Biometric Identification of Mice

Master’s Thesis in Computer Engineering

Andreas Ellmauthaler, Eric Wernsperger
Preface

This Master thesis represents work carried out at the School of Information Science, Computer and Electrical engineering at Halmstad University and completes the Masters program in Computer Engineering.

We would especially like to thank our supervisor Kenneth Nilsson for his help, his ideas and most important his patience. Furthermore we want to thank everyone who cheered us up in moments of difficulty and made us believe in ourselves again. Without you this work would not have been possible.
Abstract

The identification of laboratory mice has been an important issue in pharmaceutical applications ever since tests have been performed on animals. As biometric identification has become an increasingly important issue over the past decade, attempts are underway to replace traditional identification methods, which are mostly invasive and limited in code space. This thesis discusses a project that aims at identifying mice by biometrically examining the blood vessel patterns in their ears.

In the proposed algorithm, firstly, the blood vessel structure within the obtained images got enhanced before segmenting the image in blood vessel and non-blood vessel portions. In the next step a sufficient amount of unique feature points got extracted from the segmented image. The obtained feature points were afterwards used for the actual identification procedure.
Out of 20 mice, 18 could be identified successfully using the proposed algorithm.
# Table of contents

1. INTRODUCTION ........................................................................................................... 1  
1.1 Previous work ............................................................................................................. 2  
1.2 Project goals .............................................................................................................. 2  

2. BACKGROUND STUDY .......................................................................................... 3  
2.1 Image enhancement ................................................................................................ 3  
2.2 Image segmentation .................................................................................................. 6  
2.2.1 Threshold techniques ............................................................................................ 6  
2.2.2 Tracking ................................................................................................................ 7  
2.2.3 Other segmentation techniques ............................................................................ 8  
2.3 Image post-processing ............................................................................................. 8  
2.4 Enrollment ................................................................................................................ 9  
2.4.1 Feature extraction ................................................................................................. 9  
2.5 Matching .................................................................................................................. 10  

3. ALGORITHM ............................................................................................................. 11  
3.1 Image Enhancement ............................................................................................... 12  
3.2 Image Segmentation ............................................................................................... 16  
3.2.1 Starting points ...................................................................................................... 17  
3.2.2 Tracking ................................................................................................................ 18  
3.3 Image Post-Processing ............................................................................................ 23  
3.4 Enrollment process .................................................................................................. 25  
3.4.1 Description of the sample .................................................................................... 25  
3.4.2 Input parameters .................................................................................................. 25  
3.4.3 Feature extraction ............................................................................................... 26  
3.4.3.1 Extraction of bifurcation points using morphological operations ..................... 26  
3.4.3.2 Extraction of branch orientation angles ............................................................ 26  
3.4.3.3 Extraction of inter-bifurcation point connection angles .................................... 28  
3.4.3.4 Extraction of inter-bifurcation point distances .................................................. 29  
3.4.4 Comparison of images showing the same ears ...................................................... 29  
3.4.4.1 Comparison based on rotation variant branch orientation angles .................. 29  
3.4.4.2 Comparison based inter-bifurcation point distances ....................................... 31  
3.4.4.3 Preparing data for enrollment ....................................................................... 32  
3.4.4.4 Enrollment ....................................................................................................... 33  
3.5 Matching process ..................................................................................................... 33  
3.5.1 Bifurcation point analysis .................................................................................... 34  
3.5.2 Statistical examination of segmentation results .................................................... 37  
3.5.2.1 Determination of rotation and translation parameters ...................................... 37  
3.5.2.2 Statistics regarding points abandoned by false rotation and translation parameters ........................................................................................................... 37  
3.5.2.3 Statistics regarding common distances between vessel points ............................ 38  
3.5.2.4 Statistics regarding amounts of image points vs. amounts of transformed points ........................................................................................................... 40  
3.5.2.5 Statistics regarding transformed points that cannot be aligned ....................... 41  
3.5.2.6 Statistics regarding transformed points that cannot be aligned..................... 41

---

3.5.2.6 Statistics regarding transformed points that cannot be aligned..................... 41

3.5.2.5 Statistics regarding transformed points that cannot be aligned ....................... 41

3.5.2.4 Statistics regarding amounts of image points vs. amounts of transformed points ........................................................................................................... 40

3.5.2.3 Statistics regarding common distances between vessel points ............................ 38

3.5.2.2 Statistics regarding points abandoned by false rotation and translation parameters ........................................................................................................... 37

3.5.2.1 Determination of rotation and translation parameters ...................................... 37

3.5.1 Bifurcation point analysis .................................................................................... 34

3.5 Matching process ..................................................................................................... 33

3.4.4.4 Enrollment ....................................................................................................... 33

3.4.4.3 Preparing data for enrollment ....................................................................... 32

3.4.4.2 Comparison based inter-bifurcation point distances ....................................... 31

3.4.4.1 Comparison based on rotation variant branch orientation angles .................. 29

3.4.3.4 Extraction of inter-bifurcation point distances .................................................. 29

3.4.3.3 Extraction of inter-bifurcation point connection angles .................................... 28

3.4.3.2 Extraction of branch orientation angles ............................................................ 26

3.4.3.1 Extraction of bifurcation points using morphological operations ..................... 26

3.4.2 Input parameters .................................................................................................. 25

3.4.1 Description of the sample .................................................................................... 25

3.4 Enrollment process .................................................................................................. 25

3.3 Image Post-Processing ............................................................................................ 23

3.2 Image Segmentation ............................................................................................... 16

3.2.2 Tracking ................................................................................................................ 18

3.2.1 Starting points ...................................................................................................... 17

3.2 Image Segmentation ............................................................................................... 16

3.1 Image Enhancement ............................................................................................... 12

2.5 Matching .................................................................................................................. 10

2.4 Enrollment ................................................................................................................ 9

2.3 Image post-processing ............................................................................................. 8

2.2.3 Other segmentation techniques ............................................................................ 8

2.2.2 Tracking ................................................................................................................ 7

2.2.1 Threshold techniques ............................................................................................ 6

2.2 Image segmentation .................................................................................................. 6

2.1 Image enhancement ................................................................................................ 3

1.2 Project goals .............................................................................................................. 2

1.1 Previous work ............................................................................................................. 2

1. INTRODUCTION ........................................................................................................... 1
1. Introduction

In today’s pharmaceutical laboratories, mice are used to test the effects of newly developed drugs. In order to keep track which medication is used on which mouse, an efficient labelling procedure is needed so that all laboratory rodents can be identified uniquely. Up to now this has been achieved by using several methods [1].

A commonly used procedure in this respect is ear punching, where either a small notch near the edge of the ear or a hole in the middle of the ear is punched. The holes and notches are placed according to a predetermined code.

Another frequently used method is ear tagging. Here, a special applicator places an earring or metal tag with an identification number in the pinna of the mouse’s ear.

Tattooing is the preferred method for identifying neonatal mice. The tattoo is placed either on the tail, toe or ear.

Subcutaneously implanting a microchip is another method of identifying laboratory mice. After implantation, these chips, which have approximately the size of a grain of rice, transmit a unique alphanumeric code for each mouse which can be easily identified with the help of a computer.

Although all the mentioned methods serve the purpose, they are to some degree invasive and also restricted in code space. A good identification method for laboratory animals should therefore consist of the following properties [2]:

- Permitting the rapid and easily applicable individual identification of large populations.
- Capability of allowing a large range of numbers for unique identification of many animals.
- Be easy to read.
- Be humane, and in compliance with regulatory agencies for the care and use of laboratory animals.

One possibility, which meets those requirements is the use of biometric identifiers, like fingerprint, iris, retina, etc. for human identification, since animals also have individual physical characteristics. For instance, iris scans can be used for identifying horses and retinal scans can be used for identifying cattle. However, laboratory animals, which are in four out of five cases laboratory mice, impose some specific requirements on the application. The measurement must be quick and not unpleasant for the animal since the identification will be made several times a week. A retinal scan e.g. is due to the small eyes completely unfeasible to do for each rodent in the lab [3].

In this work a method is proposed which uses the visible vein structure in the ears of laboratory mice as biometric identifiers.

