TOWARDS AUTOMATED THREE-DIMENSIONAL TRACKING OF NEPHRONS THROUGH STACKED HISTOLOGICAL IMAGE SETS

A dissertation submitted to the Faculty of Engineering and the Built Environment, University of Witwatersrand for the degree of Master of Science in Engineering.

Charita Bhikha

August, 2015

DECLARATION

I declare that this research proposal is my own unaided work. It is being submitted to the Degree of Master of Science in Engineering to the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination to any other University.

……………………..

Signature

……... day of ……………… year ………….

ABSTRACT

The three-dimensional microarchitecture of the mammalian kidney is of keen interest in the fields of cell biology and biomedical engineering as it plays a crucial role in renal function. This study presents a novel approach to the automatic tracking of individual nephrons through three-dimensional histological image sets of mouse and rat kidneys. The image database forms part of a previous study carried out at the University of Aarhus, Denmark. The previous study involved manually tracking a few hundred nephrons through the image sets in order to explore the renal microarchitecture, the results of which forms the gold standard for this study. The purpose of the current research is to develop methods which contribute towards creating an automated, intelligent system as a standard tool for such image sets. This would reduce the excessive time and human effort previously required for the tracking task, enabling a larger sample of nephrons to be tracked. It would also be desirable, in future, to explore the renal microstructure of various species and diseased specimens.

The developed algorithm is robust, able to isolate closely packed nephrons and track their convoluted paths despite a number of non-ideal conditions such as local image distortions, artefacts and connective tissue interference. The system consists of initial image pre-processing steps such as background removal, adaptive histogram equalisation and image segmentation. A feature extraction stage achieves data abstraction and information concentration by extracting shape descriptors, radial shape profiles and key coordinates for each nephron cross-section. A custom graph-based tracking algorithm is implemented to track the nephrons using the extracted coordinates. A rule-base and machine learning algorithms including an Artificial Neural Network and Support Vector Machine are used to evaluate the shape features and other information to validate the algorithm's results through each of its iterations.

The validation steps prove to be highly effective in rejecting incorrect tracking moves, with the rule-base having greater than 90% accuracy and the Artificial Neural Network and Support Vector Machine both producing 93% classification accuracies. Comparison of a selection of automatically and manually tracked nephrons yielded results of 95% accuracy and 98% tracking extent for the proximal convoluted tubule, proximal straight tubule and ascending thick limb of the loop of Henle. The ascending and descending thin limbs of the loop of Henle pose a challenge, having low accuracy and low tracking extent due to the low resolution, narrow diameter and high density of cross-sections in the inner medulla. Limited manual intervention is proposed as a solution to these limitations, enabling full nephron paths to be obtained with an average of 17 manual corrections per mouse nephron and 58 manual corrections per rat nephron.

The developed semi-automatic system saves a considerable amount of time and effort in comparison with the manual task. Furthermore, the developed methodology forms a foundation for future development towards a fully automated tracking system for nephrons.

ACKNOWLEDGEMENTS

I would like to thank my supervisors Robyn Letts, Prof. David Rubin and Adam Pantanowitz, for their invaluable support, advice, feedback, and constant interest and motivation.

I would also like to thank all members of the Biomedical Engineering Research Group for their inspiring discussions and stimulating ideas.

Thank you to all of my friends and family for providing support, enthusiasm and comfort throughout the course of my studies.

Finally, many thanks to the team at the Departments of Cell Biology, Connective Tissue Biology, and Neurobiology, Institute of Anatomy, University of Aarhus, Aarhus, Denmark, for providing the data which forms the core of this project.

CONTENTS

Appendices

List of Figures

Figure No. Page

List of Tables

List of Symbols

Vectors are indicated by variables in bold.

z	Image number
f_i	Shape factor i where i={1,…,6}
K	Number of clusters or centroids or nodes per segment
f	Nephron number
m_V	Number of observations Number of examples Number of elements in a vector V
n	Number of features
i_z	Identity number of a single nephron cross section in image z
r	Residual Radius in shape profile
$\mathbf{X}$	Input for machine learning algorithm
$\mathbf{Y}$	Output for machine learning algorithm
$\boldsymbol{\Psi}$	Automatically tracked path
$\boldsymbol{\Upsilon}$	Manually Tracked Path
α	Accuracy
β	Extent
θ	Polynomial coefficients in machine learning Angle in shape profile
δ	Angle increment for shape profile
Iz	Image z
C	General constant Set of centroids
T_{bgrnd}	Threshold for background removal

List of Abbreviations

3D	Three-dimensional
2D	Two-dimensional
PCT	Proximal Convoluted Tubule
PST	Proximal Straight Tubule
DTL	Descending Thin Limb
LH	Loop of Henle
ATL	Ascending Thin Limb
TAL	Thick Ascending Limb
DCT	Distal Convoluted Tubule
ICT	Interstitial connective tissue
BV	Blood vessels
ANN	Artificial Neural Network
SVM	Support Vector Machine
ML	Machine Learning

CHAPTER 1

Introduction

The kidney performs the vital bodily functions of water and solute exchange, blood pressure regulation and urine concentration through the functional unit of the nephron. Approximately one million nephrons intricately populate each human kidney, producing distinct regions with differing functionalities [**1**] [**2**]. The spatial distribution of nephrons within the kidney forms its microarchitecture.

The microarchitecture of the kidney has recently been the focus of a number of studies [**3**] [**4**] [**5**]. In particular, the functional implications of the renal microstructure on the underlying physiological mechanisms involved are of great interest [**6**] [**7**] [**8**]. Nephrons are the target for many drugs which regulate blood pressure and solute concentrations and hence important bodily functions [**2**]. A deeper understanding of its anatomy may lead to a better understanding of physiological function and disease, which may be beneficial to drug development, disease diagnosis and treatment.

A deeper characterisation of the microarchitecture also enables further development of models and simulations that accurately describe the functionality of the nephron and the kidney. This is a fundamental step towards the development of an artificial kidney or dialysis device. For researchers studying and modelling kidney function, some of the most useful statistics are the ratio of long-looped nephrons to short-looped nephrons and the change of this ratio across different individuals and species, the distribution in lengths within these categories, and the relative lengths of different parts of the nephron [**7**] [**9**].

A previous study carried out by the Department of Biomedicine at the University of Aarhus, Denmark, involved manually tracking the paths taken by a few hundred nephrons through histological image sets of mouse and rat kidneys, and thereafter performing an in-depth analysis of the findings [**9**] [**10**] . The manual tracking task required an exhaustive amount of time and effort per dataset, which posed a limit on the amount of data that could be acquired. This created the need for an automatic tracking tool which could be used as a standard tool on multiple datasets. This would allow the renal characterisation of multiple species as well as diseased specimens. Since the microstructure of nephrons can vary in the same kidney, it is important to obtain large samples when taking measurements such as nephron lengths, in order to render the findings more statistically accurate and representative of a variety of kidney specimens.

The image database [**11**] has been made available for use through collaboration between the University of Aarhus, Denmark, and the University of the Witwatersrand, Johannesburg. This study attempts to aid and improve the process of modelling the renal microstructure by creating an automatic software tool to track nephrons through the image sets. The manually tracked nephrons form the gold standard comparison for this study.

Various potential methodologies have been investigated and tested. The system developed in this dissertation comprises three main stages; image pre-processing, feature extraction and nephron tracking. Machine learning algorithms have been employed to accurately guide the tracking algorithm. The final system is semi-automated, occasionally requiring user input for tracking in the inner medulla where the small size, dense nephron cross-sections prove to be difficult to track automatically.

Chapter 2 introduces basic concepts of the kidney on both macroscopic and microscopic scales in order to highlight details that are relevant to the problem. An overview of existing methodologies in related fields is also discussed. The aims, objectives, rationale and scope of the study are presented in Chapter 3. Particular characteristics of the images which make the tracking task complex and introduce a number of non-ideal factors are explored in Chapter 4. A brief overview of the system is included in Chapter 5, and the developed methodology consisting of the stages of image pre-processing, feature extraction and tracking is detailed in Chapters 6, 7 and 8, respectively. Chapter 9 is dedicated to the machine learning aspects of the system. Results are presented in Chapter 10, followed by a detailed analysis and discussion in Chapter 11. Final conclusions are drawn and recommendations made in Chapter 12.

CHAPTER 2

Background

This chapter serves to provide background knowledge on concepts and fields relevant to this study, and to explore existing solutions, methodologies and applications.

2.1 An Overview of Renal Histology

A basic understanding of the anatomy and histology of the kidney is required in order to correctly model the problem, identify structures in the images and interpret results of the study in light of their biological implications.

The kidneys are a pair of bean-shaped organs lying posteriorly in the abdominal cavity [**12**]. From a high-level perspective, one of the main functions of the kidneys is to take in unfiltered blood, and produce urine and filtered blood as outputs. This filtering and reabsorption function is performed by the kidney's functional unit called the nephron. Approximately 1 million nephrons populate each human kidney [**13**].

A nephron is a long, tortuous, unbranched tubular structure, varying in diameter along its length [**1**]. Its length is broken up into seven parts, namely the proximal convoluted tubule (PCT), proximal straight tubule (PST), descending thin limb (DTL), ascending thin limb, (ATL), thick ascending limb (TAL) and distal convoluted tubule (DCT) [**1**]. The nephrons are arranged such that the PCT, PST, TAL and DCT occur in the outer part of the kidney called the cortex, while the DTL and ATL form loops of Henle in the inner region called the medulla [**1**], as illustrated in Figure 2.1. Water and various solutes are exchanged between the filtrate and the blood along the length of the nephron [**14**].

A glomerulus and Bowman's capsule (making up a renal corpuscle) occurs at the start of each PCT; this is the site at which blood is filtered to form the renal filtrate which fills the nephron tubule lumen. The renal corpuscle has a vascular pole at which the glomerulus meets blood vessels (afferent and efferent arterioles) and a urinary pole where the Bowman's capsule fuses with the nephron tubule [**1**]. The glomeruli are clearly visible in the image sets.

At its distal end, each DCT joins a collecting duct which is a common structure collecting the filtrate from a family of nephrons [**1**]. This is the only site at which branching will be seen in the nephron network [**1**]. The collecting ducts drain into the minor and major calyces of the kidney, which then empty into the ureters and subsequently the bladder.

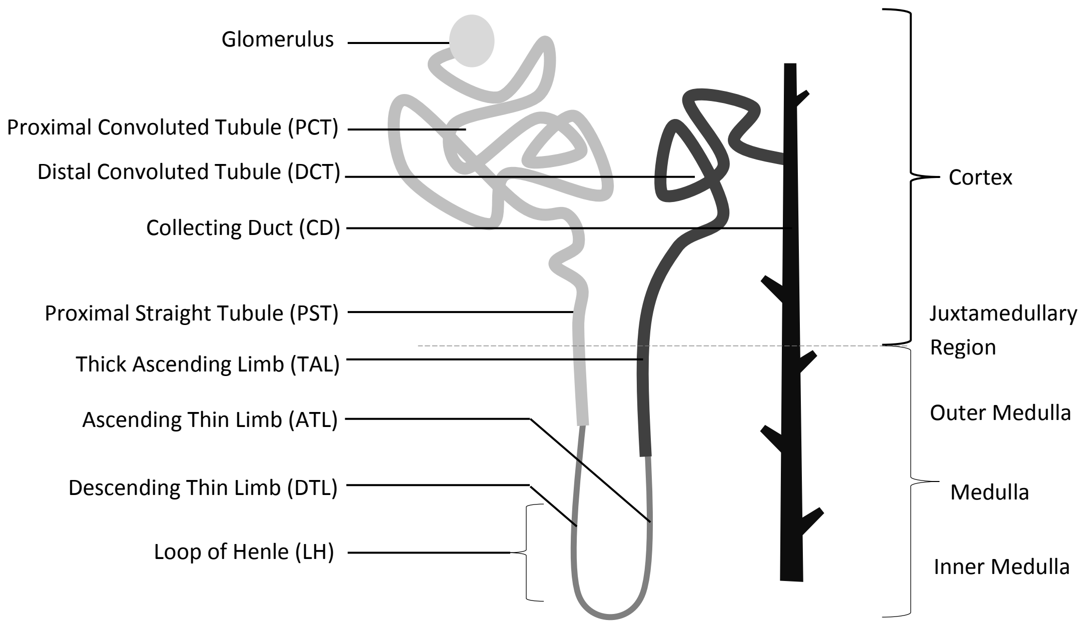

Glomerulus
Proximal Convoluted Tubule (PCT)
Distal Convoluted Tubule (DCT)
Collecting Duct (CD)
Proximal Straight Tubule (PST)
Thick Ascending Limb (TAL)
Ascending Thin Limb (ATL)
Descending Thin Limb (DTL)
Loop of Henle (LH)
Cortex
Juxtamedullary Region
Outer Medulla
Medulla
Inner Medulla

Figure 2.1: Basic anatomy of the nephron. Adapted from **[**1**]**.

Toluidine blue is the dye used in preparation of the image sets. It is a basic stain commonly used in renal pathology [**15**]. It has a high affinity for acidic tissues, producing a bluish purple stain [**16**]. It also increases the sharpness of histological images [**16**]. In a typical Haematoxylin and Eosin (H&E) stained kidney specimen, the various parts of the nephrons can be distinguished by the number of nuclei, diameter, thickness of the wall and types of cells making up the tubule [**1**]. The given images stained with toluidine blue results in the diameter and wall thickness being the only differentiating features.

The nephrons are in close contact with the renal blood supply in order to perform the filtering and solute exchange functions [**1**]. The arteries, veins and capillary networks are seen in the image sets, having varying sizes and are more irregularly shaped compared to the nephrons. However, many blood vessels, especially those emerging to and from the glomeruli, are very similar in appearance to the nephrons and may be confused.

The presence of loops of Henle in the inner medulla and the convolutions in the cortex are high-level examples of structure influencing renal function [**14**]. Looking closer, there are cortical nephrons with short loops and juxtamedullary nephrons with long loops. These have differing filtering rates [**10**]. Deeper characterisation of the renal microarchitecture may reveal additional structural aspects which have important functional implications.

2.2 Existing Solutions

2.2.1 Nephron Tracking and Three-Dimensional Reconstruction

The spatial distribution of nephrons has been explored in previous studies although all instances of tracking were performed manually and therefore the resulting statistics were based on a limited number of nephrons. The mouse or rat kidney is commonly used as it is small enough to fit on microscopic slides while adequately representing the structure of mammalian kidneys.

One of the previous studies carried out at the University of Aarhus, Denmark (from which the image sets were obtained) involved reconstructing 151 complete nephrons from the manually tracked data of a mouse kidney [**10**]. The tracking was done on 30 families of nephrons, where a family refers to all nephrons emptying into a common collecting duct [**10**]. The glomeruli were used as starting points. A number of statistics were calculated and the spatial interrelations of each part of the nephrons were thoroughly discussed, revealing some important features of the kidney [**10**]. A later study involved manually tracking 56 nephrons of a rat kidney and taking a variety of measurements such as the lengths of different parts of the nephron and glomerular volumes [**9**]. Computer-aided 3D reconstruction was also carried out for visualisation purposes.

In a different set of studies by Pannabecker and Dantzler [**4**] [**5**], the 3D architecture of the rat kidney was investigated. Various cross-sections of rat nephrons were physically labelled using differential staining/immunocytochemistry techniques. Immunofluorescence allowed visual differentiation between parts of the nephron by means of distinct fluorescence during microscopy. The digitised images were used to manually track the TDL and TAL near the papillary tip. 3D reconstruction involved creating a mesh of three-dimensional cylinder-like objects which were created for each individual nephron cross-section in each image. Existing imaging software called Amira visualisation was used. Although immunofluorescence aided the tracking process, the tracking procedure was not automated in any obvious manner.

Both these sets of studies involved manually tracking nephron cross-sections in different areas of interest in the kidney. The tracking processes were computer-aided in the sense that the software provided a user-interface; the tracking was not automated or predictive and no machine learning was used.

2.2.2 Glomeruli Detection

The glomeruli need to be detected as they serve as good starting points for tracking. Automated glomerulus detection is an important step during computer-aided diagnosis of kidney disease during a biopsy [**17**]. The change in size and shape of glomeruli is an indicator of the degree of damage in the kidney [**17**]. The biggest challenge for accurate detection is the fact that the surrounding contours are not continuous [**18**] and that other surrounding tissue produce strong noise levels [**17**]. The shape and size of the glomeruli also vary.

A set of papers [**19**] [**20**] document using a log edge detector and wavelet transform to produce a low resolution image with enhanced glomeruli edges. Spline curve fitting is applied through a genetic algorithm to obtain an accurate closed curve around the glomeruli. Another study [**17**] has shown that the watershed algorithm can produce a more accurate closed glomerulus edge.

These methods often require a starting seed and are not suitable for purely automated glomeruli detection. The images in this study differ widely from those used in other studies (usually H&E images). In contrast to images in previous studies, the nephrons produce stronger edges than glomeruli. Also, accurate closed curves around the glomeruli are not necessarily needed, merely indicate coordinates. A custom glomerulus detection method is therefore devised for this study.

2.2.3 Automated Tracking of other Biological Structures

It is important to note the difference between automatic tracking and automatic segmentation. Automatic segmentation is the isolation of independent structures in images, such as the separation of organs in CT and MRI images [**21**] [**22**], or the differentiation between tissue types in histological images, mostly for purposes of visualisation or further processing. The segmentation can be pixel (2D) or voxel (3D) based. Commonly employed techniques for segmentation include edge detectors [**23**], histogram-based methods, the watershed transform, region growing [**21**], morphological operations and active contour modelling [**22**].

In contrast, automatic tracking utilises segmentation results 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 the automatic tracking of other biological structures do currently exist, although these are for one or a few objects in a single image.

A common example is the tracking of blood vessels in retinal images [**23**]. One study [**24**] makes use of a Kalman filter as the basis for tracking, using the position and orientation of vessel fragments as states. Gradient information and expected vessel structure are used to estimate the next state during tracking. System noise is also taken into account. A number of verification or correctness checks specific to the problem are used to improve results [**24**]. Another study [**25**] uses correlations with rotated templates to track vessels iteratively in local pixel areas, in order to avoid image-wide operations which are generally slow.

The portal and hepatic venous trees of the liver has also been automatically tracked. One approach uses Laplacian-based contraction to obtain a skeleton of the vessel system [**26**], which is then broken up into nodes. Tracking consists of using orientation and diameter consistency metrics to model continuity between nodes. Maximisation of the continuity function provides the best candidate.

2.3 The Nephron Tracking Problem

The methods from the aforementioned applications cannot be directly applied to the current nephron tracking problem due to a number of factors. The nephrons are sectioned transversely, enabling one to track individual nephron cross-sections from image to image. In contrast, retinal images and CT images of the hepatic venous tree capture a single longitudinal view of the entire structure in question. Another crucial difference is the vast number of independent nephrons needing tracking versus one or a few structures in other applications. Moreover, the tortuosity of the nephrons poses a major challenge. The vast amount of data (700-3000 high resolution sections through the kidney per dataset) also poses a limitation on how the data is to be processed in an efficient manner.

Although existing methodologies cannot be used directly and completely to fulfil the requirements of the automated nephron tracking problem, several of the methods have been adopted and combined in the current approach. This includes graph-based tracking, various metrics to indicate confidence per iteration and a set of validation rules to eliminate error. In addition to this analytic heuristic technique, the high modelling capability of machine learning is employed for path validation. Machine learning is highly appropriate for such a problem as it can automatically model the complex system with high accuracy through training. The machine learning component is discussed in greater detail in Section 2.5.

2.4 Graph Theory

The primary structure of the designed tracking algorithm in this study adopts basic concepts used in graph theory as described by [**39**] and is summarised below.

A graph (G) consists of a collection of *nodes* (V) interconnected through *edges* (E). In general, a node is an object which possesses certain attributes. An edge connects two nodes, establishing a relationship between them, i.e. G = (V, E) and E = (V_1, V_2). An edge can be undirected or directed where the edge points from a *parent* node to a *child* node. Each node has a potential to have 0-1 parent/s and 0-

n children. In terms of nephron tracking, each individual nephron cross-section can be seen as a node. The 'nodes' are then progressively linked, or tracked, to form a list of parent-child pairs.

A *walk* is a sequence of nodes and edges as shown in Figure 2.2. Given a set of directed edges, a walk can be reconstructed through inference of the parent-child pairs. The resultant nephron path can be seen as a bidirectional *walk* in 3D space through the nodes making up a nephron, starting at some initial seed and ideally ending at the glomerulus and collecting duct.

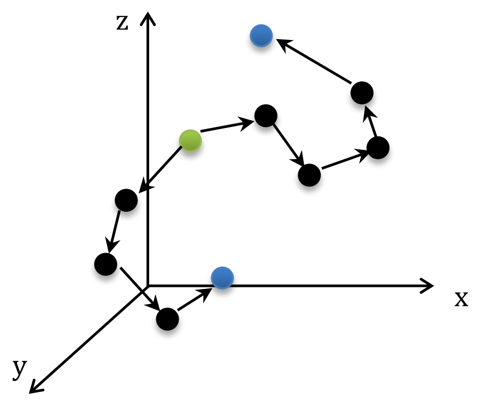


Figure 2.2: A nephron's path can be seen as a walk through a set of nodes in 3D space. The walk occurs in two directions from a starting seed (green) towards endpoints (blue) which should be a glomerulus and collecting duct.

2.5 Machine Learning

2.5.1 An Overview of Basic Machine Learning Principles

A machine learning algorithm forms a hypothesis, or a prediction function, based on experience through given inputs and outputs [**27**], i.e. a training set {**X**,**Y**}. Once a learning algorithm has been trained, it can be used to predict new unseen examples. The process is summarised in Figure 2.3.

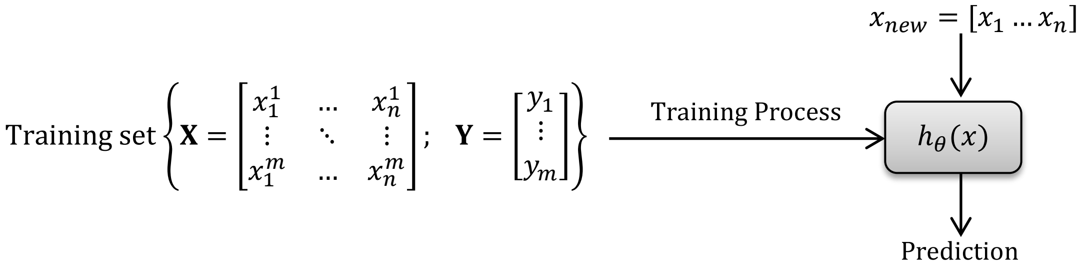

$x_{new} = [x_1 \dots x_n]$
Training set
$\mathbf{X} = \begin{bmatrix} x_1^1 & \dots & x_n^1 \\ \vdots & \ddots & \vdots \\ x_1^m & \dots & x_n^m \end{bmatrix}$; $\mathbf{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$
Training Process
$h_\theta(x)$
Prediction

Figure 2.3: The general process followed when using machine learning algorithms. $h_\theta(x)$ is the prediction function.

The weights (**θ**) of the generalised polynomial function $h_\theta(x)$ as in equation (2.1) are adjusted with each example, such that some cost/error objective function as in equation (2.2) is minimised [**27**]. This is done through methods such as gradient descent and back-propagation [**27**], and is termed 'learning'. Popular learning algorithms include Logistic Regression, Decision Trees, Bayesian Classifiers and many more [**28**].

$$h_\theta(x) = \text{sigmoid}(\theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n) = \text{sigmoid}(\boldsymbol{\theta}^T\mathbf{X}) \tag{2.1}$$

$$\boldsymbol{\theta} = \arg\min_\theta \frac{1}{m}\sum_{i=1}^{m} y^i \log\left(h_\theta(x^i)\right) + (1-y^i)\log\left(1-h_\theta(x^i)\right)$$
$$\dots + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{2.2}$$

where x^i and y^i are the input features and output of the *i*th example, respectively. *m* is the number of examples and *n* is the number of features.

Learning can be supervised, where the correct outputs are provided [**29**], or unsupervised, where intrinsic patterns are sought for within the given data [**29**]. A supervised problem may be of a regression type, where there is a continuous valued output, or a classification type, where the output is a discrete label [**27**]. Randomisation and normalisation (feature scaling) of the input is essential for good results during training [**28**]. Once the machine learning algorithm is well-trained, it can be used to classify new, unseen examples.

An underfit hypothesis is one that is too simple or of a low order [**28**]. It has high bias and cannot even represent the training set well. An overfit hypothesis is one that has too high an order. It works very well for the training set but cannot accurately predict new examples [**28**]. It is said to have high variance as it captures noise and outliers [**28**]. The regularisation parameter of a machine learning algorithm controls the level of generalisation of the hypothesis and can be adjusted to address under- or over- fitting [**28**]. Additional features or polynomial features can also solve a high bias problem, while decreasing features and adding more training examples can resolve overfitting. In addition to the training set, a validation and test set is also used during training to prevent bias towards the training set.

Additional theory on machine learning can be found at [**27**] and [**28**].

2.5.2 Application to Medical Imaging

Artificial intelligence, or machine learning, has found application in the medical imaging field. It is particularly advantageous because biological structures cannot usually be described with high accuracy through simple predictive equations. Large modelling capacity combined with flexible input and output choices make these algorithms highly desirable.

Feature-based machine learning (FML) involves computing features of objects in the images which are then used as inputs to the machine learning algorithm. The output is typically not in the image space but rather a classification or numerical value [**30**]. One such application involved using a multi-layer perceptron neural network to classify breast lesions as either malignant, fibroadenoma, fibrocystic disease or benign [**31**]. Features such as cellularity, cohesiveness, clump thickness and uniformity were computed from images of fine needle aspirate smears [**31**].

Pixel, or voxel, based machine learning (PML) uses image pixels as direct inputs, or features. PML can automatically infer features and hence reduces error and data loss that occurs through feature extraction [**30**]. The output can be a classification or a processed image containing, for example, a detected coordinate, boundary curve or enhanced object [**30**]. For example, a feed-forward neural network has been used to aid detection of boundaries during automatic segmentation of the colon in CT images [**32**]. Using a processed binary image as an input, the network is able to extract fluid filled regions of the colon [**32**]. The training time and computational power required for PML is very large due to the high dimensionality produced by image inputs.

2.5.3 Application to the Nephron Tracking Problem

The nephron tracking problem has a large number of inputs (either raw or processed images, or features such as shape, colour, position, size) and complex unknown functions. A non-linear, high dimensional machine learning algorithm is able to model these functions through supervised learning on the datasets.

For this study, two supervised classifiers are chosen for performance comparison. These are an Artificial Neural Network (ANN) and a Support Vector Machine (SVM), which are the most popular and powerful non-linear machine learning algorithms [**28**]. Both ANNs and SVMs are capable of modelling complex systems with high accuracy through supervised learning.

An ANN is a biologically inspired non-linear machine learning algorithm. It consists of multiple calculating units called neurons, each of which outputs a weighted sum of its inputs [**27**]. The neurons are arranged into multiple interconnected layers. Propagation of the input through the layers results in an intricate interrelation of the inputs dependant on the weighting factors at each neuron [**27**]. ANNs are capable of representing highly complex hypotheses, as the input features are progressively mapped into more complex features by the deeper layers [**27**]. It mimics the notion of the brain's plasticity, using 'one algorithm for all learning' [**28**].

An SVM is one of the most powerful machine learning algorithms available [**28**]. It is sometimes cleaner than logistic regression and ANN for complex hypotheses [**28**]. It is also known as a large margin classifier as it maximises the distance of the boundary from the examples. It uses computed features called landmarks, which can be computed with or without a variety of kernels.

For each of the algorithms, regularisation, the chosen features, the number of examples and the degree of the hypothesis need to be carefully chosen to optimise performance. It has been shown that most algorithms in the same class seem to perform equally well, provided that there are a large number of training examples (>10000) [**28**].

The large amount of data available (3 sets of mouse kidneys of ≈1000 images/set and 3 sets of rat kidneys of ≈4000images/set) means that sufficient training can occur in order to optimise a highly complex hypothesis function. Some sets can be used for training and others for independent testing. Multiple datasets will result in an algorithm that is not an over-fit to one particular set of images.

Feature-based learning is chosen as image-based outputs are not required (nephron detections are more easily obtained through other methods due to the homogenous, easily identifiable nature of the cross-sections). The approach is to use the machine learning classification as a validation step post-tracking. This also reduces the dimensionality of the problem.

ANNs have low transparency – the optimised weightings cannot be easily interpreted to infer a model of the system. SVMs are slightly more transparent as the landmarks and margins can be interpreted [**28**]. However, transparency is not an issue, as this problem does not require an understanding of the underlying mathematical model. The algorithm simply relies on an output of tracking accuracy for purposes of path validation.

CHAPTER 3

Project Framework

3.1 Research Question

The work presented in this study forms part of a larger research goal, which aims to aid the process of exploring the spatial microstructure of the kidney in order to advance research findings in the fields of renal physiology and pathology. It also aims to verify the existing conclusions drawn from the small sample of nephrons in the previous study, on a larger more representative sample set.

In terms of this study, the research aims to:

- Determine how 3D structures, or representations, of individual nephrons can be automatically extracted from serial slices of the kidney.
- Develop methodologies towards an automated nephron tracking system.
- Determine how effectively and accurately an automated approach to tracking can be compared to the manual method.
- Quantify how much manual intervention is necessary in the automatic approach to obtain the paths of entire nephrons.

Once tracked, the results can be processed to extract useful statistics or reconstruct a 3D representation of the renal microstructure.

3.2 Rationale

Why does software need to be developed?
The manual tracking problem requires an exhaustive amount of effort per dataset. Each mouse and rat dataset has on average 1000 and 3000 images, respectively. Manually tracking one long-looped mouse nephron requires tracking about 1800 nephron cross-sections. This poses a limit on the amount of data that can be acquired (the number of nephrons and kidneys analysed). This creates the need for an automatic tracking algorithm which could be used as a standard tool on

multiple datasets, requiring little to no human effort apart from operation and occasional human intervention in the tracking process, although this should be minimised. Larger sets of results are needed in order for the extracted characteristics to be statistically representative of all kidney specimens.

Methods for 3D reconstruction from 2D images have been widely established, such as in 3D magnetic resonance (MRI) and computed tomography (CT) scans. However, these tools are not suitable to this problem due to a number of reasons:

- The problem is not limited to visualisation, but requires accurate measurements to be made per nephron, which requires accurate tracking of each individual nephron's cross-sections through the images.
- Existing tools are adapted to isolating only a few objects with relatively simple shapes/contours, e.g. the gross structure of the liver or heart, whereas the kidney has thousands of densely packed, intertwined nephrons, each of which takes a tortuous path in 3D space.
- MRI/CT image sets are not typically as large in volume (hundreds to thousands of high resolution images for the kidney data sets). This poses a challenge in terms of memory.
- Due to the large number of intertwined nephrons surrounded by interstitial tissue, generic algorithms could very easily incorrectly link nephrons or misjudge the correct path.

Existing software packages could perhaps be used on the results of the tracking algorithm rather than the raw images for purposes of visualisation. This research focuses on the development of the methods required for automated tracking rather than research on aspects of nephrology. Developing these methods is an essential step towards fully automated nephron tracking.

The resources that were required for this research project include the image sets and software development tools both of which were readily available. In particular, MATLAB Version R2012a [**33**], the Image Processing Toolbox, Neural Network Toolbox and Statistics Toolbox were used.

3.3 Objectives

The designed system needs to be:

- Automated to a high degree: Minimal effort must be required for setup and calibration, and user input must be minimised during tracking.
- Robust: It is able to track the convolutions in the tortuous path of nephrons and is capable of handling a wide range of cases, accommodating variability in the input data.
- Intelligent: The system makes use of modern techniques and makes informed decisions through computed models rather than depending on hard-coded rules.
- Practical: The code is reasonably efficient and user interaction is made easy.

3.4 Assumptions

- For purposes of verification, the assumption is made that the manually tracked data is absolutely correct. The accuracy of the algorithm will be measured against this gold standard. Visual inspection can also be used to verify results on nephrons which have not been previously tracked.
- The algorithm is only expected to work for datasets with reasonably clear data, which follows the constraints outlined in Section 12.1.2.
- A few parameters can be adjusted at the start of automated tracking in order to optimise the code for a particular dataset, i.e. calibrate the system to the input.

3.5 Success Criteria

The solution will be deemed successful if:

- The algorithm is able to track large portions of the paths of the manually tracked nephrons in an automatic manner.
- Complete nephron paths can be obtained using limited manual intervention.
- The algorithm works with a variety of datasets, with a minimal number of parameters needing to be adjusted.
- The algorithm has high specificity and sensitivity.
- The results can be used to provide a visual representation of the spatial distribution of the nephrons.
- The developed methodologies contribute to future work in this field.

CHAPTER 4

Analysis of the Problem Domain

In order to construct a working solution to the problem, the available data must first be analysed to identify requirements, constraints and limitations posed by the images.

4.1 The Image Sets Acquired from the University of Aarhus

The image sets acquired from the University of Aarhus, Denmark consist of images from three mouse kidney specimens and three rat kidney specimens. According to their previous studies [**9**] [**10**], tissue blocks were cut from each of the six kidneys perpendicular to the longitudinal axis extending from the cortical capsule to the papillary tip [**10**]. The tissue blocks were then fixed with glutaraldehyde, post-fixed with OsO4, stained én bloc with uranyl acetate, and embedded in flat molds in Epon [**9**] [**10**]. The blocks were then sliced transversely into consecutive sections using a microtome equipped with a Diatome histoknife [**9**]. Each slice was then stained with toluidine blue [**9**], digitised using a microscope and digital camera and labelled sequentially. A custom software interface was used for the manual tracking and labelling task; which is discussed in detail in [**9**] [**10**].

The animal experiments were carried out in accordance with the animal care license provided by the Danish National Animal Experiments Inspectorate [**9**] [**34**] (ethics clearance number 2004/561-818). Due to the work being purely computational, additional ethics clearance was not required on part of the University of Witwatersrand.

Some noteworthy characteristics of the datasets are tabulated in Table 4.1.

