
    curr_coord = open(newCoord(1),:);%open(1,1:6);
    ii = curr_coord(1,3);
    open(newCoord(1),:)=[];

    % OR 2. Choose first node in the list
    %curr_coord = open(1,1:6);
    %open(1,:) = [];

end

looptimer(end+1) = toc; tic
fprintf(log,['\n' num2str(size(open,1)) ' Img:' num2str(ii)])

%== If extremes of image set has been reached, get a new current node ===

while ((ii+sk_up+1)>=numImgs)||((ii-1-sk_dn)<=minImgIdx)
    deadend(end+1,:) = curr_coord(1,1:3);
    fprintf(log,' endpoint ');
    closed(end+1,:) = curr_coord;
    if isempty(open)
        terminate = true;
        fprintf(log,' terminate ');
        break
    end
    curr_coord = open(1,:);
    ii = curr_coord(1,3);
    open(1,:) = [];
end

end % while loop end

runtime = [runtime toc];

manualAdjustClick2

%% 3. RECONSTRUCT PATH

[fpath, mpath] = reconstructPath(closed);

for i=1:size(fpath,2)
   fpath{i}(end,:)=[];
end

MOVES = array2struct_trackingData(capMoves);

fprintf([log,'\n\nLength of closed list: ' num2str(size(closed,1)) '\n'])
fprintf(['MinImg: ' num2str(min(closed(:,3))) '\n' ])
fprintf(['MaxImg: ' num2str(max(closed(:,3))) '\n' ])
fprintf('...Done!')

%------------- END OF CODE --------------
```

changeMode.m

```
function OPTNS = changeMode(mode)
```

```
% changeMode - Returns a struct containing a number of tracking settings
% related to a specific tracking mode, which is the area of the
% kidney/nephron being tracked. The settings for 4 transitions have been
% tuned. The settings must change at the transition between the zones
% because the morphology changes.

% Syntax:  options = changeMode(mode)
%
% Inputs:
%    mode   - A string of the mode to be changed to. The following are
%             valid modes:
%                - 'PCTPST'
%                - 'DTL'
%                - 'ATL'
%                - 'TALDCT'
%
% Outputs:
%    OPTNS  - A struct of various tracking settings tuned to the selected mode

% Other m-files required:   none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

OPTNS.mode = mode;

switch mode
  case 'PCTPST'
    OPTNS.bidirecValid = 1;
    OPTNS.distcoeff = 1;
    OPTNS.skipChange = 30;
    OPTNS.ml_sens = -0.1;
    OPTNS.IM_sens = 0.2;
    OPTNS.max_match = 0.80;
    OPTNS.clip = 80;
    OPTNS.brEnable = 1;
    OPTNS.upEnable = 1;
    OPTNS.dnEnable = 1;
    OPTNS.consecSkips = 1;
    OPTNS.skipRefrac = 1;
    OPTNS.minSize = 0;
    fprintf('Settings: PCT/PST \n')

  case 'DTL'
    OPTNS.bidirecValid = 0;
    OPTNS.distcoeff = 1.5;
    OPTNS.skipChange = 30;
    OPTNS.ml_sens = -0.4;
    OPTNS.IM_sens = 0.2;
    OPTNS.max_match = 0.70;
    OPTNS.clip = 50;
    OPTNS.brEnable = 0;
    OPTNS.upEnable = 1;
    OPTNS.dnEnable = 0;
    OPTNS.consecSkips = 1;
    OPTNS.skipRefrac = 1;
    OPTNS.minSize = 0;
    fprintf('Settings: DTL \n')

  case 'ATL'
    OPTNS.bidirecValid = 0;
```

```
    OPTNS.distcoeff = 1.5;
    OPTNS.skipChange = 30;
    OPTNS.ml_sens = -0.4;
    OPTNS.IM_sens = 0.2;
    OPTNS.max_match = 0.70;
    OPTNS.clip = 50;
    OPTNS.brEnable = 0;
    OPTNS.upEnable = 0;
    OPTNS.dnEnable = 1;
    OPTNS.consecSkips = 1;
    OPTNS.skipRefrac = 1;
    OPTNS.minSize = 0;
    fprintf('Settings: ATL \n')

   case 'TALDCT'
    OPTNS.bidirecValid = 1;
    OPTNS.distcoeff = 1.7;
    OPTNS.skipChange = 30;
    OPTNS.ml_sens = -0.1;
    OPTNS.IM_sens = 0.2;
    OPTNS.max_match = 0.70;
    OPTNS.clip = 80;
    OPTNS.brEnable = 1;
    OPTNS.upEnable = 1;
    OPTNS.dnEnable = 1;
    OPTNS.consecSkips = 1;
    OPTNS.skipRefrac = 1;
    OPTNS.minSize = 0;
    fprintf('Settings: TAL/DCT \n')

   otherwise
    disp('Invalid mode.')
end

%------------- END OF CODE --------------
```

clipImg.m

```
function imout = clipImg(imin, centre, W,imsize)

% clipImg - Simple cropping of an image about a point with a specified
% half-width W. If the required area is outside the bounds of the given
% image, only the portion of the image which exists is returned.

% Syntax:  imout = clipImg(imin, centre, width,imsize)
%
% Inputs:
%    imin           - The input image
%    centre         - The point around which the cropping must occur
%    W              - The required half-width around the centre point
%    imsize         - The size of the image
%
% Outputs:
%    imout          - The cropped sub-image
%
% Other m-files required: bound.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

    size_y = imsize(1);
```

```
    size_x = imsize(2);
    x = round(centre(1));
    y = round(centre(2));

    px = bound(1,size_x,[x-W x+W]);
    py = bound(1,size_y,[y-W y+W]);

    imout = imin(py(1):py(2),px(1):px(2),:);

%------------- END OF CODE --------------
```

findBranch2.m

```
function [branch, tips, interim] = ...
                findBranch2(CENTERS, COORD, OPEN, CLOSED)

% findBranch2 - This function implements horizontal tracking, i.e. given
% some current node in an image, it checks if other nodes lie on the same
% nephron segment as the current node. If so, all branch nodes found are
% ordered into child-parent pairs and returned as 'tip' branch coordinates
% or 'interim' branch coordinates. A flag is also returned. A check for
% inclusion in the open and closed lists is done to ensure the same set of
% coordinates are not included more than once.

% Syntax:  [branch, tips, interim] = ...
%                               findBranch2(CENTERS, COORD, OPEN, CLOSED)
%
% Inputs:
%    CENTERS    - The matrix of nodes in the image.
%    COORD      - The current node, or entering node.
%    OPEN       - The open list of coordinates (to be tracked).
%    CLOSED     - The closed list of coordinates (already tracked).
%
% Outputs:
%    branch     - A flag indicating if a branch has been found (1) or not (0)
%    tips       - A list of coordinates of end, or exit, branch points. Each
%                 coordinate is stored with its parent, which may be an
%                 interim branch point or the entry coordinate.
%    interim    - A list of coordinates of interim/central branch points. Each
%                 coordinate is stored with its parent, which may be another
%                 interim branch point or the entry coordinate.

% Other m-files required:   isIncluded.m
%                           dist.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

    branch = false;
    tips = [];
    interim = [];

    % Find row index of COORD in CENTERS
    idx1 = find( and( abs(CENTERS(:,1) - COORD(1))<1 ,...
                      abs(CENTERS(:,2) - COORD(2))<1 ));
    % Obtain the segments ID number
    seg_no = full(CENTERS(idx1,3));
%     [seg_no, ~] = getSegIDNum(CENTERS, COORD);

    % Obtain indexes of other nodes on the particular segment through ID number
```

```
    idx2 = find( abs(CENTERS(:,3) - seg_no)<1);

    % If other segments have been found
    if numel(idx2)>1

        % Remove index of COORD
        idx2 = idx2(idx2~=idx1);

        % Construct list of coordinates of all the branches
        all_branches = [CENTERS(idx2,1:2) COORD(3).*ones(numel(idx2),1)];

        % If only one other node is on the segment, return it as a tip
        if numel(idx2)==1
            branch = true;
            tips = [all_branches COORD(1:3)]; % child-parent

        % If more than 1 node is present, they need to be ordered
        elseif numel(idx2)>1

            branch = true;

            % Find angles and distances from COORD to all other branch nodes
            dis = dist(all_branches(:,1:2),COORD(1:2));
            ang = atan2d((bsxfun(@minus,all_branches(:,2),COORD(2))),...
                         (bsxfun(@minus,all_branches(:,1),COORD(1))));

            % Make all angles positive
            ang(ang<0) = ang(ang<0)+360;

            % Get an order index according to nodes closest to COORD
            [~,ord_idx] = sort(dis,'ascend');

            % Apply the ordering
            % near_dis = dis(ord_idx);
            near_ang = ang(ord_idx);
            all_br_ord = all_branches(ord_idx,:);

            % If the two closest points are opposite each other
            if abs(near_ang(1)-near_ang(2))>120 %&& abs(near_dis(1)-near_dis(2))<20

                % Two closest points' parent is COORD
                br{1} = [all_br_ord(1,:) COORD(1:3)];
                br{2} = [all_br_ord(2,:) COORD(1:3)];
                all_br_ord(1:2,:) = [];

                % Find parent-child pairs of the rest of the branch nodes
                % from the two closest nodes.
                while ~isempty(all_br_ord)

               [minima(1),minIdx(1)] = min(dist(all_br_ord(:,1:2),br{1}(end,1:2)));
               [minima(2),minIdx(2)] = min(dist(all_br_ord(:,1:2),br{2}(end,1:2)));

                    if minIdx(1)==minIdx(2)
                        [~,sidx]=min(minima(1:2));
                  br{sidx}(end+1,:) = [all_br_ord(minIdx(1),:) br{sidx}(end,1:3)];
                        all_br_ord(minIdx(1),:) = [];
                    else
                        br{1}(end+1,:) = [all_br_ord(minIdx(1),:) br{1}(end,1:3)];
                        br{2}(end+1,:) = [all_br_ord(minIdx(2),:) br{2}(end,1:3)];
                        all_br_ord(minIdx,:) = [];
                    end
```

```
                end

                % The last nodes (without children) are the end-points
                tips = [br{1}(end,:); br{2}(end,:)];
                br{1}(end,:)=[]; br{2}(end,:)=[];
                % The rest are interim nodes
                interim =  [br{1}; br{2}];

            % If the two closest points are in the same angle range, select
            % the closest one and progressively find child-parent pairs.
            else

                % The closest point's parent is COORD
                br = [all_br_ord(1,:) COORD(1:3)];
                all_br_ord(1,:) = [];
                child = br(1:3);

                while ~isempty(all_br_ord)
                    d = dist(all_br_ord,child);
                    parent = child;
                    child = all_br_ord(d==min(d),:);
                    child = child(1,:);
                    br(end+1,:) = [child parent];
                    all_br_ord(d==min(d),:) = [];
                end

                % The last node (without children) is the end-point
                tips = br(end,:);
                br(end,:)=[];
                % The rest are interim nodes
                interim = br;
            end
        end

    end

    % If the found branch nodes are already in the open or closed list,
    % they must be ignored as they (and their branches) have already been
    % processed.

    incl1=[]; incl2=[];
    if branch
            for k=1:1:size(tips,1)
                incl1(k) = isIncluded(OPEN(:,1:3), tips(k,1:3)) ...
                    || isIncluded(CLOSED(:,1:3), tips(k,1:3));
            end
            for k=1:1:size(interim,1)
                incl2(k) = isIncluded(CLOSED(:,1:3), interim(k,1:3));

            end
    end

    if (sum(incl1)+sum(incl2))~=0
            branch = false; interim = []; tips = [];
    end

%------------- END OF CODE --------------
```

trackStraight.m

```
function [flag, tr_coord, err,offset,mismatch,inclflag] = ...
            trackStraight(clipped_imgs, COORD, iiNext, ccentersNext, ...
```

```
                          tracking_rad, CLOSED, maxParams, offset)

% trackStraight - This function implements vertical tracking from a given
% node (COORD) into an image (number iiNext) with a set of nodes
% (ccentersNext). Cropped images of ii and iiNext are re-aligned using x-y
% transalation to produce more accurate tracking. The maximum tracking
% radius, offset and mismatch provided are used to regulate the tracking
% result. This function is also used during bidirectional validation, where
% the [x y] offset is provided instead of being calculated.

% Syntax:  [flag, coord, err,offset,match,inclflag] = trackStraight(clipped_imgs,
%  ... curr_coord, iiNext, ccentersNext, tracking_rad, closed, maxParams, offset)
%
% Inputs:
%    clipped_imgs   - The cropped images of the area around the current
%                     node (COORD) of image ii (current image no.) and iiNext
%    COORD          - The current node
%    iiNext         - The number of the next image, in which the nephron must
%                     be tracked
%    ccentersNext   - The array of node cordinates in the next image
%    tracking_rad   - The maximum tracking radius for vertical tracking
%    CLOSED         - The closed list of coordinates (nodes already tracked)
%    maxParams      - A 2 element vector of the:
%                          1. Maximum offset to be allowed (pixels)
%                          2. Maximum image mismatch to be allowed (0-100)
%    *offset        - The x-y offset between images ii and iiNext. If supplied,
%                     the offset is not automatically determined.
%                     * ONLY used during bidirectional validation.
%
% Outputs:
%    flag           - A flag indicating if a node has been found in the
%                     next image (1) or not (0)
%    tr_coord       - The tracked coordinate in image iiNext
%    err            - An error flag which is set if the:
%                          = mismatch is high (larger than the maximum allowed)
%                          = offset is high (larger than the offset allowed)
%                          = offset is medium and an image has been skipped
%    offset         - The calculated [x y] offset
%    mismatch       - A mismatch metric between the two images. A high value
%                     indicates a large mismatch between the images.
%    inclflag       - A flag indicating if the tracked node is (1) or is
%                     not (0) included in the CLOSED list.

% Other m-files required:   findOffset.m
%                           checkIfInNextImage.m
%                           isIncluded.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

maxoffset = maxParams(1);
maxmismatch =  maxParams(2);
% 1. Try to track the tube in the next image (straight) from curr_coord

% Cross-correlate the images to find translational differences
if nargin==7
    [x_off, y_off,mismatch] = findOffset(clipped_imgs);
%     [output, ~] = dftregistration(fft2(clipped_imgs(:,:,1)),...
%         fft2(clipped_imgs(:,:,2)),1);
%     x_off = output(3);
%     y_off = output(4);
%     match = abs(output(1));
```

```
else
    x_off = offset(1);
    y_off = offset(2);
    mismatch=0;
end

inclflag = false;
offset = [x_off y_off];

% If correlation is low, dont allow it
if any(isnan(offset)) || (mismatch>maxmismatch)% && abs(curr_coord(3)-iiNext)==1)
        flag=false; tr_coord=[]; err=true;

% If offset is too large, dont allow it
elseif  sqrt(x_off.^2 + y_off.^2)>maxoffset;
    flag=false; tr_coord=[]; err=true;

% If offset is medium with a skip, dont allow it
elseif  abs(COORD(3)-iiNext)>1 && sqrt(x_off.^2 + y_off.^2)>maxoffset/3;
    flag=false; tr_coord=[]; err=true;
else

    err = false;
% Define translation matrices (use inverse so no holes/overlap is produced)
%     T = [1     0     0;...
%          0     1     0;...        % not needed as its only translation
%          x_off y_off 1];

    % Apply translational transform to nodes of next image so
    % that a better comparison can be made to the current node
    %  tr_cent = tfm(ccentersNext(:,1:2), T);
    %  tr_cent = [tr_cent (iiNext).*ones(size(tr_cent,1),1)];
    tr_cent = [ccentersNext(:,1)+x_off ccentersNext(:,2)+y_off];
    index1 = checkIfInNextImage(COORD(1,1:3), tr_cent, tracking_rad);

    tr_coord = [];
    flag = false;
    if ~isempty(index1)

        temp = [ccentersNext(index1,1:2) (iiNext)];

        % Check for inclusion in the CLOSED list
        if ~isIncluded(CLOSED(:,1:3), temp)
            tr_coord = temp;
            flag = true;
        else
            inclflag = true;
        end

        if nargin==8
            tr_coord = temp;
            flag = true;
        end
    end

end

%------------- END OF CODE --------------
```

findOffset.m

```
function [x_off, y_off, mismatch] = findOffset(IM)
```

```
% findOffset - Finds the translational offset between two images using
% cross-correlation, implemented using a 2D fft. In addition a gaussian
% function is used to keep focus towards the centre of the reference image
% which is the current node location. A similarity metric is measured
% between the images after applying the found offset to the input image.

% Syntax:  [x_off, y_off, mismatch] = findOffset(IM)
%
% Inputs:
%    IM         - A MxNx2 array containing the reference and input images,
%                 respectively.
% Outputs:
%    x_off      - The pixel offset in the x direction (columns)
%    y_off      - The pixel offset in the y direction (rows)
%    mismatch   - A mismatch metric between the two images. A high value
%                 indicates a large mismatch between the images.