In a first step, images of the ears of all used mice are acquired with the help of a special device equipped with a camera. Afterwards these images are taken and analyzed for unique features using image analysis tools. In more detail, the vein structure of the ears is extracted and investigated for attributes, which allow distinguishing between the different mice. The identifiers found are stored and serve as an abstract representation of the ear. If a mouse, which has already got enrolled, needs to be identified in the future an image of the ear is taken, unique attributes are extracted and compared with those stored. If the computed attributes equal an already stored set of attributes within a small, predefined error the mouse can be identified successfully.
The project has been carried out in cooperation with two companies, namely DiLab and AstraZeneca R&D, which are both located in Lund, Sweden. The project supervisors from the University of Halmstad are Kenneth Nilsson and Thorsteinn Rögnvaldsson.

1.1 Previous work

A previous attempt, which was developed by Kenneth Nilsson and Thorsteinn Rögnvaldsson [3], used a complex representation to model the spatial position and orientation of the blood vessels in the acquired images. Each pixel is therefore represented by a complex number where the magnitude is a certainty measure for the existence of a blood vessel at a certain position and the argument represents the orientation of the vessel. After finding the rotation and translation parameters a pixel-by-pixel matching was used to verify the identity of each tested mouse.

In their experiments an equal error rate of 2.5% for images captured at the same time and an equal error rate of 6% for images captured on two different occasions (four months apart) was achieved.

1.2 Project goals

The main goal of this project is to develop an algorithm which is capable of automatically verifying the identity of laboratory mice. Therefore, in a first step, a fully-segmented image of the blood-vessel structure needs to be created. This is followed by the enrollment process which must be able to find a sufficient amount of unique feature points. In a last step a matching algorithm has to be implemented in which the identity of each enrolled mouse can be verified successfully using the segmented image and all found feature points.

The algorithm should furthermore be executed within a reasonable time frame and increase the recognition rate achieved in previous trials (see Chapter 1.1).

Below a short summary of the project goals is included.

- **Image Segmentation**: Fast and exact extraction of the visible vein structure
- **Feature Extraction**: Finding of a sufficient amount of unique points within the vein structure
- **Identification process**: Identification of a mouse
2. Background study

Since this project involves pioneer work in this field not much information could be found which explicitly addresses the problem of extracting unique features in mouse ear images. Therefore most of the presented literature deals with segmenting and enrolling retinal images. Even though those papers and articles do not fit entirely, enough similarities could be found to use the results and conclusions in this project.

It is important to note that this section only reflects the most important findings and is not exhaustive.

In accordance to the structure of the paper, this section is divided into the parts: image enhancement, image segmentation and image post-processing.

2.1 Image enhancement

Image enhancement can be seen as a pre-processing step which attempts to suppress the background noise and highlights the vessel structure. Depending on which segmentation method is used (see Chapter 2.2), it is not always mandatory and sometimes not even useful to enhance the image.

In [4] the authors attempt to estimate the background image and subtract it from the original image. In this approach neighbourhood intensity information is used as follows: if the grey level of a pixel exceeds the mean value of its neighbours the value of the pixel is changed to the minimum of its neighbours. This leads to the above mentioned estimation of the background image in which all visible vessels are suppressed.

In other approaches only the background noise is suppressed without highlighting the vessel structure. In [5], [6] the original image gets smoothened using mean and median filters, respectively.

The most popular approach found in the literature is the use of matched filters ([5], [7], [8], [9]). A matched filter is a type of filter matched to a known or assumed characteristic of a certain target signal, designed to optimize the detection of that signal in the presence of noise [10].

Vessels in human retinal images as well as in mouse-ear images have some distinct properties which have to be considered when designing matched filters [5]. Those are:

1. Blood vessels in the obtained images can be approximated as piecewise linear anti-parallel segments (anti-parallel since the gradient direction of the two edges composing the vessel are 180° apart).

2. Blood vessels appear darker relative to the background although the intensity profile varies by a small amount from vessel to vessel and gets less and less as the vessel length increases.

3. Blood vessels never have ideal step edges (concerning the transition from background to vessel). It edges may be approximated by a Gaussian curve.
As mentioned in point three the cross section of vessels can be approximated by a Gaussian curve. The 1D Gauss function is mathematically defined as

\[ f(x) = e^{\frac{-x^2}{2\sigma^2}}. \quad (1) \]

In Figure 1 a normalized (the area of the function is one) 1D Gauss function with standard deviation \( \sigma \) of one can be seen.

\[ \begin{array}{c}
\text{Figure 1: Normalized Gauss function with standard deviation of one}
\end{array} \]

Since images are two dimensional the function needs to be modified to successfully model the cross-section as well as the longitudinal section of a blood-vessel as follows:

\[ f(x, y) = e^{\frac{-x^2}{2\sigma_x^2}} \cdot e^{\frac{-y^2}{2\sigma_y^2}}. \quad (2) \]

The normalized 2D Gauss function with a four times larger width than height, indicated by \( x \) and \( y \), can be seen in Figure 2.
If the mouse image is convoluted with a 2D gauss function, which acts in this case as a matched filter, as can be seen in Figure 2, the signal-to-noise ratio is boosted in those parts of the image which has a structure matching the filter. This can be proven with the help of Schwartz’s inequality which is given by

\[ -1 \leq \frac{\langle f, g \rangle}{\| f \| * \| g \|} \leq 1 \]  

where \( f \) in this respect indicates the matched filter, which can be interpreted as an image (see Figure 2), while \( g \) stands for a sub-part of the original mouse-ear image with the same size then the used filter.

If the result of the inequality is one, the two images are equal. If the result is zero no similarities between the two images could be found. The images are complementary to each other if the inequality gives -1 as a result.

If the 2D Gauss function is furthermore rotated, blood vessels with different directions can be enhanced as well.

In order to enhance vessels with different widths, without designing new matched filters, this concept can be used on different scales [11]. Here the original image is taken and down-sampled several times. This has the effect that, granted that proper matched filters are used, thin vessels in the high resolution images get enhanced whereas thick vessels get highlighted in the low resolution images. Furthermore this concept helps to save computational time since the down-sampled images are reduced in size and therefore require fewer convolution operations to achieve the final enhanced image. To obtain the final image (after the enhancement procedure) the down-sampled images get up-sampled to the original size again and the maximum pixel value from all enhanced images is taken.
In [12] the authors suggest the usage of a 2D matched filter which consists of the direction perpendicular to the blood vessel of a combination of two shifted Gaussian derivative filters whereas in the direction of the blood vessel a plain Gaussian is implemented (see Figure 6). Due to the properties of the filter, it acts like a line detector where the centre of the vessel becomes highlighted whereas the border of the vessel becomes attenuated, leading to a better separation of vessel and background.

2.2. Image segmentation

In order to obtain binary images in which only the vessel structure can be seen, the images need to be segmented into vessel pixels and non-vessel pixels. This procedure is highly critical because only a good segmentation will lead to a sufficient amount of correct feature points and therefore a high possibility to achieve good identification results. In the literature a large number of different methods can be found, ranging from using a plain, static threshold to very complex tracking methods.

2.2.1. Threshold techniques

Thresholding, in the simplest case, just applies a static threshold to an image. All pixel values above this threshold are assigned as vessel pixels (a fixed intensity value is assigned) or non-vessel pixels (a different, fixed intensity value is assigned), if the threshold value is not exceeded. Although this procedure is very easy and fast, a good segmentation can only be achieved if no background pixel has the same intensity then the blood vessel pixel with the lowest intensity, which is hardly ever the case. Additionally, this technique is very sensitive to illumination changes which would lead to different intensity values in the image and a wrong segmentation.

A more advanced approach uses a statistical threshold instead of a static one. Here, a histogram of all grey values within the image is made. The threshold is then tuned to a grey-scale value, which labels a certain amount of pixels in the histogram (e.g.: ten percent) as vessel pixels and the rest as non-vessel pixels.

Even though this technique is not sensitive to illumination changes, it is still possible to obtain a high amount of false detections and false rejections where background pixels, which exceed the threshold, get assigned as vessel-pixels and actual vessel-pixels, which stay below the threshold, get labelled as non-vessel pixels. Therefore this technique is only useable for images with a rather low noise-level.

Apart from global thresholding, as described above, local thresholds can be applied to an image. Here, a histogram of all gray-level pixel values within a certain area is made and investigated for its maxima and minima. In the presence of a blood vessel the histogram contains typically two maxima and one minimum. The minimum indicates the cutting point between background and vessel and is chosen as the threshold. Should no blood vessel appear in the area, generally the histogram contains no minimum (only one maximum) and all pixels are labelled as background pixels.