Table 4.1: Characteristics of the average mouse and rat dataset **[**9**]** **[**10**]** **[**11**]**

	Mouse data	**Rat data**
Isotropic scale factor (x-y)	1.16 μm per pixel	1.53 μm per pixel
Slice thickness	2.5 μm x 0.5=5 μm (every 2nd slice present)	2.5 μm
Average no. of images	984	4392
Resolution	2500 x 1675 pixels	2750 x 2500 pixels

A nephron cross-section is defined as a cross-section through a single nephron at one location in an image. As one proceeds through an image set, it can be seen that the microstructure or morphology changes drastically from the cortex to the medulla. The Appendix contains a reconstructed view of the entire specimen in a longitudinal plane in order to illustrate the regions and the changes between them. Figure 4.1 displays examples of images in the cortex and medulla [**11**].

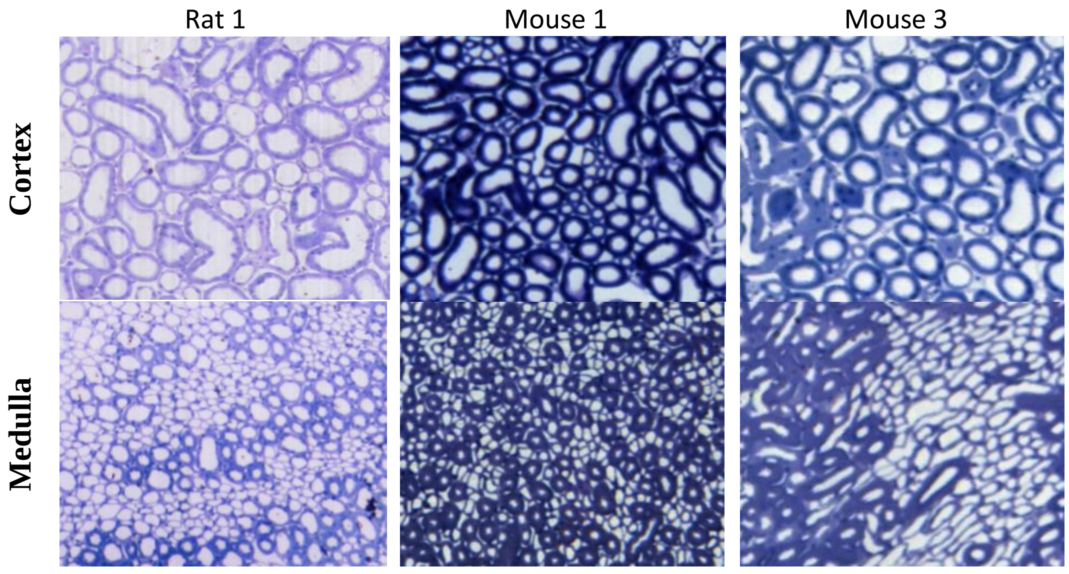


Figure 4.1: Examples of images in the cortex (left) versus the medulla (right) **[**11**]** are shown at equal magnification. A change in nephron characteristics, particularly wall intensity, tubule density and decreasing diameter can be seen.

Histological Variations

Nephrons belonging to the same collecting duct family have their loops running together in the medulla [**10**]. Cortical nephrons have shorter loops of Henle while juxtamedullary nephrons extend deeper into the medulla, have longer loops of

Henle and larger glomeruli [**1**] [**10**]. Different cross-sections of the nephron may stain with different intensities as the cell composition varies. The DTL in particular has very thin, lightly stained walls as can be seen in images of the medulla in Figures 4.1 and 4.2.

Cortex

The renal cortex is composed primarily of the PCT, DCT and glomeruli, which are relatively large in diameter (Glomeruli: 150-240μm, PCT: 40-50μm, DCT: 20-50μm [**2**]) as seen in Figure 4.2. Large blood vessels (arcuate arteries and veins) and smaller capillaries are also present. While most nephron cross-sections appear circular, there are many elliptical and elongated cross-sections in the cortex due to the turning and winding of the PCT and DCT. The PCT is longer, larger in diameter and more convoluted than the DCT [**10**], making up the majority of cross-sections in the cortex. The PCT also has a fuzzier border. The glomerulus, PCT and DCT related to the same nephron are found in the same vicinity in the cortex [**10**]. The glomeruli are randomly dispersed throughout the cortex and are easily distinguishable by eye on the microscopic images as large circles containing a ball of convoluted blood vessels. From observation, the TAL and initial DCT cross-sections are much smaller in diameter than PCT and distal DCT cross-sections and are dispersed in between these larger tubules.

Medulla

The outer medulla contains a mixture of large PCT and DCT cross-sections as well as small DTL and ATL cross-sections. Deeper in the outer medulla, the PST with an outer diameter of about 60μm, suddenly narrows to about 10-15μm and continues as the DTL into the inner medulla [**1**] [**2**].

The inner medulla primarily consists of the thin limbs of the loop of Henle. These are seen as densely packed circular structures. The descending limb has a much thinner wall than the ascending limb. All cross-sections are circular except for small elongated cross-sections at the bends of the loop of Henle. From observation, the surrounding capillary networks called the vasa recta are difficult to distinguish from nephron cross-sections as they are very similar in appearance.

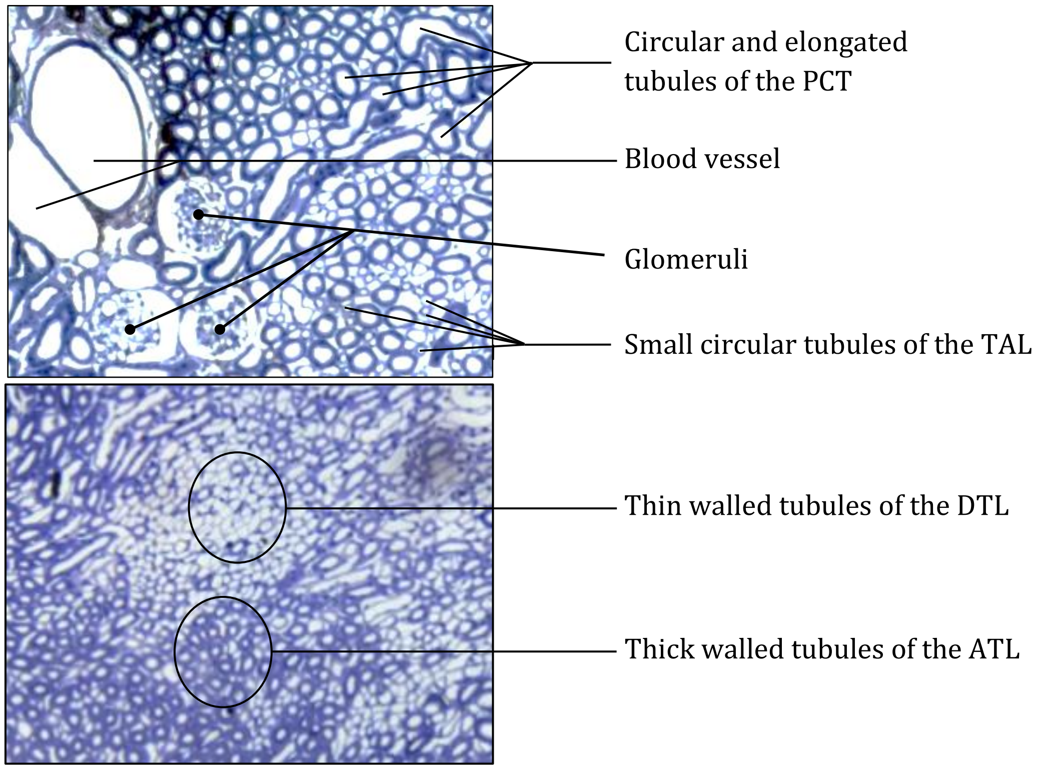

Circular and elongated tubules of the PCT
Blood vessel
Glomeruli
Small circular tubules of the TAL
Thin walled tubules of the DTL
Thick walled tubules of the ATL

Figure 4.2: A section of an image through the cortex (top) and inner medulla (bottom) showing numerous structures **[**11**]**.

The varying structure from the cortex to medulla means that processing parameters will have to change progressively through the image set in order to accommodate the varying intensities, sizes of objects and the amount of unwanted objects such as blood vessels, the interstitial connective tissue, the background, and artefacts.

Analysis of the images from the medulla poses a greater challenge compared with the cortex because the cross-sections are much smaller and concentrated, making it more difficult to isolate them accurately. Even though tracking in this area would be more prone to error (as the probability of mistakenly jumping onto the wrong cross-section is higher), the fact that the paths are mostly straight and unidirectional in this region can be used as a criterion for error checking. Other known information can also be used for guidance or error checking, e.g. slices 1-300 may consist primarily of the cortex, or a diameter of 5-10 pixels indicates a thin limb of the loop of Henle in the medulla.

4.2 An Ideal Solution

An ideal solution would consist of aligning the images and producing binary images using the required conditioning steps. A 3D segmentation algorithm such as region-growing, Watershed segmentation or the flood-fill algorithm could then be applied to the entire 3D volume, ideally isolating a particular nephron given a starting seed. Each isolated volume could then be independently analysed.

However, the data presents many complexities which do not make such a solution viable. Image misalignment, local distortions and missing data (or tissue) between adjacent images produces a definite discontinuity from image to image. This, in combination with interference from connective tissue cross-sections and other non-ideal factors result in multiple nephrons being linked using these techniques. Since there is not continuity between adjacent images (in contrast to the x-y image planes), linkage of the nephron cross-sections merely by pixel connectivity is not reliable and is error prone as it requires only a few pixels to be incorrectly connected from different nephrons. This is especially true for the inner medulla where tubule density is high. Such a solution would also not be capable of intelligently handling distorted images and artefacts. Additionally, these algorithms require the whole volume to be actively processed, which is difficult to carry out as it requires a massive amount of physical memory on the order of 25GB.

4.3 The Complexities of the Problem

Broadly speaking, the complications are firstly due to features of the specimens themselves, and secondly due to the large amount of data per dataset. The designed system must be able to counteract these complexities while accurately tracking the path of each nephron through the 3D image space.

4.3.1 Artefacts

Physical artefacts are structures, or processes, which contaminate or distort the original tissue, causing reduced visibility or complete obscuration of the tissue. They are induced during tissue preparation. An image artefact is an anomaly caused during the image capturing process.

Large physical artefacts seen in the images include tissue cuts, folds and external matter, which affect all the nephron cross-sections in the vicinity. Some artefacts only affect single nephron cross-sections, such as the presence of external matter in the lumen. A number of examples are displayed in Figure 4.3. Artefacts hinder tracking if they occur in a number of successive images. This is typically where user-input is then required.

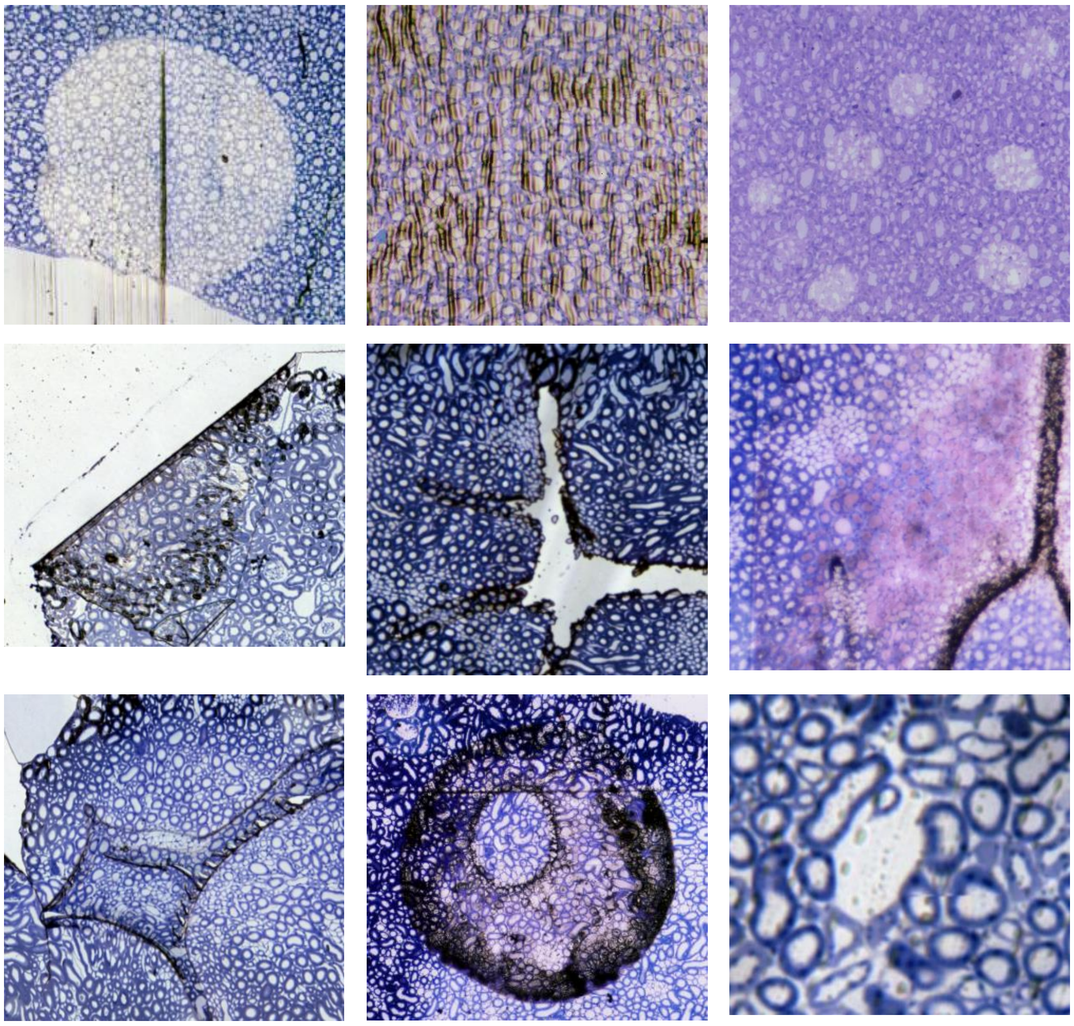


Figure 4.3: Examples of interfering physical artefacts in the image sets [11]. These include cuts, folds, external matter, blurring effects, bright spots and occluding matter in the lumens.

Some images also have areas of sharp non-uniform intensities, particularly large bright spots which could be a result of both non-ideal tissue preparation and image capturing. These cause incorrect merging of cross-sections or elimination of a large number of nephron cross-sections during pre-processing. These images cannot simply be excluded as the frequency of images with artefacts is too high (one in every 5-10 images). Also, the artefacts do not affect all of the nephron cross-sections in the image and the defective images are useful for the most part.

The presence of these vastly different artefacts require each image to be evaluated and processed individually during tracking so that the artefact can be bypassed automatically or by the user. This is another reason why a generic three-dimensional tracking algorithm such as flood-fill cannot be used.

Another anomaly is the misalignment between images. This is due to local tissue distortions (a physical artefact causing non-rigid deformation) as well as capturing slides which were not aligned (an image artefact causing translation and rotation). This is discussed in more detail in Sections 6.1 and 8.1. In addition to the nephrons, interstitial connective tissue and blood vessels are present. Although these are not artefacts, they do cause interference during tracking. Blood vessels link the glomeruli of multiple nephrons, while connective tissue causes the incorrect linking of multiple nephrons during tracking.

4.3.2 Memory

Each image set occupies about 700MB and 2GB for the mice and rat datasets, respectively when stored in a compressed form (JPEG images). In order to be processed in MATLAB (or any software), the images must be decompressed into a matrix form, where each matrix entry is a pixel value occupying at least four bytes. This then equates to a decompressed size of about $\frac{1675\times2500\times900\times4}{1024^3}$ = 14GB for a mouse dataset and $\frac{2500\times2750\times2500\times4}{1024^3}$ = 64GB for a rat dataset.

This implies that it is not possible to process the whole volume at once considering typical physical memory limitations of 8-16GB. Rather, smaller batches of images should be processed in a more intelligent, controlled manner, as is required for the complex nephron path tracking problem.

CHAPTER 5

System Overview

The task of manually tracking nephrons through an image stack is a seemingly trivial one for a human being. However, transferring the vision, interpretation and decision making abilities of the human operator into software is a very complex task. Obtaining results that are as accurate as manual tracking results is even more difficult. In order to attempt to do so, the system developed in this study uses a combination of techniques from the domains of computer vision, feature computation, graph theory and machine learning.

Although the purpose of this system is not to make an "end-diagnosis", from a methodological perspective, the problem fits the generic architecture of a Computer Aided Diagnosis (CAD) system [**35**]. CAD systems assist medical practitioners in interpreting microscope, x-ray, MRI and ultrasound images by automatically marking, measuring or detecting certain regions of interest [**29**]. These systems use a combination of image processing and artificial intelligence techniques. The architecture of a CAD system can be generalised as [**29**] [**35**]:

1. *Image Pre-processing*: Involves steps such as image registration, noise reduction, edge enhancement and intensity equalisation in order to increase quality or amplify visibility of features [**29**].
2. *Definition of Regions of Interest*: Separating or detecting the objects of interest using methods such as image segmentation or contour matching [**29**].
3. *Feature Extraction and Selection*: Computing features by measuring characteristics such as size, shape and colour [**29**] [**36**].
4. *Classification*: Involves pattern recognition through supervised classifiers such as a Decision Tree, Artificial Neural Network or Bayesian Network classifier, or unsupervised methods such as clustering.

A number of factors affect the accuracy of these systems, for example image quality, noise and complexity of the target objects [11]. Figure 5.1 describes the architecture of the designed nephron tracking system.

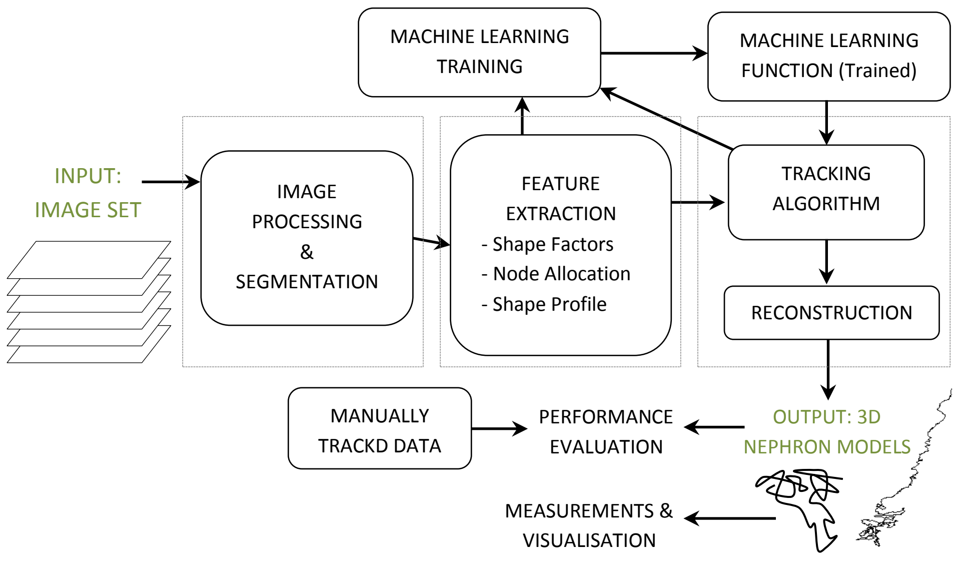


Figure 5.1: A high level overview of the nephron tracking system, showing the main sub-systems and the flow of information between them.

The system is implemented in MATLAB [**33**] as a series of independent modules where structures of information are progressively passed on 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.

There are a number of parameters in each stage which need to be calibrated to each image set. These are discussed in their relevant sections. In order to easily do so, a single *settings* file must be initialised prior to execution, which contains all of the information needed to automatically adjust and vary the parameters involved.

CHAPTER 6

Image Processing

Computer vision aims to mimic the capabilities of human vision by processing, analysing and transforming raw images into a form that can be more easily and accurately interpreted by a machine [**36**]. It forms a crucial component of many automated processes in the real world [**37**] including the current nephron tracking task.

The image processing steps prepare the images for subsequent stages by creating uniformity among all nephron cross-sections and counteracting non-ideal factors described in Section 4.3. The images are processed such that required features (nephron cross-sections) are enhanced while unwanted features (such as interstitial connective tissue (ICT) cross-sections, large blood vessels, background pixels and large artefacts) are filtered out or reduced. The final product of image pre-processing is a binary image of the lumens of the nephrons as shown in Figure 6.1.

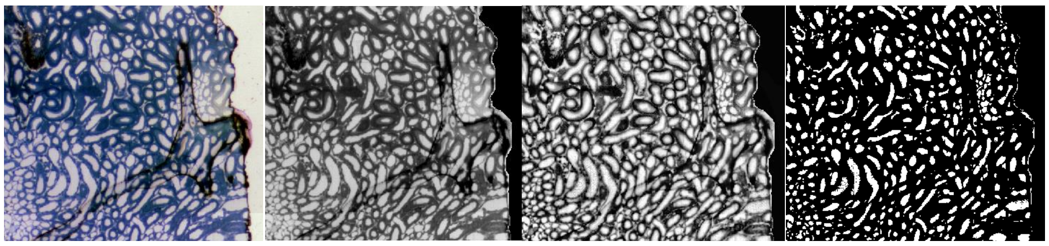


Figure 6.1: Each colour image is processed into a binary image containing nephron cross-sections of all sizes. Each raw image **[**11**]** undergoes conversion to grayscale, background removal, histogram equalisation and binarisation.

6.1 Image Registration

Image alignment was carried out on the datasets [**11**] during the previous study in order to ease the manual tracking process [**9**] [**10**]. The procedure involved iteratively estimating the translational and rotational offsets between adjacent images and applying the rigid transformation using custom software written in C

[**9**]. This alignment was apparently not sufficient due to the local distortions induced during the sectioning process [**9**]. The distortions have the effect of pinching, compressing or stretching local regions of tissue. The rat image sets were then further aligned using five manually placed landmarks which divided each image into four polygons, each of which then underwent a non-rigid transformation [**9**].

These processes have resulted in the images being sufficiently aligned from a global perspective. However, local distortions in the mouse datasets were still not fully compensated for, especially since only every second slice of the dataset [**11**] is present. This is shown in Figure 6.2, where one local area can be aligned while a nearby area is misaligned.

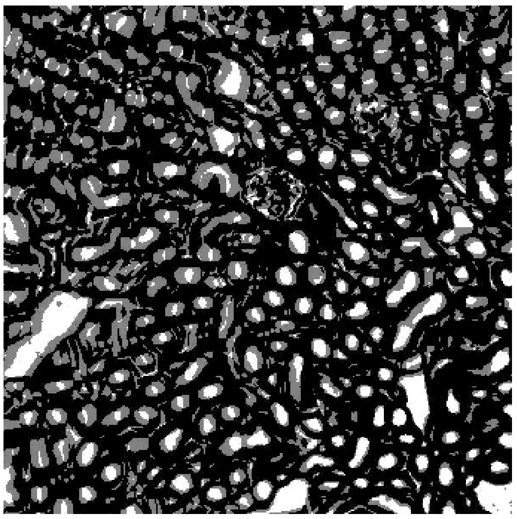


Figure 6.2: A pair of superimposed adjacent sub-images (binarised) from a mouse dataset is shown (derived from **[**11**]**). The bottom right area is well-aligned while the top areas are misaligned. This cannot be corrected using a translation and rotation only as the misalignment is due to localised stretching/compression.

Misalignment due to local distortions in the rat datasets was minimal as they were compensated for by the four-quadrant alignment method. However, this had resulted in nephron cross-sections incorrectly merging at the junctions of the four polygons, as shown in Figure 6.3. This resulted in multiple nephrons being linked during tracking. The nephrons around these junctions were therefore excluded from the study as the merge cannot be reversed.

The images were not further aligned during the pre-processing stage, although further alignment is performed during the tracking process as the local distortions require the areas around each cross-section to be handled locally and independently. This local alignment is discussed in Section 8.1 as part of the tracking system. Advanced non-linear image registration techniques such as RANSAC [**37**] were not applied as:

- Cropped local regions can be aligned using simpler methods. Non-linear alignment usually makes use of six more parameters in addition to the two employed (x and y translation), which increasing the order of the process.
- Large cumulative transforms over the image set must be avoided as they may over-morph the images.
- A small amount of the misalignment is due to the progressive change in morphology and not only due to induced distortions. A non-linear registration would counteract this change in morphology, which is undesirable as the characteristic nature of the nephrons must remain unchanged.

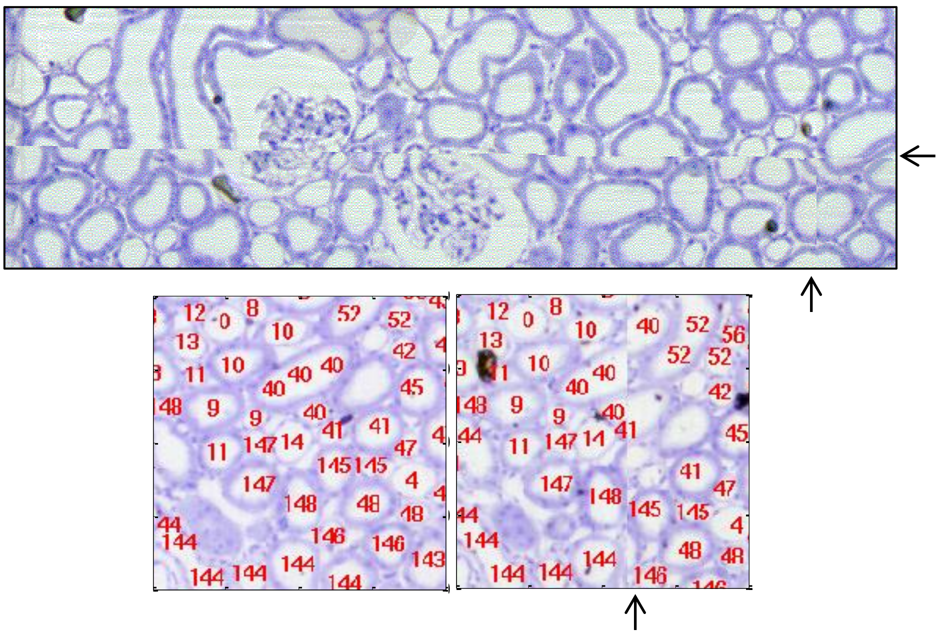


Figure 6.3: The arrows indicate the junctions of the polygons created during the four-polygon alignment method, which results in the merging of cross-sections from different nephrons. In the labelled image (below) the merge between cross-sections from nephrons 40 and 41 can be seen **[**11**]**.

6.2 Image Processing Procedure

Each nephron tubule consists of a lumen enclosed by the tubule wall, which differs in thickness depending on its location, i.e. the PCT, PST and DCT have thick walls while the DTL, ATL and TAL have very thin walls. It would be ideal to extract both the wall and lumen of each tubule but this is a difficult task due to the walls of adjacent tubules touching one another. One potential method which could be applied is spline curve fitting using a genetic algorithm, which has been used to isolate different types of tissue in histological images [**19**]. However, the vast number of single nephron cross-sections per image that would need separation is too large (≈ 8000 per image in the cortex to ≈ 36000 per image in the medulla) and the problem becomes unnecessarily complex for current purposes.

It was decided that the lumen of a nephron cross-section alone contains sufficient amount of information to represent the original structure in the colour image, i.e. location, size and shape of the nephron cross-section is provided by the lumen alone. The lumens are also more easily and accurately isolated juxtaposed to the walls of the nephrons and are thus chosen as the objects to be isolated. Each image undergoes the following procedures:

6.2.1. Conversion to Grayscale

The staining used on the specimens (toluidine blue [**10**]) results in all structures being monochrome. The colour information is thus discarded by conversion to a grayscale image by retaining the value component (or luminance) of the hue-saturation-value (HSV) image. The colour information could however be useful (e.g. if a more differentiating stain is used in future image sets) and this would require the pre-processing stage to be modified accordingly.

6.2.2. Background Removal

The tissue slice is isolated by removing the white background space. First, the image is thresholded at the image's average intensity value plus some constant C.

$$T_{\mathrm{bgrnd}}{}^{z} = \mathrm{mean}(I_z) + C \tag{6.1}$$

This is chosen instead of a constant value only as each image differs in intensity, some by a large amount. Furthermore, this value results in a sharp contrast between the background (BG) and the tissue. The C value must be chosen to suite each image set. For example, the images in one rat dataset have a very large bright tissue centre. A C value that is too low causes the nephron cross-sections to merge into one large binary element when binarised. The large component could then be mistaken for the background. Another mouse dataset has a darker background with lots of matter, and a C value that is too high results in large chunks of the background not being removed. This value must be chosen once-off during system calibration by a trial-and-error approach.

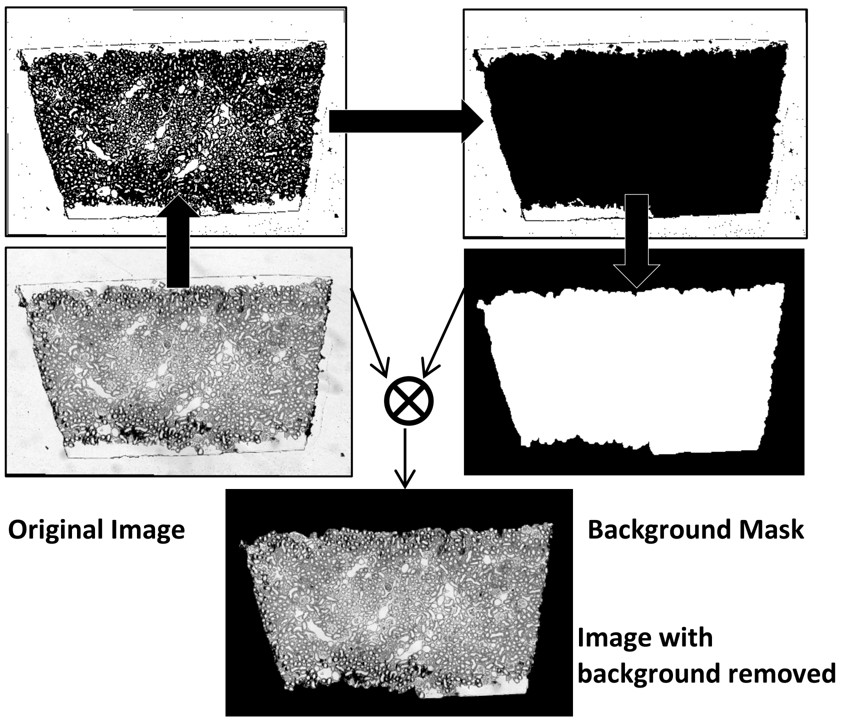


Figure 6.4: The procedure for background removal is shown. The raw image [11] is binarised. The background mask is formed by morphological closing and inversion of the largest components in the binary image. Finally the mask is multiplied with the image.

The binary image is segmented (using simple 8-neighbour connectivity), thereafter obtaining the largest cross-sections which then form a background mask. The mask first undergoes morphological image closing using a 20x20 circular kernel in order to remove small objects occurring in the background. The mask is then inverted and applied to the original image by multiplication. These steps are shown in Figure 6.4. Background removal must occur prior to (and without any) image equalisation so as not to amplify the intensity or texture of matter occurring in the background.

6.2.3. Histogram Equalisation

Histogram equalisation involves normalising the histogram of an image such that all intensity values are equally distributed among the pixels in the image. It is the most crucial image processing step and is required in order to counteract uneven intensities on both a local and global scale as illustrated in Figure 6.5.

1. ‘Globally’: Uneven intensity may occur across large areas in the image, such as random large bright spots or a brighter centre as a result of the image acquisition process. Furthermore, some images are irregularly bright or dark in comparison to the rest of the image set. Global equalisation is achieved by using a large equalisation window [**36**], about a tenth of the size of the image.

2. ‘Locally’: Uneven intensities may occur in small local areas (especially in the inner medulla) as a result of narrow diameter nephron cross-sections having a much lighter wall. Local equalisation is applied by using a much smaller equalisation window of about 5 times the size of the average nephron cross-section in the image.

The sizes of the equalisation windows must be suited to each image per dataset. Images of the cortex require a large local equalisation window (≈ 40 pixels), as the nephron cross-sections are larger than in the medulla. Too small a window results in ‘hyper-equalisation’, where large white areas (such as large nephrons) acquire a rough, broken texture. The nephron cross-sections in the inner medulla are much smaller and have very thin, light tubule walls, and are thus much more dependent on good equalisation. This requires a smaller local equalisation window (≈ 20 pixels). A window that is too large will not adequately equalise the intensity of these nephron walls, resulting in the cross-sections being removed or merged when binarisation occurs.