% Other m-files required:   none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

IM = double(IM);

% Multiply the reference image by a 2D gaussian function to concentrate the
% alignement towards the current node (which is the centre of the image)
gauss = fspecial('gaussian',size(IM,1),0.2*size(IM,1));
gauss = gauss./(max(max(gauss)));
IM(:,:,1) = IM(:,:,1).*gauss;

% Cross-correlate the two images to find translational difference which
% occurs at maximum correlation
% B = xcorr2(A(:,:,1),A(:,:,2));
s = size(IM(:,:,1))+size(IM(:,:,2))-1;
B = ifft2(fft2(IM(:,:,1),s(1),s(2)).*conj(fft2(IM(:,:,2),s(1),s(2))));
B = fftshift(B);
B = abs(B);
[yy,xx] =ind2sub(size(B),find(B==max(max(B))));
x_off = xx(1)-size(IM,2);
y_off = yy(1)-size(IM,1);

% Limit the offset found
maxOffset =  50;
% if abs(x_off)>maxOffset || abs(y_off)>maxOffset
if sqrt(x_off.^2 + y_off.^2)>maxOffset
    x_off = nan;
    y_off = nan;
    mismatch = 0;
else

    % Apply translational transform to the input image
    IM(IM~=0)=1;
    IM=logical(IM);
    tform = maketform('affine',[1 0 0; 0 1 0; x_off y_off 1]);
    temp = imtransform(IM(:,:,2),tform,'XData',...
        [1 size(IM,2)],'YData',[1 size(IM,1)]);

    % Calculate mismatch metric
    mismatch = sum(sum(abs(temp-IM(:,:,1)))./...
               sum(sum(logical(IM(:,:,1)+temp)));%corr2(A(:,:,1),temp)
```

```
end

%------------- END OF CODE --------------
```

checkIfInNextImage.m

```
function index = checkIfInNextImage(coord, centers, tracking_rad)

% checkIfInNextImage -  Returns the index of the found node, i.e.
% centers(index,:) is the new coordinate found

% Syntax:  index = checkIfInNextImage(current_coord, centers, tracking_rad)
%
% Inputs:
%    coord          - The reference node
%    centers        - The array of node cordinates in the queried image
%    tracking_rad   - The maximum tracking radius for vertical tracking
%
% Outputs:
%    index          - The row index in centers of the found coordinate

% Other m-files required:   none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

% Find all nodes in centers within the tracking radius around coord
coord = round(coord);
potential = find(dist(centers(:,1:2),coord(1:2))<tracking_rad);
index = [];
if ~isempty(potential)
    % find the radius of all possibilities
    rad_poss = ((centers(potential,1)-coord(1)).^2 + ...
                (centers(potential,2)-coord(2)).^2);

    % find the node closest to current_coord
    closest = find(rad_poss==min(rad_poss),1);
    index = potential(closest);
end

%------------- END OF CODE --------------
```

reconstructPath.m

```
function [maxpath, multpath] = reconstructPath(list)

% reconstructPath - This function forms a linked list from an array of
% child-parent coordinates. Ambiguous paths are removed and the longest
% path is constructed and returned in maxpath. All paths constructed are
% returned in mpath.

% Syntax:  [maxpath, multpath] = reconstructPath(list)
%
% Inputs:
%    list   - The array of child-parent coordinates/nodes (Mx6)
%
% Outputs:
%    maxpath    - A cell array of the reduced paths
```

```
%    multpath   - A cell array of the all the paths found

% Other m-files required:   isIncluded.m
%                           sortCell.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

if sum(list(1,:))==0, list(1,:)=[]; end

% Find all nodes that are not parents, i.e. end points
p=[];
for i=1:1:size(list,1)
    p(i) = ~isIncluded(list(:,4:6),list(i,1:3));
end
ki = find(p==1);

% Track path backwards from each end point
multpath=[];

for kii = 1:numel(ki)
%     idxs = ki(kii);
    path = list(ki(kii),1:3);
    k = ki(kii);
    while ~isempty(k)
        [~, k] = isIncluded(list(:,1:3), list(k(1),4:6));
        path = [path; list(k,1:3)];
%         idxs(end+1) = k;
    end
    multpath{kii} = path;
%     I{kii} = idxs;
end

% Sort segmented paths according to size
multpath = sortCell(multpath);
temp0 = multpath;

% Allocate common pathways to longest path segments
[p,q] = meshgrid(1:1:size(temp0,2),1:1:size(temp0,2));
p = p(:);
q = q(:);

for i1=1:1:size(temp0,2)^2

    r1 = p(i1);
    r2 = q(i1);

    if r1~=r2
        temp1 = temp0{r1};
        temp2 = temp0{r2};
        diff = size(temp1,1) - size(temp2,1);

        if diff>0
            temp2 = [zeros(abs(diff),3); temp2];
            mask = logical(repmat(mean(temp1-temp2,2),1,3));
            mask(end,:) = [1 1 1];
            temp2 = temp2.*mask;
%             temp2(bsxfun(@eq,temp2,[0 0 0]))=[];
        elseif diff<0
            temp1 = [zeros(abs(diff),3); temp1];
```

```
            mask = logical(repmat(mean(temp1-temp2,2),1,3));
            mask(end,:) = [1 1 1];
            temp1 = temp1.*mask;
%             temp1(bsxfun(@eq,temp1,[0 0 0]))=[];
        elseif diff==0
            mask = logical(repmat(mean(temp1-temp2,2),1,3));
            mask(end,:) = [1 1 1];
            temp2 = temp2.*mask;
%             temp2(bsxfun(@eq,temp1,[0 0 0]))=[];
        end
        temp0{r1} = reshape(temp1,[],3);
        temp0{r2} = reshape(temp2,[],3);
    end
end

% Remove empty coordinates/paths
for i=1:1:size(temp0,2)

    temp = temp0{i};
    temp(bsxfun(@eq,temp,[0 0 0]))=[];
    temp = reshape(temp,[],3);
    temp0{i} = temp;

   if size(temp0{i},1)<8%isempty(kkk{i})
       temp0{i}=[];
       i=1;
   end

end
temp0 = temp0(~cellfun('isempty',temp0));

% Sort cells according to size
temp0 = sortCell(temp0);
maxpath = temp0;

%------------- END OF CODE --------------
```

validationSteps.m

```
function [pass, flags, MLpred, vals] = validationSteps...
                        (offset, classifier, tracking_rad, ...
                         curr_coord, ccentersC, shapefacC, shapeprofC,...
                         next_coord, ccentersN, shapefacN, shapeprofN,...
                         options, set, mismatch)

% validationSteps -  The current node (curr_coord) in image ii has been
% potentially tracked to a node (next_coord) in image iiN by the function
% trackstraight.m. This function implements the five validation steps for
% this potential move from one nephron cross-section to another.
% The features relating to the two nodes (other nodes in the image, shape
% factors, shape profiles) as well as other information (see inputs) are
% used by the validation rules to evaulate characteristics about the move
% and hence its validity.

% Syntax:  [pass, flags, MLpred, vals] = validationSteps...
%                         (offset, net, mu, sigma, tracking_rad, ...
%                          curr_coord, ii, ccentersC, shapefacC, shapeprofC,...
%                          next_coord, iiN, ccentersN, shapefacN, shapeprofN,...
%                          options, set, match)
%
% Inputs:
%     offset          - The [x y] translational offset between the images
%     net             - The neural network structure created using the
```

```
%                       Neural Network Toolbox
%     mu              - The mean value to be used to normalise the data
%                       (derived from the training data)
%     sigma           - The sigma (standard deviation) value to be used to
%                       normalise the data
%     tracking_rad    - The tracking radius (used during bidirectional
%                       validation)
%     curr_coord      - The current node coordinate
%     ccentersC       - The array of nodes in image ii
%     shapefacC       - The array of the 6 shape factors for image ii
%     shapeprofC      - The array of shape profiles for image ii
%                       (contains 4 columns: angle, radii, node ID,
%                       segment ID)
%     next_coord      - The coordinate of the next potential node
%     ccentersN       - The array of nodes in image iiN
%     shapefacN       - The array of the 6 shape factors for image iiN
%     shapeprofN      - The array of shape profiles for image iiN
%                       (contains 4 columns: angle, radii, node ID,
%                       segment ID)
%     options         - A struct of settings related to the current mode,
%                       created using changeMode.m
%     set             - A struct of properties related to the image set
%     mismatch        - The image similarity metric from
%                       trackStraight>findOffset
%
% Outputs:
%     pass      - A flag indicating if all validation steps have been
%                 passed(1) or not (0)
%     flags     - An array of 5 flags indicating which validation steps
%                 were passed (1) and which were not (0)
%     MLpred    - The output of the machine learning predictor
%     vals      - 5 numerical values relating to the result of each of the
%                 validation steps

% Other m-files required:   getSegIDNum.m
%                           trackStraight.m
%                               findOffset.m
%                               checkIfInNextImage.m
%                               isIncluded.m
%                           formulateFeatures.m
%                           Neural Network Toolbox v8.0.1 (R2013a)
%                          (optional) Parallel Computing Toolbox  (R2013a)

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

ii = curr_coord(3);
iiN = next_coord(3);
pass    = true;
flags   = [0 0 0 0 0];
vals    = [0 0 0 0 0];
MLpred  = [0 0 0 0 0]';
[seg1, ida] = getSegIDNum(ccentersC,curr_coord(1:3));

% ============== 1. Size Validation ================
if options.minSize~=0 %&& pass==true
    [~, idaa] = getSegIDNum(ccentersN,next_coord(1:3));
    seg_area = shapefacN(idaa,2);
    vals(1) = seg_area +0.000001;
    if seg_area<options.minSize
        flags(1) = 1;
        pass = false;
    end
```

```
end

% ============== 2. Distance Validation ================
 if options.distcoeff~=0 %&& pass==true
     [~, idb] = getSegIDNum(ccentersN,next_coord(1:3));
     xydis = sqrt(sum((curr_coord(1:2)-(next_coord(1:2)+offset)).^2))...
                    /abs(iiN-ii);

     % Use half the minor axes as radii approximations
     sumradii = (shapefacC(ida,6)+shapefacN(idb,6))./2;

%      Use shape profile radii instead of (minor axes)/2
%      [~,sef] = formulateFeatures(classifier.userdata.mean,...
%                                  classifier.userdata.stddev, ...
%                      curr_coord(1:3), next_coord(1:3), offset,mismatch,...
%                      ccentersC, ccentersN, shapefacC, shapefacN, ...
%                      shapeprofC, shapeprofN,set );
%      sumradii = min(sef{1}.SP1(:,2)) + min(sef{1}.SP2(:,2));

     vals(2) = xydis-sumradii*options.distcoeff +0.000001;
     if xydis > sumradii*options.distcoeff
         flags(2) = 1;
        pass = false;
     end
 end

% ============== 3. Skip Validation ================
if (abs(iiN-ii)>1) && options.skipChange~=0 %&& pass==true
    [~, idb] = getSegIDNum(ccentersN,next_coord(1:3));
    change = mean(100.*abs(shapefacC(ida,:)-shapefacN(idb,:))./...
             min([shapefacC(ida,:);shapefacN(idb,:)]));
     vals(3) = change +0.000001;
    if change>options.skipChange
        flags(3) = 1;
        pass = false;
    end
end

% ============== 4. Bidirectional Validation ================
if options.bidirecValid~=0 %&& pass==true
    [~, val_coord, ~,~,~,~] = trackStraight(1, next_coord, ...
            ii, ccentersC, tracking_rad, [0 0 0 0 0 0],[200 2], -offset);
    [seg2, ~] = getSegIDNum(ccentersC,val_coord(1:3));
    if ~(sum(val_coord-curr_coord(1:3))==0 || (seg1==seg2))
        vals(4) = 1;
        flags(4) = 1;
        pass = false;
    end
end

% ============== 5. Shape Validation ================
if options.ml_sens~=0 %&& pass==true

    Xnorm = formulateFeatures(classifier.userdata.mean,...
                              classifier.userdata.stddev, ...
                    curr_coord(1:3), next_coord(1:3), offset,mismatch,...
                    ccentersC, ccentersN, shapefacC, shapefacN, ...
                    shapeprofC, shapeprofN,set );

      if strcmp(classifier.name, 'Pattern Recognition Neural Network')
        MLpred = classifier(Xnorm','useParallel','yes');
      elseif strcmp(classifier.name, 'RBF Kernel Support Vector Machine')
        MLpred = svmclassify(classifier, Xnorm);
      end
```

```
    vals(5) = sum(MLpred(3:4))-max(MLpred(1:2)) ;
    if vals(5) < options.ml_sens
        pass = false;
        flags(5) = 1;
    end
end


%------------- END OF CODE --------------
```

formulateFeatures.m

```
function [Xnorm, segfeat] = formulateFeatures(mu, sigma, ...
                            curr_coord, next_coord , offset, mismatch, ...
                            ccentersC, ccentersN, shapefacC, shapefacN, ...
                            shapeprofC, shapeprofN ,set)

% formulateFeatures - This function takes in two nodes and all the features
% of all the data of the images they belong to. The features relating to
% the two nodes are extracted and made available in the struct segfeat. The
% raw features of the two nodes (the nodes themselves, shape factors, shape
% profiles) are combined into a 67-element feature vector Xnorm which is
% normalised to the training data.

% Syntax:  [Xnorm, segfeat] = formulateFeatures(mu,sigma, curr_coord, next_coord ,
off,...
%           match,centers_curr, centers_next, sf_curr, sf_next, sp_curr,
sp_next,set)
%
% Inputs:
%     mu              - The mean value to be used to normalise the data (derived
from the training data)
%     sigma           - The sigma (standard deviation) value to be used to
normalise the data
%     curr_coord      - The current node coordinate
%     next_coord      - The coordinate of the next potential node
%     offset          - The [x y] translational offset between the images
%     mismatch        - The image similarity metric from trackStraight>findOffset
%     ccentersC       - The array of nodes in image ii
%     ccentersN       - The array of nodes in image iiN
%     shapefacC       - The array of the 6 shape factors for image ii
%     shapefacN       - The array of the 6 shape factors for image iiN
%     shapeprofC      - The array of shape profiles for image ii
%     shapeprofN      - The array of shape profiles for image iiN
%     set             - A struct of properties related to the image set
%
% Outputs:
%     Xnorm      - An array of the normalised features of the move
%     segfeat    - A struct containing the raw features of the move
%
% Other m-files required:   getSegIDNum.m
%                           isIncluded.m
%                           combineFeatures.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

    % Order features into a struct
    move = [curr_coord next_coord offset];
    segfeat{1}.coord1 = move(1:3);
```

```
    segfeat{1}.coord2 = move(4:6);
    segfeat{1}.offset = move(7:8);

    t1 = ccentersC;
    t2 = shapeprofC;

    [segNo, idx] = getSegIDNum(t1,move(1:2));
    [~, row] = isIncluded(t1(t1(:,end)==segNo,:), [move(1:2) segNo]);
    segfeat{1}.SF1 = shapefacC(idx,:);
    segfeat{1}.SP1 = t2(and(t2(:,3)==row,t2(:,end)==segNo),1:2);

    t1 = ccentersN;
    t2 = shapeprofN;

    [segNo, idx] = getSegIDNum(t1,move(4:5));
    [~, row] = isIncluded(t1(t1(:,end)==segNo,:), [move(4:5) segNo]);
    segfeat{1}.SF2 = shapefacN(idx,:);
    segfeat{1}.SP2 = t2(and(t2(:,3)==row,t2(:,end)==segNo),1:2);
    segfeat{1}.match = mismatch;

    % Combine the struct parameters into move features
    Xcomb = combineFeatures(segfeat,set.setsize);

    % Normalise the features
    Xnorm = bsxfun(@minus, Xcomb, mu);
    Xnorm = bsxfun(@rdivide, Xnorm, sigma);

%------------- END OF CODE --------------
```

combineFeatures.m

```
function f = combineFeatures(rf,maxImg)

% combineFeatures - Raw features (node coordinates, shape factors and shape
% profiles) of moves (node pairs) are combined into the features
% representing a move from one nephron cross-section to another. 67
% combined features are formulated:
% x1-x6:  The difference in the shape factors of area, eccentricity,
%         solidity, aspect ratio, minor axis and circularity
% x7-x12: The mean of the shape factors of area, eccentricity, solidity,
%         aspect ratio, minor axis and circularity
% x13:    The minimum area between the two cross-sections
% x14:    The Euclidean distance between the two nodes in the x-y plane
% x15:    The image difference
% x16:    The magnitude of image alignment offset
% x17:    The position of the pair (average z coordinate) relative to the
%         image set, which indicates depth into the kidney
% x18:    The correlation coefficient between the two shape profiles
% x19:    The correlation coefficient between the two sub-images
% x20-x43: Shape profile at 15 degree intervals of cross-section 1
% x44-x67: Shape profile at 15 degree intervals of cross-section 2
%                                         * Not in this order in the code
%
% Syntax:  featVector = combineFeatures(rawFeats,maxImg)
%
% Inputs:
%     rf        - A cell array of structs containing the raw features
%     maxImg    - The maximum image number in the set
%
% Outputs:
%     f      - An array of the normalised features where each row is an
%              example and each column is a feature
%
% Other m-files required:   none
```

```
% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

if nargin==1
    maxImg=700;
end

f = zeros(size(rf,1),11);
for i = 1:1:size(rf,1)

    f(i,1) = abs(rf{i}.SF1(1) - rf{i}.SF2(1));  %circularity
    f(i,2) = abs(rf{i}.SF1(2) - rf{i}.SF2(2));  %area
    f(i,3) = abs(rf{i}.SF1(3) - rf{i}.SF2(3));  %eccentricity
    f(i,4) = abs(rf{i}.SF1(4) - rf{i}.SF2(4));  %solidity/extent
    f(i,5) = abs(rf{i}.SF1(5) - rf{i}.SF2(5));  %aspect ratio

    f(i,6) = min([rf{i}.SF1(2) rf{i}.SF2(2)],[],2); %minimum Area

    % xy distance
    f(i,7) = sqrt(sum((rf{i}.coord1(1:2) - (rf{i}.coord2(1:2))).^2));

    % image difference
    f(i,8) = abs(rf{i}.coord1(3) - rf{i}.coord2(3));

    % image alignment offset
    f(i,9) = sqrt(sum((rf{i}.offset).^2));

    % metric from shape profile (temporary)
%     a = mf{i}.SP1(:,2);
%     tt=[];
%     for t=1:3, tt(end+1)=abs(corr([a(t:end); a(1:t-1)],mf{i}.SP2(:,2))); end
%     for t=22:24, tt(end+1)=abs(corr([a(t:end); a(1:t-1)],mf{i}.SP2(:,2))); end
%     f(i,10) = max(tt);
%     f(i,10) = (corr(mf{i}.SP1(:,2),mf{i}.SP2(:,2)));
    f(i,10) = sum(abs(rf{i}.SP1(:,2)-rf{i}.SP2(:,2))<3)/numel(rf{i}.SP1(:,2));

    % image position in z plane
    f(i,11) = 100.*((rf{i}.coord1(3)+rf{i}.coord2(3))/2)/maxImg;

    % minor axis length
    f(i,12) = abs(rf{i}.SF1(6) - rf{i}.SF2(6));

    f(i,13) = (rf{i}.SF1(6) + rf{i}.SF2(6))/2;  %minor axis length
    f(i,14) = (rf{i}.SF1(1) + rf{i}.SF2(1))/2;  %circularity
    f(i,15) = (rf{i}.SF1(2) + rf{i}.SF2(2))/2;  %area
    f(i,16) = (rf{i}.SF1(3) + rf{i}.SF2(3))/2;  %eccentricity
    f(i,17) = (rf{i}.SF1(4) + rf{i}.SF2(4))/2;  %solidity/extent
    f(i,18) = (rf{i}.SF1(5) + rf{i}.SF2(5))/2;  %aspect ratio

    f(i,19) = rf{i}.match;

    f(i,20:43) = rf{i}.SP1(:,2)';
    f(i,44:67) = rf{i}.SP2(:,2)';

end

f(isnan(f))=0;
```

```
%------------- END OF CODE --------------
```

manualAdjustClick2.m

```
% manualAdjustClick2 -  Reconstructs a path (ordered list of coordinates)
% from the interim closed list and uses the paths to get end-points which
% correspond to the dead ends of the tracking process. Asks the user to
% link these end-points to their correct nephron cross-section in images up
% to 3 before and after the end-point, through a simple click-and-capture
% interface. Uses these user corrected points as new seeds for another
% tracking instance. Re-initialises relevant variable/settings/lists as is
% required before another tracking instance can occur.