Note that, in order to obtain a correct result, the amount of pixels which are labelled as vessel pixels has to be in accordance with the vessel-portion in the image.
Threshold techniques usually require an enhanced image as well as a final post-processing step.

The above mentioned information is obtained from [13], [14], [15].

2.2.2 Tracking

Tracking is another, more powerful approach to obtain segmented, binary images. Here, local operators are applied on a focus known to be a vessel which is tracked from this point on [16].

The ideal vessel tracking algorithm should have the following characteristics [17]:

- *Automatic initialization:* The user effort for initializing the algorithm and the tracking process should be minimal.
- *Automatic termination* where vessels end or when all vessels have been tracked

The most common tracking algorithm used for blood vessel segmentation on retinal images uses gradients.

The gradient of an arbitrary 2D image at an arbitrary point $u(x,y)$ is given by

$$\nabla u = \frac{\partial u}{\partial x} \hat{e}_x + \frac{\partial u}{\partial y} \hat{e}_y$$

(4)  

whereas $\hat{e}_x$ and $\hat{e}_y$ refer to the Cartesian coordinates of the gradient vector $\nabla u$ which points in the direction of the maximum change in gray scale.

Figure 3a and Figure 3b show a piece of a mouse ear image with a clearly visible blood-vessel and the gradient magnitude image for the same area, respectively.

![Figure 3a: Original mouse ear image](image1)

![Figure 3b: Gradient magnitude image of Figure 3a](image2)

It can be seen in Figure 3a that the transition from background to vessel is characterized by a rapid change in the grey value. Since the magnitude of the gradient grows considerably...
Background study

whenever a vessel border is reached, the gradient image can be used as an edge detector (see Figure 3b and therefore also serves as a vessel detector.

Enrico Grisan et al. [18] propose an algorithm which extracts starting points which neighbourhood pixels consist of two vectors of opposite direction (light to dark and dark to light transitions in the gray-scale image) close to each other in the gradient signal. The vessel points are tracked by moving in the most likely direction of the vessel and analyzing the grey-level value of the new point in comparison to the previous one. Furthermore, the proposed method is capable of connecting vessel segments together by exploiting the distance between two end-points, their respective calibre, their direction and the difference in their mean grey level values. Crossings are found by identifying all pairs of vessel points that are on different sides of another vessel. Furthermore, bifurcations are identified by analyzing the end-points remaining after the connection step. Those close to a vessel, with a direction incident to the vessel in a reasonable neighbourhood are considered branching.

Compared to human expert labelling, this algorithm reaches a vessel recognition rate of 93% on average.

In a different approach [17], a tracking method is proposed which uses fuzzy logic to detect all valid vessels in a fundus image. The starting points are obtained by locating the optic nerve and taking the brightest points in this region. This is followed by the fuzzy vessel tracking which is based on the optimal clustering of a vessel profile in two clusters: vessel and non-vessel. Therefore, three adaptive thresholds are defined, which specify whether a point belongs to the vessel or not. If the vessel contrast is lower than the minimum contrast resolved by the human eye, this point is considered a termination point and tracking for this vessel segment stops.

2.2.3 Other segmentation techniques

Another approach to extract the vein structure is the use of artificial intelligence-based approaches (AIBA). Here, an a priori knowledge of the anatomical structure is needed to guide the segmentation process. These systems employ some low-level image processing algorithms, such as tresholding, thinning, and linking while guiding the segmentation process using high-level knowledge. AIBA perform well in terms of accuracy, but the computational complexity is much larger than in other segmentation techniques [16].

Neural networks, a further segmentation method, are networks which consist of numerous nodes [16]. Each node input is assigned a weight and the output is a function of weighted sums of the inputs. In order to find the right weights for each input the network has to be trained to deliver a correct recognition. If a good training set is selected, including all possible features, the network can learn the classification boundaries in its feature space.

The main advantage of this system, apart from its capability to learn, is its ability to use nonlinear classification boundaries obtained during the training of the network. One of the disadvantages of neural networks is that a new training of the system is needed every time a new feature is entering the system.

2.3 Image post-processing
After the segmentation procedure is finished, it is in some cases, depending on the chosen method, necessary to post-process the image. Post-processing normally tries to remove false vessel detections from the binary image or to connect lose vessel branches together, etc.

The most common post-processing tool is the use of mathematical morphology schemes. Morphology in general relates to the study of object forms or shapes [16]. Morphological operators apply structuring elements to binary images but can also be extended to gray-level images.

Below the most common morphologic operations are listed:

- **Dilation**: Expands objects by a structuring element, filling holes and connecting disjoint regions
- **Erosion**: Shrinks objects by a structuring element, eliminating small objects
- **Closing**: Dilation followed by erosion
- **Opening**: Erosion followed by dilation

Note, that neither after the closing nor after applying the opening operation, respectively, the size of the object is changed.

### 2.4 Enrollment

Enrollment is referred to as the process of registering feature data extracted from an image in a database [22]. In this thesis the features that are stored in the database are extracted from segmented and skeletonized images.

#### 2.4.1 Feature extraction

In biometric applications the characteristics of feature data depend on which type of object is examined. As an example, iris texture analysis as proposed in [23] yield feature vectors containing data regarding brightness values. Thereby, the iris is subdivided into a number of blocks each containing a certain amount of pixels. The mean and standard deviation values of each block are stored in the feature vector.

Another example of feature extraction in biometric applications is pointed out in [24]. The images in use are retinal images. Among others, the extracted features involve landmarks. Landmarks consist of a center location that is met by three blood vessel branches as wells at the orientations and the widths of the corresponding blood vessel branches. Landmarks are features that are especially important in applications that analyze blood vessel structures [25]. In [26] and [25] landmarks are found by the tracing procedures. In contrast, [27] suggests the use of mathematical morphology for shape extraction. The differences between the images used in [27] and in [26] and [25] is that latterly mentioned involve non-binary grey scale value images as opposed to the image used in [27]. The use of mathematical morphology on shapes with ambiguous boundaries can yield undesired results in applications that require accuracy [27]. However, skeletonized binary images are suitable for feature extraction using mathematical morphology [28].
Additionally, distances between landmarks are useful features as they allow computation of rotation and translation parameters needed for the alignment operations [24].

2.5 Matching

Image matching is generally referred to as a technique that examines the similarity of two images [29]. As extracted features are commonly used to represent images, matching can as well involve extracted features. In order to compute similarity measures between extracted features following formulas are used [23].

\[
d_1(f, f_i) = \sum_j |f^j - f_i^j| \quad (5)
\]

\[
d_2(f, f_i) = \sum_j (f^j - f_i^j)^2 \quad (6)
\]

Where \(f\) and \(f_i\) represent features of an unidentified image and the \(i\)th database image respectively, \(f^j\) and \(f_i^j\) resemble the \(j\)th component of the feature array of the unidentified image and of that of the \(i\)th database image, respectively. A more generic approach to computing similarity measures is by considering relative frequencies of occurrence of certain attributes. The relative frequency of occurrence is given by [28]:

\[
foc_j := \frac{h_j}{n} \quad (7)
\]

where \(h_j\) denotes the number of attributes that fulfill given criteria among the total sample, \(n\). In [26] seed points are generated by crossing grid lines with blood vessels. One example of computing a similarity measure by using the frequency of occurrence would be to use those seed points.

Whereas formula (7) describes a linear measure, further emphasizing on a certain score value can be performed by using Gaussian weighting [30]. Thereby, the most desirable value is the center value, also known as the expectancy value \(\mu\). As opposed to linear functions, all values that are in proximity to the expectancy value yield high scores whereas values which are in proximity to the standard deviation value face high decays. Therefore, in some cases Gaussian weighting is a method that yields a more representative similarity measure than the frequency of occurrence does.
3. Algorithm

As already mentioned in the introductory part this work proposes a method which is able to identify laboratory mice in pharmaceutical companies. Therefore images of the ears are taken and analysed for unique features in the blood vessel structure which can afterwards be used for the matching procedure.

The two participating companies in this project DiLab and AstraZeneca provided therefore 120 RGB images of 20 different mice (6 images per mouse) at a resolution of 3072 x 2048 pixels on which the developed algorithm should be tested. The images were made on two different occasions (image 1 to 5 at the first and image 6 at the second occasion), taken four months apart. This was done since it needed to be proven that the vessel structure inside the ear does not change with the age of the animal.

In a first step the original images were cropped manually to a size of 600 x 600 pixels to include most of the blood vessel pattern in the ear.