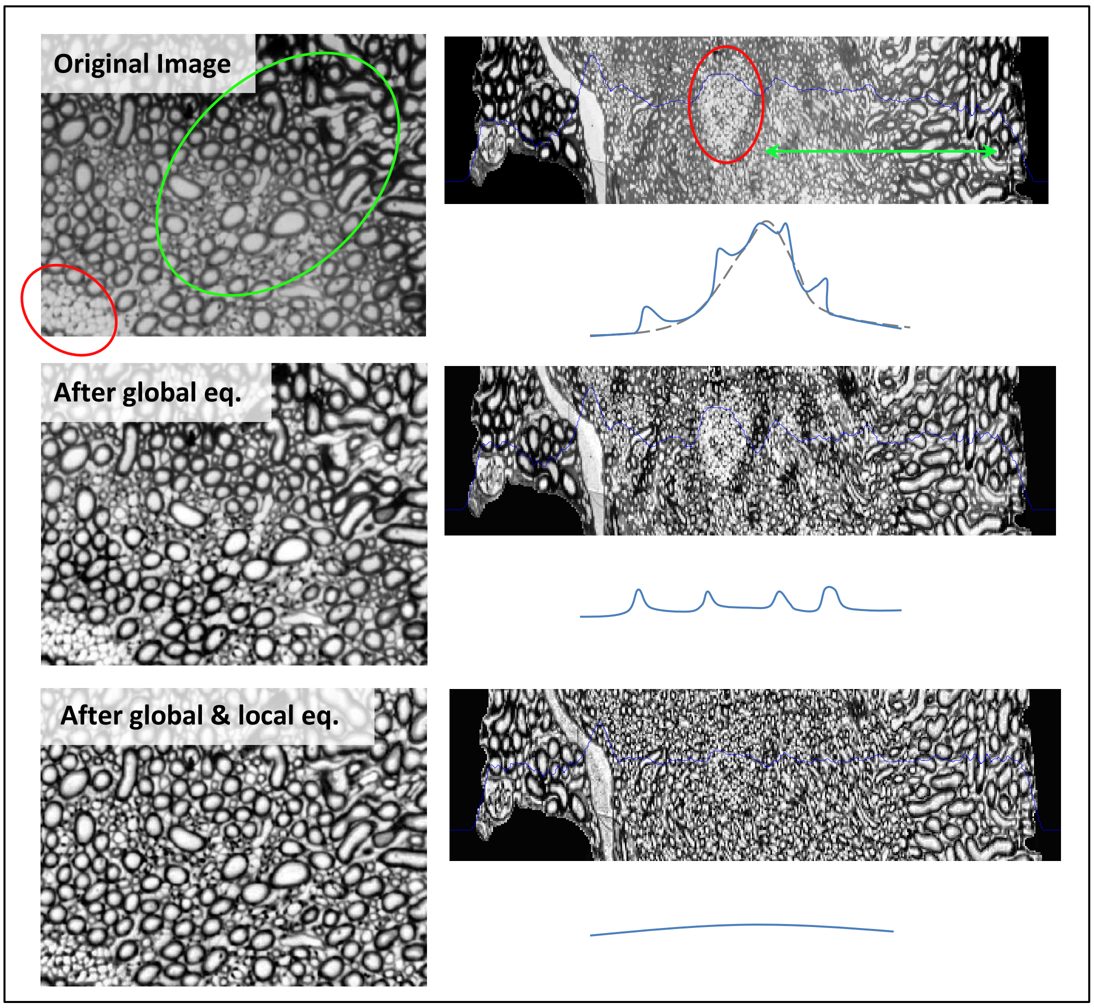


Figure 6.5: Top: There are large regions of uneven intensity (shown in green) which require equalisation with a large window. There are also much smaller, local uneven intensities due to thin walled cross-sections. If not equalised locally, these groups of nephrons will merge into large binary cross-sections and would not be able to be differentiated. Middle: After global equalisation, the image intensity is uniform over large areas as seen in the real and conceptual histograms. Bottom: After local equalisation, uniform intensity is achieved across small areas as well. Images adapted from [**11**].

6.2.4. Thresholding

The image is thresholded at a constant value T to create a binary image. Adaptive thresholding is not used as uniformity was achieved through the equalisation steps. The threshold value is chosen such that it does not allow independent lumens to merge while also not letting small nephron cross-sections disappear. It also varies through the image set such that the inner medulla images have a slightly higher value to prevent the dense, thin-walled cross-sections from merging, while the cortex has a lower value to prevent segmentation of large nephron cross-sections.

6.2.5. Removal of Unwanted Cross-Sections

Unwanted cross-sections include those of the blood vessels and connective tissue. Connective tissue cross-sections appear between nephron cross-sections in the cortex. They are characteristically irregularly shaped, fragmented and typically small and thin, as shown in Figure 6.6. It is difficult to remove these cross-sections without also removing some nephron cross-sections which are similar in appearance (particularly cross-sections of the TAL and DCT). It is important not to remove these nephron cross-sections as the TAL and DCT will otherwise not be able to be tracked.

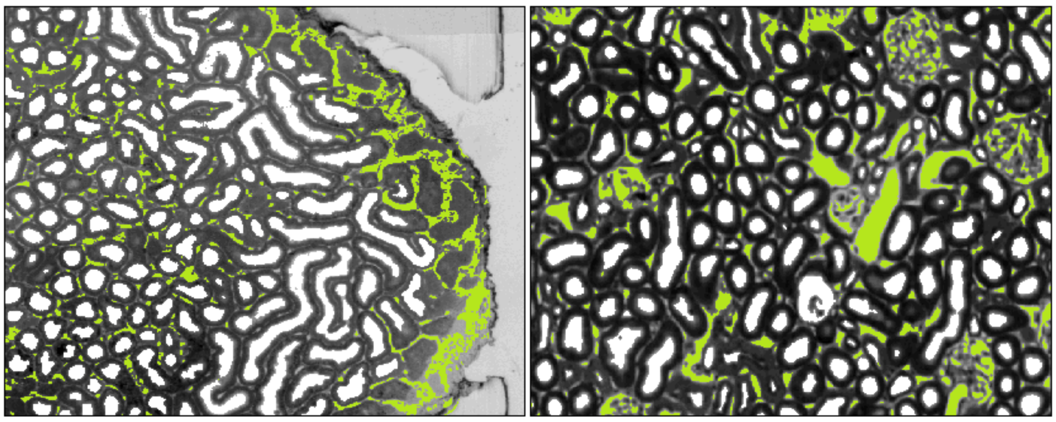


Figure 6.6: Connective tissue cross-sections have been manually marked in green in images of the cortex from a rat (left) and a mouse (right) image set (adapted from **[**11**]**). The irregularly shape connective tissue cross-sections are distinct, although there are some nephron binary cross-sections which have similar characteristics.

Size-based component exclusion: Binary components that are very small (<10 pixels) and very large (> 100 000 pixels) can be confidently identified not to be nephron cross-sections and are removed. Large components are normally large blood vessels while small components are small connective tissue cross-sections or noise from the equalisation.

Morphological erode/dilate cycles: Performing morphological eroding *p* times, followed by a dilation *p* times results in the elimination of larger connective tissue cross-sections. While this works well, it is applied very sparingly as it also results in the removal of small nephron cross-sections. This cannot be done in the inner medulla where the cross-sections are only a few pixels wide.

Shape-based clustering: Clustering the cross-sections based on their shape factors is another possible way to eliminate unwanted cross-sections.

Morphological operations and clustering invariably results in the loss of some nephron cross-sections as well (usually elongated or C-shaped cross-sections). This is highly undesirable, as the tracking process depends on each cross-section along the nephron's path to be present, especially elongated ones. It is therefore decided to include all possible data (including unwanted ICT) rather than to have missing data.

Obtaining the final binary image is one of the most important tasks, as the accuracy of following stages depends on how well cross-sections are isolated from one another. A compromise must be made between the number of connective tissue cross-sections present and the number of small nephron cross-sections which do not get eliminated.

Further pre-processing involves the removal of highly distorted images and replacing them with the image above or below. An average of 4 images per dataset had to be been manually replaced. However, an automatic method can be devised if a larger number of images are defective, for example by analysing the mean intensity of each image in the image set.

6.3 Image Segmentation

Connected component segmentation [**36**] (using a 4-connected neighbourhood) is used to segment the image into independent nephron cross-sections. Although this is the simplest segmentation technique, it produces satisfactory results because a good binary image was obtained in the previous steps. A labelled binary image is formed, where all the pixels belonging to one component have a unique value.

Watershed segmentation or region-growing on the equalised image are other possible segmentation techniques, which could perform better in cases where independent lumens incorrectly merge through a few connected pixels. However, they do tend to over-segment the image [**17**], resulting in the division of elongated nephron cross-sections.

6.4 Automatic Parameter Variation

A very important factor during pre-processing is accommodating for the change in morphology from the cortex to the medulla. The cortico-medullary boundary is an area where the proximal straight tubule (≈ 60μm in diameter) suddenly narrows to a diameter of 10-15μm to form the thin descending limb of the loop of Henle [**1**] [**2**]. This change requires almost all parameters of the pre-processing steps to vary accordingly to ensure that cross-sections of all sizes are extracted.

In order to accommodate for this relatively sudden change, the parameters are made to vary along the image set according to a modified sigmoid function $Y(n)$ (also known as a generalised logistic function [**38**]) as in equation 6.2, which has its inflection point set at the transition zone. This transition zone must be chosen empirically through observation during system calibration.

$$Y(z) = \frac{UL-LL}{1+e^{\frac{k-z}{\delta}}} + LL \qquad (6.2)$$

Where z is the image number, UL is the upper limit, LL is the lower limit, k is the z value at which the inflection point occurs and δ is a steepness coefficient [**38**].

The steepness coefficient is chosen by the best outcome during experimentation on a few images. The sigmoid function allows relatively constant parameter values in the cortex and inner medulla as shown in Figure 6.7. Some parameters of the tracking algorithm, such as the tracking radius, also make use of this function.

The suitability of this function can be validated by examining the change in certain characteristics along the image set, such as the number of cross-sections and average nephron cross-section width. As shown in Figure 6.8, these characteristics have an inherent sigmoidal shape; hence the function models the change well. The curve seen near the initial images is due to the tissue slices progressively growing from the edge and is not due to differing tubule characteristics, and is therefore not modelled.

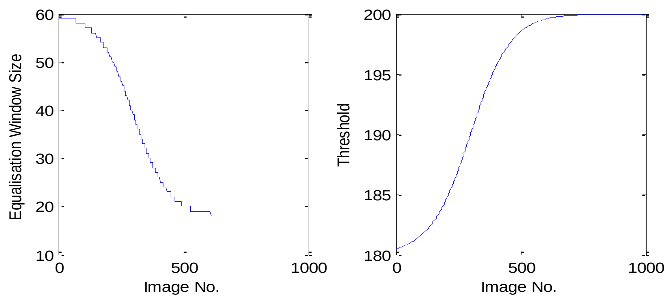


Figure 6.7: The equalisation window size and threshold value are made to vary according to custom sigmoid functions. The equalisation window is quantised. The parameters are for a mouse dataset with its inflection point at the 350th image.

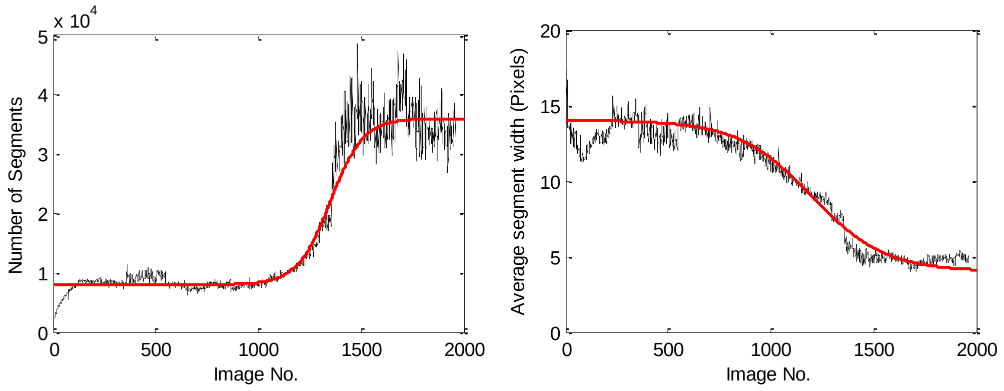


Figure 6.8: Features of the datasets (black) such as the average nephron cross-sectional width (right) and number of cross-sections per image (left) have an inherent sigmoidal characteristic, as shown in red. The graphs are of a rat dataset.

Modelling the processing parameters using tailored sigmoid functions for each image set results in uniform results (quality and accuracy of binary images) across all datasets, i.e. it serves as a calibration mechanism. Subsequent stages therefore do not have to cater for any particular dataset and can be designed in a generalised manner for standard input data.

CHAPTER 7

Feature Extraction

Feature extraction aims to simplify and concentrate useful information from raw data. It forms an *intrinsic image* of the input data, which is an array of information representing important physical characteristics of the objects involved [**37**]. 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 nephron cross-section is its size, shape, colour and location.

7.1 Node Allocation

A *node* is defined as a point coordinate in the 3D image space. The pixel locations per cross-section can be reduced into a set of nodes allocated along the cross-section, hence modelling their location. For example, a circular nephron cross-section can be represented by one central coordinate, instead of a few hundred pixel locations, and an elongated cross-section can have multiple nodes allocated along its length. This abstraction greatly simplifies the problem, reduces the size of the data, decreases the computational load on subsequent stages and concentrates significant information.

Node allocation was first achieved by using the circular Hough transform [**37**] to locate circles occurring within a certain radius range in the binary images. At first glance the results seemed good, but deeper inspection revealed that many cross-sections were not allocated a node, and elongated cross-sections were not handled adequately. This method was computationally expensive due to its iterative, analytical nature and offered no control over how many nodes would get allocated per cross-section.

Finally, K-means clustering was used to allocate nodes. K-means clustering is an unsupervised learning method which groups *m* observations into *k* clusters [**39**].

Each cluster is defined by its centroid, which is the mean of all the observations assigned to it. These centroids are used as the nodes and are found by solving the optimisation problem in equation 7.1.

$$\mathbf{C} = \arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \epsilon \mathbf{S}_i} \|\mathbf{x} - \mu_i\|^2 \tag{7.1}$$

Where **x** is the observations, **S** is the observations partitioned into k sets $\{S_1, S_2, \ldots S_k\}$, μ_i is the mean of the observations in S_i and **C** = $\{c_1, c_2, \ldots c_k\}$ is the set of k centroids [**39**].

Each non-zero pixel on an isolated binary cross-section is made an observation. Shape criteria (area and circularity, which are discussed in Section 7.2.1) are used to decide on the number of nodes (K) to allocate to a cross-section. Cross-sections with an area below 5 pixels are ignored (K=0) as these are typically pixels due to equalisation noise. If the cross-section is circular (circularity>0.95) or small (area<300), one centroid is requested (K=1). For elongated cross-sections, the requested K value begins at 2 and is incremented until the average pairwise distance between adjacent nodes is less than a desired minimum value. This value is chosen to be close to the average cross-sectional width in the current image ($\approx$ 20 pixels in the cortex and $\approx$ 8 pixels in the medulla). This ensures that an adequate number of nodes are allocated per cross-section depending on its size and shape. Figure 7.1 displays examples of allocated nodes on cross-sections of various shapes and sizes.

A characteristic of the K-means clustering method is that it groups data into Voronoi cells [**39**], which is suitable to the node allocation problem. Some disadvantages are the susceptibility to local minima and varying results due to randomised initial conditions [**28**]. This is overcome by running the clustering multiple times per cross-section until certain conditions are met. An important advantage of the method is that a desired number of centroids can be requested.

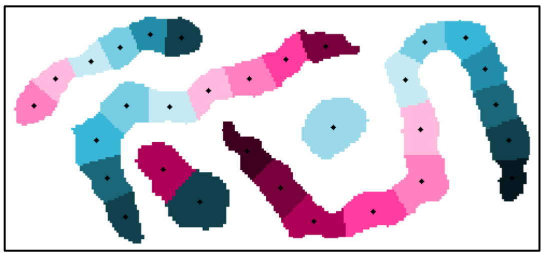


Figure 7.1: K-means clustering divides cross-sections into similarly sized Voronoi cells. The centroids from K-means clustering are used as nodes for the cross-sections. A suitable number of nodes are allocated automatically per cross-section.

7.2 Shape Measurements

Tracking of a nephron using only the 3D set of nodes results in the linkage of multiple nephrons, blood vessels and connective tissue due to the close proximity of the intertwining nephrons. By only considering the point cloud, the algorithm is blind to a large amount of available information. Shape information about each cross-section is thus also captured. The idea behind incorporating shape information into the tracking is to retain information about the original cross-sections so that the algorithm can make intelligent, confident and informed decisions at each incremental step of the process.

7.2.1 Shape Factors

A shape factor, or metric, refers to a dimensionless value that is dependent on an object's shape but is independent of its size [**40**]. It usually indicates the degree to which an object deviates from an ideal shape, such as a square or circle [**40**]. Various shape factors are calculated per cross-section to capture abstract information about each cross-section along with its nodes.

Shape metrics are calculated using various measurements of an object. Primary measurements include the object’s area, perimeter and chord lengths [**40**]. Secondary measurements which are useful include the convex area (the area of a polygon of the lowest degree which covers the object), equivalent diameter, centroid and moments of the object about the centroid [**37**] [**40**]. The first moment is the mean and the second is the variance. Moments are used to calculate an equivalent ellipse which has the same variance as that of the object [**37**]. The minor and major axes are then the perpendicular axes of the aforementioned ellipse, and the orientation is the angle that the major axis makes with the horizontal axis [**37**].

Shape metrics combine these measurements to describe a number of shape characteristics. 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. These values are extracted using the MATLAB *regionprops* function [**33**]. The shape factors are briefly described:

The *aspect ratio* is the ratio of the major axis to the minor axis and indicates the length and symmetry of an object [**36**]. For a circle or bisymmetric object, the aspect ratio is 1 while it is >1 for an elongated object.

Circularity measures how close an object's shape is to a circle [**36**] and is given by equation 7.2. A value close to 1 indicates a very circular object while an elongated object has a value near 0.

$$\mathrm{f_{circ}} = \frac{4\pi \mathrm{Area}}{\mathrm{Perimeter}^2} \tag{7.2}$$

Solidity is the ratio of the area to the convex area of an object, and measures its virtual hardness or density [**36**]. Generally, this area is larger than or equal to the area of the object. Objects with holes and multiple concave areas have a low solidity. Waviness is also sometimes used, where the perimeters of the object and polygon are used instead of the areas [**36**]. Waviness does not take holes into account.

Eccentricity is typically associated with an ellipse, being defined as the ratio of the distance between the foci of the ellipse and its major axis length [**37**]. Using moments, an equivalent ellipse can be calculated for an arbitrary shape, hence an eccentricity value. Eccentricity is a value between 0 and 1, with 0 indicating a circular shape and 1 indicating an elongated shape.

The *extent* is an area ratio between the object's area and the region area, where the region is the bounding box of the object [**37**]. It generally describes the degree to which the object fills the bounding box. Extent can give a combined indication of elongation, orientation and solidity [**37**]. It is useful for detecting highly irregularly shaped objects. Note that this metric is orientation sensitive, e.g. an elongated object lying horizontally or vertically will have high extent, while it would have low extent if lying at 60^0.

7.2.2 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 irregularly shaped, such as glomeruli or connective tissue cross-sections. Also, a move from one cross-section to another cannot be adequately described by merely comparing shape factors. As an additional feature the shape profile around each node is calculated.

The shape profile of an object is a polar plot of the distance to its boundaries with respect to a reference point [**36**]. It is commonly used to compare the structure of two objects. It transforms a 2D shape representation into a 1D plot [**36**], hence reducing computational effort during matching. It allows shape comparison of objects of different sizes by normalisation of the distance.

First, the edge of the cross-section is obtained using a Sobel edge detector [**37**]. 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 equations 7.3 and 7.4.

$$\boldsymbol{\theta} = \arctan\left(\frac{\mathbf{P_{edge}}(y)-\mathrm{P_{ref}}(y)}{\mathbf{P_{edge}}(x)-\mathrm{P_{ref}}(x)}\right) \tag{7.3}$$

$$\mathbf{r}(\theta) = \left\|\mathbf{P_{edge}} - \mathrm{P_{ref}}\right\| \tag{7.4}$$

where $\mathbf{P_{edge}}$ is the vector of edge coordinates, $\mathrm{P_{ref}}$ is the reference coordinate, $\boldsymbol{\theta}$ is the vector of angles and $\mathbf{r}(\theta)$ is the vector of radial distances.

If the object is concave, the plot will be multivalued [**36**]. This redundancy increases the order of the matching procedure back to 2D. This problem is solved by simply taking the nearest r value for a given θ, and is called *unwinding* [**36**]. In order to produce a consistent feature set, the radius at constant angle increments is interpolated. The degree of abstraction is dependent on the chosen angle increment δ [**36**]. The unwinding and interpolation is shown in Figure 7.2.

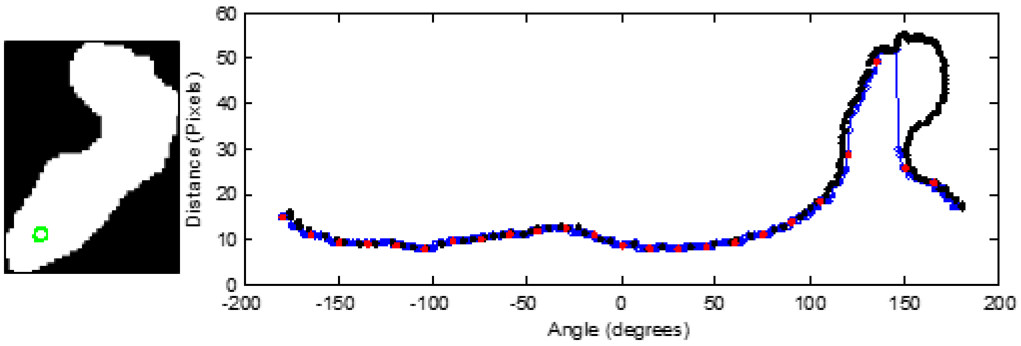


Figure 7.2: The shape profile of the cross-section on the left is shown, with the reference point indicated by the green dot. The original shape profile, shown in black, is unwound to form the plot in blue. This is then interpolated to acquire the distances at desired angle increments (15°), forming the final shape profile shown in red.

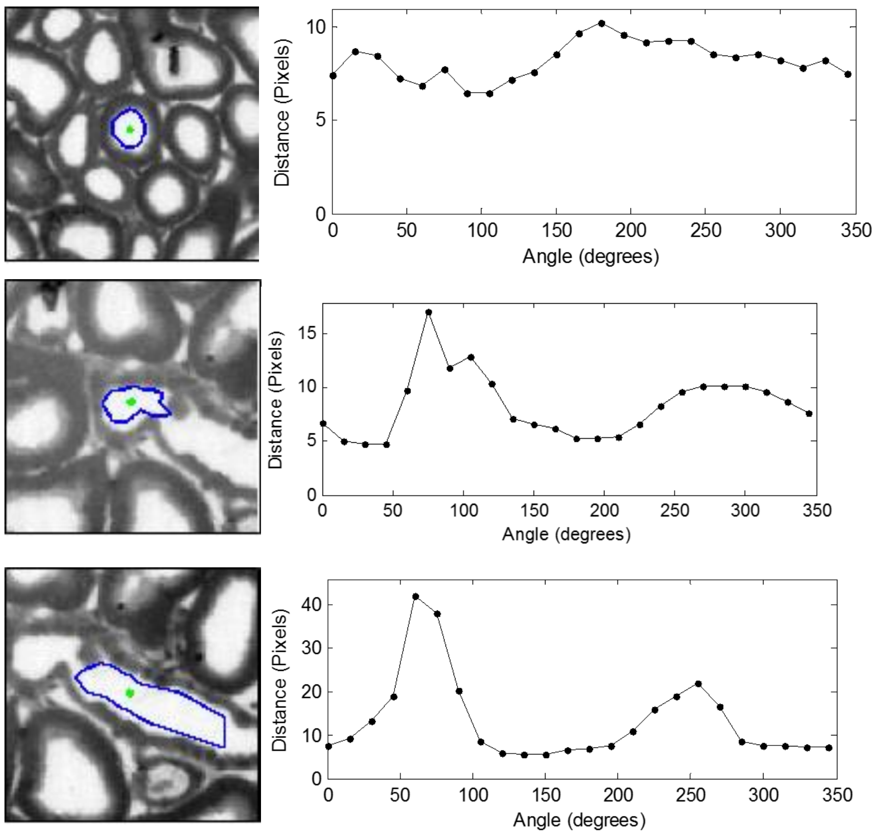


Figure 7.3: The shape profiles relative to nodes on the cross-sections **[11]** (green dots) are shown. 15 degree increments were chosen.

Another problem is pixel quantisation leading to discontinuous angles in small sized cross-sections. This problem is solved by scaling each cross-section up to 50x50 pixels prior to shape profile calculation. The final **r** vector is then down scaled. Examples of shape profiles are shown in Figure 7.3

The shape profile of a given object will be different depending on the chosen reference point. The centroid is commonly selected [**36**], but the allocated nodes have been chosen instead as they are more relevant to the problem and allow an accurate relative comparison of shape profiles between linked nodes. This is illustrated in Figure 7.4.

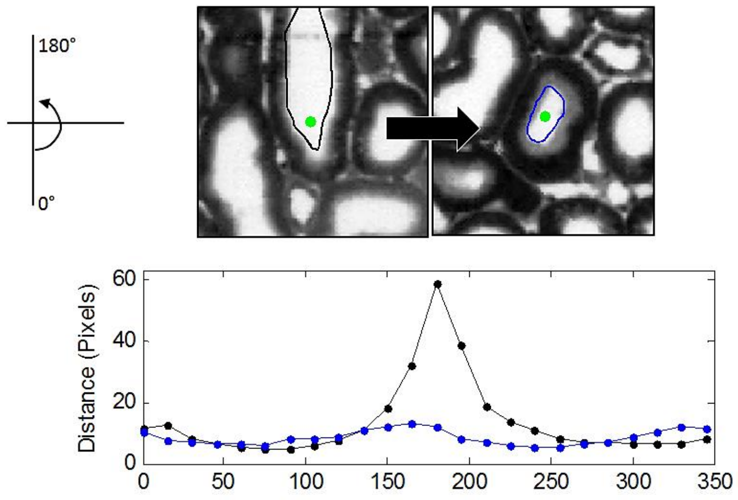


Figure 7.4: If a tracking iteration attempted to move from cross-section 1 (top left) to cross-section 2 (top right), the shape profile of the nodes involved would need to be compared, rather than the profiles around the centroids. As can be seen, when the respective nodes are used, the shape profile is similar for a large range of the angles and can hence be compared. The move shown is a legitimate move between nephron cross-sections of the same nephron. Images adapted from **[**11**]**.

Correlation of the shape profiles (equation 7.5) and a measure depending on their absolute difference (equation 7.6) are two similarity measures which can be used to compare a pair of shape profiles.

$$S_{\mathrm{r}_1,\mathrm{r}_2} = \mathrm{corr}(\mathbf{r_1}(\theta), \mathbf{r_2}(\theta)) \tag{7.5}$$

$$S_{\mathrm{r}_1,\mathrm{r}_2} = 100 \times \frac{360}{\delta} \times \mathrm{count}(|\mathbf{r_1}(\theta) - \mathbf{r_2}(\theta)| < 3) \tag{7.6}$$

Where S_{r_1,r_2} is the similarity measure between profiles r1(θ) and r2(θ), corr is the correlation function, count is a function returning the number of elements satisfying the condition in its argument and δ is the angle increment.

7.3 Data Structures

Each cross-section gets assigned a group of nodes and shape metrics with a shape profile per node as shown in Figure 7.5. Ideally, all the information belonging to a node should be stored along with it. However, the nodes, shape factors and shape profile occupy different amounts of memory and require different access speeds.

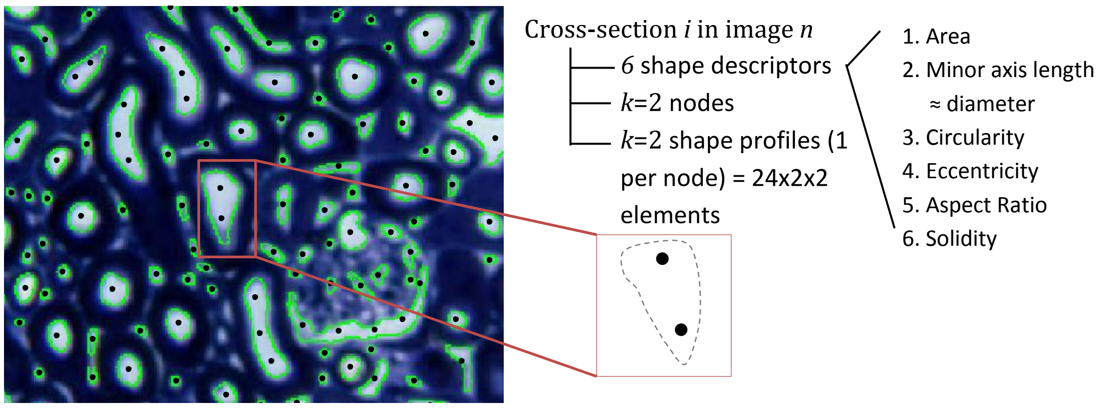

Cross-section *i* in image *n*
6 shape descriptors
k=2 nodes
k=2 shape profiles (1 per node) = 24x2x2 elements
1. Area
2. Minor axis length ≈ diameter
3. Circularity
4. Eccentricity
5. Aspect Ratio
6. Solidity

Figure 7.5: A clip of a raw image is shown **[11]**. The extracted cross-sections after pre-processing 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. As can be seen, many cross-sections in the cortex are not of actual nephrons but rather of the connective tissue between them. The glomeruli are also highly segmented.

The nodes need to be constantly and quickly accessed during tracking. Since they simply consist of x-y-z coordinates, they can be stored in working memory (RAM) for efficient access. In order to link the nodes with the other information, each cross-section is allocated an identity number, and each node is allocated an identity number relative to the cross-section it lies on. The nodes are stored in a cell array with each cell containing a fixed array for the node data for one image. Each row in the fixed array contains information about one node. Nodes lying on a common cross-section have the same identity number (ID), hence allowing for easy detection of elongated cross-sections during tracking.

The shape factors are accessed often and consume moderate memory. It is thus stored in working memory in the same manner as the nodes. Each row of the fixed arrays corresponds to the row for the nodes and is hence implicitly linked.

The shape profiles occupy a massive amount of memory as they consist of 50 elements per node (24 angles, 24 distances, 1 cross-section ID, 1 node ID). The shape profiles of an average rat dataset would occupy $2500\text{images} \times 20\,000\,\frac{\text{nodes}}{\text{image}} \times 50\,\frac{\text{elements}}{\text{node}} \times 8\,\frac{\text{bytes}}{\text{element}} = 18.63\text{GB}$ if stored uncompressed in working memory. Since they are only accessed during machine learning validation, they are stored on hard disk and only the required elements are accessed using an input-output *matfile* structure in MATLAB.

The chosen storage format together with custom-coded utility functions (see Appendix) provides ease of access and complete traceability of information related to any particular nephron cross-section.

7.4 Glomeruli Detection

The glomeruli can be distinctly located by eye and possess unique characteristics which can distinguish them from other tissue. The glomeruli have a characteristic rough texture from the clump of tiny entangled blood vessels. A C-shaped white space is present in most glomeruli. This is the urinary pole which is often seen fusing with the nephron tubule in the span of 1-3 images. A glomerulus spans 15-30 images in the mouse datasets and 50-70 images in the rat datasets, depending on the depth into the kidney (cortical glomeruli are smaller while juxtamedullary glomeruli are larger [**1**]).

In order to fully model a glomerulus, an ellipsoid should be fitted in the 3D image space to cover its full volume. For purposes of tracking, the presence of a glomerulus must be detected when the nephron tubule merges with the urinary pole, so that tracking can be terminated. This termination is critical as it has been seen that the afferent/efferent arterioles are sometimes tracked out of the glomerulus, through connecting vessels and to the glomeruli of other nephrons.

This linking of multiple nephrons is highly undesirable. Two methods suitable to the given image sets have been devised for glomeruli detection.

The first method involves measuring the peak densities of an edge image. A log edge detector is applied to the binary image. In the edge image, the glomeruli have more edge pixels than the surrounding nephrons due to the numerous small binary fragments making up the glomerulus. Therefore, the average pixel density at the glomeruli is higher than elsewhere. An averaging filter is applied after which the peaks are extracted. Distance-based clustering is then used to obtain single point representations of the glomeruli.

The second method involves grouping alike cross-sections using shape factors as features during unsupervised learning. K-means clustering is used. If 5 clusters are chosen, clustering groups the cross-sections into those that are:

1. very circular – high circularity
2. slightly elliptical - low eccentricity and circularity
3. very elongated – high eccentricity and aspect ratio
4. very small in size – small area
5. concave, C-shaped structures – low solidity

The last class distinctly contains most of the C-shaped urinary poles of the glomeruli, along with some C-shaped nephron cross-sections.

An example of the results produced using these two methods is shown in Figure 7.6. Both methods have false positives and negatives but could perhaps be combined to improve accuracy. Issues with these methods include:

1. Nephrons become smaller deeper into the image set and hence their mean edge values go up as well, producing false positives. Connective tissue also sometimes looks like glomeruli. Parameters of the algorithm need to be varied per image to get accurate results.
2. Some glomeruli are missed (false negatives) as their edges are not pronounced or the C-shape is not present.

The glomeruli coordinates found from the union of the results of the two methods can be used as starting seeds for tracking. Another method was found to be more suitable for glomeruli termination during tracking. The method forms part of the validation and machine learning stages of the system and is discussed in Chapter 9. It basically involves detecting the fragments making up the glomerulus during tracking through a trained classifier.