% Instructions: Re-run the section named '2. RUN TRACKING' in
% TrackerFinal.m after this script (and the manual intervention process)
% has finished.

% Other m-files required:   isIncluded.m
%                           findBranch2.m
%                           reconstructPath.m
%                           getEndPoints.m
%                           dist.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

%% 1. Use deadends to get endpoints
% Remove unnecesary dead ends
% size(deadend,1);
% tempdeadend = deadend;
% rem=[];
% for p=1:size(deadend,1)
%
%     [branch, tips, interim, ~] = findBranch2(ccenters{deadend(p,3)}, ...
%         deadend(p,1:3), [0 0 0 0 0 0],[0 0 0 0 0 0]);
%     if size(tips,1)==1 && isempty(interim)
%         rem(p) = 1;
%     elseif isIncluded(ignored, deadend(p,1:3))
%         rem(p) = 1;
%     elseif 0%isIncluded(skipBlock, deadend(p,1:3))
% %         isIncluded(misMatch, deadend(p,1:3)) ||...
% %        isIncluded(distMeas, deadend(p,1:3)) ||...
% %        isIncluded(biDirInv, deadend(p,1:3))
% %         rem(p) = 1;
%    else
%        rem(p)=0;
%    end
% end
% sum(rem)
% deadend(rem==1,:)=[];
% endPoints = deadend;

%% OR 2. Use endpoints of the reconstructed path
[fpath, ~] = reconstructPath(closed(1:end,:));
endPoints = getEndPoints(fpath)

%% Display each end point with 3 images before and after

potential = [];
for k=1:size(endPoints,1)
```

```
    coord = endPoints(k,1:3);
    for frame=1:7
        subplot(1,7,frame)
        displayCoord([coord(1) coord(2) coord(3)-4+frame], set);
        hold on
        scatter(coord(1),coord(2),'.','r')
        ind = find(closed(:,3)==(coord(3)-4+frame));
        scatter(closed(ind,1),closed(ind,2),'o','b')
        scatter(ccenters{coord(3)-4+frame}(:,1),ccenters{coord(3)-
4+frame}(:,2),'.','y')
        hold off
    end

[xx,yy] = ginput(1);
ax = input('axis? ');
% ax = ax.Children;

if ax~=0
    potential(k,:) = [xx yy endPoints(k,3)-4+ax endPoints(k,1:3)];
else
    % If labelled 0, it is not a real end point
    potential(k,:) = [0 0 0 0 0 0];
end

end

%% Get nodes closest to the locations clicked on by the user

potential(sum(potential,2)==0,:)=[];    % Remove incorrect end points
old = potential;
new = [];
for ll = 1:size(old,1)
    k1 = dist(ccenters{old(ll,3)}(:,1:2),old(ll,1:2));
    new(ll,1:2) = ccenters{old(ll,3)}(find(k1==min(k1),1),1:2);
end
new = [new old(:,3:6)]



%%  Re-initialise

% Store manual interventions
manualCorrec = [manualCorrec; new];

% Get new current node
open = new;
newCoord = find(open(:,3)==min(open(:,3)),1);
curr_coord = open(newCoord(1),:);%open(1,1:6);
ii = curr_coord(1,3);
open(newCoord(1),:)=[];
endPoints = [];

%  Re-initialise tracking varaiables
predictionU = 0;
predictionD = 0;

terminate = false;
up = false;     down = false;       branch = false;
up_coord = [];  down_coord = [];    branch_coord = [];

skip = false;   su=0;   sd=0;   prvs_sku=[0 0 0 0]; prvs_skd=[0 0 0 0]; sk=0;
up_buff = ones(1,8);   down_buff = ones(1,8);

mup = 0; mdn=0;
```

```
upMisErr = 0;
dwMisErr = 0;
have = [0 0 0];
img = zeros(set.imsize(1),set.imsize(2),3);

%------------- END OF CODE --------------
```

getEndPoints.m

```
function endpoints = getEndPoints(mpath)

% Obtains the end points of the recontructed paths created using
% reconstructPath.m. Only the end points of significant path fragments are
% returned (short fragments are ignored). The input (mpath) is a cell array
% of groups of coordinates of the various path fragments.

% See also: reconstructPath.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

endpoints=[];
for i = 1:size(mpath,2)
    if size(mpath{i},1)>=0.05*1*size(mpath{1},1)
        endpoints(end+1,:) = mpath{i}(1,:);
    end
end
```

Note: getSetProperties.m is included in the Pre-processing section

getSectionNo.m

```
function [setidx,offset] = getSectionNo(imgNo,imSet)

% getSectionNo - Management function to move through sets of shape profiles
% for 500 images at a time (to reduce memory/RAM requirements)

% Syntax:  [setidx,offset] = getSectionNo(imgNo,imSet)
%
% Inputs:
%    imgNo     - The image number in the set
%    imSet     - The number of the image set being used (1-6)
%
% Outputs:
%    setidx  - Index for batches of 500 images
%    offset  - Image number in the reduced set

% Other m-files required:   none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

if (imSet==4) || (imSet==5)
    if imgNo<501
        setidx = 1;
        offset = imgNo;
    elseif imgNo<1001
```

```
        setidx = 2;
        offset = imgNo-500;
    elseif imgNo<1501
        setidx = 3;
        offset = imgNo-1000;
    elseif imgNo<2001
        setidx = 4;
        offset = imgNo-1500;
    elseif imgNo<2501
        setidx = 5;
        offset = imgNo-2000;
    elseif imgNo<3001
        setidx = 6;
        offset = imgNo-2500;
    else
        setidx = 7;
        offset = imgNo-3000;
    end

else
    setidx = 1;
    offset = imgNo;
end

%------------- END OF CODE --------------
```

getShapeProfileCells.m

```
function shapeprof = getShapeProfileCells(set,OutVersion)

% getShapeProfileCells - Returns a cell array containing a matfile
% input-output struct to the files containing the shape profiles (SP) for a
% specific image set (set) and the version of the preprocessing and feature
% extraction output (OutVersion). The files were saved in groups, e.g. SP
% for images 1-500 in one file and SP for images 501 to 1000 in another
% file, so that reading the matfiles is quicker (reading one very large
% matfile is slow). The function getSectionNo.m then manages the switching
% from one file to another depending on the image number.

% Syntax:  shapeprof = getShapeProfileCells(set,OutVersion)
%          Then, shapeprof{n} should have 'data' and 'idx' field names
%                         where n = 1 ... size(shapeprof)

% Inputs:
%    set         - The image number of the set (1-5) as a numerical or string
%    OutVersion  - The preprocessing and feature extraction version number
%
% Outputs:
%    shapeprof  - A cell array of matfile io objects

% Other m-files required:   none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

if isnumeric(OutVersion)
    OutVersion = num2str(OutVersion);
end
v = OutVersion;
```

```
switch(set)

case {1,'1'}
shapeprof{1} = matfile(['dataOut/set1data' v '/SP1TO801.mat'],'Writable',true);

case {2,'2'}
shapeprof{1} = matfile(['dataOut/set2data' v '/SP1TO990.mat'],'Writable',true);

case {3,'3'}
shapeprof{1} = matfile(['dataOut/set3data' v '/SP1TO1000.mat'],'Writable',true);

case {4,'4'}
shapeprof{1} = matfile(['dataOut/set4data' v '/SP1TO500.mat'],'Writable',true);
shapeprof{2} = matfile(['dataOut/set4data' v '/SP501TO1000.mat'],'Writable',true);
shapeprof{3} = matfile(['dataOut/set4data' v '/SP1001TO1500.mat'],'Writable',true);
shapeprof{4} = matfile(['dataOut/set4data' v '/SP1501TO2000.mat'],'Writable',true);
shapeprof{5} = matfile(['dataOut/set4data' v '/SP2001TO2500.mat'],'Writable',true);

case {5,'5'}
shapeprof{1} = matfile(['dataOut/set5data' v '/SP1TO500.mat'],'Writable',true);
shapeprof{2} = matfile(['dataOut/set5data' v '/SP501TO1000.mat'],'Writable',true);
shapeprof{3} = matfile(['dataOut/set5data' v '/SP1001TO1500.mat'],'Writable',true);
shapeprof{4} = matfile(['dataOut/set5data' v '/SP1501TO2000.mat'],'Writable',true);

otherwise
    disp('Invalid set.')

end

%------------- END OF CODE --------------
```

getTrackingParams.m

```
function TRparams = getTrackingParams(imgNo,imSet)

% getTrackingParams - Returns a struct containing a number of variables
% related to a specific image (imgNo) in a specific image set (imSet). These
% parameters are tracking parameters which vary through the image set in a
% sigmoidal manner. The parameters for the 6 image sets have been tuned
% through the sigmoid function parameters.

% Syntax:  TRparams = getTrackingParams(imgNo,str)
%
% Inputs:
%    imgNo     - The image number in the set
%    imSet     - The number of the image set being used (1-6)
%
% Outputs:
%    TRparams  - A struct of the tracking variables for image number imgNo
%                and image set imSet

% Other m-files required:   custSigmoid.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------
```

```
TRparams = struct;

switch imSet
    case 1
        TRparams.trackRad  = custSigmoid(imgNo, 1, 15, 10, 250, 5);
        TRparams.maxOffset = custSigmoid(imgNo, 1, 40, 15, 350, 5);
        TRparams.areaLim   = custSigmoid(imgNo, 1, 8, 1, 300, 5);
        TRparams.skipArea  = custSigmoid(imgNo, 1, 10, 1, 300, 1);

    case 2
        TRparams.trackRad  = custSigmoid(imgNo, 1, 20, 15, 300, 5);
        TRparams.maxOffset = custSigmoid(imgNo, 1, 35, 15, 300, 5);
        TRparams.areaLim   = custSigmoid(imgNo, 1, 8, 1, 300, 5);
        TRparams.skipArea  = custSigmoid(imgNo, 1, 10, 1, 350, 1);

    case 3
        TRparams.trackRad  = custSigmoid(imgNo, 1, 20, 15, 300, 5);
        TRparams.maxOffset = custSigmoid(imgNo, 1, 40, 15, 350, 5);
        TRparams.areaLim   = custSigmoid(imgNo, 1, 8, 1, 300, 5);
        TRparams.skipArea  = custSigmoid(imgNo, 1, 10, 1, 300, 1);

    case 4
        TRparams.trackRad  = custSigmoid(imgNo, 1, 20, 10, 1000, 4);
        TRparams.maxOffset = custSigmoid(imgNo, 1, 80, 50, 1300, 5);
        TRparams.areaLim   = custSigmoid(imgNo, 1, 12, 6, 1300, 5);
        TRparams.skipArea  = custSigmoid(imgNo, 1, 12, 1, 1100, 1);

    case 5
        TRparams.trackRad  = custSigmoid(imgNo, 1, 20, 10, 1000, 4);
        TRparams.maxOffset = custSigmoid(imgNo, 1, 80, 50, 1300, 5);
        TRparams.areaLim   = custSigmoid(imgNo, 1, 12, 6, 1300, 5);
        TRparams.skipArea  = custSigmoid(imgNo, 1, 12, 1, 1100, 1);

    case 6
        TRparams.trackRad  = custSigmoid(imgNo, 1, 20, 10, 1000, 4);
        TRparams.maxOffset = custSigmoid(imgNo, 1, 80, 50, 1300, 5);
        TRparams.areaLim   = custSigmoid(imgNo, 1, 12, 6, 1300, 5);
        TRparams.skipArea  = custSigmoid(imgNo, 1, 12, 1, 1100, 1);

    otherwise
        disp('Invalid set.')
end


%------------- END OF CODE --------------
```

Plotting & Analysis Functions

Plotting_and_Analysis_Tools.m

```
% Plotting and Analysis Tools
% *Uses data from execution of TrackerFinal.m

% M-files used: tubeplot.m
%               tubeplot1.m
```

```
%               getSetProperties.m
%               getShapeProfileCells.m
%               custSigmoid.m
%               comparePaths2.m
%               getShapeProfile.m
%               getSegIDNum.m
%               displayCoord.m
%               displayMove.m
%               array2struct_trackingData.m
% DATA required: Workspace data from a tracking instance of TrackerFinal.m
%                OR data loaded from saved results

% >>>>>>>>>>>>>>>>>>>>>>> INSTRUCTIONS <<<<<<<<<<<<<<<<<<<<<<<<<

% 1. Run the first section 'LOAD SAVED RESULT' if tracking had not occured
% 2. Only modify parameters above the '^^^^^^^^'line in each section
% 3. Run one section at a time (CTRL+ENTER). Each section runs indepedantly

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 20-Mar-2015

%% ========================================================================
% ------------------------ LOAD SAVED RESULT ---------------------------
% ========================================================================

% Choose data set
imset = 1;
version = 5;

% Uncomment the result to load
% load('other\results\2510\set3neph3FULL.mat')
load('other\results\2510\set1neph75FULL.mat')

% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
s = num2str(imset); v = num2str(version);
set = getSetProperties(s,v);
load(['dataOut\set' s 'data' v '\set' s 'feat' v '.mat'])
shapeprof = getShapeProfileCells(s,v);
MOVES = array2struct_trackingData(capMoves);

%% ========================================================================
% ------------------------ SAVE TRACKING DATA ---------------------------
% ========================================================================

note  = 'set1data5 neph61 till atl; 21 corrections';
save('set1neph59TillATL.mat','misMatch','capMoves','open','closed','fpath',...
     'tr','ss','skipBlock','skipAllow','biDirInv','distMeas','ignored', ...
     'net','note','manualCorrec')

%% ========================================================================
% ------------------ COMPARE TO LABELLED IMAGE SET ----------------------
% ========================================================================

im  = 105;
X = 12.5*100;
Y = 9.5*100;
zoom = 2*100;
highlight = closed(:,1:3);

% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ind = find(highlight(:,3)==im);
I = imread([set.imOutPath num2str(im) '.jpg'])>100;
fill = round(sub2ind(size(I),highlight(ind,2),highlight(ind,1)));
```

```
subplot 121
imagesc(~imfill(~I,fill)+0.7*I);
hold on, colormap hot, axis equal
if exist('ccenters','var'), ...
        scatter(ccenters{im}(:,1),ccenters{im}(:,2),'.','b'), end
scatter(highlight(ind,1),highlight(ind,2),'.','g')
title(num2str((im))), hold off, axis([X-zoom X+zoom Y-zoom Y+zoom])
im_num = [];
for i1 = 1:1:set.range-numel(num2str(im+set.offset))
    im_num = [im_num '0'];
end

subplot 122
imagesc(imread([set.imLabPath im_num num2str(im+set.offset) '.jpg']));
hold on, scatter(highlight(ind,1),highlight(ind,2),'.','g'),
% scatter(ccenters{im}(:,1),ccenters{im}(:,2),'.','b'),
hold off
axis equal, axis([X-zoom X+zoom Y-zoom Y+zoom])

%% ========================================================================
% ----- DISPLAY ALL NODES ON SELECTED IMAGE WITH NEPHRON HIGHLIGHTED -----
% ========================================================================

im = 60;
highlight = closed(:,1:3);

% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
I = imread([set.imOutPath num2str(im) '.jpg'])>100;
ind = find(highlight(:,3)==im);
fill = round(sub2ind(size(I),highlight(ind,2),highlight(ind,1)));
imagesc(~imfill(~I,fill)+0.6.*I);
hold on, colormap hot
scatter(ccenters{im}(:,1),ccenters{im}(:,2),'.','b')
title(num2str((im))), hold off

%% ========================================================================
% ------------------------- PLOT 3D TUBE VIEW ----------------------------
% ========================================================================

 clf
% figure
 path = fpath{1};
 smoothing = 10;

% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
xx = 1.*smooth(path(1:end,1),smoothing)';
yy = 1.*smooth(path(1:end,2),smoothing)';
zz = 1*smooth(path(1:end,3),1)';
radius = 1.5;
colVec = (size(path,1):-1:1);
% tubeplot1([xx;yy;zz],radius,8);
% saveobjtube('neph412.obj',xx',yy',zz',3,1,6)
tubeplot(xx,yy,zz,radius,colVec);

% plot3(xx,yy,zz,'g')
shading flat
colormap jet
grid on;
hold on
axis auto
view([-5 -10 3]);
scatter3(manualCorrec(:,1),manualCorrec(:,2),manualCorrec(:,3),'k*')
hold off
```

```
%% ========================================================================
% ------------------ PLOT MANUALLY TRACKED NEPHRON ----------------------
% ========================================================================