Figure 4 shows the manually resized version for the first image of mouse number 1.

![Manually cropped RGB mouse ear image at a size of 600 x 600 pixels](image)

In order to achieve a higher contrast between background and blood vessels, gray scale images were computed as the difference between the G and the R images. Additionally, the images were down-sampled to 300 x 300 pixels before applying the proposed algorithm to them.

In Figure 5 the gray-scale image of mouse number 1 (see Figure 4) at a size of 300 x 300 pixels can be seen.
The black lines represent the blood vessels with the exception of the black elliptic line in the top left corner which indicates the boarder of the ear.

The main algorithm was implemented in MATLAB version 7.1.0.246 (R14) Service Pack 3 which provides a very good framework and furthermore a comprehensive image analysis toolbox and tested on all obtained images.

In the next sub-chapters the proposed algorithm will be explained in more detail. Starting with the techniques used to enhance the obtained images followed by the segmentation process which leads to a binary image, the enrollment process and finally the matching algorithm.

3.1 Image Enhancement

As can be seen in Figure 4 and Figure 5, respectively, the original images obtained can contain a considerable amount of noise. A list with possible noise which can occur in the images can be seen below.

- Since the mouse-ear skin is pigmented the background is not uniform in gray-scale and appears therefore very noisy
- Small hairs can appear in the images which have, apart from its shorter length, the same properties than actual blood vessels.
- The boarder of the mouse ear can appear in the images which can lead to wrong blood vessel detections.
- During the image acquisition process the ear can become folded which leads to additional noise in the image; this can be confused with actual blood vessels. This can furthermore go hand-in-hand with a change of the distance between blood vessels which may, in the worst case, make it impossible to identify the mouse.

Therefore, the decision was made to include an enhancement step in order to improve the original image for the segmentation process. Since the use of matched filters seemed to
deliver the most promising results (see Chapter 2), it was also chosen as the main enhancement technique in this project.

As already mentioned in the introductory part of this paper the transition of background to vessel and vice versa is not characterized by a perfect step-edge but can be approximated by a Gaussian curve. Therefore also 2D Gaussian functions (see Figure 2) were used in this project to model the vessel structure. Since it is very crucial to achieve a good separation between the background and the actual vessel a combination of Gaussian derivative filters and plain Gaussian filters was used to enhance the vessel structure. In more detail a 2D matched filter was developed which consisted in the direction perpendicular to the blood vessel (x-direction) of a combination of two shifted Gaussian derivative filters whereas in the direction of the blood vessel (y-direction) a plain Gaussian was implemented. A visualisation of the basic shape of the used filter can be seen in Figure 6 [12].

Blood vessels are furthermore characterized as piece-wise linear segments. As a result the standard deviation along the vessel (in the y-direction) was chosen much higher than the standard deviation across the vessel (in the x-direction) which led to a bar-like shape of the filter. Additionally, vessels can appear in all possible directions. Therefore it is not enough to convolute the mouse-ear image with a matched filter in a single direction, but also to rotate the filter kernel. Since the computational complexity should be kept as low as possible a trade-off between accuracy and speed had to be made. A final important feature of blood vessels is that the veins thin out as the length of the vessel increases. Consequently it is not sufficient to create only one filter which models the vein structure but to create more matched filters which are able to enhance vessels with different widths.

After a long testing phase the decision was made to use the following two different matched filters:
Here, the first filter is in charge of enhancing the main vessels, whereas the second filter is responsible for highlighting medium width and small width vessels. Furthermore, both matched filters got rotated 16 times, which led to an angular resolution of 11.25°, by multiplying the filter kernel with the following rotation matrix $r$:

$$
\begin{bmatrix}
\cos(\varphi) & -\sin(\varphi) \\
\sin(\varphi) & \cos(\varphi)
\end{bmatrix}
$$

(8)

$\varphi$ in this respect represents the rotational angle, which is always a multiple of 11.25° and ranges from 0° to 168.75°.

Firstly, in order to enhance very thin vessels, a 3rd filter was implemented with kernel size 25, standard deviation of 1 in the x-direction and 5 in the y-direction. This unfortunately resulted in the noise, especially thin hairs, becoming enhanced as well which made it impossible to distinguish between the actual blood vessel system and the noise.

Figure 7a shows a visualization of the 1st matched filter in all 16 directions whereas in Figure 7b the resulting, enhanced images after convoluting the image in Figure 5 with these matched filters can be seen.

---

**Figure 7a:** Gray-scale visualization of the 1st matched filter in all 16 directions
Note that the white portions in Figure 7a indicate positive values whereas all black areas imply negative values. Furthermore note that since a rotation angle of 0° appears to the naked eye as a rotation of the matched filter of 90° (see top-left image), a rotation angle of 0° is referred from now on to an angle of 90° and an angle of 168.75° to 258.75°, respectively, on the following pages.

Figure 7b: Resulting enhanced images after convoluting the mouse ear-image in Figure 5 with each of the rotated matched filters seen in Figure 7a

It can be seen in Figure 7b that in each sub-image only those vessel fragments get highlighted which correspond to the direction of the filter. Therefore no vessel branches are enhanced in the top-left image whereas in the right image, 3rd row the main vessel structure can be seen clearly. It is important to note that the convolution operation in this case does not return the full two-dimensional result but gives back the central part of the convolution with the same size than the original image.

In order to obtain a single enhanced image the minimum (since blood vessels have a lower gray-value than the background) of all 16 convoluted images is taken in each pixel. The final result for both filter settings can be seen in Figure 8a and 8b, respectively.
As indicated in Figure 8a, only the main vessels become highlighted by employing a big filter whereas possible noise, as well as small vessels, gets attenuated completely. The use of a smaller filter (Figure 8b) leads to a clearer enhancement of the whole vessel system (thick vessels as well as thin vessels) but also introduces more noise to the enhanced image. Nevertheless, since the proposed segmentation algorithm operates on two different scales, as described in the next sub-chapter, both images are needed to obtain a good segmentation result.

Note, that in both enhanced images line-like segments appear which do not belong to the actual blood vessel structure and which furthermore are not visible in the original image. This is due to the fact that the mouse ear is, as already mentioned, pigmented. Since the matched filters created are designed to enhance objects with the same shape than the actual filter form, areas in which the pigmentation appears in bar or line-like shapes get enhanced as well. This phenomenon represents one of the drawbacks of this approach.

The complete enhancement algorithm can be found in Appendix A.

3.2 Image Segmentation

The next step, after enhancing the original images, consisted of creating fully segmented images of the blood vessel structures.

As described in Chapter 2.1 numerous techniques exist in literature to extract blood vessels. But since tracking was the most popular approach found and additionally delivered the most promising result, the decision was made to implement a tracking algorithm in this project as well.

Note that in order to extract the main vessels first the algorithm starts by taking the enhanced image obtained with the first filter settings (see Figure 8a) before applying the tracking algorithm to the second enhanced image.
3.2.1 Starting points

The main difficulty for each tracking algorithm is commonly to find proper starting points from which the various blood vessels can be tracked. In other words a sufficient amount of starting points has to be found which permit to track all vessel branches within the enhanced image, without introducing wrong vessel fragments to the final segmented image.

In the proposed approach the image gets therefore firstly divided into 144 sub-images with size 25 x 25 pixels. After that the darkest point of each sub-image is taken and stored in a vector, consisting of the position and the gray-scale value. Since it is not possible to act on the assumption that each sub-image actually contains blood-vessel pixels a further step is needed to eliminate wrong starting points. Therefore a threshold is applied to all 144 initial starting points which, after a long testing phase was set to 5 percent for extracting the main vessels (1st filter) and to 0.5 percent for extracting all remaining vessels (2nd filter). In more detail, firstly, a one dimensional vector with size 1 x 90000 is created in which the gray-value of each pixel is stored. In a next step the vector gets sorted starting with the pixel with the lowest gray-value. Now the proposed threshold-percentage (e.g. lower 5 percent) of 90000 is taken and the gray-value at the particular pixel number gets stored. In a last step each starting point which exceeds the stored gray-value gets removed from the starting point vector. Note, that this procedure was also used for all other thresholds in this project.

Finally, the local minimum of all starting point values within neighbourhoods of 10 pixels is taken since it can be assumed that all of them are placed on the same blood vessel.

Figure 9a to 9c show the above mentioned procedure to obtain valid starting points for the actual tracking procedure. For reasons of consistency the same mouse than previously (mouse number 1, image number 1) is taken.