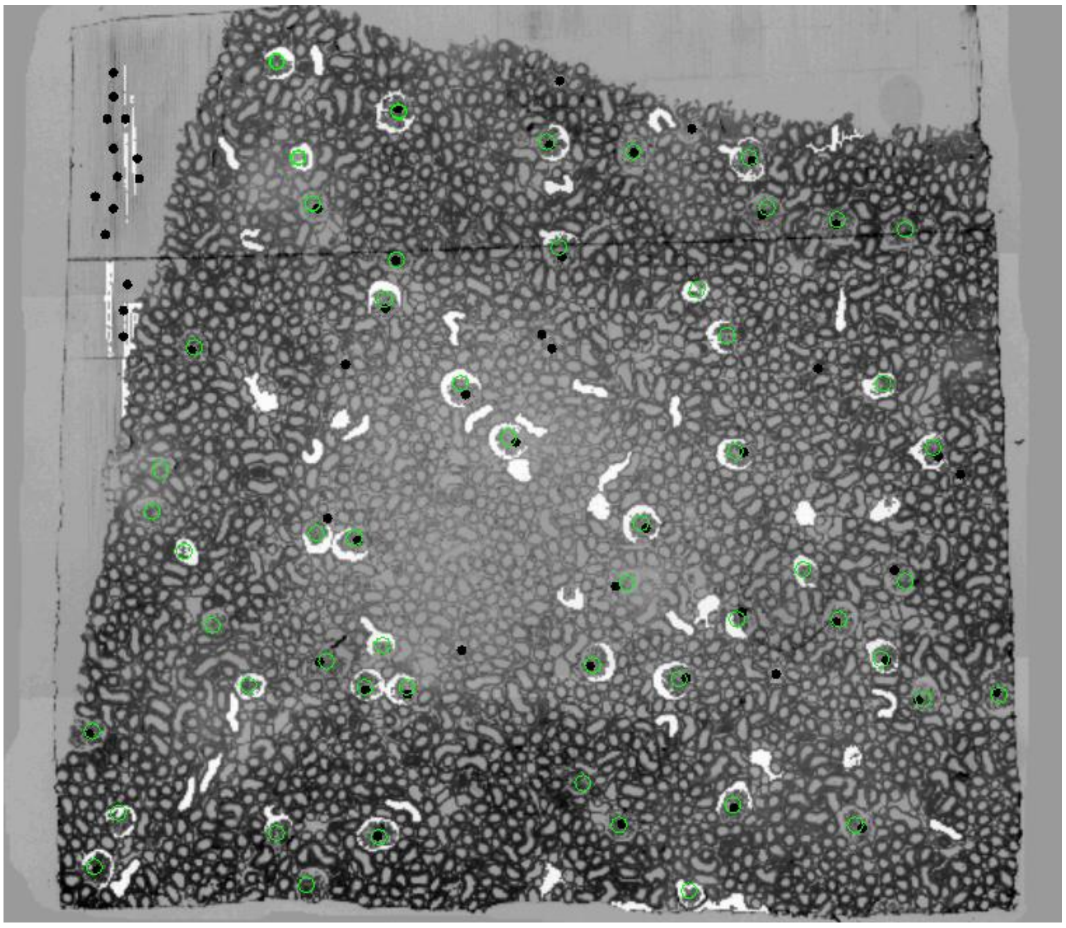


Figure 7.6: The glomeruli detection using the edge image density is shown by the black dots. The detection using shape clustering is shown by the cross-sections highlighted in white. True glomeruli locations were manually marked in green. Both methods can detect a large number of glomeruli and could be used to cross validate the results. Image from **[**11**]**.

CHAPTER 8

Tracking Algorithm

When a nephron is manually tracked by eye, an intuitive process is used by the brain. Once a nephron cross-section has been fixated on, 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 paths taken by the many nephrons. The algorithm is highly dependent on the quality of pre-processing and accuracy of feature extraction stages. The major processes of the tracking algorithm are shown in Figure 8.1. Each activity is discussed in its respective section in this chapter.

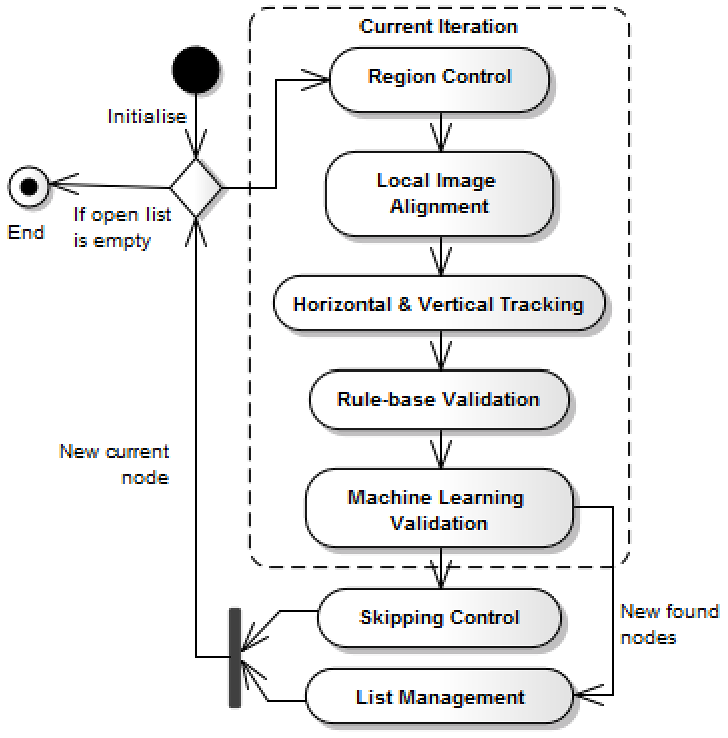


Figure 8.1: An activity diagram of the tracking algorithm. Each iteration explores possible child nodes from one parent node. Once the open list is empty, manual intervention is requested at the end points. Once the nephron is complete, the closed list is reconstructed into an ordered path.

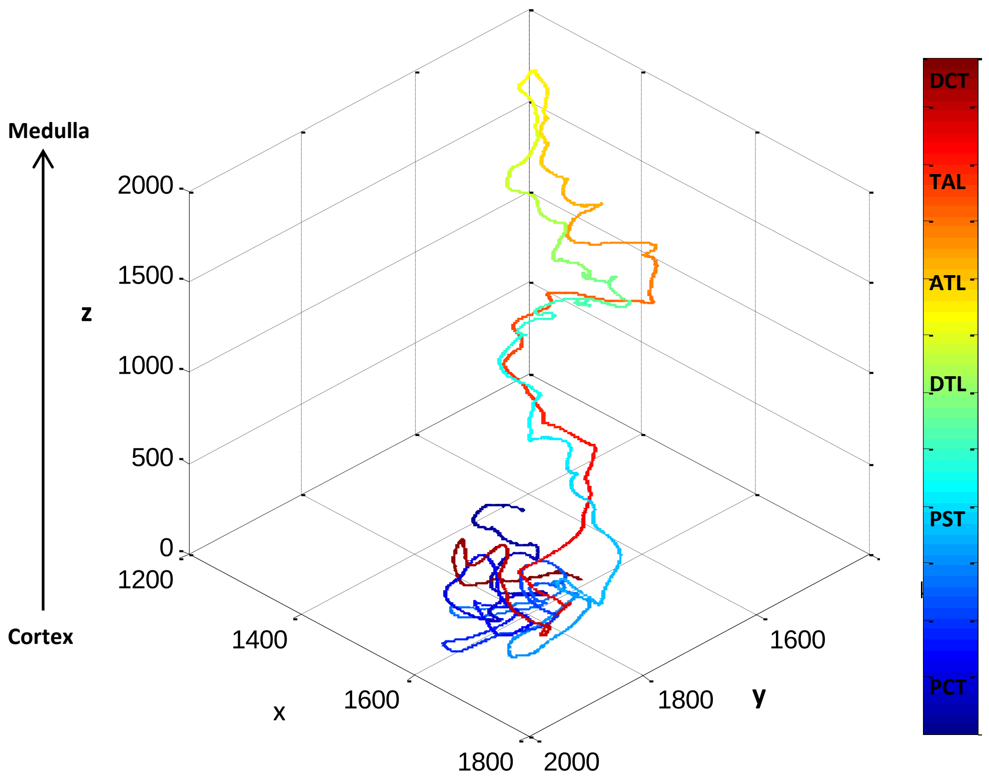


Figure 8.2: An example of a manually tracked rat nephron is shown **[**11**]**. The spatial dimensions of the x-y-z image space are illustrated. Note that the positive z direction extends from the cortex to the medulla as shown. Colour is used to highlight the different parts of the nephron as shown by the colormap on the right. This convention will be used for all 3D plots.

An example of a manually tracked nephron is shown in Figure 8.2. The tracking algorithm aims to produce such a result in an automatic manner. The z-dimension of the 3D image space refers to the dimension along the image set, where z=1 is the first image in the cortex. Note that 'upwards' (z+1) refers to movement towards the medulla while 'downward' (z-1) refers to movement towards the cortex. This is contrary to the standard orientation where the cortex is positioned superiorly and the loop of Henle is drawn extending downwards. Prior to proceeding, a few symbols are defined:

I_z	Image *z*
$\mathcal{C}_z$	The set of all nodes in image *z*
c_z^i	The set of nodes on cross-section *i* in image *z*
${c_k}_z^i$	The *k*th node on cross-section *i* in image *z*

8.1 Local Image Registration

Local alignment is needed (in addition to the previous alignments [**9**] discussed in Section 6.1) due to the presence of local image distortions and progressive change in morphology. The procedure followed involves cropping images I_z, I_{z+1} and I_{z-1} around the current node location.

Image I_z is multiplied by a Gaussian function in order to give the centre of the image (the area around the current location) a higher weight than the surroundings. This is particularly useful when tissue folds occur, where two areas of the same sub-image can be aligned differently, such as in Figure 8.4. The preference is then given to the choice which best aligns the current node.

The sub images in I_{z+1} and I_{z-1} are cross-correlated against sub-image I_z, in order to obtain two pairs of translational offsets (x_{off}, y_{off}) between the images [**41**]. The offset is obtained by extracting the peak coordinate (global maximum) of the correlation result. The transform in equation 8.1 is applied to each pixel and each node in images I_{z+1} and I_{z-1}. This local alignment only takes translation into account; it is assumed that local rotational offsets are minimal as the previous alignment had largely compensated for rotational offsets.

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_{off} & y_{off} & 1 \end{bmatrix} \tag{8.1}$$

The offsets are typically only a few pixels, but they have a large impact on the accuracy of tracking since some nephron cross-sections are also just a few pixels wide. The example in Figure 8.3 shows how a small alignment can increase tracking accuracy.

After alignment, vertical tracking is attempted. The alignment is not cumulative; it is only with respect to the three images for the current iteration, so that accurate links can be made to the cross-sections above and below. After tracking, the newly found nodes (now in the transformed axes) are mapped back to the original axes using the inverse transform T^{-1}. The transformation must be reversed because each sub-image has a different transform which is not valid for other areas in the same image or other images.

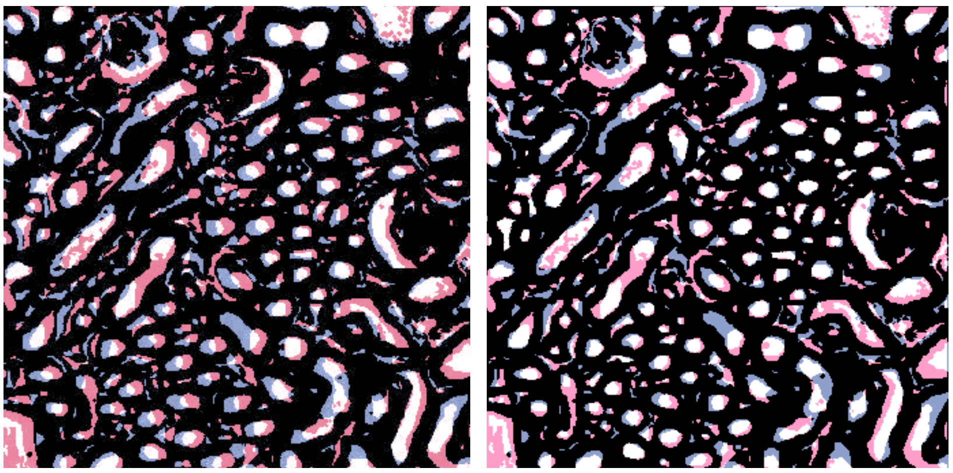


Figure 8.3: Although prior image registration was performed, there still exists an offset between adjacent images as shown on the left. After basic translational alignment (x_{off}=2, y_{off}=9) in the local region, the cross-sections are much better aligned (on the right), allowing more accurate tracking. Original image from **[**11**]**.

The transform is also applied to the sub-images themselves in order to obtain an alignment measure. This value is used to flag images which are highly misaligned even after the alignment procedure. This occurs when artefacts and tissue folds produce missing tissue and non-linear distortion such as in Figure 8.4.

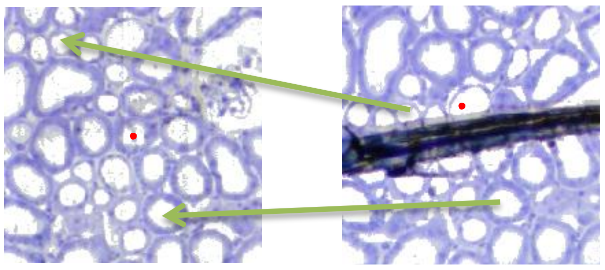


Figure 8.4: Some images cannot be aligned accurately due to artefacts. Such images can be flagged and thus ignored by measuring an alignment metric. If these images are not bypassed, tracking mistakes can easily occur as shown by the red dots. The green arrows indicate corresponding areas. Images from **[**11**]**.

8.2 Graph-based Tracking

Initial tracking attempts made use of hard-coded rules to handle bends and turns and explicitly controlled the tracking direction. Rules for a number of cases were created but these resulted in an inflexible system which was only capable of handling ideal turns and bends (which were modelled by the rules) and tracked only small pieces of the nephron. Inspection of the results showed that the majority of errors occurred when the tracking direction was not chosen correctly. The final tracking algorithm abandoned the concept of direction decision making. Instead, all directions are explored at each node and the final direction is implicitly determined in the reconstructed path.

A graph-based approach is employed for tracking that is similar to the structure of path finding algorithms [**42**]. The algorithm forms a graph in 3D space by establishing edges between the nodes previously allocated during feature extraction. Edge formation is described in Section 8.3.

The algorithm processes one node per iteration and continuously updates an open and closed list. It is given a starting seed, or node. Once the current node has been explored, it is added to a *closed* list, which contains all explored nodes. The new found nodes (child nodes of the current node) are added to an *open* list, which consists of all unexplored nodes. A new current node is selected from the open list at each iteration of the algorithm. This continues until the open list is empty. Ideally, given a starting seed, edges should be formed such that all nodes belonging to the nephron being tracked are collected in the closed list. Each node is stored along with its parent node, forming a linked list. The trajectory can then be reconstructed post-tracking.

By using linked lists and exploring all possible routes, the direction of tracking does not have to be explicitly controlled or limited in any way. The direction can be arbitrary, enabling the tracking of highly convoluted curves in variable directions, as well as branched structures. This accommodates for increased variability in the plane in which the nephrons are sliced, whereas previous methods did not have any tolerance for tissue that was not sliced in cross-section.

8.3 Edge Formation

The edges are established through a controlled set of criteria. Given a particular node ${c_k}_z^i$ in image I_z, it has the potential to connect to three other nodes through two types of edges as shown in Figure 8.3.

Vertical edge – Includes potential connections to cross-sections in the image above (I_{z-1}) and below (I_{z+1}) the current cross-section. Nodes are searched for which lie within some tracking radius around the current node, i.e. a node satisfying equation 8.2 will become a child node of the current node. The tracking radius varies from the cortex to medulla according to a sigmoidal function.

$$\min\left(\left\|C_{z\pm1} - {c_k}_z^i\right\| < r_{track}\right) \tag{8.2}$$

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 in the presence of image distortions and misalignments.

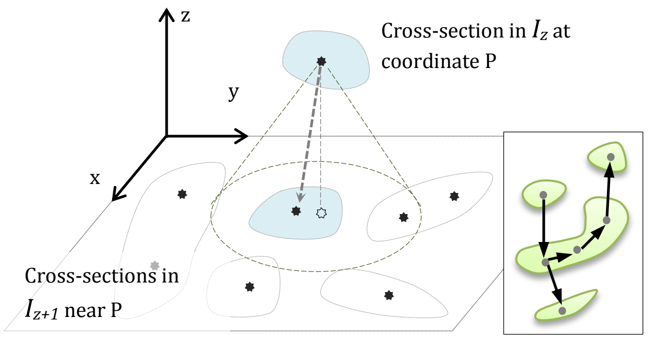

z
y
x
Cross-section in I_z at coordinate P
Cross-sections in I_{z+1} near P

Figure 8.5: Each node in image *z* has the potential to connect to 2 nodes vertically (in images *z+1* and *z-1*) within some tracking radius and 1 node horizontally on the same cross-section as itself. This allows cross-sections to easily be linked through turns and bends.

Horizontal edge – This involves linking together all nodes that lie on the same cross-section as the current node, i.e. c_z^i. The current node is termed the 'entering' node. The pairwise distances between all nodes are used to establish the linkage from the entering node towards the outermost nodes. Only the outermost nodes are added to the open list for exploration. The intermediate nodes are not explored above and below, i.e. they are ‘locked’. This prevents incorrect linking.

8.4 Skipping Images

An image may be termed defective if it has a large amount of interfering artefacts or distortion, which obscures cross-sections of the nephron at hand. These images can safely be skipped while tracking the nephron. However, a maximum of 5 μm of the specimen may be skipped at a time (1 image for the mouse and 2 images for the rat datasets), 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).

Since the tracking algorithm makes use of nodes rather than the images themselves, it is unaware of the presence of an image artefact. Skipping is thus attempted whenever tracking in the upward or downward direction is not successful. This results in skip attempts occurring too frequently at every dead end, for example on the last cross-section of a U-shaped bend. A set of skipping criteria are established to prevent skip attempts from occurring too frequently. A direction buffer is used to ensure that skips occur only on straight cross-sections, by checking if there have been recent successful tracking iterations in a particular direction. A refractory period is also used, which is the time (in number of iterations) after a successful skip during which other skips cannot occur.

8.5 Validation Steps

Tracking using the nodes alone would work if the images were exactly aligned and the data only contained information of the nephron cross-sections. However, many of the cross-sections actually belong to connective tissue (ICT) and blood vessels (BV), which are randomly dispersed between the nephron cross-sections and lie in close proximity to the nephrons. Even though the correct nephron path may be found, much interference is caused by connective tissue cross-sections, 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 in order to eliminate incorrect moves from one cross-section to another.

a. *Distance Validation* – The Euclidean distance (in x, y space) between a parent and potential child node must be less than the sum of their radii, as in equation 8.3. A coefficient D_{coeff} allows some leeway on this rule. 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 ICT cross-sections linking to nephrons are eliminated by this rule as shown in Figure 8.6.

$$\|c_P - c_C\| < D_{coeff} \times \frac{MA_P + MA_C}{2} \tag{8.3}$$

Where c_p and c_c are the parent and child nodes and MA_p and MA_c are the minor axes of the parent and child node. The radii are approximated as half the minor axes lengths.

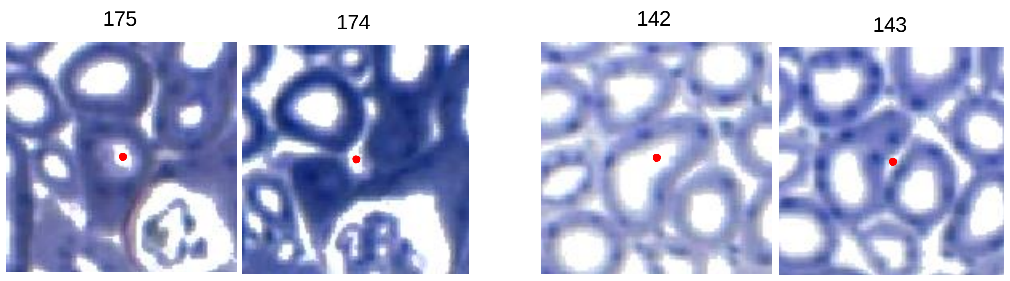


Figure 8.6: Examples of moves blocked by the distance validation rule. Images from **[**11**]**.

b. *Bidirectional Movement Validation* – If a move is made from node A in image I_z to node B in image I_{z+1}, then an attempted move from node B into image I_z must lead back to node A (i.e. bi-directionality must be maintained). If not, the move is discarded. Moves between ICT cross-sections and glomeruli are typically not bidirectional and are hence largely eliminated as in Figure 8.7.

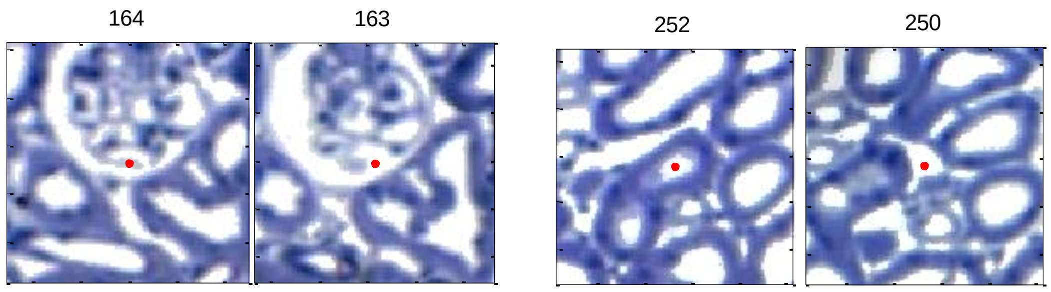


Figure 8.7: Examples of moves blocked by the bidirectional validation rule. Images from **[**11**]**.

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 according to equation 8.4. This is so that skips are not allowed on turns and bends, as this presents a high chance of error.

$$\triangle SF = \frac{100}{6}\sum_{i=1}^{6}\frac{|f_i^p - f_i^c|}{\min(f_i^p, f_i^c)} \tag{8.4}$$

Where $\triangle SF$ is the overall change in shape factors, f_i^p and f_i^c are the ith shape factors of the parent and child cross-sections, respectively. For a skip to be valid, $\triangle SF$ must be less than 30%. Figure 8.8 displays two invalid skip attempts.

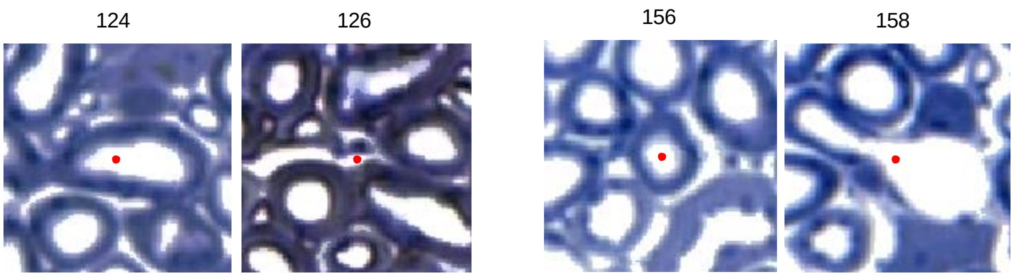


Figure 8.8: Examples of moves blocked by the skipping validation rule. Images from **[**11**]**.

8.6 Region Control

Certain control variables of the tracking algorithm are altered when a transition is made between the cortex and medulla, in order to make the algorithm more suitable to tracking in the respective regions. For example, once the PST narrows into the DTL, horizontal edge formation and vertical edge formation in the downward direction (from I_z to I_{z-1}) is disabled. This is so that only an upward, unidirectional path is allowed to be formed up to the loop of Henle. Using this known information about the nephrons structure prevents tracking errors in the inner medulla, which are common due to small nephron cross-sections merging (when separating walls are too thin) and being very close to one another. This of course assumes that no large bends occur in the medulla. The conditions activated for different regions are tabulated in Table 8.1.

Table 8.1: Different modes of tracking are created by altering conditions at transitions between different parts of the nephron.

Conditions	Cortex (PCT)	Cortex→Medulla (PST→DTL)	loop of Henle (DTL→ATL)	Medulla→Cortex (ATL→TAL)
Upward vertical tracking	Enabled	Enabled	Disabled	Enabled
Downward vertical tracking	Enabled	Disabled	Enabled	Enabled
Horizontal tracking	Enabled	Disabled	Disabled	Enabled
Size of cropping window for alignment	Initialised to 80	Reduce to 50	Stay at 50	Increase to 80

The transitions are detected using the 5th output of a machine learning classifier (discussed in Section 9.2) that produces a continuous valued output with '0' being a move in the cortex and '1' being a move in the medulla. A vector of this output along the tracking iterations produces a *real-time* 'region signal' which is used to indicate the transition from the cortex to the medulla. The signal is smoothed through a running average filter with an *m* sized window to produce a signal as in Figure 8.9. Hysteresis thresholding (with an upper medulla threshold and a lower cortex threshold) is applied to the signal to activate different modes of tracking. This method prevents a rapid fluctuation of activations.

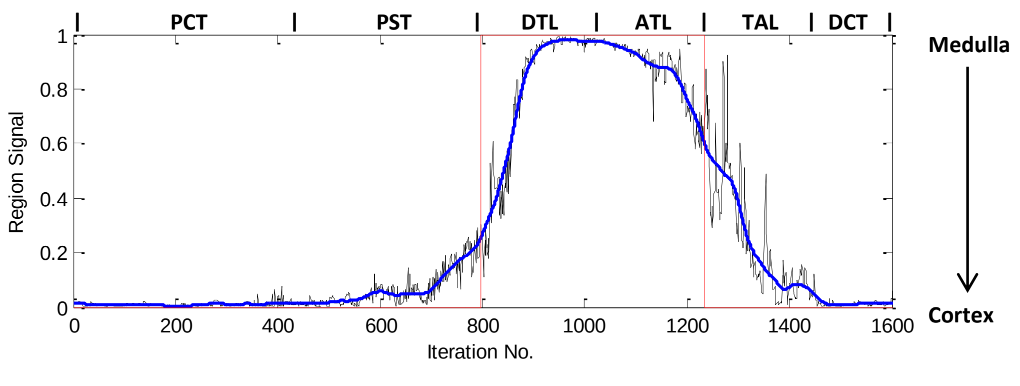


Figure 8.9: The output of the region classifier (black) is smoothed to form a region signal (blue). It is thresholded with hysteresis (red) to produce a binary decision for the activation of different conditions during tracking. The graph shown is for tracking of a whole mouse nephron from the glomerulus to the DCT.

8.7 Reconstruction

Once tracking is completed, the final 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 near turns or pieces of connective tissue that were mistakenly linked. This reconstructed graph is a skeleton structure of the tracked nephron. Each coordinate in the skeleton can be linked to the original cross-section in the binary image as well as its shape factors, either of which can be used to reconstruct a 3D rendering of the tracked nephron.

Lastly, the automatically tracked path must be evaluated in 3D space. At this stage, known information about the kidney can be used, e.g. the proximal and distal convoluted tubules intertwine and must thus be in the same vicinity in the cortex [**10**], or the PCT is longer and more convoluted than the DCT [**10**]. If the results do not adhere to one or more of these expectations, it could then be that the result is incorrect.

Since each vertical move has some associated uncertainty, the reconstructed path can be seen as having weighted edges. The path can then be broken at points with a high uncertainty. In this way, multiple nephrons may be able to be separated if incorrectly linked during tracking.

8.8 Manual Intervention

Premature termination of tracking (due to non-ideal/inadequate pre-processing, feature extraction, image artefacts and distortions) commonly occurs in the inner medulla. One way to overcome premature termination during tracking without introducing error is to allow the user to manually link the end point/s of the automatically tracked path onto the correct path. This of course steers away from a purely automated system, but it 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 8.10.

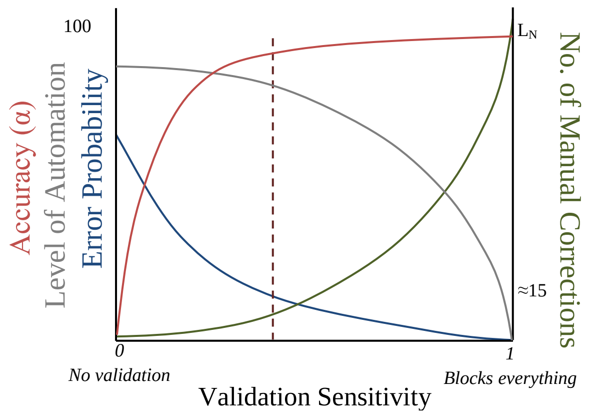


Figure 8.10: The number of false positives increases with increasing validation sensitivity, resulting in premature termination of tracking. This means only a portion of the nephron is tracked, but with a low error, where error refers to the deviation onto an incorrect path. If manual correction is used, the number of corrections required for continuation of tracking will increase with sensitivity (up to LN, the length of the nephron). This means a decreased level of automation but also a decreased chance of error. Note that this graph is merely conceptual.

More than two endpoints are sometimes detected as the last cross-section making up a bend is usually seen as an endpoint. Manual corrections are implemented by displaying to the user the main endpoints of the automatically tracked path, along with 2-3 cropped images before and after the problematic cross-section. The user then simply clicks on the cross-sections which should have been linked by the algorithm. These are added to the open list and the algorithm continues from those points. Correct endpoints (termination at the glomerulus) can simply be ignored.

CHAPTER 9

Machine Learning Validation

Tracking nephrons using the rule-base for validation results in some nephrons being correctly tracked while others are incorrectly linked to other nephrons, ICT cross-sections or blood vessel networks. A large amount of information is not yet taken into account, such as the shape profile and some 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. Machine learning is a suitable technique for this application, as it can automatically amalgamate the large amount of information into a generalised rule through training. The formed rule, or hypothesis, may be unintuitive to a human being and too complex to model using hard-coded logic or inflexible heuristics.

The rule-base eliminates invalid moves between pairs of cross-sections. Likewise, machine learning is incorporated such that a trained learning algorithm assesses the shape descriptors and other features of the cross-sections and classifies the move into one of five classes. This classification result is used by the tracking algorithm to make decisions during tracking. A supervised ANN and SVM are the chosen classifiers as they are non-linear and able to form complex hypotheses.

9.1 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 their combined features. A total of 67 features are formulated which include:

- x_1-x_6: the difference in the shape factors of area, eccentricity, solidity, aspect ratio, minor axis and circularity
- x_7-x_{12}: the mean of the shape factors of area, eccentricity, solidity, aspect ratio, minor axis and circularity
- x_{13}: the minimum area between the two cross-sections

- x_{14}: the Euclidean distance between the two nodes in the x-y plane
- x_{15}: the image difference, which is normally 1 but can be 2 or 3 if images have been skipped
- x_{16}: the magnitude of image alignment offset – a high offset coupled with other odd features may be a flag for an invalid move
- x_{17}: the position of the pair (average z coordinate) relative to the image set, which indicates depth into the kidney, i.e. cortex, outer medulla, inner medulla
- x_{18}: the correlation coefficient between the two shape profiles
- x_{19}: a correlation coefficient between the two sub-images around the location of the move
- x_{20}-x_{43}: shape profile at 15 degree intervals of cross-section 1
- x_{44}-x_{67}: shape profile at 15 degree intervals of cross-section 2

9.2 Training Set Formation

The training set is created by capturing moves, or pairs of nodes, during unsupervised tracking (without any machine learning validation) of a chosen set of nephrons. Five classes are chosen for classification as described in Table 9.1. Each parent-child pair is assigned a label as shown in the examples in Figure 9.1.

Table 9.1: The intermediate output classes of the learning functions and their combination into final classes

Final Class	Intermediate Class
Nephron (Valid)	1. A normal move between circular cross-sections 2. A normal move involving elongated cross-section/s
Non-nephron (Invalid)	3. An abnormal move typically involving ICT or blood vessel cross-sections 4. A move involving a glomerulus cross-section
x	5. A move in the inner medulla

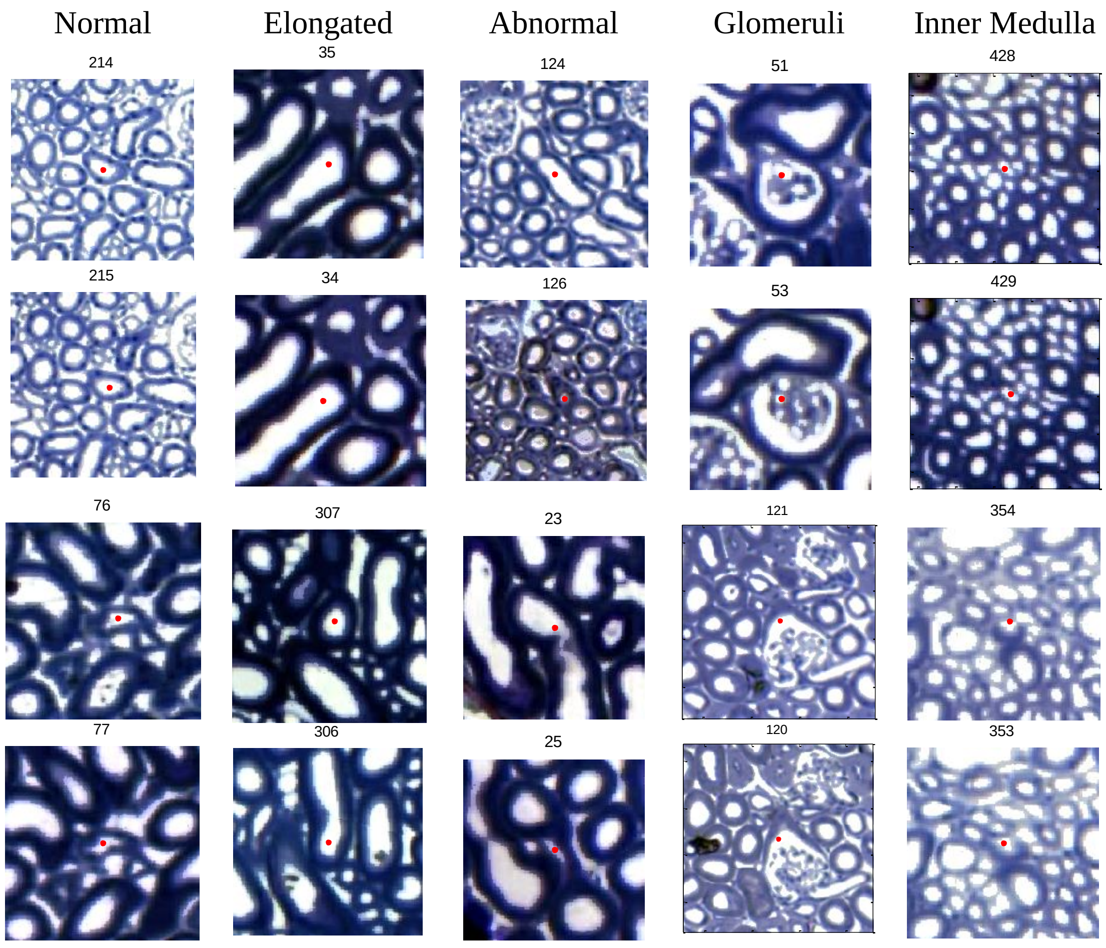


Figure 9.1: The moves attempted by the unregulated tracking algorithm are captured, displayed and labelled to form training examples for the machine learning algorithms. Two examples of moves from each of the five classes are shown. Images from **[**11**]**.