%     set1 --> mouse1
%     set2 --> mouse3
%     set3 --> mouse4
%     set4 --> rat5
%     set5 --> rat8
%     set6 --> rat4 (excluded from tracking results)

% figure
load 'other/Manual Data/complete/mouse1.mat'
nephNo = 75;     % Not every consecutive number may exist
offset_start = 0;
offset_end   = 0;
radius = 1.5;
smoothing = 20;

% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
x = getfield(nef,['num' num2str(nephNo)]);
x(:,1) = 1*smooth(x(:,1),smoothing);
x(:,2) = 1.*smooth(x(:,2),smoothing);
x(:,3) = 1.*smooth(x(:,3)-0,1);

m = (offset_start/100)*size(x,1)+1;
n = size(x,1)-(offset_end/100)*size(x,1);
tubeplot(x(m:n,1),x(m:n,2),x(m:n,3),radius,1:n-m+1)
grid on, axis auto, shading flat, colormap jet
view([1 1 0.5]);

%% ========================================================================
% -------- MEASURE SIMILARITY TO MANUALLY TRACKED NEPHRON ----------------
% ========================================================================

load 'other/Manual Data/complete/mouse1.mat'
nephNo = 75;     % Not every consecutive number may exist
offset_start = 0;
offset_end   = 0;
radius = 1.5;
smoothing = 20;
compareTo = closed(:,1:3);

% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
k = custSigmoid(1:5000, -1, 20, 30, 1300, -10);
x = getfield(nef,['num' num2str(nephNo)]);
y = compareTo(:,1:3);
y(:,3) = (y(:,3)+set.offset);
[alpha, r1] = comparePaths2(x,y,k);
[ beta, r2] = comparePaths2(y,x,k);
fprintf(['\nAlpha: ' num2str(alpha) ' %%\n'])
fprintf(['Beta: ' num2str(beta) ' %%\n\n'])

%% >>>>>>>>>>>>>>>>>>>>>> VIEW MOVE DATA <<<<<<<<<<<<<<<<<<<<<<<<<<<<
%% ========================================================================
% ------------------ Get shape factors of a move ------------------------
% ========================================================================

clc
% close all
% figure(1)
n = 43;
coord = capMoves(n,1:6);
```

```
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
[~, id1] = getSegIDNum(ccenters{coord(3)},coord(1:3));
[~, id2] = getSegIDNum(ccenters{coord(6)},coord(4:6));
fields = ['Circularity  '; 'Area         '; 'Eccentricity '; ...
          'Solidity     '; 'AspectRatio  '; 'MinorAxLen   ' ];
[fields num2str([ shapefac{coord(3)}(id1,:)' shapefac{coord(6)}(id2,:)'],3)]

%% ========================================================================
% ----------- Show moves from child-parent node array --------------------
% ========================================================================

n = 43;
coord = capMoves(n,:);

% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
displayMovedisplayMove(coord(1:3), coord(4:6),set);
coord(7:end);

%% ========================================================================
% --------------- Look at shape profile of a move ------------------------
% ========================================================================

%  figure(2)
n = 23;
coord = capMoves(n,1:6);

% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
[ang1,sp1]=getShapeProfile(coord(1:3), ccenters{coord(3)},...
    shapeprof{1}.data(SPidx(coord(3)):SPidx(coord(3)+1)-1,:) ,1);
hold on
[ang2,sp2]=getShapeProfile(coord(4:6), ccenters{coord(6)},...
    shapeprof{1}.data(SPidx(coord(6)):SPidx(coord(6)+1)-1,:) ,1,[0 0]);
scatter(0,0,'xr')
hold off
grid on
% Compare various correlation metrics
match(1) = corr(sp1,sp2);
tt=[];
for t=1:3, tt(end+1)=abs(corr([sp1(t:end); sp1(1:t-1)],sp2)); end
for t=22:24, tt(end+1)=abs(corr([sp1(t:end); sp1(1:t-1)],sp2)); end
match(2) = max(tt);
match(3) = 1*sum(abs(sp2-sp1)<2)/numel(sp1);
match(4) = 1/abs(sqrt(prod(sp2-sp1)));
match(5) = 1-0.01.*sum(abs(sp2-sp1)./mean([sp1; sp2]))


%% >>>>>>>>>>>>>>>>>>>>>> VIEW NODE DATA <<<<<<<<<<<<<<<<<<<<<<<<<<<
%% ========================================================================
% ------------------------- Display a node -------------------------------
% ========================================================================

n = 23;
select = capMoves(n,:);

% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
displayCoord(select(1:3),set);
select(4:end)'

%% ========================================================================
% ----------------- Get shape factors of a node ------------------------
% ========================================================================
```

```
n = 23;
coord = capMoves(n,1:3);
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
[~, id] = getSegIDNum(ccenters{coord(3)},coord);
fields = ['Circularity  '; 'Area         '; 'Eccentricity '; ...
          'Solidity     '; 'AspectRatio  '; 'MinorAxLen   ' ]
[fields num2str( shapefac{coord(3)}(id1,:)',3.) ]



%% ========================================================================
% ****************** FOR EASE OF VIEWING OTHER MATRICES ******************
% ========================================================================
%%
a=1;
displayMove(skipBlock(a,1:3), skipBlock(a,4:6),set);
skipBlock(a,end)

%%
a=1;
displayMove(skipAllow(a,1:3), skipAllow(a,4:6),set);
skipAllow(a,end-1)

%%
a=1;
displayMove(biDirInv(a,1:3), biDirInv(a,4:6),set);

%%
a=5;
displayMove(distMeas(a,1:3), distMeas(a,4:6),set);
distMeas(a,7:end)

%%
a=10;
displayMove(misMatch(a,1:3), misMatch(a,4:6),set);
misMatch(a,7:end)'

%%
a=1;
displayMove(closed(a,1:3), closed(a,4:6),set);

%%
a=1;
displayMove(misAligned(a,1:3), misAligned(a,4:6),set);
misAligned(a,7:end)

%%
a=3;
subplot 131, displayCoord(manualCorrec(a,4:6), set);
axis off
subplot 132, displayCoord([manualCorrec(a,4:5) ...
    round((manualCorrec(a,6)+manualCorrec(a,3))/2)], set);
axis off
subplot 133, displayCoord(manualCorrec(a,1:3), set);
axis off
```

displayCoord.m

```
function [dummy] = displayCoord(COORD, SET, W)

% displayCoord - A custom utility for displaying a sub-image of the area
% around COORD using the width W. COORD must be a 3D coordinate (x,y,z)
% where z denotes the image number in the set. The struct SET of the image
```

```
% set properties must be provided in order to obtain the image paths and
% properties (as is created by the function getSetProperties). The image
% shown is a combination of the original colour image and the binary image.
% The image number is displayed as the figure title.

% Syntax:  displayCoord(COORD, SET, W)
%
% Inputs:
%    COORD  - The point around which the image must be shown
%    SET    - The struct of image set properties
%    W      - The required half-width around COORD (optional; default = 50)
%
% Outputs: none to the command window; diplayed figure
%
% Example:
% displayCoord([250 321 85], getSetProperties(1,5), 80)
%     Will display image number 85 from image set 1, version 5. A 160x160
%     pixel area around the point [250 321] will be shown. The colour image
%     with a transparent version of the black and white image will be
%     shown.

% Supporting m-files: getSetProperties.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

dummy=[];

if nargin<3
    W = 50;
end

im_num = [];
for i1 = 1:1:SET.range-size(num2str(COORD(3)+SET.offset),2)
    im_num = [im_num '0'];
end

colIm = (imread([SET.imInPath im_num num2str(COORD(3)+SET.offset) '.jpg']));
bwIm = 50*uint8(imread([SET.imOutPath num2str(COORD(3)) '.jpg'])>200);
temp = cat(3,bwIm,bwIm);
bwIm = cat(3,temp,bwIm);
imagesc(colIm+bwIm);
axis([COORD(1)-W COORD(1)+W COORD(2)-W COORD(2)+W])
axis square
title(num2str(COORD(3)))
hold on
scatter(COORD(1),COORD(2),'*','g')
hold off

%------------- END OF CODE --------------
```

displayMove.m

```
function [dummy] = displayMove(coord1, coord2, set, W)

% displayMove - A custom utility for displaying two sub-images
% side-by-side, as is required to display a move of a nephron from one
% image to another. The sub-images are of the area around coord1 and coord2
% using the width W. coord1 and coord2 must be 3D coordinates of (x,y,z)
% where z denotes the image number in the set. The struct SET of the image
```

```
% set properties must be provided in order to obtain the image paths and
% properties (as is created by the function getSetProperties).

% Syntax:  displayMove(coord1, coord2, set, W)
%
% Inputs:
%    coord1     - The point around which the first image must be shown
%    coord2     - The point around which the second image must be shown
%    SET        - The struct of image set properties
%    W          - The required half-width around COORD (optional; default = 50)
%
% Outputs: none to the command window; diplayed figure
%
% Example:
% displayMove([120 359 62],[122 354 63], getSetProperties(2,3), 60)
%     Will display image numbers 62 and 63 from image set 2, version 3
%     side-by-side. A 120x120 pixel area around each of the points will be
%     shown.

% Other m-files required:   displayCoord.m
% Supporting m-files:       getSetProperties.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 17-Mar-2015

%------------- START OF CODE --------------

dummy=[];

if nargin<4
    W = 50;
end

subplot(1,2,1)
displayCoord(coord1, set);
axis([coord1(1)-W coord1(1)+W coord1(2)-W coord1(2)+W])
axis off
subplot(1,2,2)
displayCoord(coord2, set);
axis([coord1(1)-W coord1(1)+W coord1(2)-W coord1(2)+W])
axis off

%------------- END OF CODE --------------
```

comparePaths2.m

```
function [metric, residual] = comparePaths2(x,y, tol)

% comparePaths2 - Calculates the residual, or difference between two sets
% of coordinates representing paths in 3D space (nephron trajectories). The
% residual is calculated as the minimum Euclidean distance between each
% point in y to the path x (the residual is a vector of the size of y). The
% similairty metric is then a threshold applied to the residual using the
% provided tolerance value/s.

% Syntax:  [metric, residual] = comparePaths2(x,y, tol)
%
% Inputs:
%    x      - A Mx3 matrix of coordinates of the first path.
%    y      - A Nx3 matrix of coordinates of the second path.
%    tol    - The tolerance or threshold (Euclidean distance) used to
%             calculate the similarity metric. This can be a single value
```

```
%             or a vector with N elements.
%
% Outputs:
%    metric     - The similarity of y to x (%)
%    residual   - The 1xN vector of residuals of y with respect to x

% Other m-files required:   none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% February 2015; Last revision: 12-Feb-2015

%------------- START OF CODE --------------

if nargin==2
    tol = 10;
end

residual=100.*ones(1,size(y,1));    % Preallocate a vector

%Compare each elemnt in y to coordinates in relevant image in x
for i=1:1:size(y,1)

    % Get coordinates in images i, i+1 and i-1 in x
    t1 = or(or(x(:,3)==y(i,3)-1,x(:,3)==y(i,3)+1),x(:,3)==y(i,3));
    %t1 = x(:,3)==y(i,3);
    xi = x(t1,1:3);

    % Calculate Euclidean distances to those coordinates from y(i)
    dis = dist(xi,y(i,1:3));

    % Residual is the minumum distance (sum of square difference)
    if ~isempty(dis)
        residual(i) = min(dis);
    end

end

% The metric is a threshold of the residual
 metric = 100.*sum(residual<tol(y(:,3)))./size(y, 1);

%------------- END OF CODE --------------
```

getShapeProfile.m

```
function [ang, dis] = getShapeProfile(coord, centers, SPmat, display,off)

% getShapeProfile - This function is used to obtain the shape profile (ang and
% dis) related to a given node (coord) from the nodes and shape profile
% matrices (centers and SPmat). There is an option for automatically
% plotting the extracted profile (if display=true) as well as applying an
% x-y offset (off) to the profile. This function is used purely for display
% and analysis pre- or post-tracking.
%
% Inputs:
%    coord      - The node at which the shape profile is desired.
%    centers    - The array of nodes on the current image.
%    SPmat      - The shape profile array for the current image in which
%                 the queried shape profile is contained. This is a Mx4
%                 array created during the feature extraction stage by the
%                 extractFeatures6 function.
%    display    - A flag to plot the shape profile (1) or not (0)
```

```
%    off        - The (optional) x-y offset in a 2 element array
%
% Outputs:
%   ang         - The angles (degrees) of the extracted shape profile
%   dis         - The radii related to ang
%   diplayed figure

% Other m-files required:   getSegIDNum.m
%                           isIncluded.m

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% February 2015; Last revision: 12-Feb-2015

%------------- START OF CODE --------------

   if nargin==4
       off=[0 0];
   end

   [segNo, ~] = getSegIDNum(centers,coord(1:2));
   [~, row] = isIncluded(centers(centers(:,3)==segNo,:), [coord(1:2) segNo]);
   SP = SPmat(and(SPmat(:,3)==row,SPmat(:,4)==segNo),1:2);
   ang = SP(:,1);
   dis = SP(:,2);

   if nargin==5

       x = (dis.*cosd(ang))-off(1);
       y = (dis.*sind(ang))-off(2);

       ang = atan2d(y,x);
       dis = sqrt(x.^2+y.^2);
   end

   if display==1
       plot(dis.*sind(ang),-dis.*cosd(ang),'-b.')
   end

%------------- END OF CODE --------------
```

array2struct_trackingData.m

```
function struc = array2struct_trackingData(arr)

% extractFeatures6 - Extracts nodes, shape factors and shape profiles for
% each component in a binary image. This function is custom-coded for binary
% images of kidney cross-sections.
%
% Syntax:  S = array2struct_trackingData(A)
%
% Inputs:
%    arr     - The input array (capMoves from TrackerFinal.m)
% Outputs:
%    struc   - The parsed output structure
%
% Other m-files required: none

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 22-Mar-2015
```

```
%------------- START OF CODE --------------

if nargin==0
    arr = capMoves;
end

struc = cell(size(arr,1),1);
for i = 1:size(arr,1)



   ss.coord1 = arr(i,1:3);
   ss.coord2 = arr(i,4:6);
   ss.offset = arr(i,7:8);
   ss.ML_abn = arr(i,9);
   ss.ML_glom = arr(i,10);
   ss.ML_elong = arr(i,11);
   ss.ML_norm = arr(i,12);
   ss.ML_innmed = arr(i,13);
   ss.mismatch = arr(i,14);
   ss.childArea     = arr(i,15);
   ss.distMetric    = arr(i,16);
   ss.changeMetric  = arr(i,17);
   ss.BidirecFail   = arr(i,18);
   ss.MLfail        = arr(i,19);

   struc{i} = ss;

end

%------------- END OF CODE --------------
```

sortCell.m

```
function out = sortCell(in)

% Sorts a cell array according to size of the contents of the cells.

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 22-Mar-2015

%------------- START OF CODE --------------
for ib=1:1:size(in,2)
    for ia=1:1:size(in,2)-1
        if size(in{ia},1)<size(in{ia+1},1)
            temp = in{ia};
            in{ia} = in{ia+1};
            in{ia+1} = temp;
        end
    end
end
out = in;
%------------- END OF CODE --------------
```

Note: Functions tubeplot.m, frenet.m, frame.m, tubeplot1.m and saveobjtube.m are used. These functions are open source plotting tools available online.

Glomeruli Detection

GlomeruliDetection.m

```
%% %%%%%%%%%%%%%%%%%%%%%% GLOMERULI DETECTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This is the main script for running glomeruli detection using the devised
% methodology.