![Figure 9a: Initial 144 starting points by taking the darkest points of each sub-image](image1)

![Figure 9b: Remaining starting points after removing all points from the vector which do not exceed a certain gray-value](image2)
Figure 9c: Final starting points after taking the local minimum in a neighbourhood of 10 pixels

Figure 9c indicates that with the proposed algorithm all starting points, except one, lie well-distributed on the main vessels and deliver therefore a good basis for the following tracking procedure. The vector containing all valid starting positions is finally sorted in order to start the tracking process with the most significant point (with the lowest gray-value).

### 3.2.2 Tracking

The actual tracking algorithm starts off by taking the first starting point, which was obtained using the procedure described in Chapter 3.2.1 applied to the 1st filter image, out of the vector. Additionally the point gets marked as a vessel pixel and the most likely angle for the direction of the blood vessel at this point gets calculated. The directional information is obtained from a feature of the enhancement process. Since the convolution of the original image with the matched filter only boosts a vessel segment if the direction of the filter matches the direction of the vessel in the area covered by the filter, information on the vessel propagation can be taken from there.

In the proposed enhancement algorithm the minimum operation, applied to the 16 filtered images, does not only return the final enhanced image but also a two dimensional matrix with size 300 x 300 pixels in which the number of the taken filter for each pixel is stored. Therefore, in a next step, the filter number only at the position of the starting point has to be mapped to a quantized angle (quantized in the respect that only 16 different values are possible) in order to obtain a correct vessel direction.

In a next step initial neighbouring pixel of the starting point are selected, according to the obtained, quantized vessel direction. Note that the obtained angle can only vary between 90° and 270°. An angle of 90° indicates that the vessel propagates in the direction of the y-axis whereas an angle of 180° implies that the vessel is located parallel to the x-axis (see Figure 7a). Since the proposed tracking algorithm operates pixel-wise, in other words only in a neighbourhood of 1 pixel, the tracking is restricted to four different directions in the upper tracking path and, again, to four directions in the lower tracking path. Therefore a mapping has to be made in which the quantized 16 angles are clustered into four angular groups where each group represents one tracking direction.

Figure 10 shows the four different angular groups, the tracking direction in respect to the angular group and the according, initial neighbour pixels.
Note that the tracking algorithm firstly starts in the upper direction before moving in the opposite direction.

Now, out of the three neighbourhood pixels the point with the lowest gray-scale value is set as the new reference pixel and furthermore labelled as a vessel pixel. In a next step the direction of the vessel at the new point is looked up in the previously described filter matrix and compared to the direction at the previous reference point. If the angle did not change significantly, in other words did not leave its angular group (e.g.: between 157.5° and 202.5°), the algorithm uses the same tracking direction then before. Should the direction have changed in the way that the angle is located in a new angular group, different neighbourhood pixels than before need to be selected. Since the approach shown above only covers vessel propagation in the upper direction, the method has to be modified in the following way for all proceeding reference pixels:

The red dots make it now possible to track vessels also in the lower direction. Should the angle group change, the neighbourhood matrix is taken which is closest to the previous
neighbourhood matrix. An example for the change in the direction of the vessel and the effect on the tracking direction and the respective neighbour pixels can be seen in Figure 12.

![Diagram of blood-vessel tracking](image)

1) The initial angle is located between 157.5° and 202.5° and the red neighbour pixels were chosen. Since the neighbourhood pixel, located top-right to the reference pixel has the lowest gray-value it is chosen as the new reference pixel and gets furthermore labelled as a blood vessel pixel.

2) The angle at the new reference point changed and lies now between 112.5° and 157.5°. Out of the two possible neighbourhood matrices the one was chosen which was closest to the old matrix. Finally, out of the three candidates the neighbour pixel above the old reference point was chosen as the new reference pixel and is again, labelled as a blood vessel pixel.

3) Similar to 2)

**Figure 12: Example for the tracking of a blood-vessel with change in the tracking directions**

This procedure (finding new neighbourhood pixels, choosing a new reference point, labelling the found point as a vessel pixel) is now repeated until one of the following exit conditions applies:

- The new reference point exceeds a certain gray-scale value
- The angular difference between the previous and the current reference point exceeds a certain value
- The reference pixel got already labelled as a blood-vessel pixel
- The border of the ear is reached

For the tracking of both main vessels and small and medium vessels, the gray value threshold is set to 20 percent and the maximum angular difference to 45 degrees.

Since the tracking for a starting point initially moves in the upper direction, the tracking also has to resume in the lower direction. Therefore, as soon as the tracking in the upper direction is finished, the starting point is set as the reference point again and the original tracking
direction gets inverted. This leads to the fact that the tracking moves in the opposite direction until an exit condition applies.

In order to make the tracking procedure more resistant against possible noise which might still be present in the enhanced image the following additional features, regarding direction and gray-value, were included in the algorithm.

- The angle at each reference point is set to the mean of the four, previous angles and the actual angle at the new position. This measure aims on smoothing the tracking direction since vessels are usually not suspect of fast changes in the angular direction.
- A “tolerance bit” is included in the algorithm which is set to one whenever a new reference point exceeds the specified gray value threshold. The bit is set back to zero if the proceeding reference point undercuts the threshold again. If not, the tracking stops and continues either in the other direction or with a new starting point.

Figure 13a shows all pixels labelled as blood vessel pixels drawn in the enhanced image whereas Figure 13b shows the segmented, binary image after applying the algorithm to the first, enhanced image (see Figure 8a).

Figure 13a and 13b show clearly that all vessel segments, on which starting points were located, were tracked. Furthermore it can be seen that the tracked portion, indicated by white points in both figures, is always located in the centre of the vessel.

Note that in Figure 13b the tracked part of the mouse ear border got already removed due to a first post-processing step (see Chapter 3.3).

As a next step the enhanced image obtained by using the 2nd matched filter (see Figure 8b) needed to be added to the segmentation process to extract all remaining vessels.

In order not to segment the main vessels again, which are contained in the second enhanced image as well, the already segmented vessel portions had to be excluded for the second part of the tracking operation.

For this purpose the tracked image got firstly post-processed according to the algorithm described in Chapter 3.3. After that the tracked region in the segmented, post-processed image got morphologically dilated with a quadratic structure element with size 13 x 13 pixels,
leading to an expansion of the tracked area. Figure 14 shows the segmented, post-processed image shown in Figure 13b after applying the morphologic operation.

![Figure 14: Tracked image after dilation with a quadratic structure element of size 13 x 13 pixels](image)

As a final step the pixels in the 2nd filtered image, which position matches with the white portions in the expanded, segmented image, got set to the highest originally appearing gray value. This had the effect that after applying the tracking algorithm to the modified, enhanced image, not a single starting point is positioned on an already tracked vessel and therefore only new vessel portions get tracked.

Figure 15 shows the second enhanced image after applying the above mentioned operations to it.

![Figure 15: Figure 8b after “blackening” all pixels which belong to an already tracked blood vessel](image)
Now the tracking algorithm was applied to the modified image and all newly found vessel pixels got added to the segmented image, obtained in the first tracking phase. Note that, as already mentioned before, this time only points with a gray-value below a threshold of 0.5%, applied to the modified 2nd filtered image (see Figure 12), were used as starting points.

The newly found vessel portions drawn in the second, enhanced image as well as the final, tracked image can be seen in Figure 16a and Figure 16b, respectively.

It can be seen in Figure 16b that after applying the tracking algorithm to the 2nd filtered image a segmented image could be achieved in which the most parts of the vessel structure was extracted successfully.

Unfortunately also some incorrect segmentation results are visible in the image. This is due to the fact that parts of the enhanced image, which clearly do not belong to the blood vessel structure, have the same gray-value than actual vessel pixels and got therefore picked as starting points.

The next chapter therefore introduces a post-processing step which is in charge of cleaning the segmented image from all occurring noise.

The whole tracking algorithm can be found in Appendix A.

3.3 Image Post-Processing

Since the segmented image still contains a certain amount of noise, or in other words invalid vessel portions, a post-processing algorithm got applied to the binary images in order to obtain an image which will be used in the enrollment and matching process at a later stage.

Note that the algorithm described below gets applied, firstly, on the binary image obtained after extracting the main vessels and secondly on the final binary image obtained after extracting also the remaining vessels.