A voting scheme [**29**] between the five 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 (parameters of the algorithm) from the cortex to inner medulla (as in Section 8.4). The shape factors and descriptors belonging to each cross-section in the pair can be extracted as required and the 67 features are then combined to form the input matrix. A multi-class classifier is produced using the one-vs.-all approach.

9.3 Training

The training set consisted of 9424 examples, with a ratio of 0.58 : 0.10 : 0.11 : 0.07 : 0.13 for classes 1 to 5, respectively. Although the types of examples are skewed, there are a sufficient number of examples per class (at least 650). The input is randomised and each feature is normalised. The labelled data set was split into training, validation and test sets with a 0.7:0.15:0.15 ratio, respectively. Training of the ANN and SVM was carried out using built in MATLAB functions [**33**].

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 a radial basis function (RBF) kernel has the analogous effect. It is critical that false positives are minimised as a false positive 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 branch is typically halted due to the presence of many invalid moves through connective tissue, and is therefore not as critical as a false positives.

9.4 Reinforced Learning

In addition to manual selection of examples, a method involving feedback from the tracking algorithm and the training process was used in order to collect a sufficient number of examples per output class. This reinforced learning procedure prevented the formation of a skewed dataset or the under-representation of a certain class, which may have affected classification accuracy. This feedback process is illustrated in Figure 9.2.

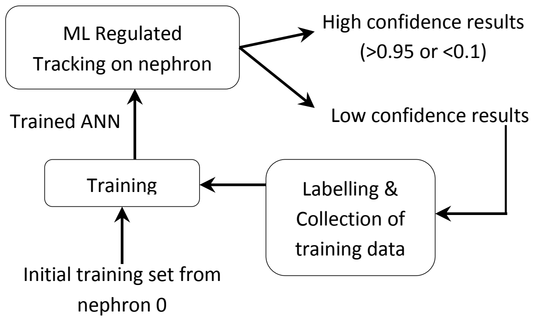

ML Regulated Tracking on nephron
High confidence results (>0.95 or <0.1)
Low confidence results
Trained ANN
Training
Labelling & Collection of training data
Initial training set from nephron 0

Figure 9.2: A schematic showing the method employed for reinforced learning, which aims to decrease skewness among the five output classes.

9.5 Feature Analysis

A number of the features may be redundant or irrelevant. A subset of the most useful features can be determined through various feature selection techniques. The RELIEFF feature selection method [**43**] has been used which ranks each feature by its importance. The analysis shows that shape profile correlation, average solidity, the pair's z-position and average eccentricity are the most useful features. The raw shape profiles have a medium importance, while the image difference, difference in area and difference in minor axis length have the least impact on classification and could be removed. However, the number of features is not too excessive (only 18 features and 2 shape profiles) and so all features are included.

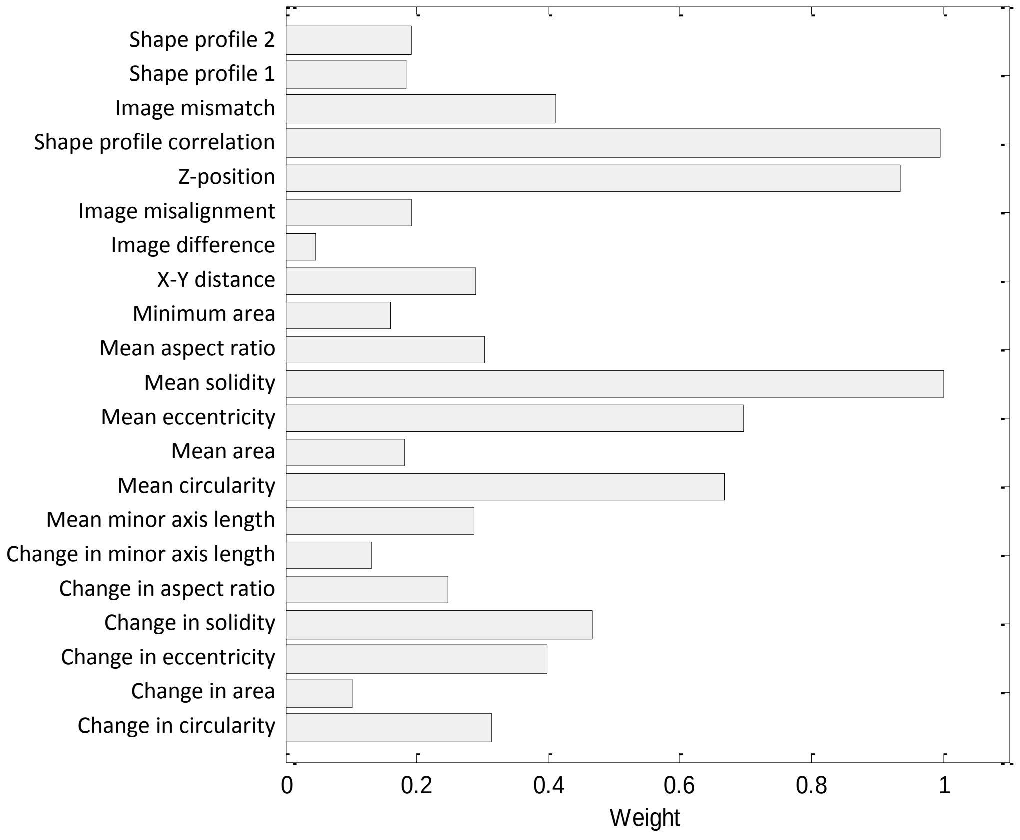


Figure 9.3: The RELIEFF feature selection method allocates a weight to each of the 67 features indicating its importance during classification of a move.

It is also useful to visualise the separability of the five classes and the impact of the features on the five classes of examples. One could simply view plots of two or three features at a time and try to deduce their relationship and their impact on classification. However, this is limiting and does not represent the overall effect of all features on the classification problem.

Principal Component Analysis (PCA) is a dimensionality reduction technique, useful for visualising high dimensional data. It uses Single Value Decomposition (SVD) to obtain eigenvectors and eigenvalues. A training set of 9424 examples and 67 features was reduced to 2 dimensions yielding the plot in Figure 9.4.

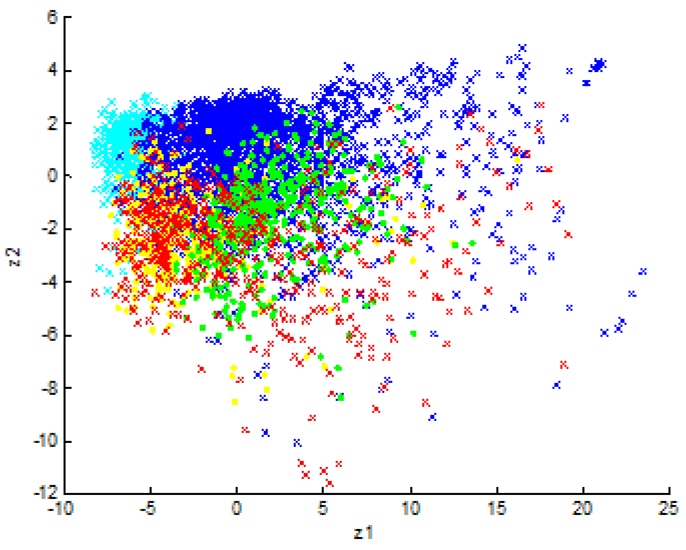


Figure 9.4: The scatter plot displays 9400 examples in terms of reduced features z1 and z2, which were projected from the first 2 eigenvectors of the data. Moves that are normal (blue), involving elongated cross-sections (green), connective tissue (red), glomeruli (yellow) and moves in the inner medulla (cyan) are shown.

Figure 9.4 shows that there are large overlaps between the different classes of moves especially between connective tissue and glomeruli types. However, this is expected as the cross-sections in these two classes are very similar. Despite overlapping, separating lines can be visualised between the other classes. Additionally, a machine learning algorithm using all 67 features will have better resolving ability between the classes than that seen in the plot. The PCA plot uses only two principal features which do not retain variance. That is, the number of eigenvectors (K) needed to adequately represent the original data should satisfy equation 9.1.

$$\frac{\sum_{i=1}^{K} S_i}{\sum_{i=1}^{n} S_i} \geq 0.99 \qquad (9.1)$$

Where S_i is the ith eigenvalue. Additional feature analysis plots can be found in the Appendix. For K=2 and n=67, the value is only 0.4677, therefore variance is not retained and the two reduced features do not represent the information offered by all 67 features.

An SVM with an RBF kernel is well suited to creating the arbitrarily shaped hypothesis function required. A neural network with a large number of neurons could also form a complex hypothesis function.

9.6 Optimisation

Initially, the training data was categorised into valid and invalid moves (instead of the five intermediate classes). This resulted in low classification accuracy even with an increasing number of training examples and hidden units of the ANN. This was because there are different types of valid and invalid moves (as indicated by the five final classes), each of which has its own unique characteristics. The different types could not accurately be modelled into one hypothesis function and hence a multi-class classifier had to be used. This resulted in improved classification accuracy.

Under certain conditions, obtaining more training examples improves performance significantly. If the features provide sufficient information to accurately predict the output, and if the learning algorithm can fit a complex function (so that underfitting is addressed), then a large training set will optimise performance as it will minimise overfitting [**28**].

From the Principal Component Analysis of the features, it can be seen that the features do adequately model a move from one nephron cross-section to another. The features are also informative enough for a human operator to correctly classify a move. Both an ANN with a large number of hidden layers (50) and an SVM are capable of forming complex, non-linear hypotheses. The number of training examples was thus increased until there was no longer an increase in performance as shown in Figure 9.5.

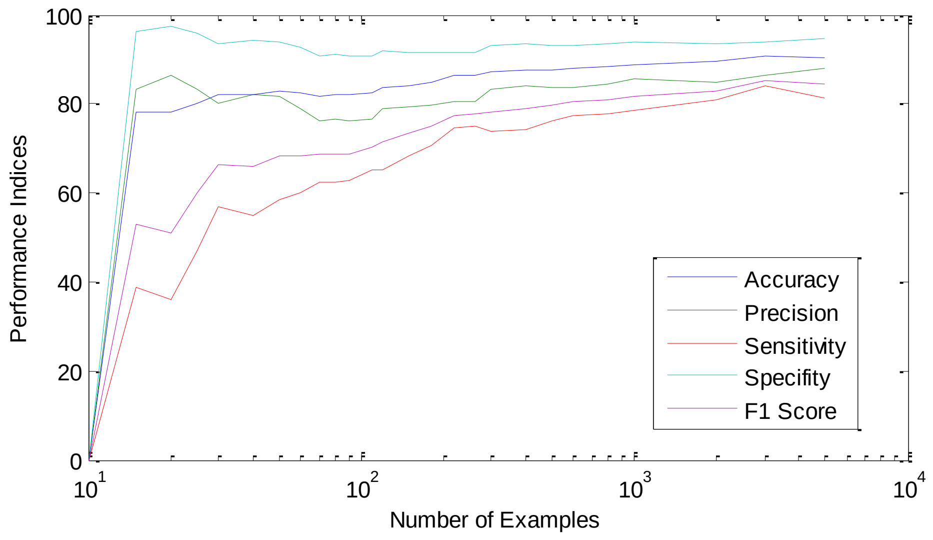


Figure 9.5: Five performance indices were measured on a test set after training the SVM (using an RBF kernel of a width of 5) with a varying number of randomised training examples. The performance converges around a 1000 examples.

CHAPTER 10

Results

The accuracy of each stage is dependent on the accuracy of previous stages as each stage's output is an input for the following stage. The results of each stage are analysed and discussed.

10.1 Pre-Tracking Stages

The pre-tracking stages include the pre-processing and feature extraction stages.

10.1.1 Pre-Processing

The goal of the pre-processing stage was to produce binary images in which each binary component represents one nephron cross-section. This has been achieved in the majority of cross-sections in each image by careful selection and variation of the pre-processing parameters using sigmoid functions.

These automatically varying parameters are suitable to the majority of images in the set but may not be suitable for a few outlying images resulting in the occasional merging of binary components which were meant to be independent. This is termed 'under-segmentation'. Also, the compromise between the equalisation window size and the threshold value sometimes results in the over-segmentation of some binary components which were meant to be whole. These two cases are depicted in Figure 10.1.

The performance of the pre-processing stage is thus evaluated by measuring *segmentation accuracy*, which is defined as the percentage of binary components correctly representing the nephron cross-sections. Due to the impracticality of manually evaluating each cross-section in any one image, 12 samples were selected from different datasets for evaluation. The samples were chosen such that the cross-sections contained were representative of all cross-sections in the respective area. Table 10.1 presents the results.

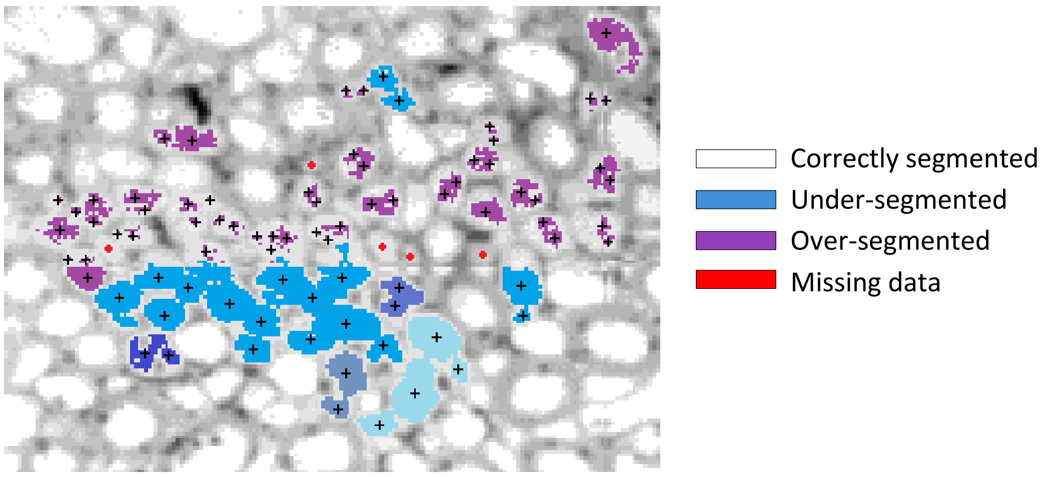


Figure 10.1: A clip of an image in the medulla of a rat dataset is shown with its binary image superimposed. Binary components correctly produced are shown in white. Purple components are those which were over-segmented, producing multiple binary components per nephron cross-section. Components in blue are those which have been incorrectly merged into single components. Red dots indicate nephrons which have no overlying binary components, thus producing missing data. Nodes allocated subsequent to binary image formation are shown by the '+' symbols. Original image from **[**11**]**.

Table 10.1: The segmentation accuracy of samples from 4 datasets in the cortex, outer and inner medulla.

	Segmentation Accuracy (%)		
	Cortex	**Outer Medulla**	**Inner Medulla**
Mouse 1	98.59	98.75	97.24
Mouse 2	98.48	96.62	95.01
Rat 5	95.13	82.92	88.86
Rat 4	96.01	93.61	90.30

The majority of extracted binary components correctly represents the structures in the original image. Under-segmentation occurs when groups of nephrons have very thin walls in comparison to surrounding nephrons (such as in the DTL) causing independent cross-sections to merge. Over- and under-segmentation mostly occurs in images of the medulla of the rat datasets because the centre of the images is much brighter than the periphery.

10.1.2 Feature Extraction

K-means clustering in combination with the shape criteria has resulted in a highly flexible and adaptive node allocation method. The majority (>99%) of cross-sections receive an ideal number of nodes, especially since the randomly initialised K-means is repeated if the adjacent node criterion is not met. Even on under-segmented binary components (such as the large blue components in Figure 10.1), a node is correctly allocated at the location of each nephron cross-section. On incorrectly segmented binary components, nodes are not allocated per nephron cross-section as each binary component undergoes independent node allocation.

The degree to which the shape factors and shape profiles represent the original nephron cross-section in the colour image is dependent on the extracted binary components during pre-processing. Accuracy of feature extraction stage is thus dependant on the pre-processing stage or segmentation accuracy. Given ideal binary components, the measured shape factors and shape profiles are ideal (100% accurate).

10.2 Measuring Similarity between Paths

In order to evaluate the outcome of a tracking instance, the automatically tracked path must be compared to the corresponding manually tracked nephron. Single number evaluation metrics are used to indicate performance and similarity.

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_f	The manually tracked path of nephron *f*
Ψ_f	The automatically tracked path of nephron *f*

where a path is a set of coordinates in 3D space.

When the result has a low degree of correctness, it is either because tracking terminated prematurely or the path deviates onto an incorrect one (linkage with another nephron, blood vessel, or ICT cross-sections), or a combination of these. The outcome of tracking a particular nephron is hence evaluated using two measures:

1. α_f = % of Ψ_f that is correct – ‘accuracy’
2. β_f = % of Υ_f, that Ψ_f covers – ‘extent’

These are calculated by obtaining the per image residuals as in equations 10.1-2.

$$\boldsymbol{r}_{\Psi\Upsilon}(i) = \min\left(\left\|\boldsymbol{\Psi}_{\mathrm{i}} - \boldsymbol{\Upsilon}_{\in \mathrm{z}_{\Psi_{\mathrm{i}}}}\right\|\right) \quad \forall i \in \{1, \dots, m_{\Psi}\} \tag{10.1}$$

Where $\boldsymbol{\Psi}_{\mathrm{i}}$ is the ith coordinate in the automatically tracked path $\boldsymbol{\Psi}$, m_{Ψ} is the number of coordinates in $\boldsymbol{\Psi}$, $\mathrm{z}_{\Psi_{\mathrm{i}}}$ is the z coordinate (image number) of the ith coordinate, $\boldsymbol{\Upsilon}_{\in \mathrm{z}_{\Psi_{\mathrm{i}}}}$ is the subset of the manually tracked path containing all coordinates in image $\mathrm{z}_{\Psi_{\mathrm{i}}}$ and $\boldsymbol{r}_{\Psi\Upsilon}$ is the per image residual of $\boldsymbol{\Psi}$ with respect to $\boldsymbol{\Upsilon}$**.** Similarly the residual of $\boldsymbol{\Upsilon}$ with respect to $\boldsymbol{\Psi}$ is:

$$\boldsymbol{r}_{\Upsilon\Psi}(i) = \min\left(\left\|\boldsymbol{\Upsilon}_{i} - \boldsymbol{\Psi}_{\in z_{\Upsilon_i}}\right\|\right) \quad \forall i \in \{1, \dots, m_{\Upsilon}\} \tag{10.2}$$

The residual indicates the minimum distance of each node in one set to nodes in the same image of the other set. It provides a measure of the discrepancy between the two paths on a per coordinate basis. In order to obtain a single valued similarity measure between whole paths, the residuals are thresholded at some tolerance. This is to allow for differences due to slight image misalignments and differing node positions (the manually placed coordinates may not be in the centre whereas the automatically allocated nodes are more towards the centre), as they should not technically contribute to the error. Accuracy and extent are then defined as in equations 10.3 and 10.4.

$$\alpha = \frac{100}{m_{\Psi}} \mathrm{count}(\boldsymbol{r}_{\Psi\Upsilon} < \mathrm{tol}) \tag{10.3}$$

$$\beta = \frac{100}{m_{\Upsilon}} \mathrm{count}(\boldsymbol{r}_{\Upsilon\Psi} < \mathrm{tol}) \tag{10.4}$$

Where tol is the tolerance in pixels and count is a function returning the number of elements satisfying the condition in its argument.

10.3 Possible Outcomes

The outcome of a tracking instance can be one (or a combination) of four types of cases as shown in Figure 10.2.

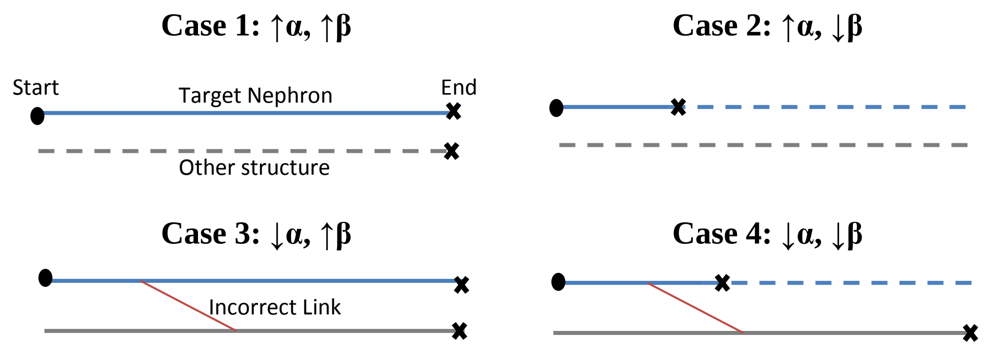


Figure 10.2: The target nephron's path is shown in blue while the path of other structures is indicated by the grey line. Solid lines indicate tracked paths and broken lines indicate untracked paths. The circles and crosses indicate the beginning and end of tracking, respectively.

Case 1: Ideally, the tracking algorithm should track the full length of the target nephron without mistakenly tracking the path of any other nephron or blood vessel (case 1). This would lead to a high α value (the tracked path is well correlated to the target path) and a high β value (a large percentage of the target path has been tracked). However, this is not achieved due to a number of hindering factors.

Case 2: Most often, only a portion of the target path is correctly tracked with no incorrect paths being formed (case 2). This produces a high α value but a low to medium β value depending on where along the path premature termination had occurred. This premature termination is usually caused by artefact interference and large local distortions, which trigger one of the validation steps, causing tracking to terminate. Examples can be seen in the middle column of Figure 10.5.

Cases 3 and 4: Sometimes incorrect links are made to one or more structures other than the target nephron (cases 3 and 4). This is caused by artefacts and distortions as shown by examples in Figure 10.3. This leads to a low α value as the tracked path does not fully correlate to the target path. The β value can vary depending on how much of the target path has been found in addition to the incorrect paths.

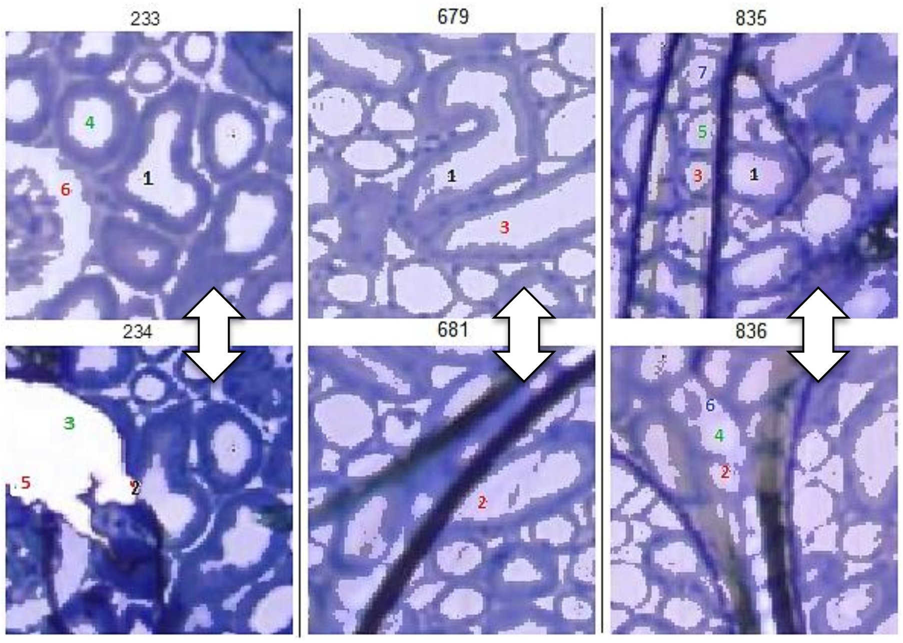


Figure 10.3: Three examples of incorrect linkage to multiple structures are shown. The numbers indicate the order of moves/iterations during tracking, while their colours indicate analogous structures. In the first example (left), the horizontal edges of the large cross-section created by the artefact results in the target nephron being linked to another nephron and an unrelated glomerulus. This move passes all validation stages, with the ANN output of (0.384 0.015 0.277 0.531 0.001) for the five classes. In the other two examples (middle and right), a tissue fold obscures the cross-section of the target nephron and brings a cross-section of another nephron directly beneath the current nephron cross-section. The algorithm sees these as valid moves and incorrect links are made. Images from **[**11**]**.

It is difficult to correct cases 3 and 4 as it cannot be detected without the use of manual data (there are no α or β measurements for unseen data) or manually inspecting the tracked path. Therefore, the algorithm has been designed to minimise the possibility of these cases by establishing the four validation steps. Despite these preventative measures, there are still some incidents of incorrect linking as it is difficult to model each of a variety of cases without hindering normal tracking. Also, some of the invalid moves appear to be valid according to the model for each validation step, especially incorrect links made to other nephrons.

As a result of the ‘strictness’ imposed by the validation steps, there is a higher occurrence of premature termination (as opposed to tracking without the validation steps). If the problematic areas are not bypassed, the extent of tracking remains low (as only a portion of the nephron is tracked) even though the algorithm is capable of tracking the rest of the nephron. Manual intervention is used to manually bypass such points so that tracking can continue.

10.4 Tracking Results

The tracking algorithm successfully tracks large portions of the nephrons automatically, occasionally requiring manual correction in order to obtain full nephron paths. Different parts of the nephrons were tracked with varying accuracies and extents as shown in Tables 10.1 and 10.2 due to differing tubule characteristics. In particular, the PCT and PST were tracked well, while the DTL and ATL were more problematic in both the mouse and rat datasets.

16 nephrons from 2 mouse datasets and 11 nephrons from 2 rat datasets were chosen to form a test set to test the tracking algorithm. Only short-looped nephrons were chosen as the long-looped nephrons proved to be too error prone to track as the cross-sections become increasingly difficult to track deeper in the inner medulla. The chosen nephrons were ones for which manual tracking had been performed and ones that were not used to form the training set for the machine learning algorithms. The nephrons were tracked automatically to various points, i.e. some nephrons were only tracked to the DTL while others were tracked to the DCT, etc. This is because a number of consecutive cross-sections of some nephrons became so small that corresponding binary components were not extracted and tracking (without extensive manual intervention) could not proceed as a result.

The path of each tracked nephron was broken up into the six components (PCT, PST, DTL, ATL, TAL and DCT). α and β values were measured in isolation for each component, along with the number of manual corrections required in each component. The Appendix contains a spreadsheet of the detailed results. Summarised results are presented in Tables 10.2 and 10.3.

Table 10.2: Test results on a chosen set of 16 mouse nephrons. The number of manual corrections is given as the mean ± 1 standard deviation.

	MOUSE				
Area of Nephron	**Accuracy - α_{MEAN} (%)**	**β_{MEAN} (%)**	**β_{IDEAL} (%) [9]**	**Extent - β_{MEAN}/ β_{IDEAL} (%)**	**Average Number of Manual Corrections**
PCT	95.14	27.36	25	109.44	1.20 ± 1.11
PST	98.24	16.33	18	90.72	0.50 ± 0.71
DTL	80.57	13.90	19	73.16	5.44 ± 1.69
ATL	85.67	14.94	14	106.71	2.46 ± 1.87
TAL	96.32	13.19	14	94.21	3.64 ± 1.55
DCT	72.13	14.29	10	142.90	5.86 ± 3.00
Full	87.49	100	100	100	19.09 ± 1.65
PCT to DTL		57.59	62		7.67 ± 1.25
PCT to TAL		87.38	90		13.25 ± 2.00

Table 10.3: Test results on a chosen set of 11 rat nephrons. The number of manual corrections is given as the mean ± 1 standard deviation.

	RAT				
Area of Nephron	**Accuracy - α_{MEAN} (%)**	**β_{MEAN} (%)**	**β_{IDEAL} (%) [9]**	**Extent - β_{MEAN}/ β_{IDEAL} (%)**	**Average Number of Manual Corrections**
PCT	96.32	28.48	25	113.92	5.20 ± 4.70
PST	90.17	14.64	18	81.33	5.00 ± 2.75
DTL	84.63	15.83	19	83.32	24.00 ± 8.19
ATL	88.47	15.63	14	111.64	13.50 ± 6.95
TAL	97.48	11.50	14	82.14	6.67 ± 3.09
DCT	95.23	13.91	10	139.10	4.33 ± 2.49
Full	80.85	100	100	100	58.70 ± 4.70

α indicates how much of the tracked nephron is correct by measuring similarity to the manually tracked nephron. It is low if the path deviates onto other structures and is 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.

Note that α and β are measured by comparing individual coordinates of the manual and automatically tracked nephrons. The automatically traced path typically contains more coordinates since the algorithm tracks all cross-sections related to the nephron rather than just those required, i.e. all 3-4 elongated cross-sections making up a bend are automatically tracked, while the manual path will label only 1-2 of the elongated cross-sections at a bend. The algorithm also tracks glomeruli cross-sections, whereas the manual path terminates on the last PCT cross-section. This has the overall effect of producing a higher than ideal β value.

The indistinct locations of some transitions (e.g. PCT to PST, or the end of the DCT) also results in the measured beta values being higher than the ideal values. Note that the beta values are relative to the entire nephron length and not the segment in question, for example the PST makes up 25% of the total length, and hence a measured beta value of 24% means that 96% of the PST was tracked.

The number of manual corrections varies with the sensitivity of the validation steps. For example, decreasing the ANN threshold, increasing the coefficient of the 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 branching onto incorrect structures (decreases α) as shown conceptually in Figure 8.10. The settings/conditions for the validation steps were therefore chosen such that the algorithm tracks with high accuracy while also not requesting for manual intervention at unnecessary/unreasonable points.

The frequency of manual intervention is dependent on the number of image artefacts and distortions encountered along the path of the nephron, and the visibility of the cross-sections. Examples of manual corrections are shown in Figure 10.5. A longer path (in terms of the number of moves) requires more corrections, e.g. the rat nephrons are on average 4.7 times longer than mouse nephrons and long-looped nephrons are at least 1.5 times longer than short-looped nephrons from observation of the manual data. The cross-sections of the DCT in the mouse are very small and hence harder to track than the larger cross-sections in the rat. The number of corrections required in each area of the nephron in the mouse and rat is compared in Figure 10.4, where the rat data is normalised (divided by 4.7) in order to highlight differences other than the image set size.

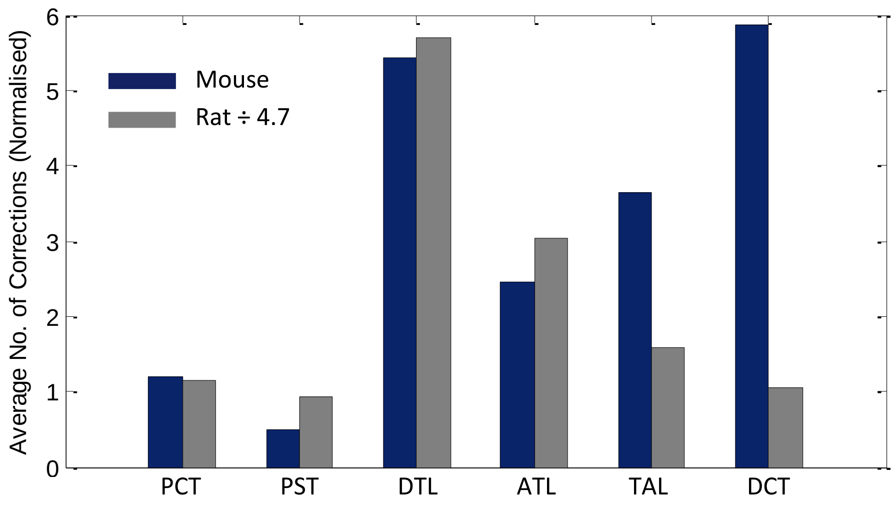
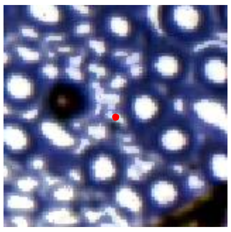
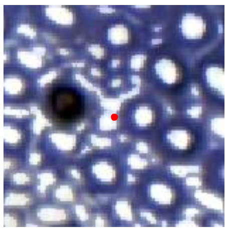
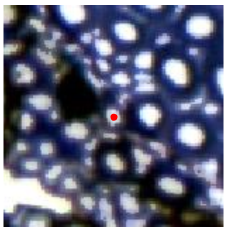
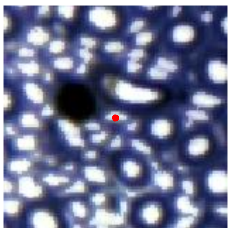
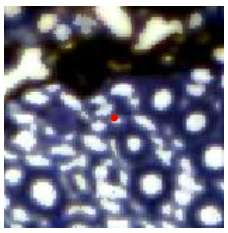
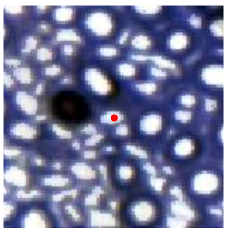
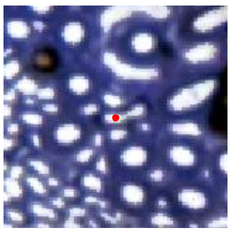
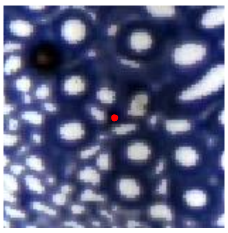
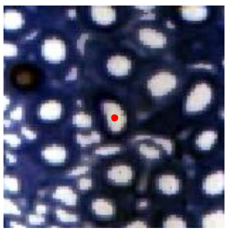
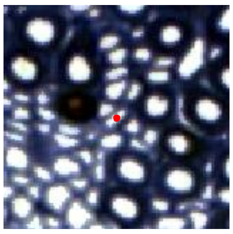
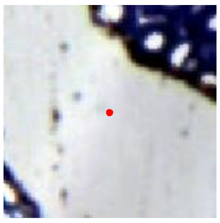
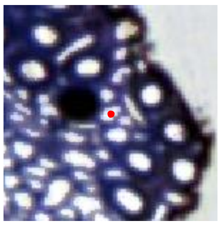
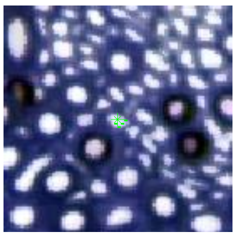
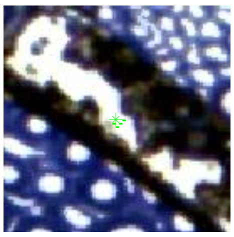
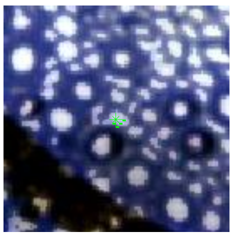


Figure 10.4: The number of manual corrections required for mouse and rat nephrons is shown. The rat data is normalised by the ratio of a mouse and rat dataset (1:4.7) in order to make a better comparison unrelated to image set size.