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 12-Mar-2015

% Other m-files required:     getSetProperties.m
%                             getProcessingParams.m
%                                 custSigmoid.m
%                             ProcessImgStd.m
%                             detectGlomeruli.m
%                                 predictCluster.m
%                                 classifyImgSegments.m
%                                     Image Processing Toolbox
%                                     Statistics Toolbox
%                                     featureNormalize.m
%                                     featureUnnormalize.m
%                                     predictCluster.m

% Instructions:
%     1. Set im to the image number
%     2. Set the parameter set to the image set being used
%     3. Run the first section of code (CTRL + ENTER); max. 30 seconds
%     4. Set the 4 parameters for glomeruli detection. These affect the
%        sensitivity of detection
%     5. Run the second section of code (CTRL + ENTER); max. 15 seconds

%------------- START OF CODE --------------

%% 1. Acquire binary image from colour image

im = 7;
set = 'Test';

% ---------------------------------------------------------
setprops = getSetProperties(set);
params   = getProcessingParams(set,im);
img  = imread([setprops.imInPath num2str(im) '.jpg'], 'jpg');
imin = ProcessImgStd(img,params);

%% 2. Run glomeruli detection

% Choose parameters for glomeruli detection
glom_rad = 50;
peak_th  = 0.75;
clus_lim = 400;
display  = 1;

% ---------------------------------------------------------
[imGlom, G1, G2] = detectGlomeruli(imin, glom_rad, peak_th, ...
                                          clus_lim, display);

%------------- END OF CODE --------------
```

detectGlomeruli.m

```
function [imGlom, G1, G2] = detectGlomeruli(imin, glom_rad, peak_th, ...
                                            clus_lim, display)

% detectGlomeruli - Glomeruli are detected by means of a density plot
% created using the edges of the equalised image in imin. High density
% occurs at glomeruli locations. Regions of high density are detected and
% clustered into points through a custom method.
%
% A second method which clusteres and classifies segments based on their
% shape is applied, and used to valid the results of the first method (see
% classifyImgSegments.m).
%
% Syntax:  [imGlom, G1, G2] = detectGlomeruli(imin, glom_rad, peak_th, ...
%                                             clus_lim, display)
%          [imGlom, G1, G2] = detectGlomeruli(imin)
%
% Inputs:
%    imin       - The mxnx6 matrix of input images created using
%                 ProcessImgStd.m
%    glom_rad   - The average radius of the glomeruli to be detected
%                 (default = 50)
%    peak_th    - The threshold to apply to the density function; must
%                 be a value from 0-1 (default = 0.75)
%    clus_lim   - The minimum number of points in the density point
%                 cloud that must be present within glom_rad to form a
%                 valid cluster (default = 400)
%    display    - An option for display, 1 = display desired (default = true)
%
% Outputs:
%    imGlom     - The output image which is the original image with the
%                 potential glomeruli segments highlighted
%    G1         - Glomeruli detections of method 1
%    G2         - Glomeruli detections of method 1 validated by method 2
%
% Other m-files required:   classifyImgSegments
%                           predictCluster
%
% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 28-Mar-2015

%------------- START OF CODE --------------

if nargin==1
    glom_rad = 50;
    peak_th = 0.75;
    clus_lim = 400;
    display = 1;
end

% >>>>>>>>>>>>>>>>>>>>>> METHOD 1 <<<<<<<<<<<<<<<<<<<<<<<<

% Find edge image
I = double(edge(imin(:,:,3),'log'));
% imagesc(1000.*I+double(proc(:,:,3)))

% Apply averaging filter
% H = fspecial('average',glom_rad);
% kkk = imfilter(I,H);
% kkk = kkk./(max(max(kkk)));
% III = kkk;
```

```
win = glom_rad;
kern = 1/win.*ones(win,win);
kern(round(win*win/2)) = 1;
III = conv2(double(I),kern);
III = III(win/2:end-(win/2),win/2:end-(win/2));
III = III./(max(max(III)));

% Isolate peaks
aa = III>peak_th;

% Get coordinates of 1's
[v,u] = ind2sub(size(aa), find(aa==1));
step = 1;
x = [v(1:step:end),u(1:step:end)];
x = double(x);

% Cluster through custom method (hierachical)
centroid = [];
k = 1;
while ~isempty(x)

    rndidx = round(1 + (size(x,1)-1).*rand);
    y = sqrt((x(:,1)-x(rndidx,1)).^2 + (x(:,2)-x(rndidx,2)).^2);
    pc = find(y<glom_rad);
    if ~isempty(pc)
        if numel(pc)<clus_lim, x(pc,:)=[];
        else
            c1 = x(pc,:);
            x(pc,:)=[];
            % clus{end+1} = c1;
            confidence = double(numel(pc))./double(clus_lim);
            centroid(end+1,:) = [round([mean(c1(:,1)) mean(c1(:,2))]) confidence];
        end
    end
    k = k+1;
end

% >>>>>>>>>>>>>>>>>>>>>> METHOD 2 <<<<<<<<<<<<<<<<<<<<<<<<
% Use kmeans clustering on binary segments, then obtain the segments
% classified to the centroid that most represents glomeruli. Use these to
% verify the detections of method 1.

imlab = bwlabel(imin(:,:,6)>180,4);
[imtag,C] = classifyImgSegments(imlab, 10);
% Glomeruli centroid obtained through experimemtation
p = predictCluster([860  0.58  3.3  0.91 0.54],C);

clusdata = uint8(imtag==p(1));
[v,u] = ind2sub(size(clusdata), find(clusdata==1));
cx = [v(1:1:end),u(1:1:end)];
cx = double(cx);

th = 1*100;
val = zeros(size(centroid,1),1);
for i=1:1:size(centroid,1)

   if sum(((cx(:,1)-centroid(i,1)).^2+(cx(:,2)-centroid(i,2)).^2)<glom_rad^2)<th
...
           && centroid(i,3)<=5
       val(i) = 0;
   else
       val(i) = 1;
   end
```

```
end

val = logical([val' val']);
CC = centroid(val);
CC = reshape(CC,[],2);

imGlom =0.1.*imin(:,:,1)+10.*clusdata;
G1 = centroid;
G2 = CC;

% Display
if display
    imagesc(imGlom)
    colormap gray
    hold on
    scatter(G2(:,2),G2(:,1),'.','r')
    scatter(G1(:,2),G1(:,1),'o','g')
    hold off
    axis equal
end

%------------- END OF CODE --------------
```

predictCluster.m

```
function p = predictCluster(xtest, centroids)

% Given an array of centroids and test points, this function return
% the index of the centroid that is closest to each of the test points.
% The centroids are obtained using some clustering algorithm on a number of
% example points.

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 28-Mar-2015

    for i=1:size(xtest,1)
        dist = sum((bsxfun(@minus,centroids,xtest(i,:))).^2,2);
        [~, p(i)] = min(dist, [],1);
    end
end
```

classifyImgSegments.m

```
function [imtag, C] = classifyImgSegments(imin, K, ownC)

% classifyImgSegments - This function performs classification of binary
% image components based on five shape factors of area, solidity,
% aspectRatio, eccentricity and circularity. These are used as features per
% component. The components' features are then clustered using the K-means
% clustering method. Classification is then based on the nearest cluster
% centroid. The binary image components are then labelled according to the
% classification.
%
% Syntax:  [imtag, C] = classifyImgSegments(imin, K)
%          [imtag, C] = classifyImgSegments(imin, K, ownC)
%
% Inputs:
%    imin           - The input binary image
%    K              - The number of cluster centroids to be used
%    ownC           - (optional) An array of user-supplied cluster centroids
%                     which bypasses kmeans clustering
```

```
% Outputs:
%    imtag          - The output image tagged by cluster
%    C              - The centroids of the clusters formed
%
% Other m-files required:   Image Processing Toolbox (bwlabel, regionprops)
%                           Statistics Toolbox (kmeans)
%                           featureNormalize
%                           featureUnnormalize
%                           predictCluster
%
% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 12-Mar-2015

%------------- START OF CODE --------------

% Obtain shape factors
imin = imin>180;
stats = regionprops(logical(imin), 'Area', 'Perimeter',  ...
    'PixelIdxList',...
    'Eccentricity', 'MajorAxisLength',...
    'EquivDiameter', 'MinorAxisLength','BoundingBox');

for i=1:1:size(stats,1)
    area(i) = stats(i).Area;
%     convexArea(i) = stats(i).ConvexArea;
    perimeter(i) = stats(i).Perimeter;
    equivDiameter(i) = stats(i).EquivDiameter;
    majorAxisLength(i) = stats(i).MajorAxisLength;
    minorAxisLength(i) = stats(i).MinorAxisLength;
%     extent(i) = stats(i).Extent;
    eccentricity(i) = stats(i).Eccentricity;

    % Alternate solidity measure to increase speed
    tm = round(stats(i).BoundingBox);
    boxSize(i) = tm(3)*tm(4);
    solidity(i) = (stats(i).Area)./boxSize(i) + 0.21;
    if solidity(i)>1, solidity(i)=1; end

    pixelIdxList{i} =  stats(i).PixelIdxList;
end
aspectRatio = majorAxisLength./minorAxisLength;
circularity = 1./(perimeter./(pi.*equivDiameter));
    circularity(circularity==inf) = 0;

% Define Features to use
X = [area' solidity' aspectRatio' eccentricity' circularity' ];

if nargin==2
    % Run Kmeans to find hidden structure
    % max_iters = 500;
    [X_norm, mu, sigma] = featureNormalize(X);
    [~, centroids] = kmeans(X_norm, K,'EmptyAction','drop');
    C = featureUnnormalize(centroids, mu, sigma);
else
    C = ownC;
    K = size(C,1);
end

% Classify each segment based on centroids
class = zeros(size(X,1),1);
for i=1:1:size(X,1)
    xtest = X(i,:);
    class(i) = predictCluster(xtest, C);
end
```

```
%Produce image of classified segments
imtag = bwlabel(imin);
imtag(imtag>0)=1;

for tag=1:1:K
    nn=find(class==tag);
    idxs=[];
    for k=1:1:numel(nn)
        idxs = [idxs ;pixelIdxList{nn(k)}];
    end
    imtag(idxs) = tag;
end

%------------- END OF CODE --------------
```

featureNormalize.m

```
function [X_norm, mu, sigma] = featureNormalize(X)

%FEATURENORMALIZE Normalizes the features in X
%   FEATURENORMALIZE(X) returns a normalized version of X where
%   the mean value of each feature is 0 and the standard deviation
%   is 1. This is often a good preprocessing step to do when
%   working with learning algorithms.

% Andrew NG
% Coursera Machine Learning Course
% https://www.coursera.org/course/ml

mu = mean(X,1);
X_norm = bsxfun(@minus, X, mu);

sigma = std(X_norm,1);
X_norm = bsxfun(@rdivide, X_norm, sigma);

end
```

featureUnnormalize.m

```
function [XX] = featureUnnormalize(X_norm, mu, sigma)

% featureUnnormalize - Inverses the normalisation procedure that had
% occured on X_norm with a mean of mu and standard deviation of sigma.

% Author: Charita Bhikha
% email address: charita.bhikha@gmail.com
% March 2015; Last revision: 28-Mar-2015

XX = bsxfun(@times, X_norm, sigma);
XX = bsxfun(@plus, XX, mu);