In a first step, as already mentioned in Chapter 3.2, a first post-processing method was included in the tracking algorithm deleting tracked vessel portions from the segmented image.
which do not exceed a length of 40 pixels for the 1st filtered image and 20 pixels for the 2nd filtered image. This was done by simply counting the amount of tracked points for each starting point. Since, due to an applying exit condition, certain vessel branches might not get connected, a method had to be introduced which closes the gap between to lose vessel portions. This is especially the case when the tracking algorithm approaches, coming from a small vessel branch, the main vessel. Here, the angle can change for more than 45 degrees which would lead to a stoppage of the tracking for the certain vessel branch. Therefore a MATLAB® function gets applied to the image, namely ‘imclose’. Imclose performs morphological closing (dilation followed by erosion) using a previously defined structuring element. The chosen structuring element was a quadratic object with size 5x5 pixels, which implies that all lose vessel portions within a neighbourhood of 5 pixels get connected to each other. Now the image is skeletonised in the sense that all pixels on the boundaries of tracked vessels got removed leaving one pixel wide vessels. This post-processing step was included since it is needed for the extraction of all branching points contained in the image (see also Chapter 4). Since all vessel portions are now connected and a skeletonised image was created all remaining connected pixels with an overall size less than 130 pixels got removed. In a last step all vessel branches were removed which do not exceed a certain length. This operation got included since short hairs might intersect with an actual vessel creating invalid branching points which would lead to wrong enrollment results and an incorrect identification (see Chapter 3.4 and 3.5). Therefore, all branching points were firstly extracted and the length of each vessel branch proceeding from the point got investigated. Should the length of a vessel branch lie below a total length of 15 pixels, the branch get removed from the tracked image.

Figure 17 shows the segmented image in Figure 16b after applying the post-processing algorithm to it.

![Segmented image](image-url)

**Figure 17: Final segmented image after applying the post-processing algorithm**

Note that not all mentioned post-processing techniques affected the resulting image since the image obtained from the tracking process did not contain all described kind of noise. Still all methods are necessary to acquire an image in which most wrong segmented portions are removed.
Algorithm

The complete post-processing algorithm can be found in Appendix A.

3.4 Enrollment process

In this application the process of enrollment is about extracting features from segmented images and transforming them into representative information which is then stored in a database. The information stored in the database is used for identification purposes. The identification process compares an arbitrary image to all representations stored in the database. The goal of that process is to find out which mouse that is represented by feature data stored in the database the arbitrary image belongs to. Thus, best possible representations for storage in the database are desired in order to enhance the likelihood of recognition for each arbitrary image that is run through the enrollment algorithm.

3.4.1 Description of the sample

The overall sample used in this project is described in Chapter 3. In order to utilize the fact that multiple images of the same ear are available each instance is used, whereas five are used for enrollment and one is used for identification. For enrollment, images one through four obtained by the first measurement as well as the image obtained from the second measurement are used. The fifth image belonging to the first measurement is reserved for identification. The setup showed in Figure 18 is an extract of the MATLAB workspace after loading all images belonging to mouse #19.

![Figure 18: MATLAB workspace variables belonging to loaded images](image)

3.4.2 Input parameters

Due to the fact that the segmentation of each image used in this project involves a trade-off between including noise and neglecting thin blood vessels, several segmented images contain noise. While the blood vessel structures displayed in each segmented image belonging to the same mouse is similar in each image, the noise found in each segmented image differs from the noise found in others. Consequently, the comparison of two or more segmented images showing the same ear and the enrollment of features that can only be found in at least two of the five images involved is a simple measure that can be taken in order to avoid the enrollment of noise. Additionally, this approach maximizes the amount of bifurcation points that can be used for enrollment. Therefore, this enrollment algorithm compares five segmented images of each mouse and only enrolls the features that are found in at least two of the five segmented images.
The input parameters required by the enrollment algorithm are as follows:

- five segmented images of each mouse
- the orientation information matrix introduced in Chapter 3.2.2

### 3.4.3 Feature extraction

#### 3.4.3.1 Extraction of bifurcation points using morphological operations

Morphological operations are common image processing tools for extraction or deletion of defined shapes from images [19]. The most popular morphological operations are listed and briefly described in Chapter 2.3. In order to find the position of all bifurcation points in the segmented images used in this application, morphological opening is used. The reason for using opening rather than erosion is due to the fact that in contrast to erosion, opening retains the shape of each bifurcation point. This is required for the implementation of angle extraction techniques used for branch orientation angle extraction which are described in subsequent chapters. In order to find all types of bifurcation points, eight structuring elements are created and used for morphological opening. Thereby, the structuring elements resemble the shape of the bifurcation points. The vessel pixels are marked by ones and the non-vessel pixels by zeros. As a result of the fact that morphological opening is an operation that involves erosion for determining the position of each bifurcation point and dilation for restoring the shape, the segmented image in use must be skeletonised. Otherwise, the erosion operation would yield multiple results for each bifurcation point. The structuring elements in use can be seen in Figure 19.

![Figure 19: Morphological structuring elements used to find bifurcation points](image)

#### 3.4.3.2 Extraction of branch orientation angles

As described in Chapter 3.2.2, pixel wise orientation information for each segmented image is provided by the orientation information matrix, created during the enhancement procedure. The orientation information is computed for 30 pixels per branch in each direction. The contribution angles for each branch are summed up and averaged over the corresponding distance. The illustrations shown in Figure 20a and 20b visualize the result of summing up and averaging the orientation information of each pixel (Figure 20a) as well as how the branch orientation angles are to be interpreted (Figure 20b). Finally, Figure 20c is a legend referring to Figure 20a and 20b.
As can be seen in Figures 20a through 20c, branch orientation information is rotation variant. Therefore, any two images which are matched and comprise a rotation when compared to each other cannot use branch orientation angles for comparison without being aligned beforehand. As a result of this finding, the branch orientation information is altered in order to receive rotation invariant branch angle information for all vessel tree branches. The method used to abolish the rotation dependence is expressed by the formulas below

\[
\alpha_{ri} = \max(\alpha_{rv}, \beta_{rv}, \gamma_{rv}) - \alpha_{rv}
\]

(9)

\[
\beta_{ri} = \max(\alpha_{rv}, \beta_{rv}, \gamma_{rv}) - \beta_{rv}
\]

(10)

\[
\gamma_{ri} = \max(\alpha_{rv}, \beta_{rv}, \gamma_{rv}) - \gamma_{rv}
\]

(11)

The angles \(\alpha_{rv}, \beta_{rv}\) and \(\gamma_{rv}\) refer to the rotation variant branch orientation angles found in Figure 20b. Thus, \(\alpha_{ri}, \beta_{ri}\) and \(\gamma_{ri}\) are their rotation invariant counter parts. The illustrations shown in Figures 21a through 21f demonstrate the effect of the operations stated above on three sets of branches.
As can be seen in Figure 21a the maximum angle is 105°. Once subtracted from the other two branch orientation angles the previously largest branch orientation angle is set to 0°. Figures 21b and 21c show the branch structure of 21a rotated by -20° and 20° respectively. The branch structure of 21d, 21e and 21f that result from subtracting the maximum branch orientation angle comprise the same branch orientation angles and are thus rotation invariant.

3.4.3.3 Extraction of inter-bifurcation point connection angles

Additional features provided by the presence of bifurcation points are angles set up by inter-bifurcation point connections and the branch structure. The illustration in Figure 22 shows the angles that are used as features in this application whereas the red line represents an inter-bifurcation point connection.

The angles $\varepsilon$ and $\delta$ shown in Figure 22 are given by following formulas

$$\varepsilon = \frac{\pi}{2} - \alpha_v + \frac{\pi}{2} + \rho = \pi - \alpha_v + \rho$$

with $\rho = \tan^{-1}\left(\frac{\Delta x}{\Delta y}\right)$

(12)

$$\delta = \pi - \left(\beta_v - \frac{\pi}{2}\right) + \psi$$

with $\psi = \tan^{-1}\left(\frac{\Delta y}{\Delta x}\right)$

(13)
Figure 23 shows all symbols that appear in the formulas stated above where $\alpha_{rv}$ and $\beta_{rv}$ are rotation variant branch orientation angles which were computed in the previous chapter.