Automatic Endpoint	Problematic Image	Manual Correction	Notes
450 	451 	452 	Small cross-sections merged during pre-processing
465 	467 	468 	Large artefact prevents accurate alignment
517 	518 	519 	Drastic change in morphology within one image
489 	490 	491 	Missing tissue; A skip is not allowed due to sharp change in morphology
471 	473 	474 	Artefact in 472 skipped; Artefact in 473 produces a large binary cross-section which is linked to another nephron

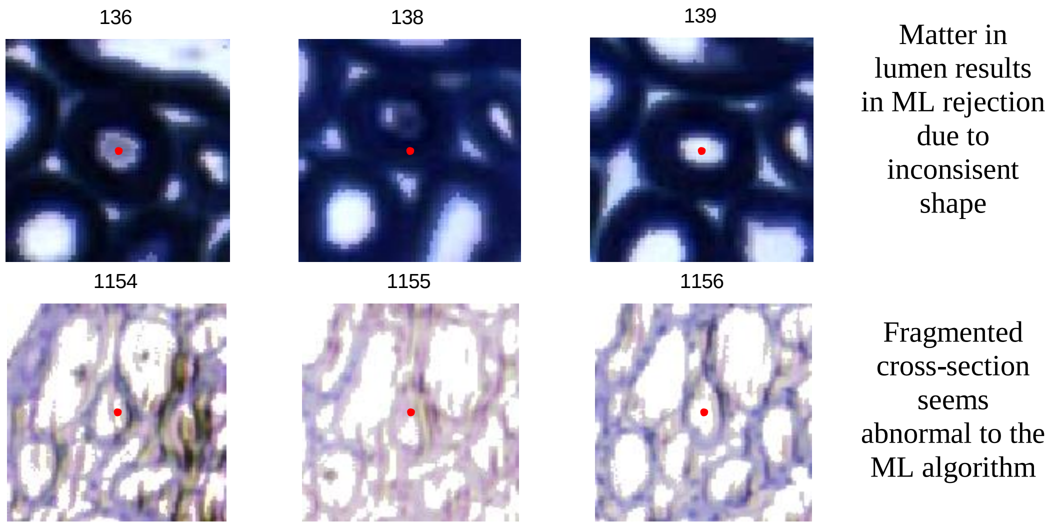

136
138
139
Matter in lumen results in ML rejection due to inconsisent shape
1154
1155
1156
Fragmented cross-section seems abnormal to the ML algorithm

Figure 10.5: Examples of premature termination during tracking requiring manual intervention. The reasons summarised on the right are indicative of the variety of non-ideal situations encountered. Original images from **[**11**]**.

A number of examples of automatically tracked nephrons compared to their manually tracked versions are shown in Figures 10.6 to 10.9. The slight discrepancies seen between the automatic and manual paths are due to different image alignments and different point coordinates used by the two methods.

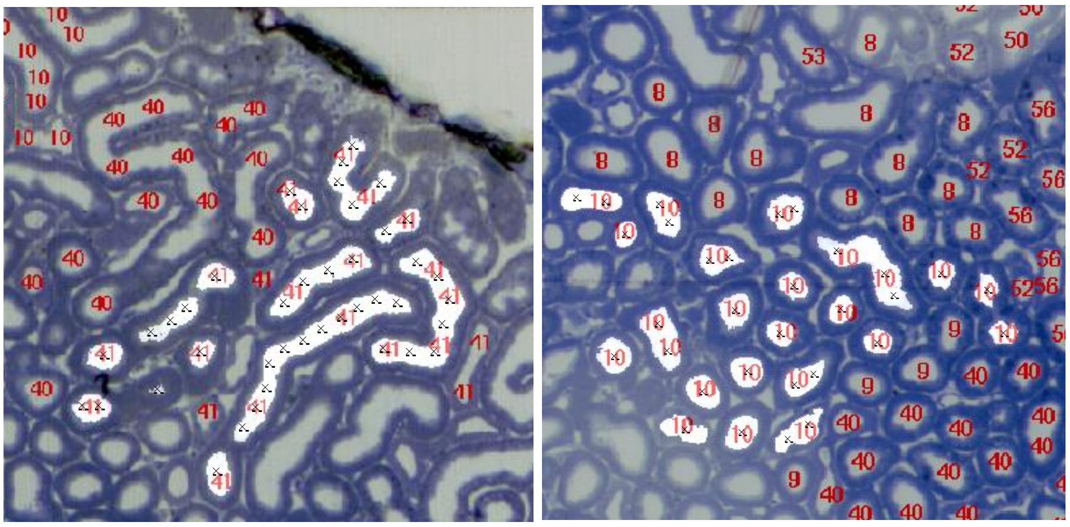


Figure 10.6: Examples of labelled images are shown with the red numbers indicating manually tracked nephrons. The automatically tracked nephrons are superimposed, shown in white with black crosses at the nodes. The automatically tracked cross-sections correspond to the manually labelled cross-sections of the PCT of nephron 41 (left) and nephron 10 (right). The cross-sections of 41 that are not highlighted are of the DCT. Original images from **[11]**.

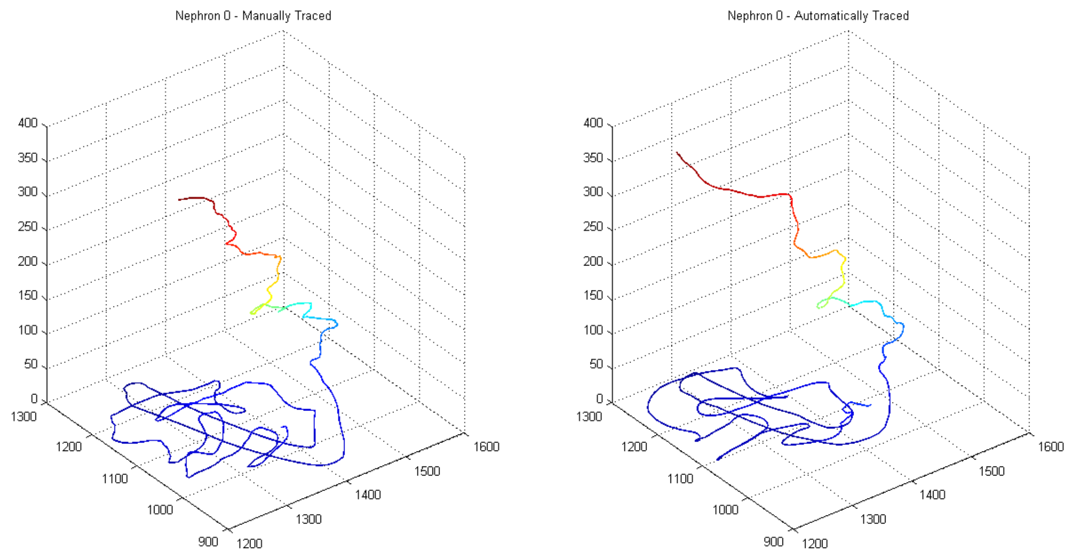


Figure 10.7: A manually tracked nephron (nephron 0 from mouse 1) is shown on the left. The same nephron is successfully tracked automatically by the tracking system. This nephron in particular was only manually tracked to the PST due to low visibility of the DTL cross-sections. The automatic tracking algorithm also experiences difficulty in tracking the DTL. The tracking terminates automatically at the glomerulus. α_0=97%; β_0>100%

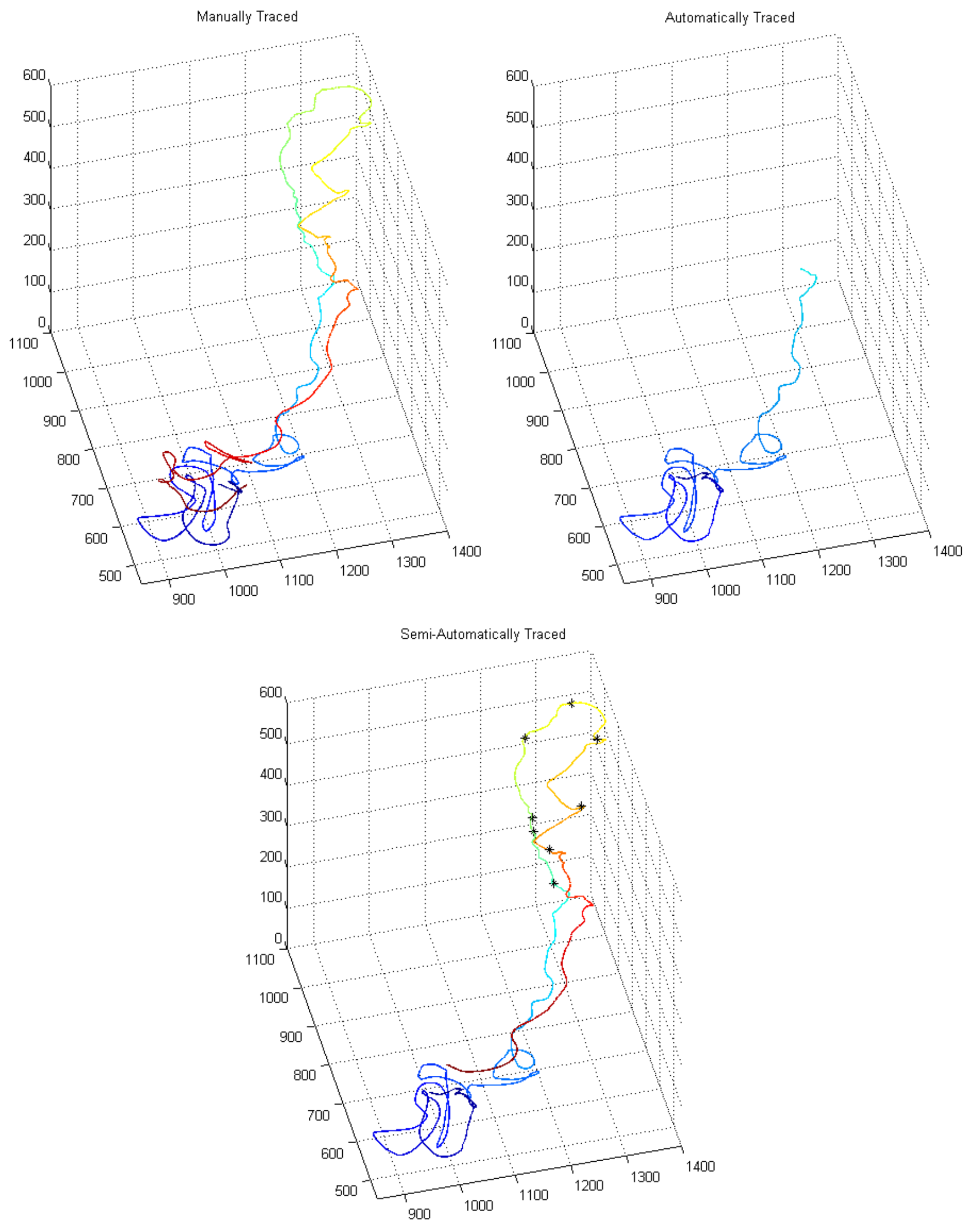


Figure 10.8: A manually tracked mouse nephron is shown on the top left. The PCT and PST are successfully tracked automatically as shown on the top right but the path terminates prematurely due to the presence of an artefact. A complete path is obtained with 5 manual corrections on the DTL and 3 on the ATL, as shown in the bottom image. This is minimal when considering a total of 1222 coordinates making up the path. α_{AUTO}=97.13%; β_{AUTO}=39.84%; $\alpha_{SEMI-AUTO}$=98.77%; $\beta_{SEMI-AUTO}$=90.23%

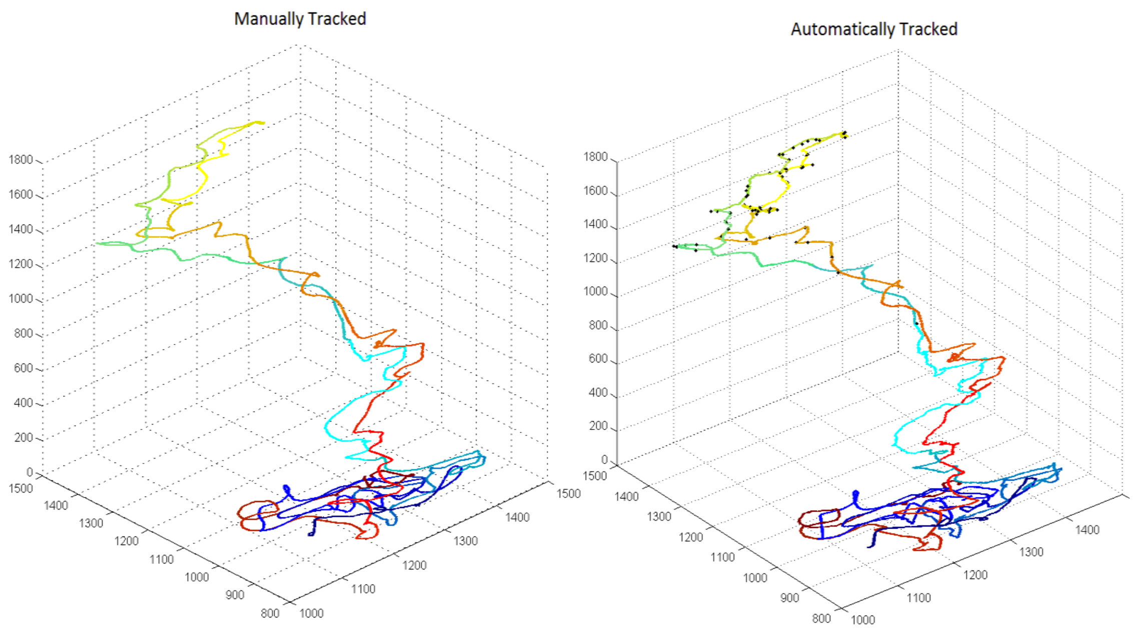


Figure 10.9: A manually tracked rat nephron is shown on the left. The same nephron is successfully tracked automatically by the tracking system with 56 manual corrections in the DTL and ATL. The paths can be seen to be almost identical.

10.5 Efficacy of Validation Steps

Although the types of invalid moves are diverse, the rule-base attempts to model the majority through hard-coded, direct rules while the machine learning validation attempts to model them in a more generalised, less rigid manner.

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. The rejection rates and accuracies are detailed in Table 10.4. The ‘accuracy’ of a validation step refers to the percentage of true positives (moves flagged as invalid that were actually invalid).

Table 10.4: The invalid move rejection rate by the various validation steps and their accuracy is shown. A total of 8017 moves were flagged as invalid in the test set of nephrons.

Validation Step		% of total invalid moves flagged	% of captured 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		
Bidirectional Val.		29.92	18.94	92.05
ML Shape Val.		57.61	42.46	93.62

All four rules have produced accuracies above 90% with the distance validation rule being the most accurate (99.67%) and the machine learning validation being the most highly 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 10.4. Of the 8017 invalid moves, 49.65% are captured by more than one rule.

Ideally, the machine learning 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 moves captured by the machine learning step are those 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.

10.6 Machine Learning Classification

The trained machine learning algorithms eliminate a large number of invalid moves which would have otherwise resulted in multiple nephrons, ICT and blood vessels being linked (42.46% of its detections are not captured by the rule-base). Both the ANN and SVM were capable of forming complex hypotheses and have performed similarly, producing classification accuracies of approximately 93% on the test set. The confusion matrices are contained in Table 10.5.

Table 10.5: The confusion matrix and classification accuracies of the ANN and SVM on a test set of 712 examples. The examples were classified into one of the five classes. Combined results for classes 1 and 2 (valid moves) and classes 3 and 4 (invalid moves) are shown in bold.

Classification Algorithm	Predicted Class	Target Class y_1	y_2	y_3	y_4	y_5	Accuracy (%)	
ANN (threshold=0.7)	y_1	384	28	14	7	0	88.7	**94.8**
	y_2	23	36	4	1	0	56.3	
	y_3	7	3	48	6	0	75.0	**85.3**
	y_4	7	0	13	32	0	61.5	
	y_5	1	0	1	0	97	98.0	**98.0**
	Accuracy (%)	91.0	53.7	60.0	69.6	100.0	83.8	
		96.3		**78.6**		**100.0**		**93.7**
SVM (RBF kernel of width 5)		y_1	y_2	y_3	y_4	y_5		
	y_1	372	18	11	7	0	91.2	**95.1**
	y_2	32	48	6	0	0	55.8	
	y_3	17	0	63	26	7	55.8	**79.7**
	y_4	1	1	0	13	0	86.7	
	y_5	0	0	0	0	90	100.0	**100.0**
	Accuracy (%)	88.2	71.6	78.8	28.3	92.8	82.3	
		96.1		**80.9**		**100.0**		**93.0**

The class 5 output of both classifiers is highly accurate and is successfully used as the region signal. The classification accuracies of the first 4 output classes are variable between the ANN and SVM. For example, the ANN is better at predicted y_4 while the SVM is better at predicted y_2. For purposes of final classification, many false positives and negatives are irrelevant, as long as they belong to another acceptable class, e.g. a move involving elongated cross-sections (y_2) can be classified as y_1 and a glomerular move (y_4) can be classified as abnormal (y_3). The first four output classes (y_1-y_4) are thus combined into a final decision Y according to equation 10.1.

$$Y = \tfrac{1}{2}(y_1 + y_2 - \max(y_3, y_4) + 1) \qquad (10.1)$$

Y ranges from 0 (an invalid move) to 1 (a valid/normal move). The threshold applied to Y determines the binary decision on validity of a move.

The ANN was made purposely less sensitive and more precise by selecting a high threshold (0.7) in order to substantially minimise false positives. The width of the RBF kernel of the SVM was also chosen to minimise false positives, but it was not as flexible as the threshold of the ANN in manipulating the achieved sensitivity and precision. The confusion matrix and performances of this final classification are detailed in Tables 10.6 and 10.7.

Table 10.6: The confusion matrix of the final classification of the test set.

Classification Algorithm	**Predicted Class**	**Target Class**	
		Valid	Invalid
ANN (threshold=0.7)	Valid	444	34
	Invalid	19	120
SVM with RBF kernel (width=5)	Valid	439	39
	Invalid	2	137

Table 10.7: Various performance indicators for the ANN and SVM. The SVM shows slightly superior behaviour. Equations from [**28**].

		Performance (%)	
Indicator	**Equation**	ANN (threshold=0.7)	SVM with RBF kernel (width=5)
Accuracy	$(TP + TN)/Total$	91.41	93.35
Precision	$TP/(TP + FP)$	95.90	99.55
Sensitivity	$TP/(TP + FN)$	92.89	91.84
Specificity	$TN/(TN + FP)$	86.33	98.56
F1 Score	$2TP/(2TP + FP + FN)$	94.37	95.54

From the indices in Table 10.7, the SVM shows slightly better performance than the ANN. It produces fewer false positives on the test set. However, it is less flexible for use than the ANN due to its binary valued output. The continuous valued output of the ANN is advantageous as the four class outputs can be weighed against one another to produce a more accurate final classification. Examples of true and false positives and negatives produced by the ANN are shown in Figure 10.10.

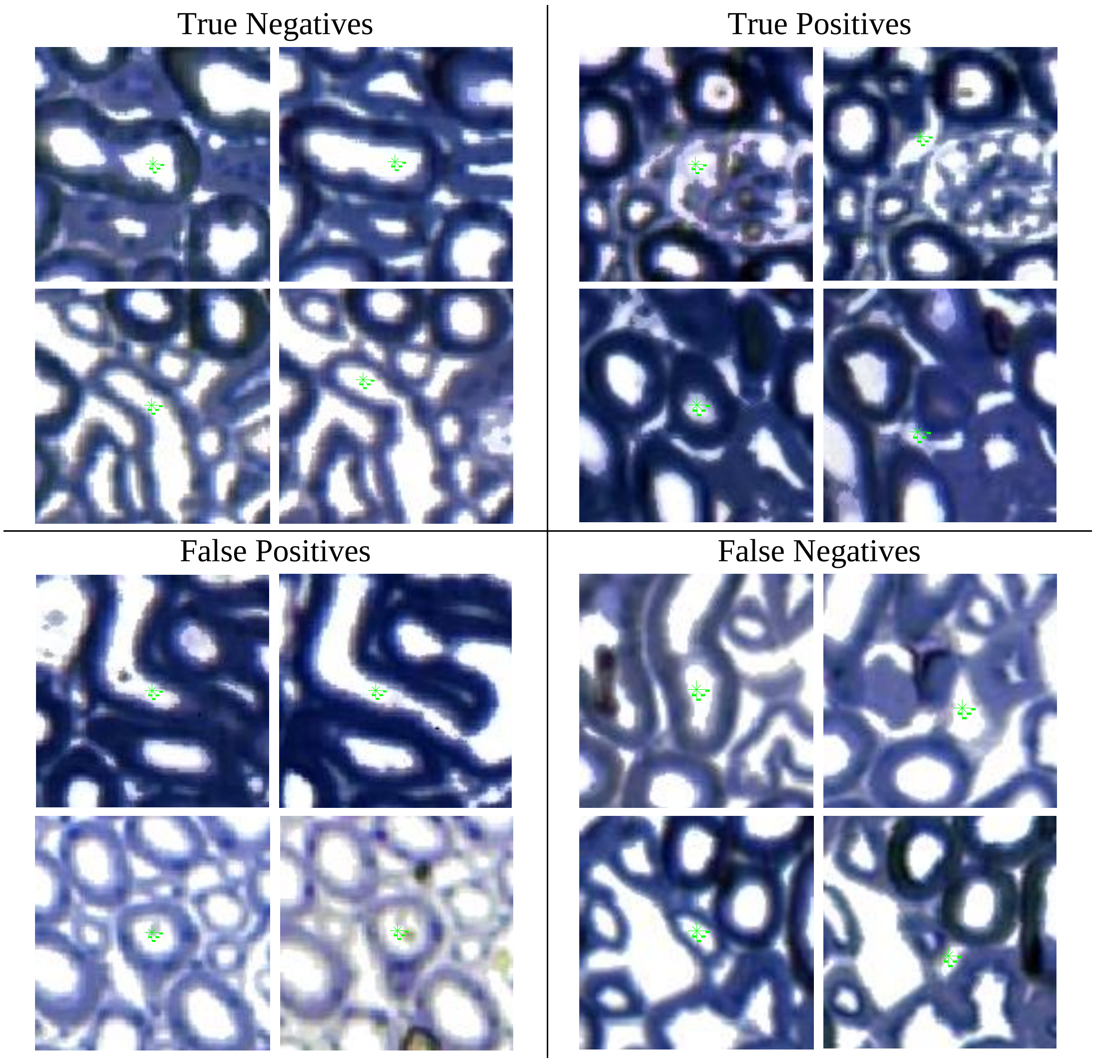
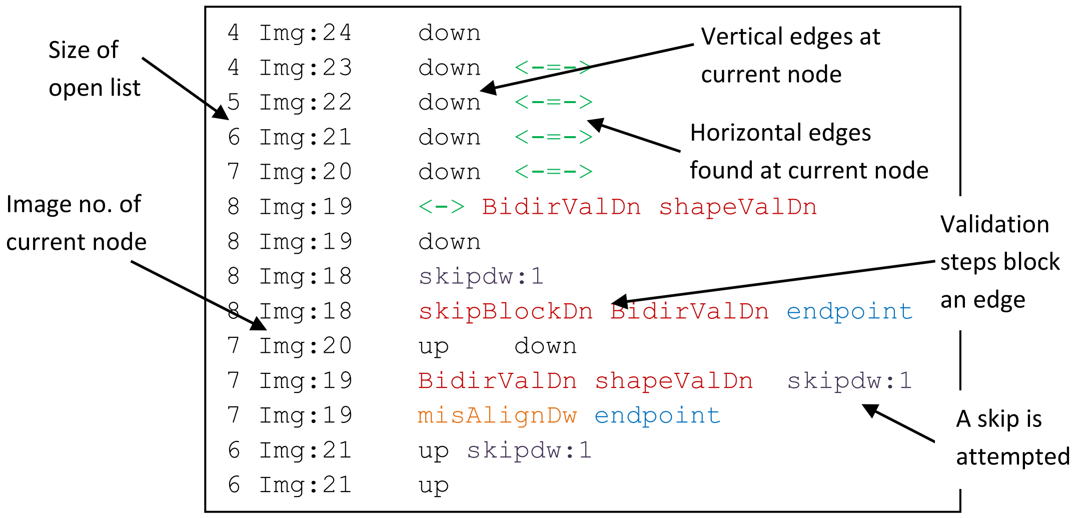


Figure 10.10: Examples of true and false positives and negatives produced by the ANN are shown. False negatives typically involve connective tissue cross-sections. False positives involve nephron cross-sections which have unusual characteristics.

Most false positives seem to consist of nephron cross-sections that were fragmented due to matter in the lumen or non-ideal pre-processing. Moves involving C-shaped, elongated nephron cross-sections were also sometimes mistaken for invalid moves due to their low solidity which is normally a characteristic of glomeruli cross-sections. False negatives typically involved ICT cross-sections which were similar in appearance to nephrons. These occur most frequently when skips are made at the last few cross-sections of a bend.

10.7 Monitoring Runtime Output

It is useful to monitor various variables and flags during the tracking process. For purposes of prototyping, verification and testing a live output log is created by the algorithm. The log is useful when analysing a path post-tracking. Figure 10.11 displays a snippet of an output log while tracking a nephron.



```
4 Img:24    down
4 Img:23    down   <-=->
5 Img:22    down   <-=->
6 Img:21    down   <-=->
7 Img:20    down   <-=->
8 Img:19    <-> BidirValDn shapeValDn
8 Img:19    down
8 Img:18    skipdw:1
8 Img:18    skipBlockDn BidirValDn endpoint
7 Img:20    up     down
7 Img:19    BidirValDn shapeValDn  skipdw:1
7 Img:19    misAlignDw endpoint
6 Img:21    up skipdw:1
6 Img:21    up
```

Size of open list
Image no. of current node
Vertical edges at current node
Horizontal edges found at current node
Validation steps block an edge
A skip is attempted

Figure 10.11: An example of an output log during the tracking of a nephron is shown. Each row is the output for one iteration of the tracking code. The left-most number is the size of the *open* list which indicates how many nodes are yet to be explored. The image number of the current node is then output. A number of strings representing the findings and validations at the current node are then shown.

The size of the open list is a useful indicator of the stability of the tracking instance, i.e. if the size diverges at a high rate (grows large very quickly), it is likely that the path has deviated onto another nephron or blood vessel. If many validation steps are being activated over a long period, it is likely that the glomerulus has been reached.

10.8 Processing Times

The main aim of this research was to develop the techniques required for automated tracking rather than to optimise efficiency for a user-end application. Nevertheless, good programming practices have been followed, such as the use of functions, pre-allocation of memory and efficient use of available memory. Parallel processing was used for the pre-processing and feature extraction stages in order to decrease execution time (by using a MATLAB pool [**33**], or cluster, operating on a *parfor* loop). However, the current implementation can be made more efficient. Computational bottlenecks include the convolution (which uses a 2D FFT) required for image alignment, continuous calling of the ANN structure and reading in three images per iteration (which processes one node) of the algorithm. Specifications of the computer that was used are detailed in the Appendix. Using this computer, the algorithm processed 3 nodes per second.

MATLAB's high level language and built-in toolboxes enabled rapid prototyping and testing. However, it is generally slow at run-time in comparison to a possible implementation in C++ or another lower level language. Further parallelisation and use of a GPU for imaging operations would also improve speed. The execution times taken by various parts of the tracking code were measured using the MATLAB Profiler, the details of which are presented in Table 10.8.

Table 10.8: The distribution of time among the main components of the code is shown for the automatic tracking of the PCT and PST of a short-looped mouse nephron (no manual interventions).

Piece of Code (MATLAB function name)	**Time (%)**	**Time (sec)**
Reading in 1-3 images (imread)	21.52	95.31
Image alignment (conv2)	20.62	91.29
Machine learning validation (nnet)	24.13	106.84
Reading shape profiles (iomatfile)	19.30	85.47
Rest of tracking code	14.43	63.92
Total	**100**	**442.83**

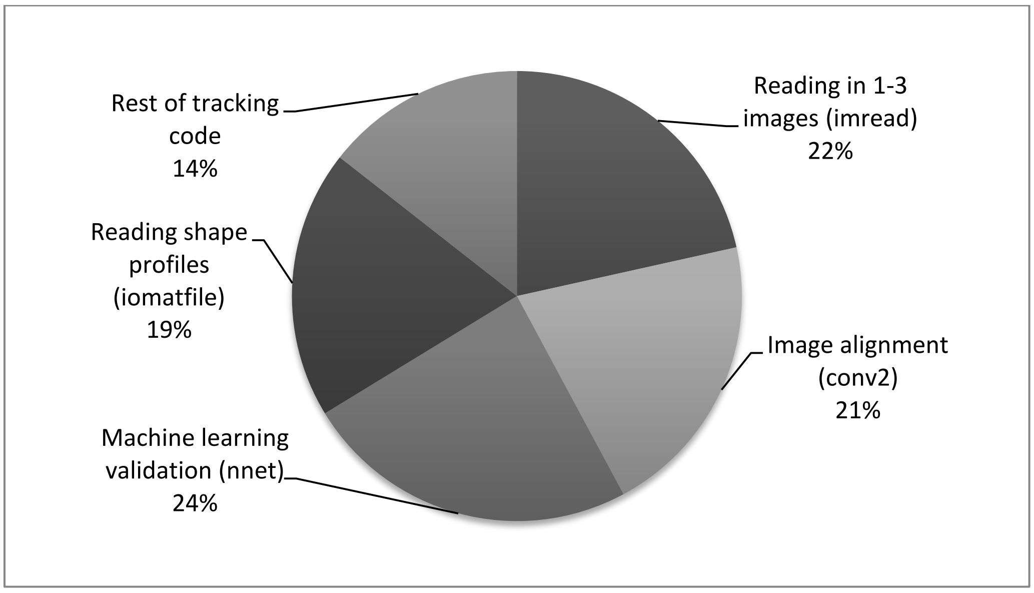


Figure 10.12: A pie chart of the distribution of time among the main components of the code is shown.

The times taken by the high level stages per mouse and rat image were also measured and are presented in Table 10.9. The timings are proportional to the number of cross-sections in the image, hence the longer processing times for inner medullary images (see Figure 6.8). While most operations in the pre-processing stage are image-wide (and therefore less dependent on the number of cross-sections), the feature extraction stage is highly dependent on the number of cross-sections as each cross-section is individually processed.

Table 10.9: The times taken to process cortical and medullary images of the mouse and rat datasets by the three stages of the nephron tracking system.

Process		**Average Time (sec/image)**	
		Mouse	Rat
Pre-processing	Cortex	2.31	3.64
	Medulla	3.94	6.21
Feature Extraction	Cortex	7.68	14.20
(using 8 parallel cores)	Medulla	13.08	55.28
Tracking one short-looped nephron		15 min/nephron	30 min/nephron

CHAPTER 11

Analysis & Discussion

The validation steps generally increase α (accuracy, or similarity to manually tracked nephron) while manual intervention increases β (the extent to which a nephron is tracked). Just as the validation steps eliminate invalid moves, they also block valid moves in the presence of artefacts, image distortions and misalignments, which cause normal morphology to appear abnormal. This is further described in Table 11.1.

Table 11.1: A summary of the implications and effects of different types of artefacts on the tracking process.

Artefact	Implication	Effect on Tracking
Bright centre	Stronger histogram equalisation needed, which over-segments larger nephron cross-sections, producing numerous independent binary components for a single nephron cross-section.	Bidirectional validation triggers which results in premature termination = more manual interventions
Tissue folds, Stretching, compression	Missing tissue and large misalignment which cannot be corrected is produced.	Jumping onto a cross-section of another structure as the fold brings other tissue directly underneath.
External matter	Obscures the nephron cross-sections either completely (missing cross-sections) or partially (change in shape of cross-sections)	Premature termination if obscuration is complete or shape validation is triggered and blocks movement for partial obscuration.
Bright spots	Under-segmentation (merging) of a small group of cross-sections	Other structures are linked to the nephron being tracked through the merged cross-sections

11.1 Performance per Area of the Nephron

Each portion of the nephron is discussed with reference to the results in Tables 10.1 and 10.2. A statement applies to both the mouse and rat datasets if it is not explicitly stated.

The majority of the cortical labyrinth is composed of the PCT and PST, which form 43% of a nephron's length (from measured β values). The algorithm is able to track the full length of the PCT and PST in the mouse and rat with averages of 2 and 10 manual corrections, respectively. The manual corrections are only required when large distortions and artefacts are encountered. Although the PCT was predicted to be the most challenging part to track due to its convoluted nature, it is tracked with high accuracy (95.14% in the mouse and 96.34 in the rat) as:

- The cross-sections are well isolated as they are large in diameter (15-30 pixels wide) and well-defined (they have thick walls).
- The average distance between neighbouring cross-sections (≈ 25 pixels) is larger than the average image misalignment of 4 pixels.

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 blood vessel (afferent/efferent arterioles), which then joins blood vessel systems and other glomeruli, which is undesirable.

The PST of the mouse is also tracked well with 98.24% accuracy 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.

As the PST narrows into the DTL, class 5 moves are successfully registered by the machine learning algorithm. 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 the inner medulla tremendously as ambiguity decreases when only one unidirectional path is allowed to be formed. Without this, incorrect links are easily made to DTL cross-sections of other nephrons, especially where cross-sections incorrectly merge when bright spots are present in the image.