end
```

Note: getSetProperties.m, getProcessingParams.m, custSigmoid.m and ProcessImgStd.m can be found under other sections

Hindawi Publishing Corporation
Computational and Mathematical Methods in Medicine
Volume 2015, Article ID 545809, 13 pages
http://dx.doi.org/10.1155/2015/545809

Research Article

Towards Automated Three-Dimensional Tracking of Nephrons through Stacked Histological Image Sets

Charita Bhikha,[1] Arne Andreasen,[2] Erik I. Christensen,[2] Robyn F. R. Letts,[1] Adam Pantanowitz,[1] David M. Rubin,[1] Jesper S. Thomsen,[2] and Xiao-Yue Zhai[3]

[1]*Biomedical Engineering Research Group, School of Electrical & Information Engineering, University of the Witwatersrand Johannesburg, Private Bag 3, Johannesburg 2050, South Africa*
[2]*Department of Biomedicine, University of Aarhus, 8000 Aarhus C, Denmark*
[3]*Department of Histology and Embryology, China Medical University, Shenyang, Liaoning 110122, China*

Correspondence should be addressed to Charita Bhikha; charita.bhikha@gmail.com

Received 26 February 2015; Revised 16 May 2015; Accepted 28 May 2015

Academic Editor: Giancarlo Ferrigno

Copyright © 2015 Charita Bhikha et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

An automated approach for tracking individual nephrons through three-dimensional histological image sets of mouse and rat kidneys is presented. In a previous study, the available images were tracked manually through the image sets in order to explore renal microarchitecture. The purpose of the current research is to reduce the time and effort required to manually trace nephrons by creating an automated, intelligent system as a standard tool for such datasets. The algorithm is robust enough to isolate closely packed nephrons and track their convoluted paths despite a number of nonideal, interfering conditions such as local image distortions, artefacts, and interstitial tissue interference. The system comprises image preprocessing, feature extraction, and a custom graph-based tracking algorithm, which is validated by a rule base and a machine learning algorithm. A study of a selection of automatically tracked nephrons, when compared with manual tracking, yields a 95% tracking accuracy for structures in the cortex, while those in the medulla have lower accuracy due to narrower diameter and higher density. Limited manual intervention is introduced to improve tracking, enabling full nephron paths to be obtained with an average of 17 manual corrections per mouse nephron and 58 manual corrections per rat nephron.

1. Introduction

The kidney performs the vital functions of water and solute transport, blood pressure regulation, and urine concentration through the functional unit of the nephron. The microarchitecture of the kidney has recently been the focus of a number of studies [1–3]. In particular, the functional implications of the renal microstructure on the underlying mechanisms involved are of great interest [4–6]. A deeper characterisation of the microarchitecture enables the development of models to accurately simulate the functionality of the kidney. Some important data includes the ratio of short- to long-looped nephrons, relative length, type, and distribution of parts of the nephron.

A large database of histological images of mouse [7] and rat [8] kidneys was made available from previous studies performed at the Aarhus University, Denmark. The previous work involved manual tracking of the paths taken by a few hundred nephrons through the image sets and thereafter performing an in-depth analysis of the findings.

The ultimate objective of this study is to improve understanding of the architecture of the human kidney; however, tracking of human nephrons is subject to a number of practical limitations and has been left for future work. It is anticipated that several structural and functional aspects of mammalian kidneys, including human kidneys, may be elucidated through these studies of rodent histology.

Each mouse and rat dataset comprises, on average, 1000 and 3000 images, respectively. Manually tracking one long-looped mouse nephron requires tracking about 1800 elements, which takes hours to carry out. The extensive time and effort required for such datasets make it impractical to track large numbers of nephrons. Therefore any semi- or fully automated tracking procedure would be beneficial.

This created the need for an automatic tracking algorithm which could potentially be used as a standard tool on multiple datasets. This would allow the renal characterisation of multiple species as well as pathological specimens. Since the microstructure of nephrons can vary in the same kidney, it is important to obtain large samples when taking measurements, such as lumen diameters and nephron lengths, in order to render the findings more statistically accurate and representative of a variety of kidney specimens.

It is important to note the difference between automatic tracking and segmentation. The latter is the isolation of independent structures in images, such as the separation of organs in computed tomography and magnetic resonance images [9, 10], or the differentiation between tissue types in histological images, mostly for purposes of visualisation or further processing. In contrast, automatic tracking utilises the results of segmentation to create an abstract computational reconstruction of the structure for purposes of accurate measurement.

Currently, there exists no method for the automatic tracking of nephrons through serial slices. However, methods for automatic tracking of other biological structures do currently exist, with a common example being that of blood vessels in retinal images [11–13]. Other structures for which automatic tracking has been attempted include the dendrites of individual neurons and the portal and hepatic venous trees of the liver [14].

However, the methods from the aforementioned applications cannot be directly applied to the nephron tracking problem due to a number of factors. A crucial difference is that there are hundreds to thousands of nephrons [15] that need to be independently tracked through serial slices (a three-dimensional problem) as opposed to one or a few structures in single images (a two-dimensional problem). In particular, the tortuosity of the nephrons poses a major challenge. Nevertheless, several concepts from existing tracking applications have been adopted in the current approach, such as graph-based tracking, metrics to indicate confidence per iteration, and a set of validation rules to reduce error.

This paper presents a methodology for automatically tracking nephrons through images obtained from serial kidney sections using image processing, feature extraction, graph-based tracking, and machine learning techniques. The combined application of these techniques presents a novel approach to the nephron tracking problem.

The research aims to determine how effectively and accurately an automated approach can be compared to the manual method and to quantify how much manual intervention is necessary in the automatic approach to track the paths of entire nephrons. Once tracked, the results can be processed to extract useful metrics and statistics.

2. Data Acquisition

The dataset was obtained from two previous projects performed at the University of Aarhus as described in the following.

Experiment 1. Kidneys from three 8-week-old male mice were fixed through the abdominal aorta with glutaraldehyde. The tissue blocks were cut perpendicular to the longitudinal axis from the surface of the kidney to the papilla. The tissue blocks were fixed overnight in the same fixative and postfixed with OsO_4, en bloc stained with uranyl acetate, and embedded in flat molds in Epon. From each of the three mouse kidneys 897, 990, and 1064 2.5 μm thick consecutive sections were obtained using a microtome equipped with a Diatome histoknife. The sections were stained with toluidine blue when heated onto the microscope slices [7].

Experiment 2. Kidneys from three 3-month-old male Wistar rats were cut into 4252, 4384, and 4541 2.5-μm thick serial sections and processed as described above [8]. All animal experiments were carried out in accordance with provisions for the animal care license provided by the Danish National Animal Experiments Inspectorate.

The multiple serial sections were digitized using a microscope equipped with a digital camera attached to a standard PC. In Experiment 1, the sections were digitized into images using a ×4 objective lens resulting in a final image size of 2500 × 1675 pixels and an isotropic pixel size of 1.16 μm. In Experiment 2, the images were recorded using a ×3 objective lens, producing images of 2750×2500 pixels with an isotropic pixel size of 1.550 μm. The multiple digitized serial images were subjected to a classic rigid registration followed by a nonrigid transformation using custom-made software written in C [16–18].

3. System Overview

From a methodological perspective, a tracking problem would fit the generic architecture of a Computer Aided Diagnosis (CAD) system [18] with stages of preprocessing, defining regions of interest, feature extraction and selection, and classification [19]. Figure 1 describes the architecture of the tracking system developed in the present study.

The system was implemented in MATLAB [20] as a series of independent modules where structures of information are progressively passed from one stage to the next. This framework is related to an object-orientated approach in that the major functions are decomposed into independent, reusable blocks. The development of the system is incremental, involving continuous reiteration through the three main stages to achieve optimal performance.

4. Image Preprocessing

The purpose of the preprocessing stage is to prepare the images for the feature extraction stage, by creating uniformity among all nephron cross sections and addressing nonideal factors. The images are processed such that required features (nephron cross sections) are enhanced while unwanted features (such as interstitial tissue cross sections, large blood vessels, background pixels, and large artefacts) are filtered out or reduced.

The lumens of the nephrons are the object chosen to be isolated because they are more easily and accurately isolated

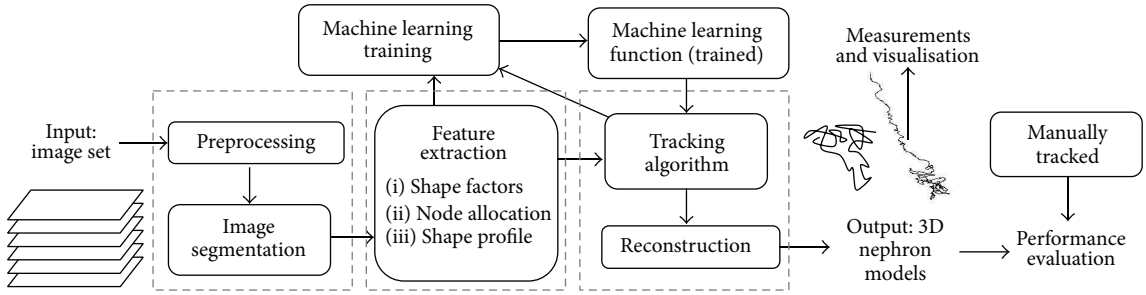


Figure 1: A high level overview of the nephron tracking system, showing the main subsystems and the flow of information between them.

than the nephron walls which touch each other. Each image undergoes the following.

(a) Conversion to grayscale is performed as the staining used on the specimens (toluidine blue [7]) results in all structures being monochrome. If a more differentiating staining method was to be used in future image sets, the colour information should be retained.

(b) Background removal is achieved by forming a background mask through a threshold filtration, large component extraction, and morphological image closing using a circular kernel. The mask is then inverted and applied to the original image by multiplication.

(c) Histogram equalisation is performed in order to counteract uneven intensities which are commonly present. Global and local adaptive equalisations are applied through the use of a large and small equalisation window, respectively [21].

(d) Simple thresholding creates a binary image. The threshold value is chosen so that it does not allow independent lumens to merge while also not letting small nephron cross sections disappear.

(e) Morphological erode/dilate cycles result in the removal of thin interstitial tissue cross sections. The kernel is chosen carefully so as to not mistakenly remove small nephron cross sections.

(f) Binary components that are very small (<10 pixels) and very large (>100 000 pixels) can be confidently identified to not be nephron cross sections and are removed.

Obtaining this final binary image is one of the most important tasks, as the accuracy of the following stages depends on how well the cross sections are isolated from one another. Many parameter values are critical when deciding on how many interstitial tissue cross sections appear in the images. A compromise must be made between the number of interstitial tissue cross sections present and the number of small nephron cross sections that do not get eliminated.

Further preprocessing involves the removal of highly distorted images and replacing them with the image above or below (so as to not have missing image numbers in the set). An average of 4 images per dataset has been manually replaced. However, an automatic method can be devised if a larger number of images are defective, for example, analysing the mean intensity of each image in the image set.

4.1. Sigmoid Function for Automatic Parameter Variation. A transition zone in the outer medulla exists where the thick descending limb (≈60 μm in diameter) suddenly narrows to a diameter of 10–15 μm to form the thin descending limb [22, 23]. This change requires almost all parameters of the preprocessing steps to change to ensure that nephron cross sections of all sizes are extracted. In order to automatically accommodate this change in morphology, the parameters of the preprocessing steps are made to vary according to a modified sigmoid function [24] which has its inflection point set at the transition zone. This also allows relatively constant parameter values in the cortex and inner medulla. The parameters of the sigmoid functions must be manually chosen through experimentation as part of system calibration.

5. Feature Extraction

Feature extraction aims to simplify and concentrate useful information from raw data. Within the images, large amounts of the data are not useful, for example, the large number of pixels making up the background. The pixel information can instead be condensed into a set of features per nephron cross section, which represent the problem to a sufficient degree. Intuitively, the most useful information about a single nephron cross section is its size, shape, colour, and location.

5.1. Image Segmentation. Connected component segmentation [21] (4-connected neighbourhood) is used to segment the image into independent nephron cross sections. Watershed segmentation is another possible segmentation technique, which could perform better in cases where independent lumens incorrectly merge through a few connected pixels. However, this method tends to oversegment the image [25], resulting in the division of elongated nephron cross sections.

5.2. Node Allocation. A *node* is defined as a point coordinate in the three-dimensional (3D) image space. The pixel locations per nephron cross section can be reduced into a set of nodes allocated along the cross section (e.g., a circular

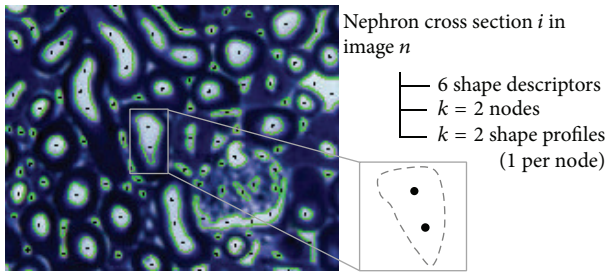


Figure 2: An example of a raw image is shown. The extracted binary cross sections after preprocessing are highlighted in green and the allocated nodes are shown as black dots. Each cross section will have k nodes, 6 shape factors, and k shape profiles. Many cross sections in the cortex are not of actual nephrons but rather of the interstitial tissue between them. The glomeruli are also highly segmented.

nephron can be represented by one centre coordinate, instead of hundreds of pixel locations). An elongated cross section can have multiple nodes along its length. This abstraction greatly simplifies the problem, reduces the size of the data, decreases the computational load on subsequent stages, and concentrates the significant information.

K-means clustering is used to allocate nodes [26]. Each nonzero pixel on a single isolated binary cross section is designated as an observation. If the nephron cross section is circular or small, one centroid is requested ($K = 1$). For elongated nephron cross sections, the K value increases until the mean distance between adjacent nodes is less than a desired value. This ensures an adequate number of nodes are allocated per nephron cross section depending on its size.

5.3. Shape Measurements. Tracking of a nephron using only the 3D set of nodes results in the linkage of multiple nephrons, blood vessels, and interstitial tissue. By only considering the point cloud, the algorithm is blind to a large amount of available information. Therefore, shape information of each cross section is also captured. Each node gets assigned a group of shape metrics and a shape profile as shown in Figure 2. The idea behind incorporating shape information into the tracking is to make the algorithm intelligent and highly confident at each incremental step of the process.

5.3.1. Shape Factors. A shape factor refers to a dimensionless value that is dependent on an object's shape but is independent of its size [27]. These metrics are calculated using various measurements of an object, such as its area, perimeter, and diameter. They usually indicate the degree to which an object deviates from an ideal shape, such as a square or circle [27]. Shape factors are extracted to capture abstract information about each cross section along with the nodes. Circularity, eccentricity, solidity, and aspect ratio were chosen as useful descriptors for the cross sections. Area and minor axis length are also captured as absolute-valued descriptors.

5.3.2. Shape Profile. The shape factors are useful for cross sections that are round and elliptical, but they do not adequately describe cross sections that are more arbitrarily shaped, such as glomeruli or interstitial tissue cross sections. As an additional feature, the shape profile, or centroidal profile, of each cross section is calculated.

The shape profile of an object is a polar plot of the distance to its boundaries with respect to a reference point [21]. It transforms a two-dimensional shape representation into a one-dimensional plot [21]. The centroid is commonly selected [21], but the nodes allocated in the previous step have been chosen instead as they are more relevant to the problem and will allow an accurate relative comparison of shape profiles between nodes.

First, the edges of a single cross section are obtained using a Sobel edge detector [28]. This method produces a well-defined closed curve around the cross section. The edge pixels are then processed into an ordered set of points. The angles and radii relative to the reference point are calculated as in

$$\boldsymbol{\theta} = \arctan\left(\frac{\mathbf{P}_{\mathbf{edge}}(y) - P_{\mathrm{ref}}(y)}{\mathbf{P}_{\mathbf{edge}}(x) - P_{\mathrm{ref}}(x)}\right),$$
$$\mathbf{r}(\theta) = \left\|\mathbf{P}_{\mathbf{edge}} - P_{\mathrm{ref}}\right\|, \tag{1}$$

where $\mathbf{P}_{\mathbf{edge}}$ is the vector of edge coordinates, P_{ref} is the reference coordinate, $\boldsymbol{\theta}$ is the vector of angles, and $\mathbf{r}(\theta)$ is the vector of radial distances. The shape profile undergoes unwinding and interpolation at desired angles in order to eliminate multivalued points and produce a consistent feature set. The degree of abstraction is dependent on the angle increment [21], which was chosen to be 15°.

6. Tracking Algorithm

When a nephron is manually tracked by the eye, an intuitive process is used by the brain. Once a single nephron cross section has been fixated, a nephron cross section within the same vicinity is searched for in the next image. Size, shape, and colour are also subconsciously compared. The tracking algorithm uses a similar process, with a number of generalised rules to accommodate the tortuous path taken by the many nephrons. The algorithm is highly dependent on the quality of preprocessing and the accuracy of feature extraction stages.

A graph-based approach similar to algorithms like the A-star search algorithm is employed for tracking [29]. The algorithm forms a graph in 3D space by establishing edges between the nodes previously allocated during feature extraction. Open and closed lists are used to manage the unexplored and explored nodes, respectively. Each node is stored along with its parent node, forming a linked list. Ideally, given a starting seed, edges should be formed such that all nodes belonging to one nephron are collected in the closed list. Prior to proceeding, a few symbols are defined:

I_n: image n,

$\mathscr{C}_n$: the set of all nodes in image n,

c_n^i: the set of nodes on cross section i in image n,

c_{kn}^i: the kth node on cross section i in image n.

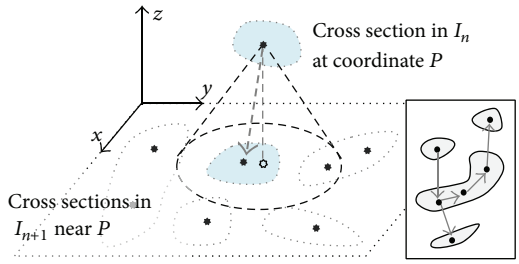


Figure 3: Each node in image n has the potential to connect to 2 nodes vertically (in images $n + 1$ and $n - 1$) within some tracking radius and 1 node horizontally on the same cross section as itself. This allows cross sections to be linked through turns and bends.

6.1. Edge Formation. The edges are established through a controlled set of criteria. Given a particular node $c_{kn}^{\,i}$ in image I_n, it has the potential to connect to three other nodes through two types of edges as shown in Figure 3:

$$\min\left(\left\|\mathscr{C}_{n\pm1} - c_{kn}^{\,i}\right\| < r_{\text{track}}\right). \tag{2}$$

6.1.1. Vertical Edge. It includes potential connections to cross sections in the image above (I_{n-1}) and below (I_{n+1}) the current cross section. Nodes are searched for which lie within some tracking radius around the current node; that is, a node satisfying the following condition will become a child node of the current node.

Only one node is allowed to be formed in each direction. If multiple nodes satisfy the condition, the one with the smallest Euclidean distance is used. The confidence of a vertical edge is <1, as the possibility of linking to an incorrect cross section exists due to the large number of closely packed nephrons.

6.1.2. Horizontal Edge. It involves linking all nodes that lie on the same cross section as the current node, that is, c_n^i. The current node is termed the "entering" node. The pairwise Euclidean distances between all nodes are used to establish the linkage between the nodes.

6.2. Local Image Registration. Local alignment is needed (in addition to the alignment in the previous study [8]) due to the presence of local image distortions and progressive change in morphology. Images I_n, I_{n+1}, and I_{n-1} are cropped around the current node location. The subimages in I_{n+1} and I_{n-1} are cross-correlated against I_n in order to obtain the translational x- and y-offset between the images [30]. These are typically only a few pixels but have a large impact on the accuracy of tracking since some nephron cross sections are also just a few pixels wide. This local alignment only takes translation into account; it is assumed that local rotational offsets are minimal. Future work could explore the increase in accuracy obtained with the use of more complex image registration methods such as a nonrigid transform. Once a link has been made between cross sections, the transformation is reversed to avoid accumulation of the offsets.

6.3. Skipping Images. An image may be termed defective if it has a large number of interfering artefacts or distortions, which obscure cross sections of the nephron at hand. These images can in general be skipped while tracking the nephron. However, a maximum of 2 images (the equivalent of 5 μm of the specimen) may be skipped at a time, as the morphology can change vastly in this span and would introduce too large a probability of error in tracking (e.g., jumping onto another nephron). A set of skipping criteria are established using a direction buffer and refractory period to prevent skip attempts from occurring too frequently (from every dead end).

6.4. Validation Steps. The steps discussed thus far would work if the data only contained information of the nephron cross sections. However, many of the cross sections actually belong to interstitial tissue and blood vessels which are randomly dispersed between the nephron cross sections and lie in close proximity to the nephron at hand. Even though the correct nephron path may be found, much interference is caused by interstitial tissue cross sections, potentially causing the path to branch from the nephron's path and even link onto other nephrons. A rule base of three validation steps is incorporated into the tracking algorithm in order to eliminate incorrect moves from one cross section to another.

(a) *Distance Validation*. The Euclidean distance (in the x-y plane) between a parent and potential child node must be less than the sum of their radii (half the minor axis length is used). This ensures that even if a cross section lies within the tracking radius, consistency in terms of size and relative displacement is maintained. Many cases of interstitial tissue cross sections linking to nephrons are eliminated by this rule.

(b) *Bidirectional Movement Validation*. If a move is made from node A in image I_n to node B in image I_{n+1}, then an attempted move from B to image I_n must lead back to node A (i.e., bidirectionality must be maintained). If not, the move is discarded. Moves between interstitial tissue cross sections are typically not connected in this manner and are hence largely eliminated.

(c) *Skipping Validation*. This ensures that a move involving a skip is only allowed if the shape of the cross section remains relatively constant during the skip. This means that skips will not be allowed on turns and bends, as this presents a high chance of error. The change in shape is measured by the percentage change in the six shape factors.

6.5. Reconstruction. The path is reconstructed through inference of the parent-child node pairs. The longest path forms the nephron path, while shorter branches are eliminated as they are most likely ambiguous nephron paths or pieces of interstitial tissue that were mistakenly linked. Each coordinate can be linked to its shape factors, enabling a 3D rendering of the nephron path with a varying lumen radius.

Table 1: The intermediate output classes of the learning functions and their combination into final classes.

Final class	Intermediate class
Valid move	(1) A normal move between circular cross sections
	(2) A normal move involving elongated cross sections
Invalid move	(3) An abnormal move typically involving interstitial tissue or blood vessel cross sections
	(4) A move involving a glomerulus cross section
x	(5) A move in the inner medulla

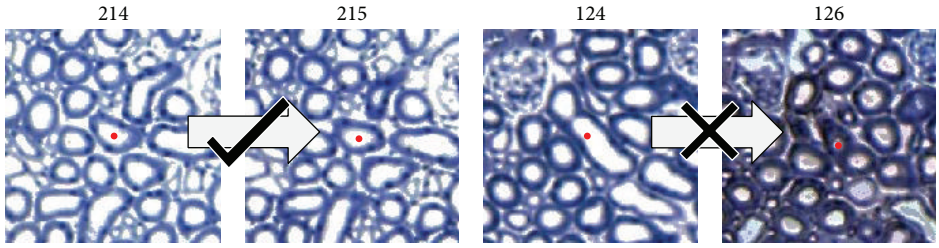


Figure 4: The moves attempted by the unregulated tracking algorithm are captured, displayed, and labelled to form training examples for the neural network. The image shows examples of a valid (a) and invalid (b) move, which will be labelled with a "1" and a "3," respectively.

Lastly, the automatically tracked path must be evaluated in 3D space. At this stage, known information about the problem can be used, for example, the proximal and distal convoluted tubules intertwine and must thus be in the same vicinity in the cortex [7], or the proximal convoluted tubule is longer and more convoluted than the distal [7]. Incorrect paths can be eliminated by comparison with typical 3D features of nephrons, such as curvatures of the bends. If the results do not adhere to one or more of these expectations, it could then be that the result is incorrect.

7. Validation Using Machine Learning

The validation rule base results in some nephrons being correctly tracked, while others are incorrectly linked to other nephrons, interstitial tissue cross sections, and blood vessel networks. A large amount of information has not yet been taken into account, such as the shape profile and shape metrics. The purpose of the machine learning (ML) stage is to incorporate some form of intelligent decision making when linking one node to another during tracking. This is done by assessing the shape descriptors and other features of the two cross sections through a trained classifier. A supervised Artificial Neural Network (ANN) and Support Vector Machine (SVM) have been used to classify a move from one cross section to another as either valid or invalid. This classification is used by the tracking algorithm to make decisions during tracking.

7.1. Feature Selection. The chosen features must fully characterise a move from one cross section to another and provide a good degree of distinction between different types of examples. Since two cross sections are being compared, it is useful to look at combined features. A total of 66 features are used including

(i) the means and differences between the shape factors,

(ii) the Euclidean distance between nodes in the x-y plane,

(iii) the z position of the nodes relative to the image set to indicate depth into the kidney, that is, cortex to medulla,

(iv) the image difference, normally 1, that can be 2 or 3 if images have been skipped,

(v) image alignment offset, high offset coupled with other odd features, which may be a flag for an abnormal move,

(vi) the shape profiles of the cross sections at 15° intervals and a correlation metric of the shape profiles.

7.2. The Training Process. The training set is created by capturing moves (pairs of cross sections) during unsupervised tracking (without any machine learning validation) of a chosen set of nephrons. Each parent-child pair is assigned a label as in Figure 4.

Five output classes listed in Table 1 were chosen to form the output matrix. A voting scheme [31] between the classes is then used to determine the final classification as valid or invalid. Class 4 is used to terminate tracking at the glomerulus while class 5 is used as a "region signal" to change the mode of tracking between the cortex and inner medulla. The shape factors and descriptors belonging to each cross section in the pair can be extracted as required and the 66 features are then combined to form the input matrix. A multiclass classifier is produced using the one-versus-all approach [32].

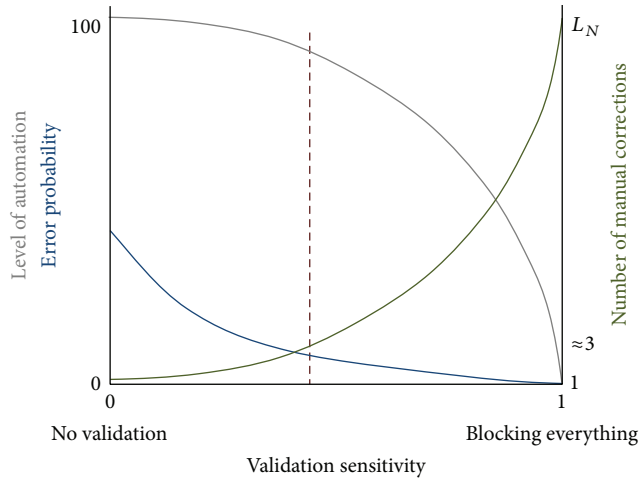


Figure 5: The number of false positives increases with increasing validation sensitivity, resulting in premature termination of tracking. This means only a portion of the nephron is tracked, but with a low error, where error refers to deviation onto an incorrect path. If manual correction is used, the number of corrections required for continuation of tracking will increase with sensitivity (up to L_N, the length of the nephron). This means a decreased level of automation but also decreased chances of error. The graph is merely conceptual.

In addition to manual selection of examples, a method involving a feedback process between the tracking algorithm and the training process is used in order to collect a fair number of examples per class. This prevents the formation of a skewed dataset or underrepresentation of a certain class, which may affect classification accuracy.

A threshold is applied to the continuous output of the ANN in order to deem the result positive or negative. This threshold has an impact on the sensitivity of invalid move rejection. For the SVM, the width of the radial basis function (RBF) kernel has the analogous effect. It is critical that false positives are minimised as these would halt the tracking process by blocking a valid move along the path of the nephron, hence preventing the rest of the nephron from being tracked. A false negative on the other hand would allow an incorrect path to be formed, but the incorrect path is typically halted due to the presence of many invalid moves through interstitial tissue and is therefore not as critical as a false positive.

8. Manual Intervention

Premature termination of tracking (due to nonideal preprocessing, feature extraction, image artefacts, or distortions) commonly occurs in the inner medulla. Image spatial resolution is a limiting factor for these small cross sections. One way of overcoming premature termination without introducing an error is to allow the user to manually bypass problematic cross sections at the end points of the automatically tracked path. This, of course, reduces the automaticity of the system but still dramatically reduces the time and effort required for the manual tracking task. The degree of automation can be controlled by sensitivity of the validation stages, as shown in Figure 5.

9. Results

9.1. Automatically versus Manually Tracked Nephrons. The accuracy of an automatically tracked nephron is measured against the manually tracked data, which forms the gold standard. The following is defined for ease of description:

Υ_n: the manually tracked path of nephron n,

Ψ_n: the automatically tracked path of nephron n.

When the result has a low degree of correctness, it is because either tracking terminated prematurely or the path deviates onto an incorrect one (linkage with another nephron, blood vessel, or interstitial tissue cross sections), or a combination of these. The outcome of the tracking of a particular nephron is hence evaluated using two correctness measures:

(1) α_n = % of Ψ_n that is correct – "accuracy,"

(2) β_n = % of Υ_n, that Ψ_n covers – "extent."

These are calculated using per image residuals between the automatic and manually tracked coordinates. α measures the similarity to the manually tracked nephron. It is low if the path deviates onto other structures and high if the tracked path contains data of only the target nephron, be it a small or large portion. β measures how much of the target nephron is tracked; it is low (relative to the ideal β value per segment) if only a small portion is tracked. It can still be high if the path branches onto incorrect structures, as long as a large part of the target nephron is found.

The tracking algorithm successfully tracks large portions of the nephrons automatically, occasionally requiring manual intervention in order to obtain full nephron paths. 16 nephrons from 2 mouse datasets and 11 nephrons from 2 rat datasets were chosen to form a test set. These were not used to form the training set for the machine learning algorithms. Different parts of the nephrons were tracked with varying accuracies and extents as shown in Table 2, due to differing tubule characteristics. In particular, the proximal convoluted tubule (PCT) and proximal straight tubule (PST) were tracked well, while the descending thin limb (DTL) and ascending thin limb (ATL) of the loop of Henle were more problematic in both the mouse and rat datasets. Automatic tracking of the PCT of a rat nephron is shown in Figure 6 and example of the PCT, PST, and DTL of a nephron tracked both manually and automatically is compared in Figure 7. The thick ascending limb (TAL) is tracked well in both the mouse and rat while the distal convoluted tubule (DCT) is only tracked well in the rat due to its larger diameter.

Tracking a full mouse nephron requires an average of 19 manual corrections while a full rat nephron requires 58 manual corrections. The frequency of manual intervention is dependent upon the number of image artefacts and distortions encountered along the path of the nephron, as well as the visibility of the cross sections. A longer path (in terms of the number of moves) requires more corrections; for example, the rat nephrons are on average 4.7 times longer than mouse nephrons.

Table 2: Test results on a chosen set of 16 mouse nephrons and 11 rat nephrons. The number of manual corrections is given as the mean ± one standard deviation. Ideal β values for the six segments for both the mouse and rat were derived from measurement of manual data and the results in the appendix of the previous study [8].

		Mouse				Rat			
Area of nephron	β_{IDEAL} (%) [8]	β_{MEAN} (%)	Extent: $\beta_{MEAN}/\beta_{IDEAL}$ (%)	Accuracy: α_{MEAN} (%)	Average number of manual corrections	β_{MEAN} (%)	Extent: $\beta_{MEAN}/\beta_{IDEAL}$ (%)	Accuracy: α_{MEAN} (%)	Average number of manual corrections
PCT	25	27.36	109.44	95.14	1.20 ± 1.11	28.48	113.92	96.32	5.20 ± 4.70
PST	18	16.33	90.72	98.24	0.50 ± 0.71	14.64	81.33	90.17	5.00 ± 2.75
DTL	19	13.90	73.16	80.57	5.44 ± 1.69	15.83	83.32	84.63	24.00 ± 8.19
ATL	14	14.94	106.71	85.67	2.46 ± 1.87	15.63	111.64	88.47	13.50 ± 6.95
TAL	14	13.19	94.21	96.32	3.64 ± 1.55	11.50	82.14	97.48	6.67 ± 3.09
DCT	10	14.29	142.90	72.13	5.86 ± 3.00	13.91	139.10	95.23	4.33 ± 2.49
Full	100	100	100	87.49	19.09 ± 1.65	100	100	80.85	58.70 ± 4.70

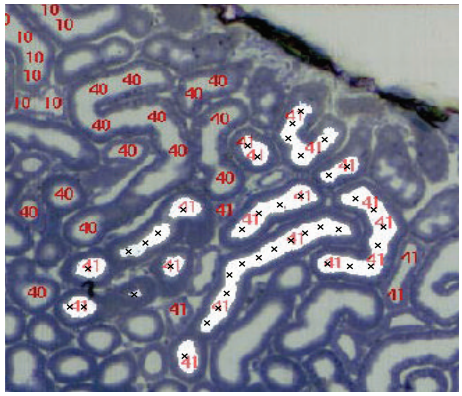


Figure 6: An example of a labelled image is shown with the red numbers representing the different manually tracked nephrons. The automatically tracked nephron (number 41) is superimposed, shown in white with black crosses at the nodes. Unlabelled "41" cross sections are of the DCT which was not tracked in this instance.

The average number of corrections required for each part of the nephron is contained in Table 2. Most corrections are for the DTL and ATL. Figure 8 displays the ability to track an entire nephron with manual intervention.

The number of manual corrections varies with the sensitivity of the validation steps. For example, decreasing the ANN threshold, increasing the coefficient of distance validation, or turning bidirectional validation off will decrease the number of requests for manual correction by the algorithm. However, this increases the chance of tracking incorrect structures (decreases α) as shown conceptually in Figure 5. The settings of the validation steps were therefore chosen such that the algorithm tracks with high accuracy (α) while not requesting excessive unnecessary manual interventions.

9.2. Efficacy of Validation Steps. The validation steps for a particular move are carried out in a set sequence with the least computationally expensive step being first. This is so that if an invalid move is detected, it does not have to go through all of the subsequent stages. However, for testing, all validation steps were carried out.

Table 3: The invalid move rejection rate and accuracies of the validation steps are shown. Results are based on 8017 invalid moves.

Validation step	% of total invalid moves flagged	% of detected invalid moves that are unique	% accuracy
Distance Val.	40.21	25.94	99.67
Skip Val.			
Total	38.59	25.38	90.01
Skips	98.97		
Bidirec. Val.	29.92	18.94	92.05
ML Val.	57.61	42.46	93.62

9.2.1. Validation through the Rule Base. Although the types of invalid moves are diverse, the rule base attempts to model the majority through hard-coded, direct rules while the ML validation attempts to model them in a more generalised, less rigid manner. The rejection rates and accuracies are detailed in Table 3.

All four rules produce accuracies above 90% with the distance validation rule being the most accurate (99.67%) and the machine learning validation being the most often triggered (captures 57.61% of all invalid moves). Given a large set of detected invalid moves, certain fractions are uniquely captured by each of the validation steps as shown in Table 3. Of the 8017 invalid moves, 49.65% were measured as being captured by more than one rule.

Ideally, the ML validation stage should be able to perform the tasks of distance and skipping validation, as the rules should be spontaneously integrated into the learnt hypothesis. Since 57.54% of the moves captured by the machine learning step are captured by other rules, it can be said that it does perform the tasks of the rule base to some degree. It can also be said that the rule base models the abnormalities to a good degree since the majority of invalid moves are eliminated even without the machine learning component.

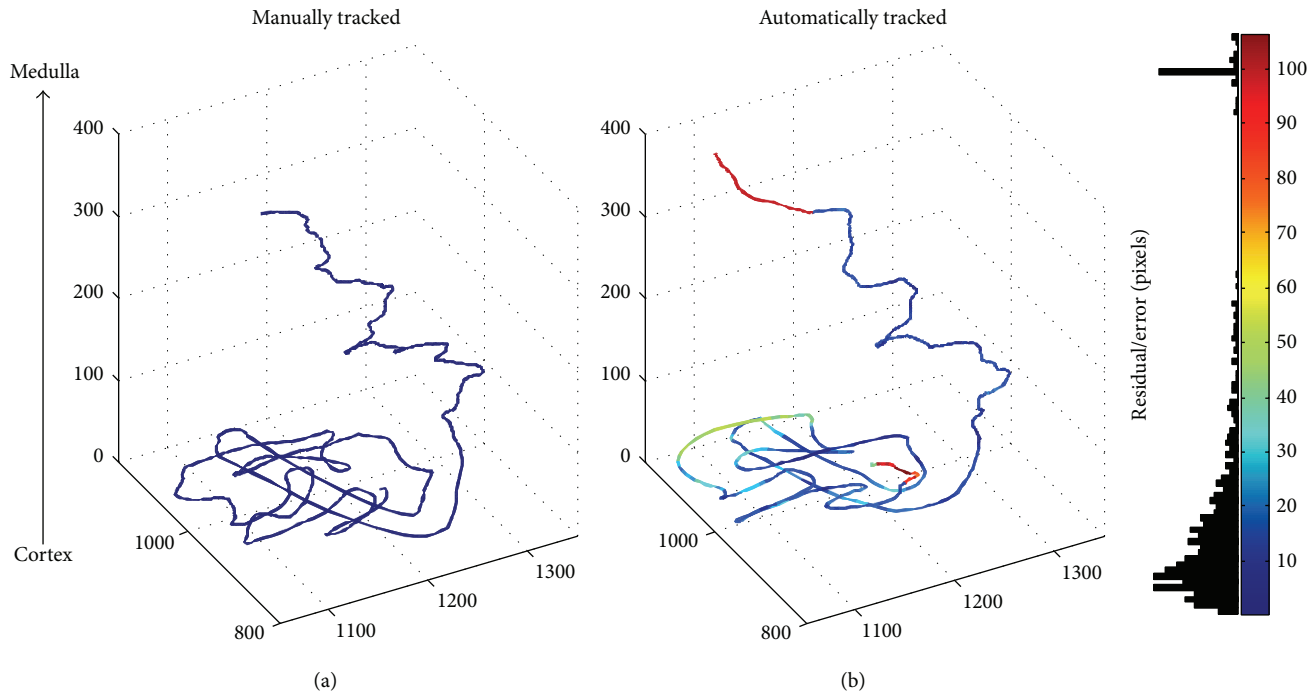


FIGURE 7: A manually tracked mouse nephron is shown on the left. The same nephron is successfully tracked automatically by the algorithm (with $\alpha = 97\%$) and is shown on the right. Tracking terminates automatically at the glomerulus. Note that, in each plot, the cortex is shown at the bottom and the DTL extends upwards. The path is coloured by the error, or residual, with respect to the manually tracked nephron. Slight discrepancies in appearance are due to different image alignments and different point coordinates used by the two methods. The distal DTL has greater error simply because the manual path was not tracked as far (therefore, $\beta > 100\%$). It can be seen in the error histogram that most of the residuals are less than 15 pixels. The correct paths of the PCT, PST, and DTL are tracked.

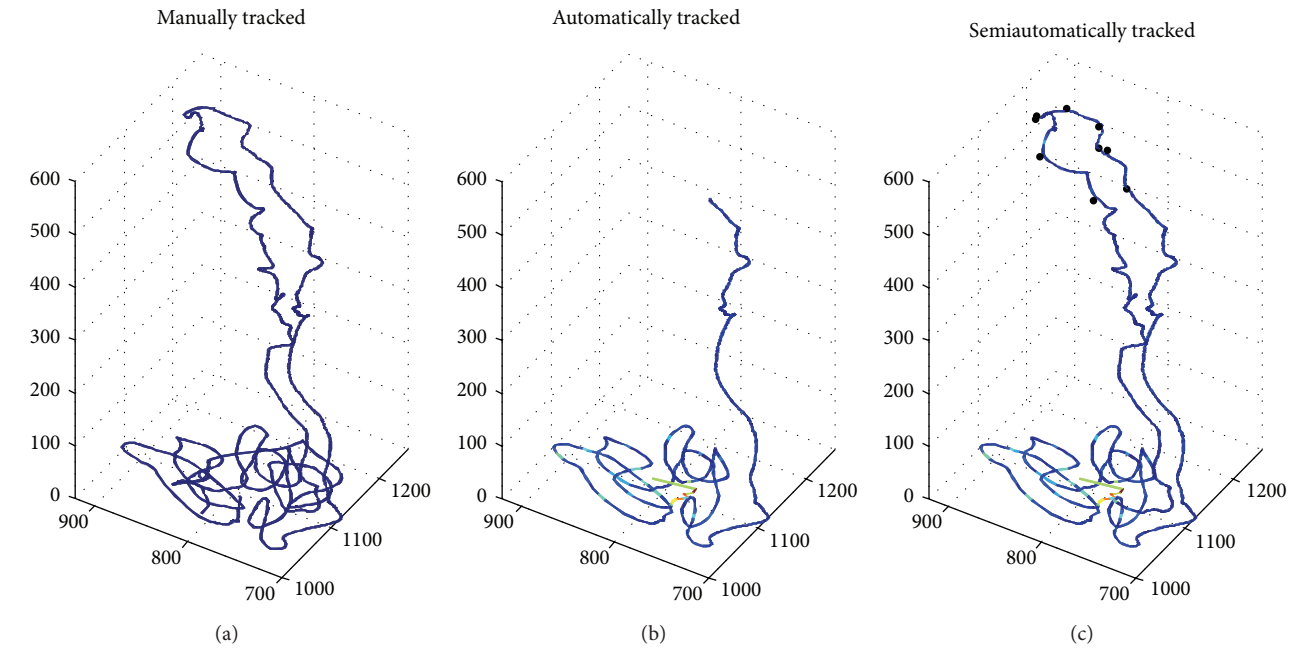