Figure 23: Angles between branch structure and inter-bifurcation point connections

3.4.3.4 Extraction of inter-bifurcation point distances

The length of the distance between two arbitrary bifurcation points is simply computed by implementing following formula whereas $\Delta x$ and $\Delta y$ can be seen in Figure 23.

$$d = \sqrt{\Delta x^2 + \Delta y^2}$$

(14)

3.4.4 Comparison of images showing the same ears

As previously mentioned, one core element of the enrollment algorithm is the examination of all five segmented images. The goal of this procedure is to find all bifurcation points that are at least contained by two images. Thereby, each image is compared to each other. In each image following features are subject to comparison:

- the rotation variant branch orientation angles of the branch structures surrounding each bifurcation point, discussed in Chapter 3.4.3.2
- the inter-bifurcation point distances, discussed in Chapter 3.4.3.4

All comparisons are carried out pair wise which means that the algorithm compares one feature of two images at once. In order to reduce computational load, the comparison techniques mentioned above are arranged in a hierarchical order. This means that while branch orientation angles referring to each bifurcation point are compared to the branch orientation angles of each other bifurcation point in each other image, only the pairs that are similar are subject to inter-bifurcation point distance comparison.

3.4.4.1 Comparison based on rotation variant branch orientation angles

In this step, the branch orientation angles of each pair of bifurcation points are examined. Following formula describes the means of comparison used in this step.
The symbol $\Delta a$ stands for the average angle difference between the branch orientation angles of the branches belonging to one particular bifurcation point in both images respectively. Therefore, throughout this thesis it is further referred to as average angle difference. The angles $\alpha_{rv1}$, $\alpha_{rv2}$, $\beta_{rv1}$, $\beta_{rv2}$, $\gamma_{rv1}$ and $\gamma_{rv2}$ are the rotation variant branch orientation angles whereas the inferior numbers stand for the involved images. In order to perform meaningful comparisons based on rotation variant branch orientation angles the images are aligned beforehand. This is done by examining rotation invariant branch orientation angles.

While carrying out all comparisons a list with all pairs of bifurcation points and all average angle differences is set up. This list is further referred to as the angle environment candidate list. Additionally, an empirically chosen threshold referring to the average angle differences is applied to filter the candidate list. Figure 24 shows an excerpt of the list that had been filtered by a threshold of 20.

As the table in Figure 24 shows, multiple entries can be found for each bifurcation point. Most likely, the correct entries are the entries which have low average angle difference values. Therefore, an algorithm which finds the most suitable pairs of bifurcation points based on the average angle difference values is required. The corresponding algorithm carries out following steps:

- All entries referring to one particular bifurcation point are collected to form a sub list. The red frame in Figure 25a shows an example of a sub list.
- The lowest average angle difference value among the thresholded sub list entries is found. The lowest value is symbolized by the green arrow in Figure 25a.
- All other sub list entries are deleted from the entire angle environment candidate list.
Next, all entries referring to the point, previously found in the second image (point 3 in image 3 in this case) are collected to form a sub list. The red frames in Figure 25b show the members of the sub list.

Again, the entry containing the lowest average angle difference value is chosen and all other entries are deleted.

All steps mentioned above are carried out until none of the points are doubly assigned.

Figure 25c shows the result of the algorithm described above.

### 3.4.4.2 Comparison based inter-bifurcation point distances

The previously described comparison algorithm yields the angle environment candidate list containing five columns whereas the first and the third columns stand for the segmented images used for comparison, the second and the fourth columns indicate the bifurcation points found in the corresponding images and the fifth column stands for the average angle difference among the branch orientation angles. In contrast to the previously described comparison algorithm which requires one bifurcation point per image, this procedure demands two bifurcation points per image. This results from the fact that the nature of a distance is that it consists of a start point, an end point and a vector between both points. Thus, in order to compare two distances, four bifurcation points – two per image – are
Algorithm

required. All possible combinations of bifurcation points provided by the previously described comparison algorithm are used to set up a list which is further referred to as distance candidate list, allowing comparison of distances. Therefore, the bifurcation points that had been filtered out of the angle environment candidate list, described in Chapter 3.4.4.1, do not appear. The following illustration shows an excerpt of that list.

![Distance Candidate List](image)

**Figure 26: Excerpt of the distance candidate list**

Finally, the distance candidate list is filtered by applying an empirically chosen threshold to the column that contains the differences between the distances.

### 3.4.4.3. Preparing data for enrollment

After setting up the distance candidate list, the data is prepared for enrollment. Each of the five segmented images contains a certain number of bifurcation points that are recognized to be contained by other segmented images as well. The segmented image containing the maximum amount of bifurcation points listed in the distance candidate list is the segmented image that is enrolled. This image is further referred to as the representative image. After determining the representative image, all bifurcation points which are not included in the representative image, yet included in at least two of the other five images are transformed to the representative image. For that purpose translation and rotation parameters are found by seeking the best relation between the lengths of the inter-bifurcation point distances used for computation of the transformation parameters and their differences in length. Figure 27 shows an example of a pair of points that most definitely would not be used for computing the transformation parameters, as well as a pair that most definitely would be used.
The distance candidate list displayed in Figure 27 shows all pairs of bifurcation points image #2 and image #3 have in common. As previously mentioned, a threshold had been applied to the last column. The threshold was set to 7. Thus, each entry qualifies as input for computation of rotation- and translation parameters between both images. The most appropriate entry is marked by the green frame. It can be seen that the inter-bifurcation point distances are rather high and the difference between both distances is rather low. Thus, the resulting rotation- and translation parameters are most likely sufficiently accurate. In contrast, the most inappropriate entry is marked by the red frame. As can be seen, the distances are comparatively short and the difference between both distances is relatively high. Thus, using this entry to compute the rotation- and translation parameters would yield an inaccurate result.

3.4.4.4 Enrollment

Having performed all steps mentioned above, all data is ready for enrollment and thus for storage in the database. Following data is enrolled:

- bifurcation points (using Cartesian coordinates)
- branch orientation angles (rotation variant & rotation invariant)
- inter-bifurcation point distances
- inter-bifurcation point connection angles
- one segmented image per mouse

The image containing the maximum amount of bifurcation points destined for enrollment is the image that is stored in the database.

3.5 Matching process

As this chapter deals with matching, various segmented images are involved and thus to be defined. The act of matching deals with two images. Thereby, one image is chosen from the database, further referred to as the database image and one image is an arbitrary input image, further referred to as the unidentified image. The process of identification resembles a series of matching operations. Thereby, one unidentified image is used as well as an arbitrary number of database images. The goal of each identification process is to verify whether or not the unidentified image matches with any of the database images. In this chapter two matching algorithms are described, namely the bifurcation point analysis algorithm and the statistical examination of the segmentation results.
3.5.1 Bifurcation point analysis

Bifurcation points are features which are commonly used for applications in biometrics such as fingerprint recognition [20] or registration of retinal images [21]. Whereas fingerprints usually have large numbers of bifurcation points, blood vessel structures which can be found in mouse ears do not. The amount of bifurcation points contained by each segmented image stored in the database varies between 4 and 11. While images that contain relatively large amounts of bifurcation points can be uniquely identified, solely by considering all features provided by the bifurcation points, images that contain smaller amounts require analysis of additional image features, in order to be uniquely identified. Therefore, for images that are poorly populated with bifurcation points, the matching procedure is carried out in two steps. The first step, being the bifurcation point analysis is carried out in all cases of matching. In some cases, the match list yielded by the bifurcation point analysis yields inconclusive results. Therefore, the second step, being the statistical examination of the segmentation results is required. Figures 28a and 28b illustrate how some blood vessel structures lack of bifurcation points whereas others are well populated. On a side note, the original grey scale blood vessel images as used in Figures 28a and 28b are solely used for demonstrative reasons. None of the previously described algorithms uses the original images.

As discussed in Chapter 3.4, bifurcation points involve following features:

- branch orientation angles
- inter-bifurcation point distances
- inter-bifurcation point connection angles

All features mentioned above are used for identification in this application, whereas the branch orientation angles used for bifurcation point analysis are rotation invariant. The illustration displayed in Figure 29 shows the corresponding features, whereas the cyan lines point out the rotation invariant branch orientation angles, the green lines the inter-bifurcation point connection angles and the red line the inter-bifurcation point distances.
The comparisons are carried out in a similar way they are in Chapter 3.4. First the rotation invariant branch orientation angles are matched. Afterwards the inter-bifurcation point distances are matched and finally, the inter-bifurcation point connection angles. As described in Chapter 3.4, the order of matching is hierarchical which means that one resulting candidate list is the input to the next comparison procedure. The final result of the bifurcation point analysis algorithm is the match list which can be seen in Table 1.