The DTL in the mouse and rat are tracked with moderate accuracies of 80.57% and 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 of the rat requires many manual corrections (≈ 24) to produce high tracking extent. Frequent premature termination occurs because over- and under-segmentation in the binary image cause the cross-sections to appear abnormal to the ANN, thus blocking many moves. Similarly, the cross-sections in the mouse are less well defined than in the cortex, making it more difficult to isolate them.

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. It requires about half the number of manual corrections in both the mouse and rat datasets.

The TAL is tracked well (with 96.32% and 97.48% accuracies in the mouse and rat, respectively) as its cross-sections are well isolated and relatively large (8-12 in the mouse and 13-20 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 artefacts), 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 manual inspection.

In principle the automatically generated path could be more correct than the manually tracked path (due to the potential for human error especially in the inner medulla), but it was assumed that the manually tracked path is absolutely correct.

11.2 Effect of Image Properties on Performance

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 connective tissue cross-sections. A larger slice thickness (e.g. every second slice in the mouse (5 μm) compared to 2.5 μm in the rat) produces less accurate results as the change in morphology is more abrupt from image to image. As shown in Figure 11.1, a certain slice thickness may be sufficient in the cortex where the nephron cross-sectional diameter is large, but it may cause too much of ambiguity for smaller cross sections in the inner medulla.

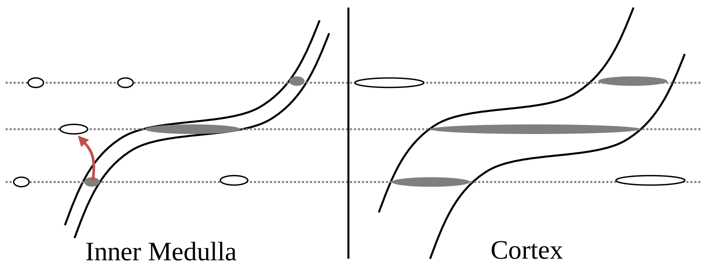


Figure 11.1: A chosen slice thickness has different implications for tracking in the cortex and inner medulla due to the different size of the structures. In the inner medulla, a larger change in morphology per image is perceived. This, along with misalignment and distortion, introduces tracking error.

This increased rate of change in morphology from the cortex to medulla can be measured by a simple ratio of the number of non-overlapping white pixels to the total number of white pixels in each pair of adjacent binary images. The measurements, as shown in Figure 11.2, indicate an increase of 20-25% in the change rate from the cortex to the inner medulla. This simple measure does, however, include the effects of misalignment and can therefore be seen as a *perceived* change in morphology. This is a large contributing factor to the high tracking error in the inner medulla.

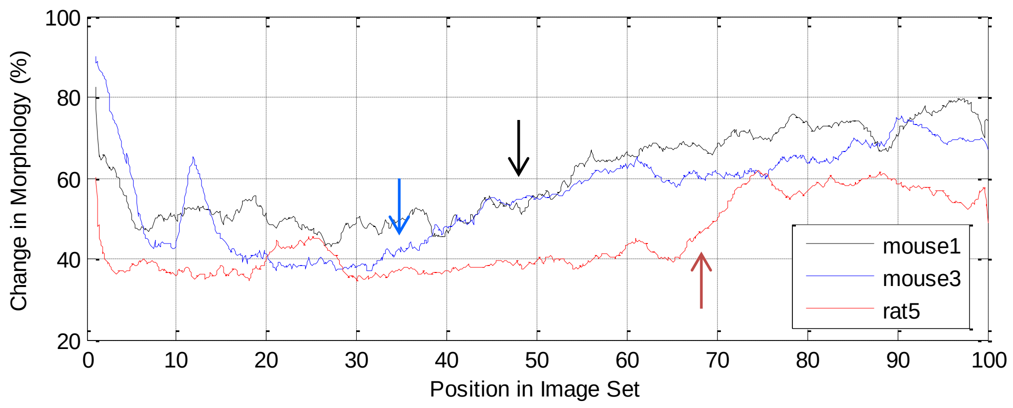


Figure 11.2: The changes in morphology for three image sets were measured. In each case, there is a noticeable increase in the morphology change rate during the transition from the cortex to inner medulla (indicated by arrows).

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 tracking process is chaotic or stability is completely dependent on results of the current iteration). This is especially applicable for tracking in the inner medulla, where the 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. In conclusion, the amount of local image distortions, spatial resolution and slice thickness of images in the inner medulla are the main determining factors of the accuracy and extent of automated tracking in the inner medulla.

11.1 Ceiling Analysis

A ceiling analysis estimates the error due to each component of a pipeline system. It can provide a good idea on which modules of the system are worth expending additional time and effort on [**28**]. A singe real number evaluation metric representing the overall system performance is measured by progressively simulating the ground truth for the previous stages to artificially produce 100% accuracy. Table 11.2 contains the ceiling analysis for the nephron tracking system based on measured performances and projections.

Table 11.2: A high-level ceiling analysis of the system. These values have been determined through careful observation and assessment of the results of the three stages as well as the images themselves.

Component	**Overall accuracy with error carried over**	**Accuracy of stage given ideal previous stages**	**Overall accuracy given ideal previous stages (%)**
Image Quality	75	75	75
Image Pre-processing	73	95	90
Feature Extraction	70	99	94
Tracking Algorithm	63	90	95
Overall	**63**	**63**	**100**

The analysis shows that the quality and resolution of the images themselves are the main limiting upper bound on the accuracy and extent of the overall system. If high quality images are to be used, the systems accuracy goes up to 90%. Thereafter, improving the tracking algorithm would improve accuracy by about 5%.

CHAPTER 12

Recommendations & Future Work

12.1 Recommendations for Future Image Sets

Creating image sets that have minimal distortions and artefacts is essential if complete automation is to be achieved. Attaining an ideal image set of thousands of images may be impractical; however, the images can be artificially manipulated by manually removing and replacing defective images, or specially pre-processing a few images that have outlying characteristics.

The inner medulla poses the biggest challenge. Even by eye, tracking the small, thin-walled, lightly stained nephron cross-sections of the DTL in the midst of hundreds of identical cross-sections proves to be confusing and challenging. A thinner slice thickness (increased resolution in the z-direction) would improve tracking in the inner medulla as the change in position at turns and bends could be better resolved.

Higher resolution images would also offer improved accuracy in isolation and tracking of cross-sections in the inner medulla. An example of a higher resolution image is shown in Figure 12.1, where there are additional features that would be useful that were not visible on the lower resolution images, such as the tubule walls and brush borders. This of course would require more time and effort in capturing the images, as well as massive processing and memory resources if each image is to be at such a high resolution. Another useful addition would be using physical markers on the slides to aid automatic image alignment.

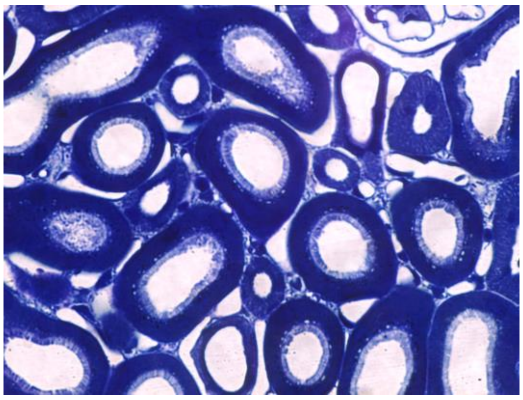


Figure 12.1: A clip of a slide from one of the mouse datasets taken at a much higher resolution is shown. Compared to the image sets used, the nephron cross-sections as well as the connective tissue could be better isolated on higher resolution images. Image from [11].

12.1.1 Staining Choice

The toluidine blue stain seems to have been ideal for the purpose of tracking as it leaves the lumens open (white in appearance), making it easy to isolate one nephron cross-section from another both by eye and automatically in software. The suitability of this stain for the tracking purpose may be emphasised by comparison with differently stained kidney specimens (such as H&E) where the lumens appear cloudy and nuclei are darkly stained [**1**] (which would then have to be compensated for during pre-processing).

One disadvantage of the toluidine blue stain is that different parts of the nephron cannot be easily distinguished. Techniques employed in the studies by Pannabecker and Dantzler [**4**] [**5**] on manually reconstructing the rat nephron may be advantageous in this regard. Immunohistochemistry techniques (using antibodies which bind to segment specific proteins) were used to stain various parts of the nephron differentially. This resulted in the DTL, ATL, collecting duct and blood vessels fluorescing with different colours. Using 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 be easily distinguished from one another and ICT interference would be virtually eliminated as only cross-sections of interest would be highlighted. Fluorescent dyes which still leave the lumens open could therefore be the best staining choice for the tracking purpose.

12.1.2 Image Constraints

If the designed system is to be used on a new image set, the images must conform to the following constraints:

- All images must have the same resolution and a uniform scaling factor.
- The images must be a serial stack labelled sequentially.
- The resolution must be high enough such that nephron cross-sections can be easily distinguished, e.g. a DTL cross-section must be at least 5 pixels in diameter.
- At least every 5μm of the specimen (or preferably a slice thickness less than the diameter of the DTL) must be included in the dataset to adequately represent the change in morphology of the nephrons.
- Transverse sections through the kidney must be used (i.e. such that nephron cross-sections appear mostly circular). Sections producing longitudinally sliced cross-sections will not be accurately tracked, as the appearance of the cross-sections is then completely as shown in Figure 12.2. Tracking longitudinal sections even by eye is difficult and error prone. Also, this type of data is not available, and so training, testing and verifying an algorithm on longitudinal sections cannot be done. Due to inadequate training and tuning, the algorithm would not handle such cases with high accuracy.

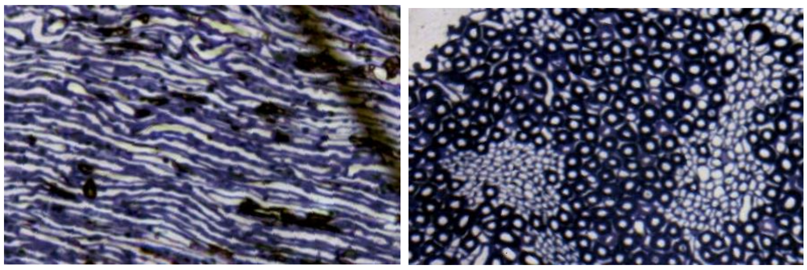


Figure 12.2: Examples of a longitudinal (left) and transverse (right) slice of the kidney. Images from **[**11**]**.

12.2 Future Work

The current approach looks at properties of the current cross-section and potential cross-sections in images above and below; it does not analyse patterns or properties of the local neighbourhood around it apart from using a local window for alignment. Looking at the surrounding area may be the key to solving problems especially in the inner medulla. Also, a machine learning algorithm that operates along a length of the detected path rather than only on a cross-section to cross-section basis may lead to more accurate results, especially in the DTL where the small cross-section diameter causes ambiguity.

The colour or intensity information from the original or equalised image may be used to compute additional features for machine learning algorithms. The colour images in combination with a full six-parameter homography could be used for a more accurate image registration, which may improve the tracking results.

Additional properties of the path around the current node, such as a 3D direction vector, can also be modelled and used for tracking conditions and validations. Even though the current algorithm can track nephrons orientated in various directions, the system still only has three degrees of freedom, for example a nephron segment that is angled 45° to the x-y plane will be tracked as a combination of horizontal and vertical edges rather than directly at 45° in the 3D image space. Future approaches for tracking could perhaps use more degrees of freedom.

This study focused on developing the methods required for (semi-) automated tracking. In order for the system to be used practically on a large number of nephrons, a more efficient version should be implemented in a language such as C++ using neural network and image processing libraries, many of which are open source. A user interface for system calibration, tracking (including manual intervention) and viewing of results should be included.

CHAPTER 13

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 was presented. A number of features were extracted in order to retain shape information during the data abstraction process. The ANN and SVM have high classification accuracies of ≈ 93% and eliminate invalid moves caused by a number of hindering factors such as artefacts and distortions.

The system is successfully able to track large portions of the nephrons automatically through both highly convoluted and straight paths. Particularly, the PCT, PST and TAL (which form more than half of the nephron length) are tracked with high extents and accuracies in both the mouse and rat datasets. The DTL and ATL prove to be problematic due to image artefacts in combination with the small nephron cross-section size, thin walls and high tubule density in the inner medulla. These are tracked with good accuracy but require many manual corrections to achieve high extent. The DCT is tracked well in the rat but not in the mouse.

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 a nephron path, each of which had to be previously manually tracked. Although complete automation is still elusive, the system saves a considerable amount of time and effort compared with the manual tracking task as it performs 99% of the task automatically. Minimising image defects is crucial in improving performance and decreasing the amount of manual intervention required.

The developed system thus serves as a semi-automatic tool to aid the tracking process, decreasing the number of user interactions from 1100 to 17 per mouse nephron and 5000 to 62 per rat nephron. The methods developed during this study form a foundation for further development towards a fully automated nephron tracking system.

REFERENCES

[1] L. C. Junqueira and J. Carneiro, "Basic Histology - Text & Atlas," in *The Urinary System.*: McGraw-Hill, 2005.

[2] William A. Beresford. Histology Full-Text, Chapter 23 Urinary System, Anatomy Department, West Virginia University, Morgantown, USA. [Online]. http://wberesford.hsc.wvu.edu/histolch23.htm

[3] Pannabecker TL, Dantzler WH, Layton HE Layton AT, "Functional implications of the three dimensional architecture of the rat renal inner medulla," 2010.

[4] Thomas L. Pannabecker and William H. Dantzler, "Three-dimensional architecture of collecting ducts, loops of Henle, and blood vessels in the renal papilla," University of Arizona Health Sciences Center, Department of Physiology, Tucson, Arizona, American Physiological Society, 2007.

[5] Thomas L. Pannabecker, Diane E. Abbott, and William H. Dantzler, "Three-dimensional functional reconstruction of inner medullary thin limbs of Henle's loop," Department of Physiology, College of Medicine, University of Arizona, Tucson, Arizona, American Physiological Society, 85724-5051, 2003.

[6] W. Kriz, "The architectonic and functional structure of the rat kidney," Z Zellforsch Mikrosk Anat, 1967.

[7] Pannabecker Thomas L., "Comparative physiology and architecture associated with the mammalian urine concentrating mechanism: role of inner medullary water and urea transport pathways in the rodent medulla," Am J Physiol Regul Integr Comp Physiol, 2013.

[8] H Ren et al., "Direct Physical Contact between Intercalated Cells in the Distal Convoluted Tubule and the Afferent Arteriole in Mouse Kidneys," PLoS One, 2013.

[9] Erik I. Christensen et al., "Three-dimensional reconstruction of the rat nephron," Am J Physiol Renal Physiol, Department of Biomedicine, Anatomy, Aarhus University, Denmark, 2013.

[10] Xiao-Yue Zhai et al., "Three-Dimensional Reconstruction of the Mouse Nephron," Departments of Cell Biology, Connective Tissue Biology, and Neurobiology, Institute of Anatomy, University of Aarhus, Denmark, American Society of Nephrology, ISSN: 1046-6673/1701-0077, 2006.

[11] Image Sets of 3 Mouse and 3 Rat Kidneys, 2013, Departments of Cell Biology, Connective Tissue Biology, and Neurobiology, Institute of Anatomy, University of Aarhus, Denmark.

[12] Keith L. Moore, Arthur F. Dalley, and Anne M. R. Agur, *Clinically Orientated Anatomy, 6th ed.*: Lippincott Williams & Wilkins, ISBN 978-1-60547-652-0, 2010.

[13] RN Douglas-Denton, B Diouf, MD Hughson, WE Hoy, and JF Bertram, "Human nephron number: implications for health and disease," http://www.ncbi.nlm.nih.gov/pubmed/21604189, 2011.

[14] Arthur C. Guyton and John E. Hall, *Textbook of Medical Physiology, 11th ed.* Philadelphia, Pennsylvania: Elsevier, ISBN 0-7216-0240-1, 2006.

[15] SB. Nicholas, JM. Basgen, and S. Sinha, "Using stereologic techniques for podocyte counting in the mouse: shifting the paradigm.," *Am J Nephrol*, California, USA, 2011.

[16] Gokul Sridharan and Akhil A Shankar, "Toluidine blue: A review of its chemistry and clinical utility," *J Oral Maxillofac Pathol*, Department of Oral Pathology and Microbiology, YMT Dental College and Hospital, Maharashtra, India 2012.

[17] Jun Zhang and Jiulun Fan, "Medical Image Segmentation Based on Wavelet Transformation and Watershed Algorithm," Department of Information and Control, Xi 'an Institute of Post and Telecommunications, , Weihai, Shandong, China, IEEE, 2006.

[18] Cathy Merritt, Tony Kasvand Hiromitsu Yamada, "Recognition of Kidney Glomerulus by Dynamic Programming Matching Method," 1988.

[19] Hong Zhu, XueMing Qian, Tao Huang Jun Zhang, "Genetic Algorithm for Edge Extraction of Glomerulus Area," Department of System Integration, Institute of Information and Automation Engineering, Xi'an, Shannxi Province, China, 2004.

[20] Jun Zhang, Jinglu Hu Jiaxin Ma, "Glomerulus Extraction by Using Genetic Algorithm for Edge Patching," School of Information, Production and Systems, WASEDA University, Kitakyshu, Fukuoka, Japan, 2009.

[21] Paola Campadelli, Elena Casiraghi, and Stella Pratissoli, "Automatic segmentation of abdominal organs from CT scans," Universita' degli Studi di Milano, Department of Computer Science, Milano, IEEE, 2007.

[22] Hae-Yeoun Lee, Noel C. F. Codella, Matthew D. Cham, Jonathan W. Weinsaft, and Yi Wang, "Automatic Left Ventricle Segmentation Using Iterative Thresholding and an Active Contour Model With Adaptation on Short-Axis Cardiac MRI," *IEEE*, vol. 57, no. 4, 2010.

[23] Bahadir Karasulu, "Automatic Extraction of Retinal Blood Vessels: A Software Implementation," vol. 8, no. 30.

[24] Tamir Yedidya and Richard Hartley, "Tracking of Blood Vessels in Retinal Images Using Kalman Filter," The Australian National University and National ICT Australia, Australia,.

[25] Ali Can, Hong Shen, James N. Turner, Howard L. Tanenbaum, and Badrinath Roysam, "Rapid Automated Tracing and Feature Extraction from Retinal Fundus Images Using Direct Exploratory Algorithms," *IEEE Transactions on Information*

Technology in Biomedicine, New York, vol. 3, no. 2, 1999.

[26] Xin Kang et al., "Automatic Labelling of Liver Veins in CT by Probabilistic Backward Tracing," Children's National Medical Center, DC, USA, IEEE, 978-1-4673-1961-4, 2014.

[27] Michael A. Nielsen, "Neural Networks and Deep Learning," in *CHAPTER 2: How the backpropagation algorithm works*.: Determination Press, 2014. [Online]. http://neuralnetworksanddeeplearning.com/chap2.html

[28] Andrew NG. (2014, April) Coursera Online Courses: Machine Learning Course. [Online]. https://class.coursera.org/ml-005

[29] Ioannis Valavanisa, Stavroula G. Mougiakakoua, Spyretta Golematia, Alexandra Nikita, Konstantina S. Nikita John Stoitsisa, "Computer aided diagnosis based on medical image processing and artificial intelligence methods," School of Electrical and Computer Engineering & Medical School, National Technical University of Athens, Athens, Greece, 2006. [Online]. http://www.sciencedirect.com/science/article/pii/S0168900206015415

[30] Kenji Suzuki, "Pixel-Based Machine Learning in Medical Imaging," *International Journal of Biomedical Imaging*, vol. Department of Radiology, The University of Chicago, 5841 South Maryland Avenue, MC 2026, Chicago, IL 60637, USA, November 2011.

[31] Mashor M.Y., Esugasini S., Mat Isa N.A., and Othman N.H., "Classification of Breast Lesions Using Artificial Neural Network," *Proceedings of International Conference on Man-Machine Systems* , 2006.

[32] K Gayathri Devi and R Radhakrishnan, "Automatic Segmentation of Colon in 3D CT images and removal of opacified fluid using cascade feed forward neural network," *Institute of Technology, India*.

[33] MATLAB Version R2012a, MathWorks, Image Processing Toolbox; Neural Network Toolbox; Statistics Toolbox.

[34] Prof. Henrik Birn, "The Danish Ministry of Food, Agriculture and Fisheries; Ministeriet for Fødevarer," Institute of Anatomy/Biomedicine 2004/561-818.

[35] K. Wagholikar, "Modeling Paradigms for Medical Diagnostic Decision Support: A Survey and Future Directions," *Journal of Medical Systems*, Aug. 2012.

[36] E. R. Davies, *Computer & Machine Vision: Theory, Algorithms, Practicalities*. Egham, UK, Surrey: Elsevier, 2012.

[37] Dana H. Ballard and Christopher M. Brown, *Computer Vision*. Rochester, New York: Prentice Hall, 1982.

[38] Peter Henderson, Richard Seaby, and Robin Somes, "Growth II," in *Types of growth curve - Logistic curve*. Penington, Lymington, Hampshire: Pisces Conservation Ltd,

2006.

[39] Guojun Gan, Chaoqun Ma, and Wu Jianhong, "Data Clustering Theory, Algorithms and Applications," in *Chapter 9: Center-based Clustering Algorithms*. Philadelphia, ASA, Alexandria, VA: ASA-SIAM Series on Statistics and Applied Probability, 2007.

[40] L. Wojnar and K.J. Kurzydłowski, *Practical Guide to Image Analysis*.: ASM International, 2000.

[41] Barbara Zitova and Jan Flusser, "Image registration methods: a survey," *Elsevier: Image and Vision Computing*, Department of Image Processing, Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic 2003.

[42] Amit Patel. (2014) Stanford Theory Group. [Online]. http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

[43] Kenji Kira and Larry Rendell, "The Feature Selection Problem: Traditional Methods and a New Algorithm," 1992.

APPENDICES

Appendix A: Longitudinal Reconstructions

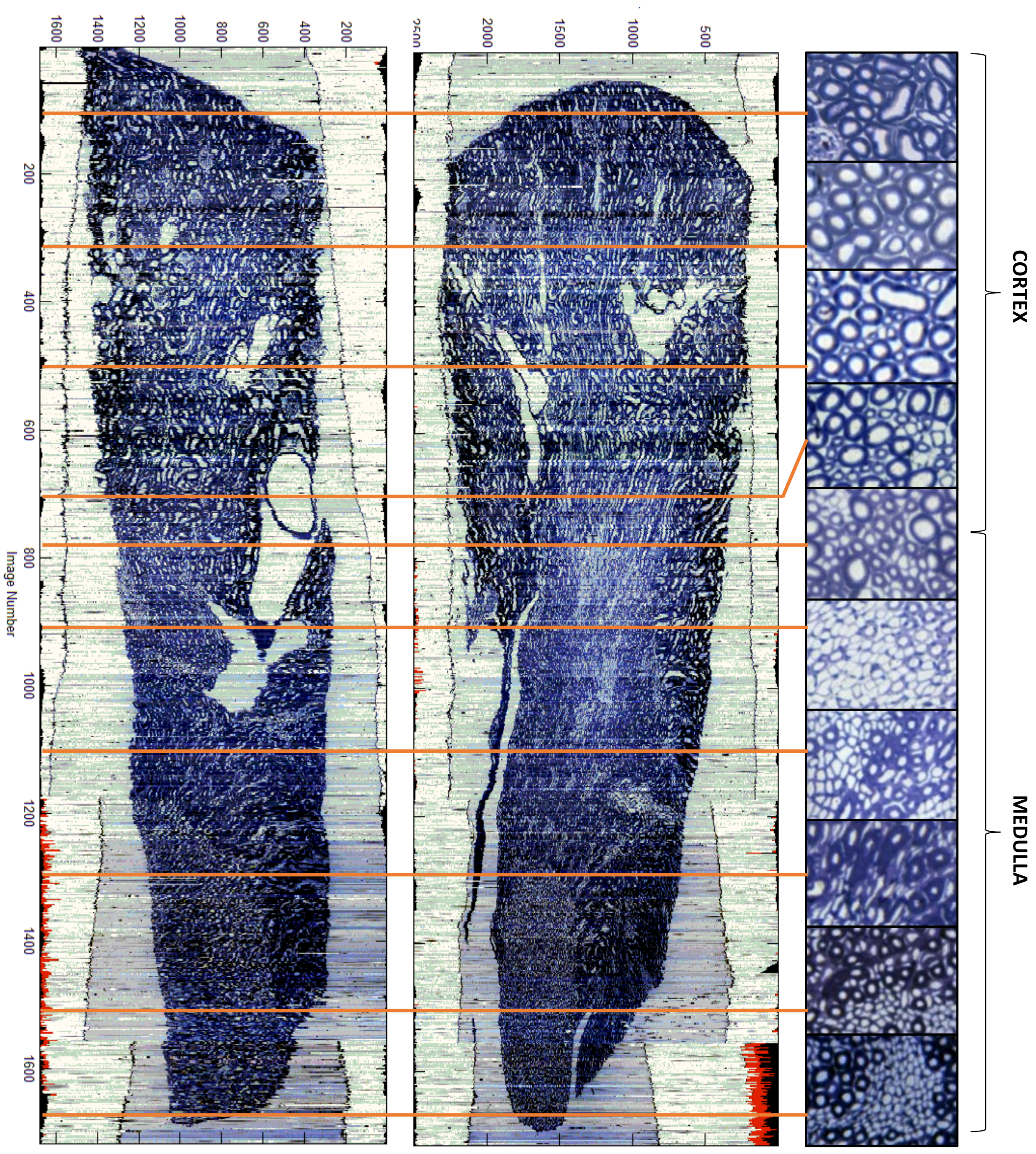


Figure A1: Two longitudinal sections were reconstructed from a mouse image set. The change in morphology can be seen, as well as the zones at which the changes occur. Clips of the transverse images are also shown.

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.

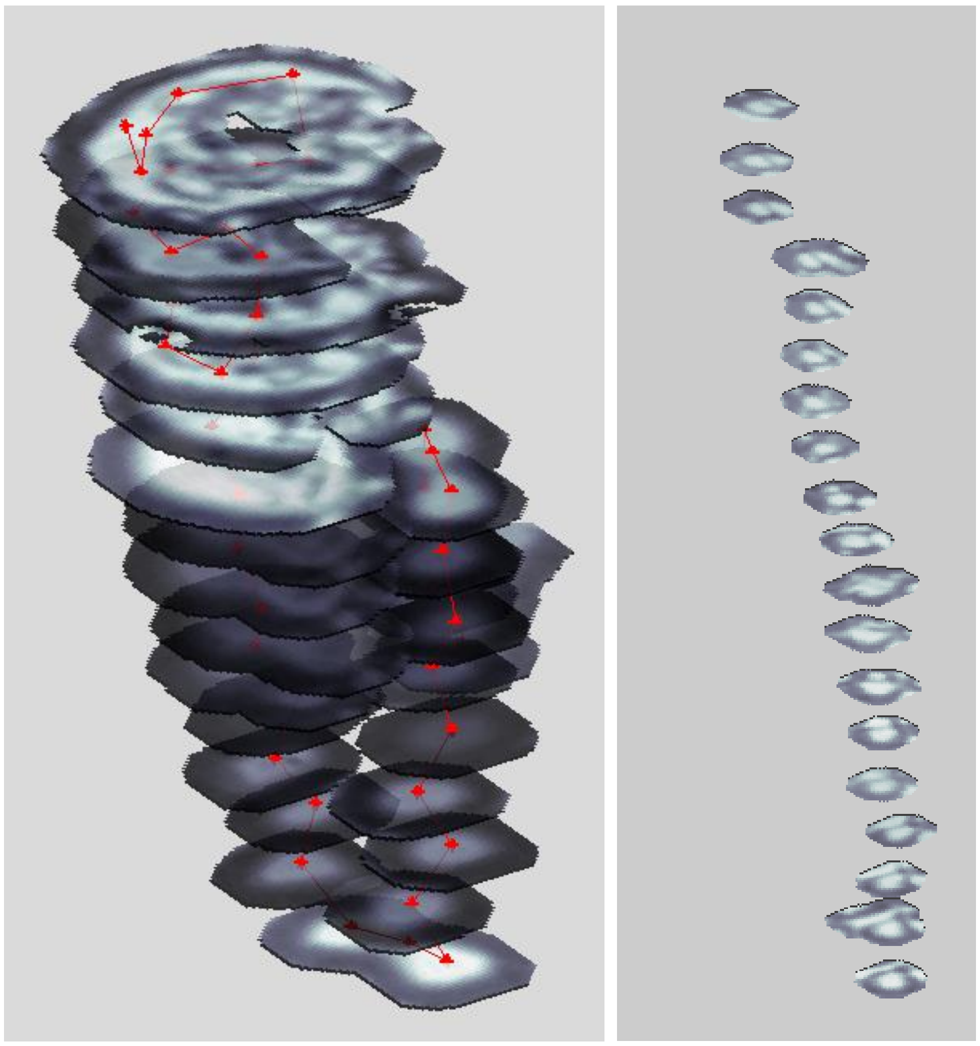


Figure B1: The results of a tracking instance were used to automatically extract the original nephron cross-sections (in the colour image) relating to the tracked nephron. The nodes and their interconnection are shown in red. The slices show an area where the nephron proceeds downwards and then turns 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.

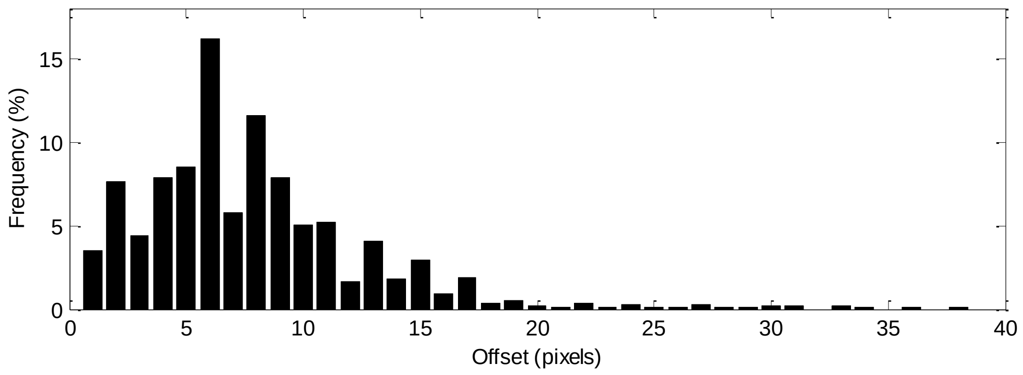
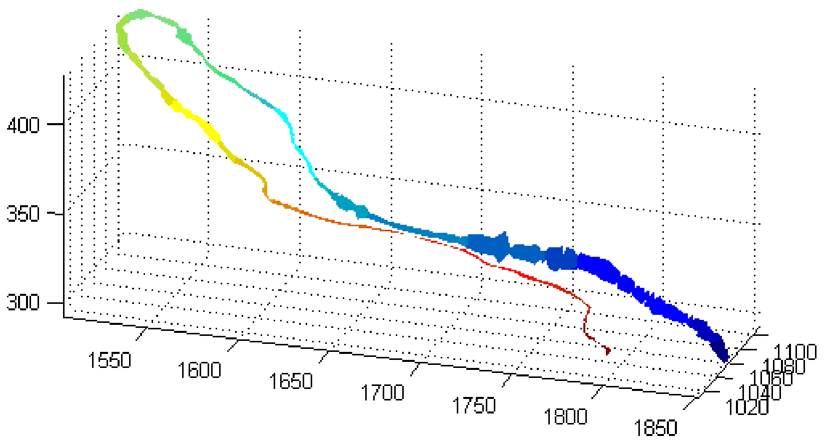


Figure B2: A histogram of the image offsets in the x-y plane between adjacent images during the tracking of 12 mouse nephrons



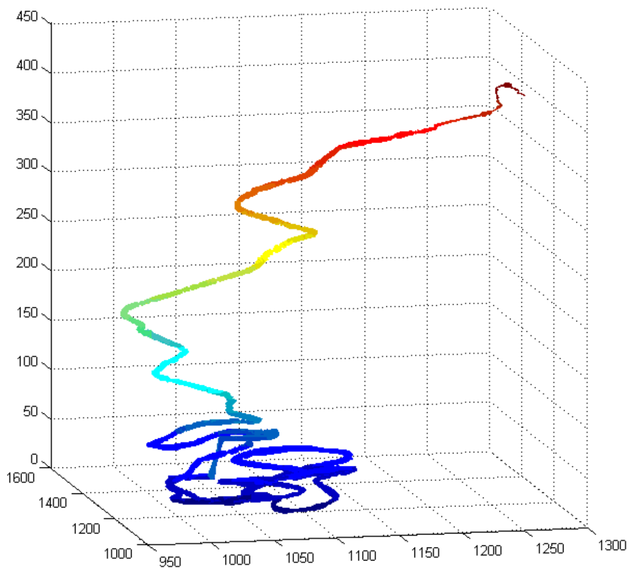


Figure B3: A loop of Henle (top) and the PCT, PST and DTL (bottom) of a mouse nephron is plotted using the extracted minor axis feature. The radius varies wit the size of the cross-sections. The transition between the thick PST and thin DTL can be seen.