FIGURE 8: A manually tracked mouse nephron is shown on the left. The PCT and PST are successfully tracked automatically as shown in the middle plot. Tracking terminates due to diminishing tubule size coupled with artefacts in the inner medulla. A more complete nephron path is obtained with 5 manual corrections on the DTL and 4 on the ATL, as shown on the right plot (semiautomatically). The paths are coloured by the error, or residual, with respect to the manually tracked nephron. The maximum residual (shown as dark red) in this instance is 35 pixels. The black asterisks are points of manual correction. This is acceptable considering that a total of 1222 coordinates make up this path. $\alpha_{\text{AUTO}} = 97.13\%$; $\beta_{\text{AUTO}} = 39.84\%$; $\alpha_{\text{SEMI-AUTO}} = 98.77\%$; $\beta_{\text{SEMI-AUTO}} = 90.23\%$.

Table 4: Results of the ANN and SVM on the test set of 712 examples. The 5 classes have been condensed into valid and invalid classes for final classification.

Classification algorithm	Predicted class	Target class		Performance indicators (%)		
		Valid	Invalid	Accuracy	Precision	Sensitivity
ANN (threshold = 0.3)	Valid	492	32	93.82	93.62	84.61
	Invalid	12	176			
SVM with RBF kernel (width = 5)	Valid	475	19	93.25	86.70	90.86
	Invalid	29	189			

9.2.2. Validation through an ANN and SVM. The machine learning algorithms eliminate a large number of invalid moves which would have otherwise resulted in multiple nephrons, interstitial tissue, and blood vessels being linked (42.46% of detections are unique). The labelled dataset consisted of 9424 examples, which was split into training, validation, and test sets with a $0.7:0.15:0.15$ ratio, respectively.

Both the ANN and SVM produced a classification accuracy of approximately 93% on the test set, with the ANN being purposely less sensitive (84% for the ANN compared to 90% for the SVM) in order to minimise the number of false positives. The confusion matrix and performances are detailed in Table 4.

The impact of different features on classifying different types of examples is visualised and deduced using Principal Component Analysis (PCA), a dimensionality reduction technique. PCA of the features revealed that the shape profile feature is most significant when differentiating between classes 1 and 2, while shape factors play more of a role in distinguishing classes 3 and 4.

9.3. Processing Times. The current implementation is not optimally efficient, although the main aim was to develop the technique rather than optimising efficiency for an end-user application. Computational bottlenecks include the discrete Fourier transform required for image alignment, continuous calling of the ANN structure, and reading in three images per iteration of the algorithm. An implementation of the system using C++ or another more efficient language would decrease execution time. Parallel processing and use of a graphics processing unit for imaging operations would also improve speed.

10. Analysis and Discussion

The validation steps generally increase accuracy (α) while manual intervention increases the extent to which a nephron is tracked (β). Each portion of the nephron is discussed with reference to the results in Table 2. A result applies to both the mouse and rat datasets if it is not explicitly distinguished.

From the measured β values, up to 43% of a nephron's length is made up of the PCT and PST. The algorithm is able to track the full length of the PCT and PST with 1–3 and 2–15 manual corrections in the mouse and rat, respectively, when large distortions and artefacts are detected.

Although the PCT was predicted to be the most challenging part of the nephron to track due to its convoluted nature, it is tracked with high accuracy ($\alpha = 95.14\%$ in the mouse and $\alpha = 96.34\%$ in the rat) as follows.

(i) The cross sections are well isolated as they are large in diameter (15–30 pixels wide) and well defined (they have thick walls).

(ii) The average distance between neighbouring cross sections (≈25 pixels) is larger than the average image misalignment of 4 pixels.

Similarly, the PST of the mouse is tracked well with $\alpha = 98.24\%$ as the cross sections are well isolated and defined and the paths have a relatively straight course. In comparison, tracking of the rat PST produced a lower accuracy of 90.17% due to a higher frequency of tissue folds leading to incorrect linking with other nephrons.

A class 2 move is successfully detected by the ML algorithms when the PCT of a nephron joins the glomerulus at its urinary pole, thus terminating the tracking. Without this, fragments in the glomerulus would be tracked towards the vascular pole, and tracking would continue through the adjoining afferent/efferent arteriole, which then joins blood vessel systems and other glomeruli, which is undesirable. When the PST narrows into the DTL, a class 5 move is successfully triggered. The level of the class 5 output is used as a region signal to change the mode of tracking into a unidirectional one for the inner medulla. This reduces error in tracking in the inner medulla tremendously as ambiguity decreases when only one unidirectional path is allowed to be formed.

The DTL in mouse and rat kidneys is tracked with only moderate accuracies of $\alpha = 80.57\%$ and $\alpha = 84.63\%$, respectively, as the cross sections are very small in diameter (3–8 pixels) and very dense (≈6 pixels between neighbouring cross sections). This results in a higher error probability during tracking as these values are comparable to the average misalignment of 4 pixels. Confusion is more likely among identical, closely packed nephrons which are not ideally aligned. The DTL requires many manual corrections (27 on average in the rat) to produce a high β value. Frequent premature termination occurs because the cross sections are less well defined, making it more difficult to isolate them (very thin nephron walls cause independent cross sections to merge in the binary image), which results in missing cross sections and invalid moves as seen by the ANN.

The ATL faces the same challenges as the DTL. However, these cross sections are slightly larger (6–12 pixels) and have thicker walls and are thus tracked more accurately in comparison to the DTL. The ATL requires about half the number of manual corrections when compared to the DTL in both the mouse and rat datasets.

The TAL is tracked well (with 96.32% and 97.48% accuracies in the mouse and rat, resp.) as its cross sections are well isolated and relatively large (8–12 pixels in the mouse and 13–20 pixels in the rat), and the path is straight.

The DCT differs vastly in the mouse and rat datasets. In the mouse, the DCT remains narrow as it progresses from the TAL. The small cross sections making up a convoluted path are difficult to track. Fast changes in morphology (due to only having every second slice) combined with small-sized cross sections trigger the distance validation rule. An average of 5 corrections is required in the mouse DCT.

The rat DCT is tracked well as its characteristics are comparable to the rat PCT. The cross sections are much larger than in the mouse. Although the DCT is longer in the rat, it also requires an average of 5 corrections. Branching is correctly handled when the DCT of multiple nephrons join through a common collecting duct.

Manual intervention is useful when the path terminates prematurely (usually due to image defects), as the user simply bypasses the problematic cross section. In cases where incorrect links are made between different nephrons, manual intervention is not useful. The latter case is difficult to identify and correct without comparison to the manually tracked data or by manual inspection.

In general, the results are highly dependent on the quality of the images, the size of the nephron cross sections, and the amount of interfering interstitial tissue. Thicker slices (e.g., every second slice in the mouse (5 μm) compared to every slice in the rat (2.5 μm)) also produce less accurate results as the change in morphology is then more abrupt from image to image. Local image distortions and low image resolution in images of the inner medulla are the main limiting factor in automatically tracking full nephron paths.

A high frequency of images containing artefacts and tissue folds decreases the accuracy of the findings tremendously, as it only requires a single incorrect move to cause the path to deviate from the nephron at hand onto another structure (i.e., the stability of the tracking process is completely dependent on the results of the current iteration). This is especially applicable for tracking in the inner medulla, where high tubule density coupled with an artefact may result in two nephron cross sections joining incorrectly and the turn being mistaken for a loop of Henle.

11. Future Work

Further studies would be required to establish if the method developed is sufficiently generic to be used to map the architecture of other anatomical structures such as blood vessel networks in tomographic CT and MRI images. The learning algorithm would require retraining on new examples, and parameters could be tuned to control algorithm sensitivity, allowing the system to adapt to the features of different structures. The applicability and adaptability of this system to other fields are an avenue for future work.

11.1. Recommendations for Future Histological Image Sets. Higher resolution images would offer improved accuracy in isolation and tracking of cross sections in the inner medulla. Another useful addition would be using markers on the slides to aid automatic image alignment, as well as eliminating or marking highly distorted images.

A previous study by Pannabecker and Dantzler [2, 3] manually reconstructed rat nephrons using immunohistochemically stained sections (antibodies which bind to segment specific proteins) to stain various parts of the nephrons. This resulted in the DTL, ATL, collecting duct, and blood vessels fluorescing with different colours. Such staining methods would provide differentiating colour information and features to the tracking and machine learning algorithms, respectively. The confidence of results would increase as different types of cross sections could easily be distinguished from one another and interstitial tissue interference would be virtually eliminated as only cross sections of interest would be highlighted. A drawback is that the morphology of the tubules may not be intact as only particular features of the tubules would be stained.

12. Conclusion

The aim of the present study was to develop an automated system for the tracking of nephrons. A proposed methodology involving image processing and a custom tracking algorithm supervised by machine learning algorithms is presented. A number of features are extracted in order to retain shape information during the data abstraction process. The ANN and SVM have high classification accuracies and eliminate invalid moves caused by a number of hindering factors such as artefacts. The presented system is able to successfully track large portions of the nephrons automatically through both highly convoluted and straight paths. Particularly, the PCT, PST, and TAL are tracked with >90% accuracies in the mouse and rat datasets and form more than half of the nephron length. While only portions of the paths can be obtained automatically from the starting seed, full nephron paths can be obtained with an average of 17 and 62 manual corrections in the mouse and rat datasets, respectively. This is reasonable considering the thousands of coordinates making up each nephron path. Although complete automation is still elusive, the system saves a considerable amount of time and effort compared to the manual tracking task as it performs 99% of the task automatically. Performance may improve with further training of the machine learning algorithms, optimising automatic parameter variation, and manually eliminating image artefacts. The methods developed during this study form a foundation for further development towards a fully automated nephron tracking system.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank the University of Witwatersrand for providing the funding and resources required

to carry out this research. The authors are grateful for the excellent technical assistance of Inger B. Kristoffersen and Birgitte Lundbøl Grann. The establishment of the histological material was supported by grants from The Danish Council for Independent Research, Medical Sciences, FSS 11-104255 (to Erik I. Christensen), the Novo Nordic Foundation, the Danish Biotechnology Program, Daloon Foundation, the European Commission (EU Framework Program 6, EuroGene, Contract no. 05085), and the University of Aarhus Research Foundation. Arne Andreasen was supported by "Maskinfabrikant Jochum Jensen og hustru Mette Marie Jensen," "F. Poulsens Mindelegat," "Søster og Verner Lipperts Fond," and "Bagenkop Nielsens Myopi-Fond." The Amira visualization system was donated to Arne Andreasen by the Toyota Foundation.

References

[1] A. T. Layton, T. L. Pannabecker, W. H. Dantzler, and H. E. Layton, "Functional implications of the three-dimensional architecture of the rat renal inner medulla," *The American Journal of Physiology—Renal Physiology*, vol. 298, no. 4, pp. F973–F987, 2010.

[2] T. L. Pannabecker and W. H. Dantzler, "Three-dimensional architecture of collecting ducts, loops of Henle, and blood vessels in the renal papilla," *American Journal of Physiology—Renal Physiology*, vol. 293, no. 3, pp. F696–F704, 2007.

[3] T. L. Pannabecker, D. E. Abbott, and W. H. Dantzler, "Three-dimensional functional reconstruction of inner medullary thin limbs of Henle's loop," *The American Journal of Physiology: Renal Physiology*, vol. 286, no. 1, pp. F38–F45, 2004.

[4] W. Kriz, "The architectonic and functional structure of the rat kidney," *Zeitschrift für Zellforschung und Mikroskopische Anatomie*, vol. 82, no. 4, pp. 495–535, 1967.

[5] T. L. Pannabecker, "Comparative physiology and architecture associated with the mammalian urine concentrating mechanism: role of inner medullary water and urea transport pathways in the rodent medulla," *The American Journal of Physiology—Regulatory Integrative and Comparative Physiology*, vol. 304, no. 7, pp. R488–R503, 2013.

[6] H. Ren, N.-Y. Liu, A. Andreasen et al., "Direct physical contact between intercalated cells in the distal convoluted tubule and the afferent arteriole in mouse kidneys," *PLoS ONE*, vol. 8, no. 9, Article ID e70898, 2013.

[7] X.-Y. Zhai, J. S. Thomsen, H. Birn, I. B. Kristoffersen, A. Andreasen, and E. I. Christensen, "Three-dimensional reconstruction of the mouse nephron," *Journal of the American Society of Nephrology*, vol. 17, no. 1, pp. 77–88, 2006.

[8] E. I. Christensen, B. Grann, I. B. Kristoffersen, E. Skriver, J. S. Thomsen, and A. Andreasen, "Three-dimensional reconstruction of the rat nephron," *American Journal of Physiology—Renal Physiology*, vol. 306, no. 6, pp. F664–F671, 2014.

[9] P. Campadelli, E. Casiraghi, and S. Pratissoli, "Automatic segmentation of abdominal organs from CT scans," in *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '07)*, vol. 1, pp. 513–516, IEEE, Patras, Greece, October 2007.

[10] H.-Y. Lee, N. C. F. Codella, M. D. Cham, J. W. Weinsaft, and Y. Wang, "Automatic left ventricle segmentation using iterative thresholding and an active contour model with adaptation on short-axis cardiac MRI," *IEEE Transactions on Biomedical Engineering*, vol. 57, no. 4, pp. 905–913, 2010.

[11] A. Can, H. Shen, J. N. Turner, H. L. Tanenbaum, and B. Roysam, "Rapid automated tracing and feature extraction from retinal fundus images using direct exploratory algorithms," *IEEE Transactions on Information Technology in Biomedicine*, vol. 3, no. 2, pp. 125–138, 1999.

[12] T. Yedidya and R. Hartley, "Tracking of blood vessels in retinal images using Kalman filter," in *Proceedings of the Digital Image Computing: Techniques and Applications (DICTA '08)*, pp. 52–58, IEEE, Canberra, Australia, 2008.

[13] B. Karasulu, "Automatic extraction of retinal blood vessels: a software implementation," *European Scientific Journal*, vol. 8, no. 30, 2012.

[14] X. Kang, Q. Zhao, K. Sharma, R. Shekhar, B. J. Wood, and M. G. Linguraru, "Automatic labeling of liver veins in CT by probabilistic backward tracing," in *Proceedings of the IEEE 11th International Symposium on Biomedical Imaging (ISBI '14)*, pp. 1115–1118, IEEE, Beijing, China, April-May 2014.

[15] R. N. Douglas-Denton, J. F. Bertram, B. Diouf, M. D. Hughson, and W. E. Hoy, "Human nephron number: implications for health and disease," *Pediatric Nephrology*, vol. 26, no. 9, pp. 1529–1533, 2011.

[16] Y. L. Zhang, S. J. Chang, X. Y. Zhai, J. S. Thomsen, E. I. Christensen, and A. Andreasen, "Non-rigid landmark-based large-scale image registration in 3-D reconstruction of mouse and rat kidney nephrons," *Micron*, vol. 68, pp. 122–129, 2015.

[17] J. S. Thomsen, L. Mosekilde, J. Barlach, C. H. Søgaard, and E. Mosekilde, "Computerized determination of 3-D connectivity density in human iliac crest bone biopsies," *Mathematics and Computers in Simulation*, vol. 40, no. 3-4, pp. 411–423, 1996.

[18] K. B. Wagholikar, V. Sundararajan, and A. W. Deshpande, "Modeling paradigms for medical diagnostic decision support: a survey and future directions," *Journal of Medical Systems*, vol. 36, no. 5, pp. 3029–3049, 2012.

[19] J. Stoitsis, I. Valavanis, S. G. Mougiakakou, S. Golemati, A. Nikita, and K. S. Nikita, "Computer aided diagnosis based on medical image processing and artificial intelligence methods," *Nuclear Instruments and Methods in Physics Research A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 569, no. 2, pp. 591–595, 2006.

[20] MATLAB Version R2012a, MathWorks, Image Processing Toolbox; Neural Network Toolbox; Statistics Toolbox.

[21] E. R. Davies, *Computer & Machine Vision: Theory, Algorithms, Practicalities*, Elsevier, Egham, UK, 2012.

[22] L. C. Junqueira and J. Carneiro, *Basic Histology—Text & Atlas*, The Urinary System, McGraw-Hill, New York, NY, USA, 2005.

[23] W. A. Beresford, "Urinary system," in *Histology Full-Text*, chapter 23, Anatomy Department, West Virginia University, 2014, http://wberesford.hsc.wvu.edu/histolch23.htm.

[24] P. Henderson, R. Seaby, and R. Somes, *Growth II: Types of Growth Curve—Logistic Curve*, Pisces Conservation Limited, Hampshire, UK, 2006.

[25] J. Zhang and J. Fan, "Medical image segmentation based on wavelet transformation and watershed algorithm," in *Proceedings of the IEEE International Conference on Information Acquisition*, pp. 484–488, IEEE, Shandong, China, August 2006.

[26] G. Gan, C. Ma, and W. Jianhong, "Center-based clustering algorithms," in *Data Clustering Theory, Algorithms and Applications*, ASA-SIAM Series on Statistics and Applied Probability, chapter 9, SIAM, Philadelphia, Pa, USA; ASA, Alexandria, Va, USA, 2007.

[27] L. Wojnar and K. J. Kurzydłowski, *Practical Guide to Image Analysis*, ASM International, 2000.

[28] D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice Hall, Rochester, NY, USA, 1982.

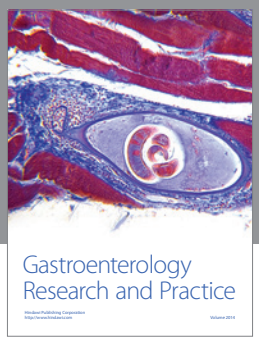
[29] A. Patel, "Stanford Theory Group: Introduction to A*," 2014, http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html.

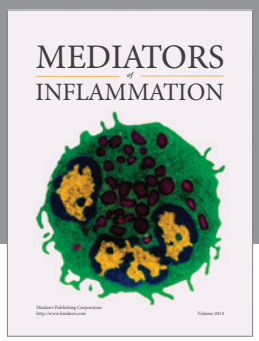
[30] B. Zitová and J. Flusser, "Image registration methods: a survey," *Image and Vision Computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[31] J. Stoitsisa, I. Valavanis, S. G. Mougiakakou, S. Golemati, A. Nikita, and K. S. Nikita, "Computer aided diagnosis based on medical image processing and artificial intelligence methods," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 569, no. 2, pp. 591–595, 2006.

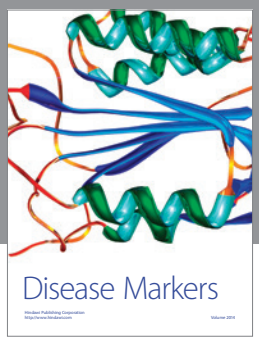
[32] N. G. Andrew, Machine Learning Course. Coursera Online Courses, 2014, https://class.coursera.org/ml-005.











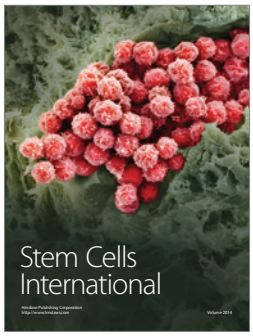


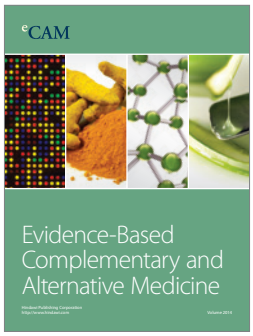




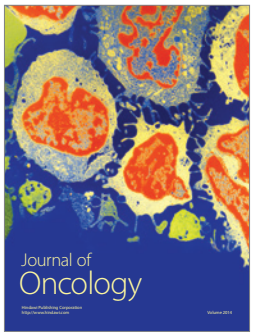


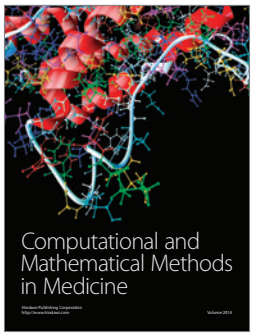


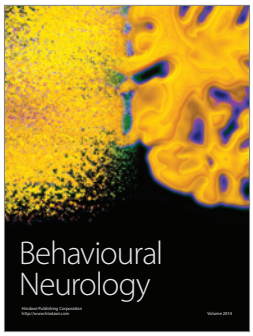




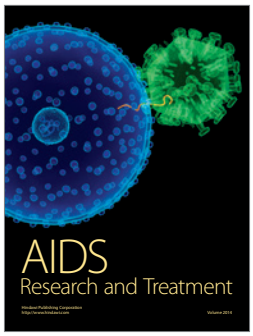


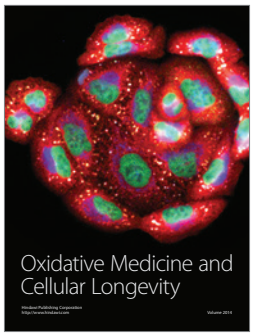












Hindawi

Submit your manuscripts at
http://www.hindawi.com