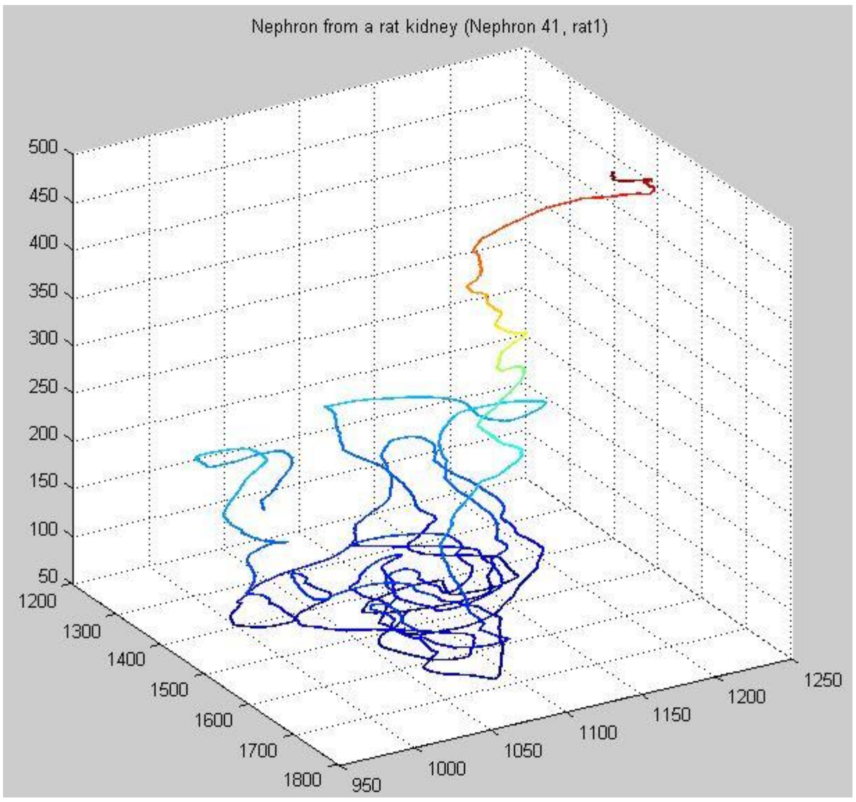


Figure B4: The PCT of a rat nephron is shown to convey the intricacy and complexity of the convolutions present in the PCT.

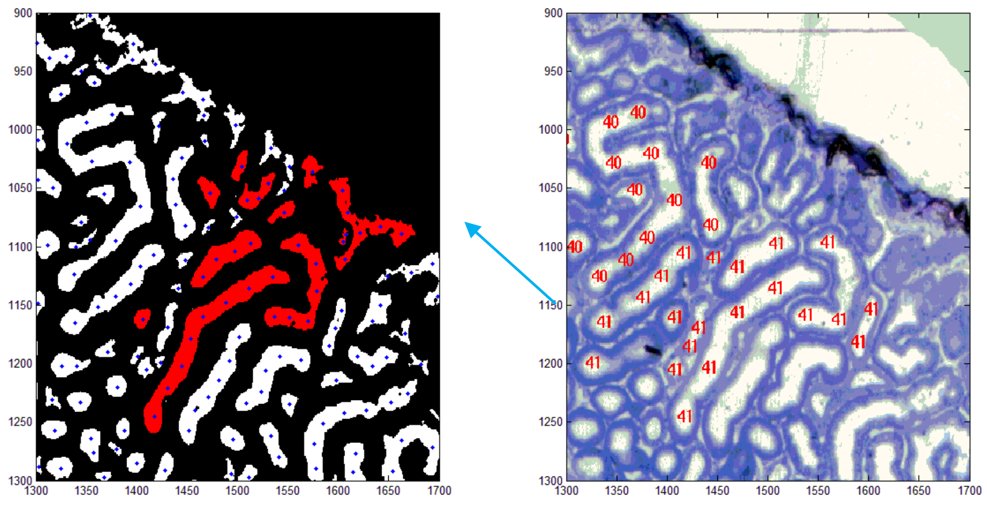


Figure B5: Without machine learning validation, incorrect linkages are often made to interstitial tissue cross-sections, which in turn link to other nephrons. Unlike machine learning validation the other validation rules are not always effective in preventing such cases.

Appendix C: Results of Nephron Tracking

Average Runtime: 5.78 minutes/nephron

<u>MOUSE</u>	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

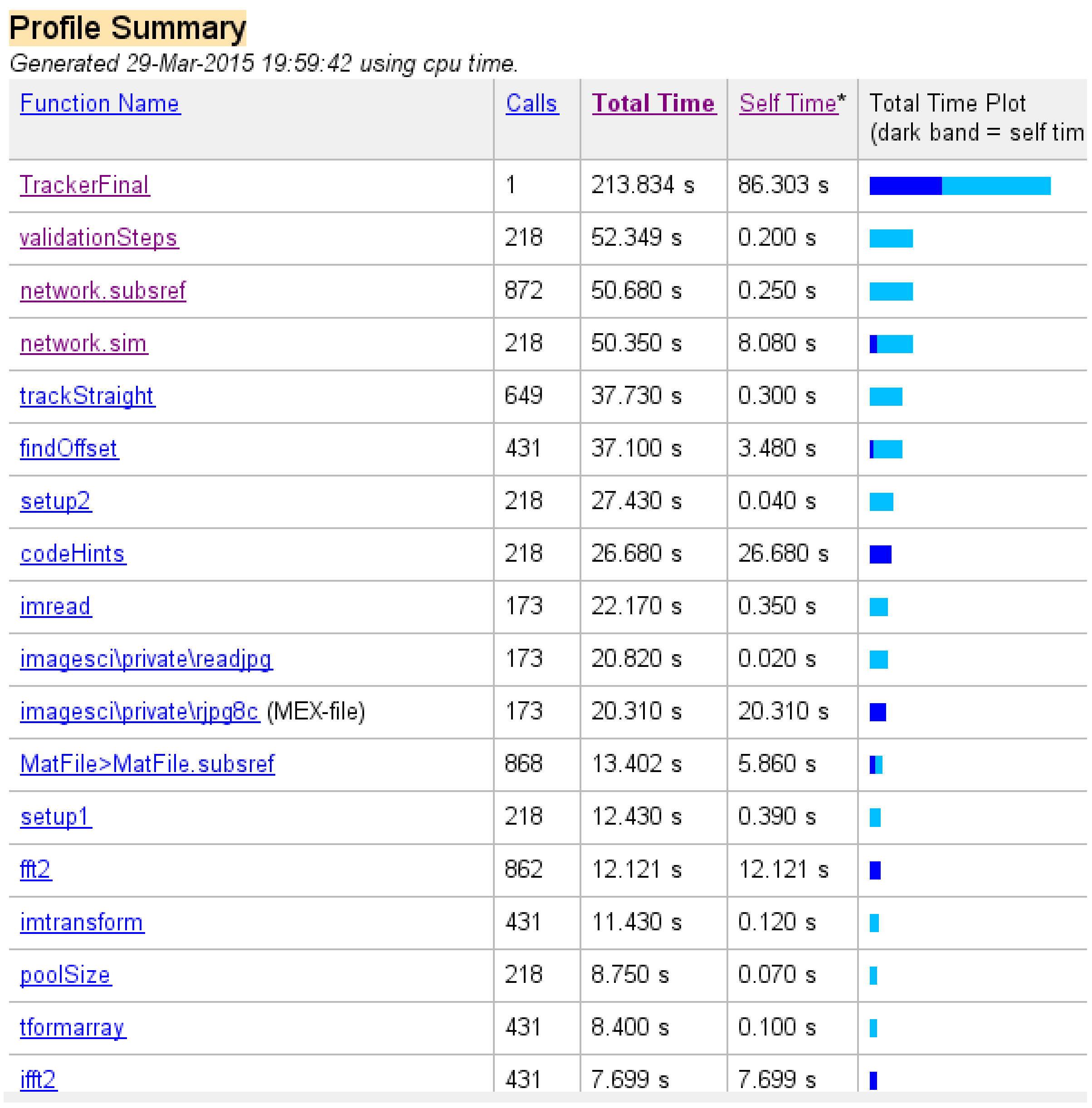
Average Runtime: 28 minutes/nephron

<u>RAT</u>	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

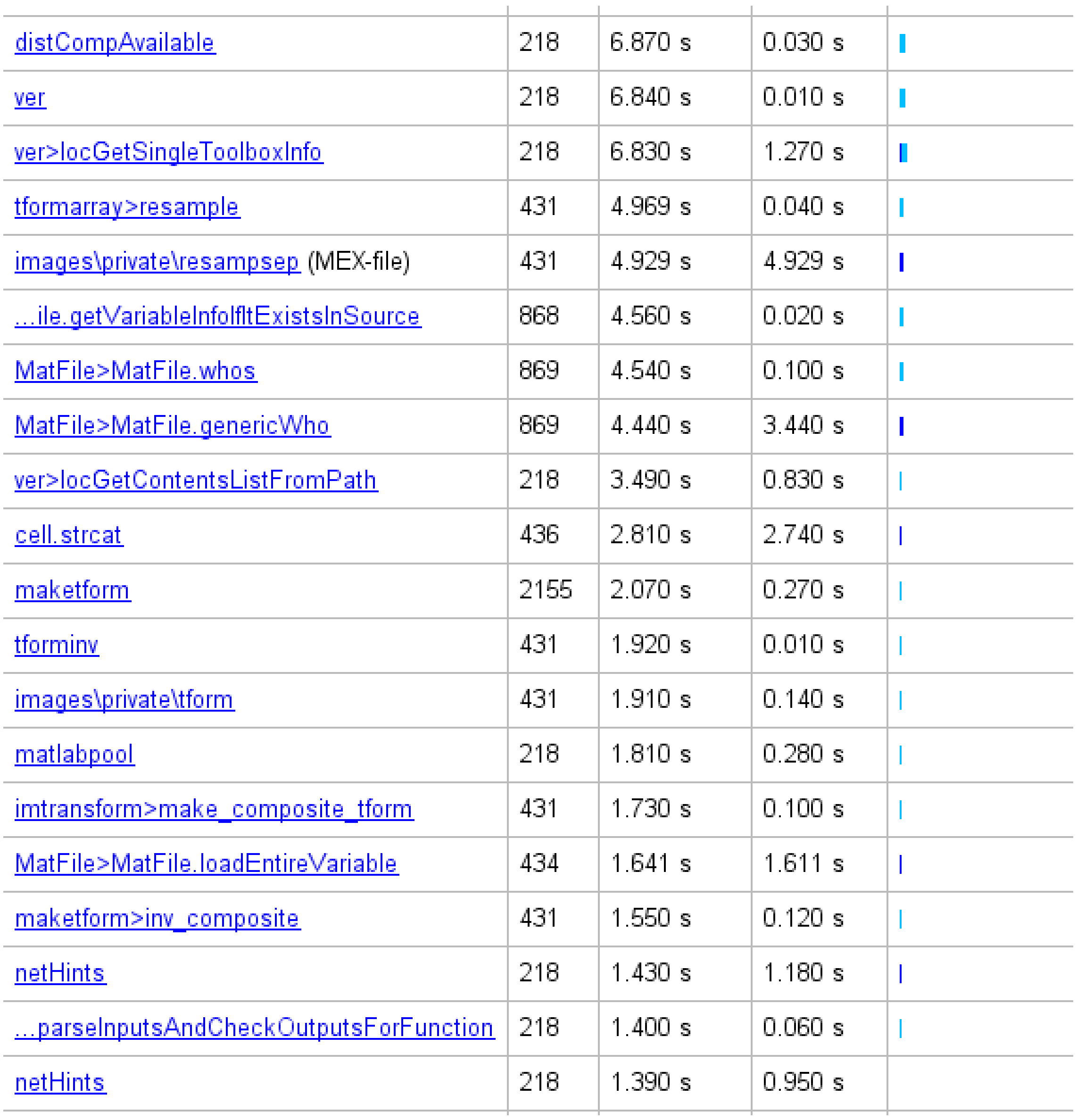
The MATLAB Profiler measures the execution time of functions, sub-functions and subroutines to aid optimisation of code. The profiler was used to measure an instance of tracking (TrackerFinal.m). The results show that neural network validation, reading images into MATLAB and the fft/ifft used during 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.

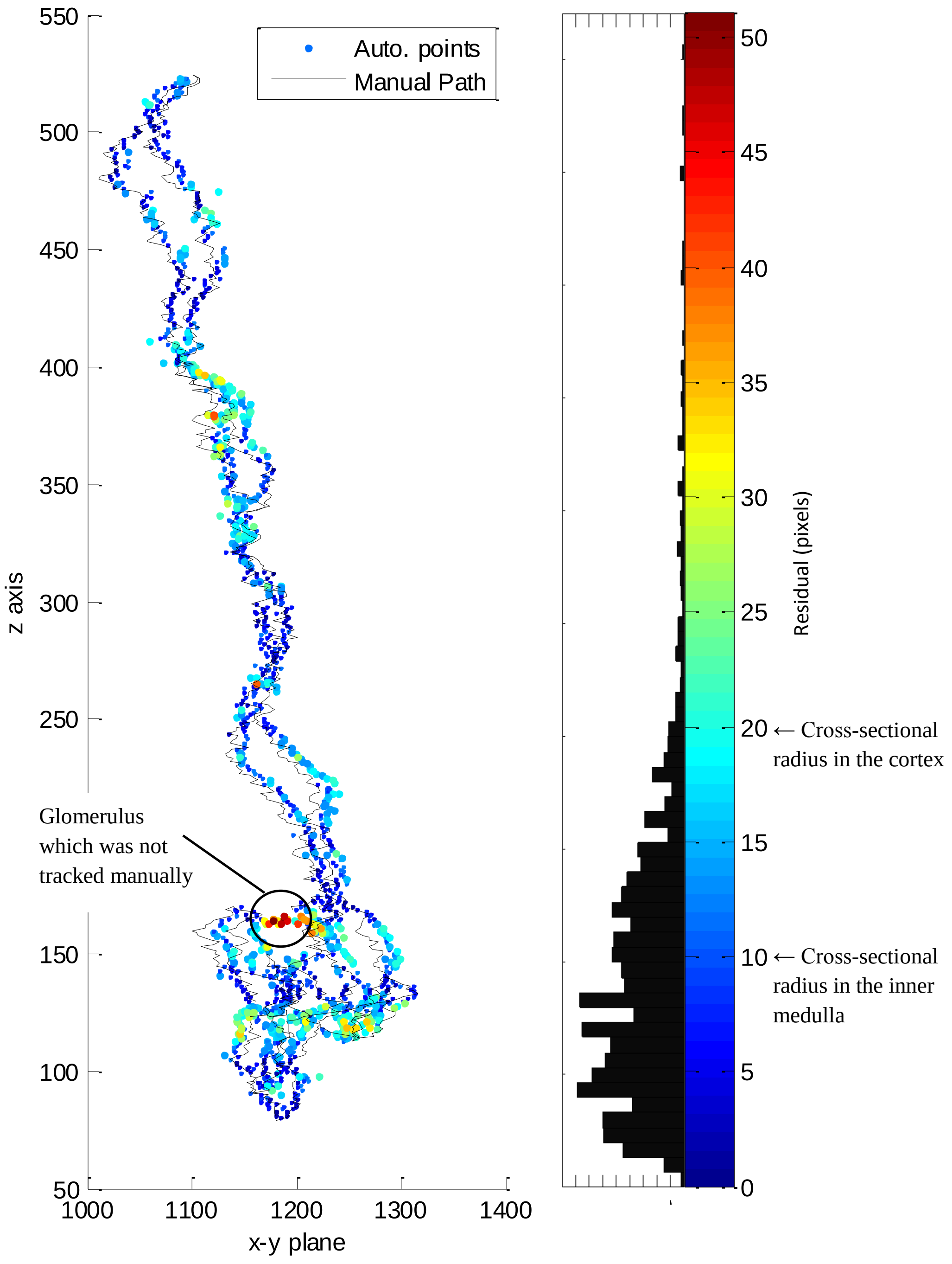


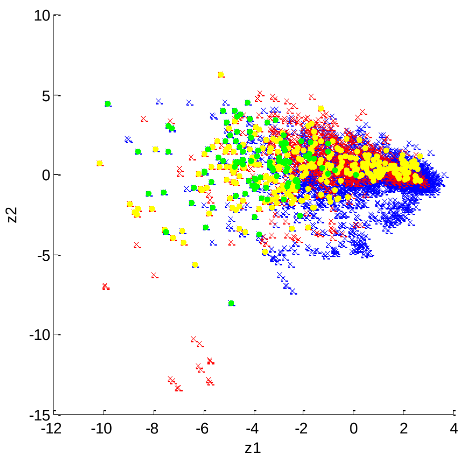
Figure E1: Points from an automatically tracked mouse nephron are compared to the manually tracked path (in black). The similarity measured by the algorithm produces α=97.45% and β=99.84% using a residual threshold of 25 pixels. The points are colour- and size- coded to its corresponding residual value.

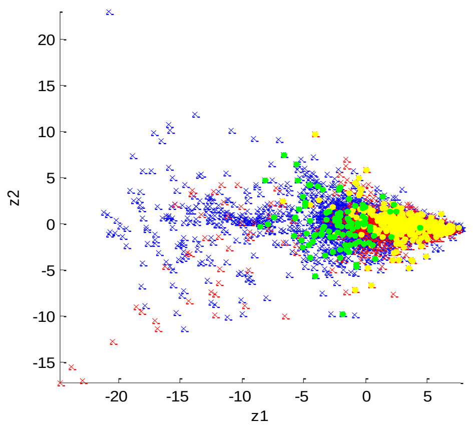
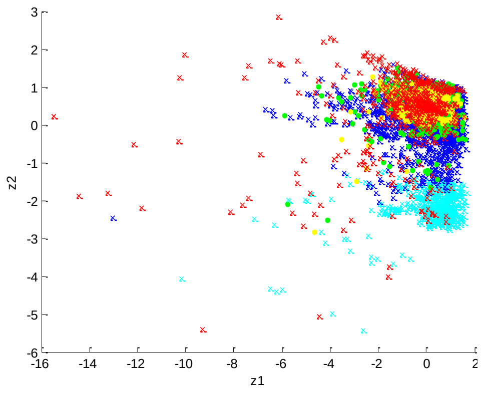
Appendix F: Additional Feature Analysis

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.

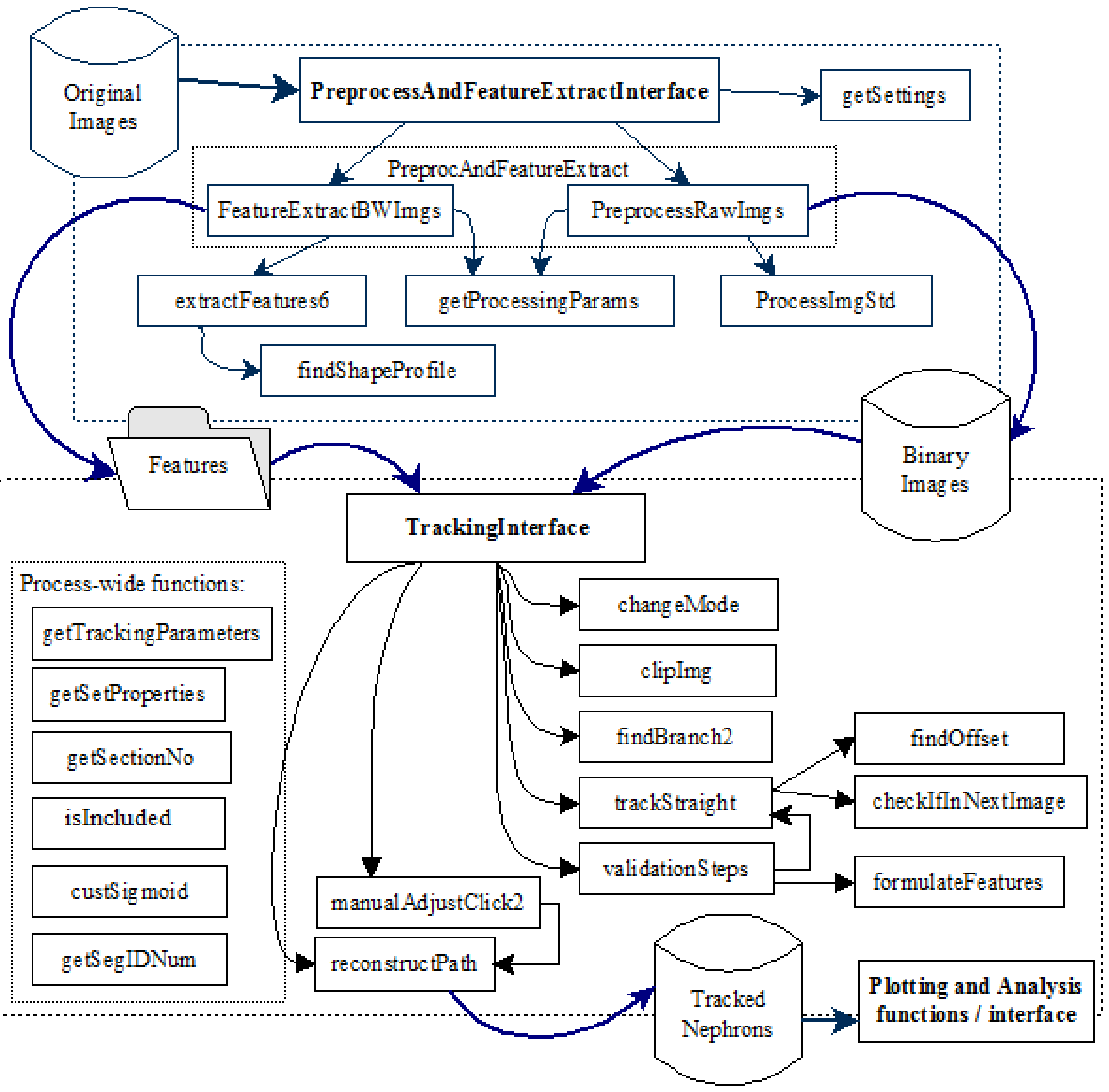


Figure H1: A code dependency graph of the system. The two main scripts of the system make use of various custom-coded functions which are independent of one another. Each function passes information to the function/script above it.

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
```

```matlab
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

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 $\times 4$ 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 $\times 3$ 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

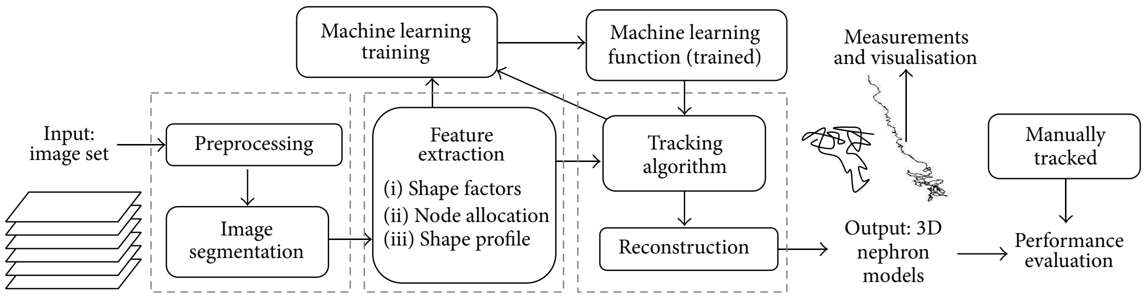


Figure 1: A high level overview of the nephron tracking system, showing the main subsystems and the flow of information between them.

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

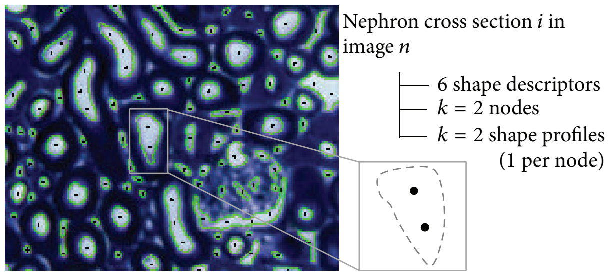

Nephron cross section i in image n
6 shape descriptors
k = 2 nodes
k = 2 shape profiles (1 per node)

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

$$\begin{aligned}\boldsymbol{\theta} &= \arctan\left(\frac{\mathbf{P}_{\mathbf{edge}}(y) - P_{\text{ref}}(y)}{\mathbf{P}_{\mathbf{edge}}(x) - P_{\text{ref}}(x)}\right), \\ \mathbf{r}(\theta) &= \left\|\mathbf{P}_{\mathbf{edge}} - P_{\text{ref}}\right\|,\end{aligned} \tag{1}$$

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.

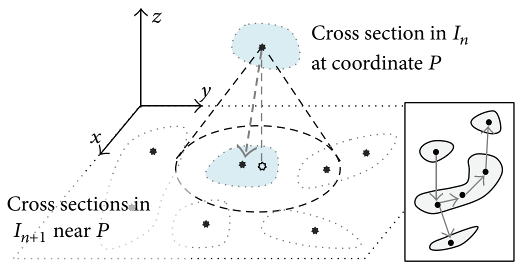

z
y
x
Cross section in I_n at coordinate P
Cross sections in I_{n+1} near P

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:

$$\min \left(\left\| \mathscr{C}_{n\pm1} - c_{kn}^{i} \right\| < r_{\text{track}} \right). \quad (2)$$

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

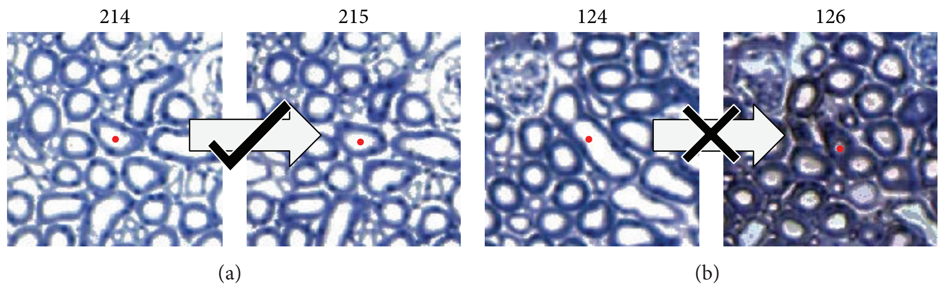


Figure 4: The moves attempted by the unregulated tracking algorithm are captured, displayed, and labelled to form training examples for the neural network. The image shows examples of a valid (a) and invalid (b) move, which will be labelled with a "1" and a "3," respectively.

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].

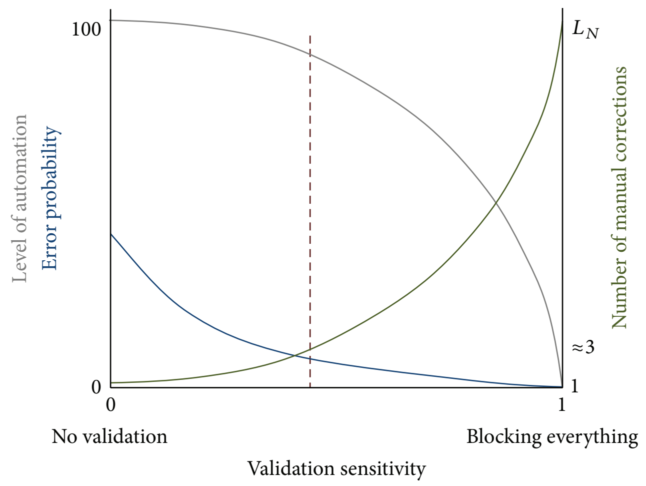


FIGURE 5: The number of false positives increases with increasing validation sensitivity, resulting in premature termination of tracking. This means only a portion of the nephron is tracked, but with a low error, where error refers to deviation onto an incorrect path. If manual correction is used, the 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: $\beta_{\text{MEAN}}/\beta_{\text{IDEAL}}$ (%)	Accuracy: α_{MEAN} (%)	Average number of manual corrections	β_{MEAN} (%)	Extent: $\beta_{\text{MEAN}}/\beta_{\text{IDEAL}}$ (%)	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

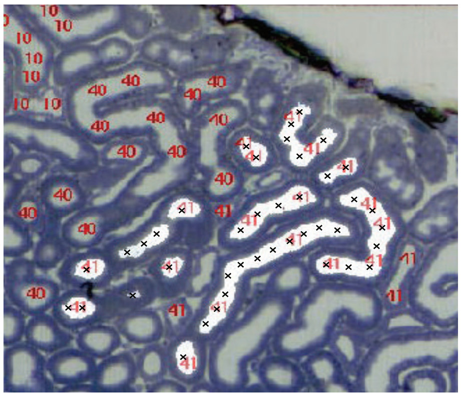


Figure 6: An example of a labelled image is shown with the red numbers representing the different manually tracked nephrons. The automatically tracked nephron (number 41) is superimposed, shown in white with black crosses at the nodes. Unlabelled "41" cross sections are of the DCT which was not tracked in this instance.

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

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.

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.

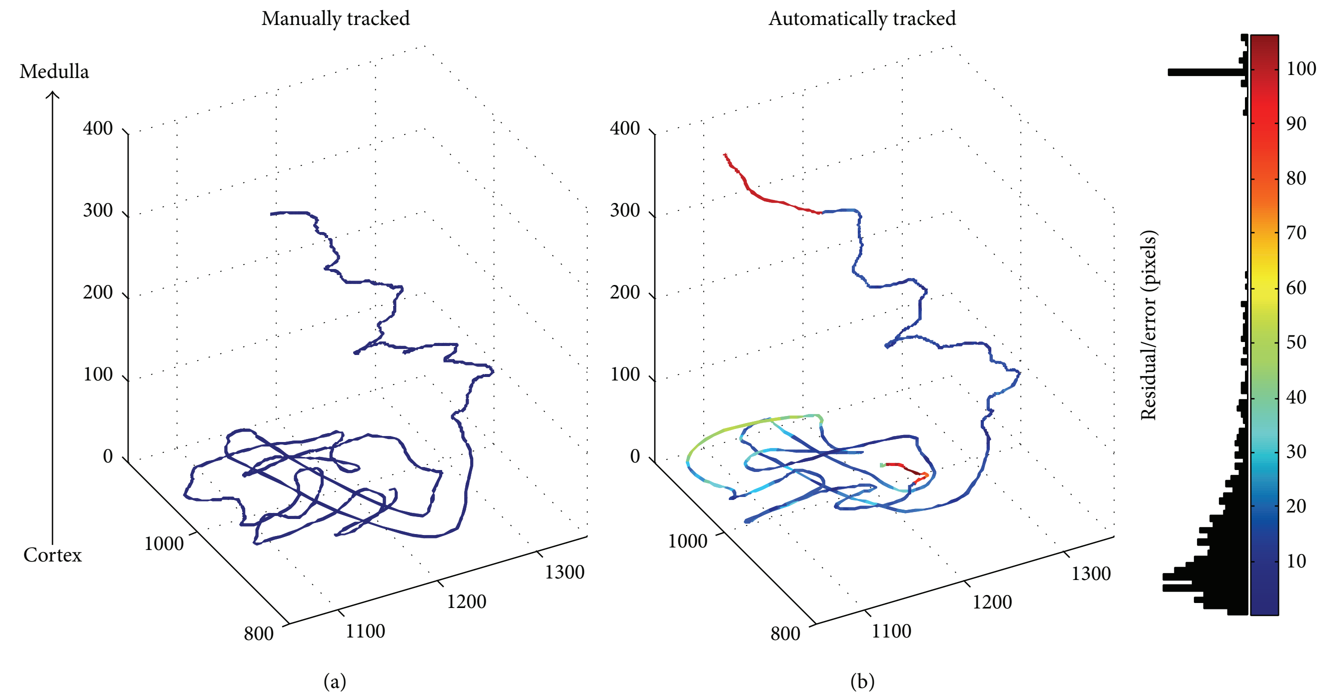


Figure 7: A manually tracked mouse nephron is shown on the left. The same nephron is successfully tracked automatically by the algorithm (with $\alpha = 97\%$) and is shown on the right. Tracking terminates automatically at the glomerulus. Note that, in each plot, the cortex is shown at the bottom and 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.

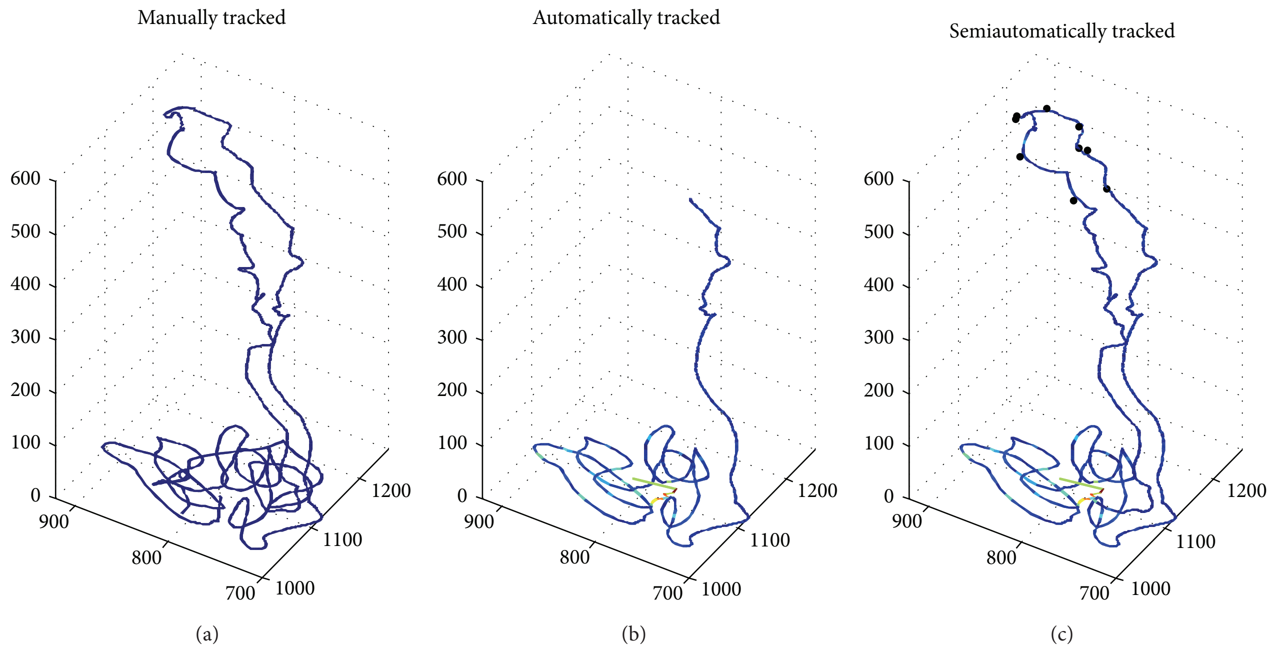


Figure 8: A manually tracked mouse nephron is shown on the left. The PCT and PST are successfully tracked automatically as shown in the middle plot. Tracking terminates due to diminishing tubule size coupled with artefacts in the inner medulla. A more complete nephron path is obtained with 5 manual corrections 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, Euro-Gene, 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.