<table>
<thead>
<tr>
<th>8</th>
<th>6</th>
<th>12</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>7</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>9</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>10</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>11</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>7</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table 1: Match list**

Each entry in the match list in Table 1 contains following information:

- number of the mouse the corresponding database image belongs to (1st column)
- two bifurcation points belonging to the database image (2nd and 3rd columns)
- two bifurcation points belonging to the unidentified image (4th and 5th columns)

Therefore, each entry of the match list is believed to indicate matching pairs of bifurcation points. The bifurcation point contained by the 2nd column is believed to be equal to the bifurcation point contained by the 4th column. The same applies to the 3rd and the 5th column.
However, the first column is the sole relevant data source for the final evaluation of the bifurcation point analysis. This is due to the fact that solely the frequency of occurrence of each mouse in the match list is considered. The bifurcation points are not only used for identification, they are also used to find the translation and rotation parameters needed by the algorithm that performs the statistical examination of the segmentation results. As opposed to the comparisons carried out in Chapter 3.4.4.1, only rotation invariant branch angle information is used in this step and thus, the likelihood that matches are also found in non-matching images is relatively high. One example for a false match based solely on information provided by bifurcation point analysis can be seen in Figures 30a and 30b.

![Figure 30a: One pair of bifurcation points, connected by a red line](image1)

![Figure 30b: A false match referring to Figure 30a](image2)

The left illustration represents the blood vessel structure of mouse #14 whereas the right illustration represents the structure of mouse #9. As the images show, the relation between the branch orientation angles are similar in both images as are the inter-bifurcation point distances as are the inter-bifurcation point connection angles. This example shows that matching operations cannot always yield unique results as long as features solely based on bifurcation points are considered. Thus, statistical measures regarding the segmentation results are used in order to perform further examination of each potential match. All mice that are to undergo further examination are found in the match list yielded by the bifurcation point analysis.

If the presence of one mouse makes up the absolute majority among all other entries and if the amount of entries is above eight, the result is seen as conclusive. Otherwise a further examination as described in the following chapters is carried out.
### 3.5.2 Statistical examination of segmentation results

#### 3.5.2.1 Determination of rotation and translation parameters

Each entry of the match list which is yielded by the bifurcation point analysis contains information based on the comparison between two pairs of bifurcation points. Thus, each entry provides sufficient information in order to determine the rotation and translation parameters between the unidentified image and the database images. If statistical examination of the segmentation results is performed on the match list, rotation- and translation parameters are computed for each entry and used by the following comparison techniques.

#### 3.5.2.2 Statistics regarding points abandoned by false rotation and translation parameters

In order to examine segmentation results, each database image is crossed with an arbitrary amount of vertical lines. Each line is 20 pixels apart. Thus, each tracked blood vessel is crossed by those lines. As a result, bifurcation points between the artificially introduced lines and the tracked blood vessels are generated and consequently subject to further examination. Figure 31a shows a segmented image with artificially introduced vertical lines and Figure 31b shows the corresponding blood vessel structure with the resulting artificial bifurcation points.

![Figure 31a: Segmented image with artificially introduced vertical lines](image1)

![Figure 31b: Blood vessel structure showing artificial bifurcation points](image2)

In most cases “wrong matches” contained by the match list yield wrong rotation and translation information. Therefore, in many cases the use of wrong rotation and translation parameters transform some of the points to coordinates which are invalid. The strategy used in this chapter is to compare the amount of artificial bifurcation points to the amount of artificial bifurcation points that are transformed to the unidentified image whilst retaining valid coordinates. Points which have invalid coordinates are not taken into account when statistical comparison operations are performed. The images that can be seen in Figures 32a and 32b illustrate the impact of the use of false parameters for rotation and translation. Figure 32a shows a database image which represents a potential albeit false match candidate. The match table entry involving that very image yields wrong rotation and translation parameters. Figure 32b shows the unidentified image. All artificial bifurcation points that are
found in Figure 32a are translated and rotated to the image using the wrong parameters. As a result, a large number of those cannot be found in the image seen in Figure 32b as all points that do not appear have invalid coordinates. As a result, the statistical measure which taken in this chapter is given by formula (16).

\[
\text{ratio} = \frac{v}{t} \quad (16)
\]

Where \(v\) is the amount of valid points that had transformed from the database image to the unidentified image \(t\) describes the total amount of artificial bifurcation points that had been generated in the database image.

3.5.2.3 Statistics regarding common distances between vessel points

In this step, the artificially introduced vertical lines discussed in Chapter 3.5.2.2 are rotated by the rotation angle which persists between each database image and the unidentified image. In the same step all artificial bifurcation points created by crossing the vertical lines with the tracked blood vessel structure are transformed to the unidentified image (Figure 33b).
The rotated lines are shifted horizontally in order to overlap with the transformed points (Figure 34a). Finally, the shifted and rotated lines are crossed with the tracked blood vessel structure of the unidentified image. The magenta points found in Figure 34b represent the blood vessel structure of the unidentified image whereas the green points represent the blood vessel structure of the database image.

Supposing all points lying on the same artificially introduced lines are interconnected, the corresponding interconnections become subject to comparison between the unidentified image and the database images. Figure 35a shows a false match whereas Figure 35b shows a correct match. The interconnections are indicated in both images by magenta lines and by green lines respectively. The magenta lines indicate the interconnections between the points which represent the blood vessel structure of the database image and the green lines indicate the interconnections inherited from the unidentified image.
Figures 35a and 35b clearly show that in the event of a match, large amounts of pairs of interconnections of equal length and orientation are found. Thus, one further method used in this project to detect whether or not two images match is to relate the amount of interconnection pairs found in the image resulting from the operations mentioned above to the amount of interconnections generated in the database image. The statistical relation can be seen in formula (17).

\[ \text{ratio} = \frac{m}{t} \]  

(17)

Where \( m \) stands for the amount of matching interconnections \( t \) stands for the total amount of interconnections found in the database image.

3.5.2.4 Statistics regarding amounts of image points vs. amounts of transformed points

In this step, artificial bifurcation points are included in the unidentified image as well. Thereby, the artificially introduced lines are rotated according to the rotation angle that persists between the database image and the unidentified image.

As in the method discussed in Chapter 3.5.2.2 the artificial bifurcation points found in the database image are transformed to the unidentified image. Finally, the amount of transformed artificial bifurcation points is compared to the amount of artificial bifurcation points generated in the image. High ratios indicate high likelihoods for a match whereas low ratios indicate low likelihoods. The mathematical expression can be seen in formula (18) below

\[ \text{ratio} = \frac{tr}{g} \]  

(18)

where \( tr \) stands for the amount of artificial points transformed to the unidentified image, \( g \) stands for the amount of points generated in the unidentified image.

In Figure 36b the points which have been generated in the unidentified image are marked in yellow and the transformed points are marked in cyan. This example clearly shows a false match due to the relatively low amount of cyan point in the image.
3.5.2.5 Statistics regarding transformed points that cannot be aligned

Supposing the translation matrix used for transforming all artificial bifurcation points extracted from the database image is correct, all artificial bifurcation points are supposed to be aligned. The statistical measure computed in this step is given formula (19).

\[ \text{ratio} = e^{\frac{\mu^2}{2\sigma^2}} \]  

(19)

The expectancy value, \( \mu \), is the input value, and thus the amount of unaligned points. The standard deviation, \( \sigma \), is set to an eighth of the amount of artificial bifurcation points generated in the database image. This value had been chosen as a result of empirical observations. In contrast to the ratios given by formulae (16) – (18) this ratio is weighted by a Gauss curve. As described in chapter 2.5 Gaussian weighted score values are more representative than ones yielded by linear functions in some cases. In this case it appears that Gaussian weighted score values are more representative.

Figures 37a and 37b demonstrate how well the points are aligned in case the match is correct.

Figure 37a shows one of the database images including its artificial bifurcation points generated for the matching algorithm. Figure 37b shows the unidentified image including its artificial bifurcation points as well as the points transformed from the database image. It can be seen that with an exception of three points all transformed points match with the original image points. The three points that do not have any counter parts in the unidentified image result from differences in segmentation. The fact that the amount of derelict points found in false matches is significantly higher eliminates the relevance of those points. A false match can be seen in Figure 38b.
The unidentified image that can be seen in Figure 38b is the same image as in Figure 37b whereas the database image in Figure 38a is a false matching candidate. The transformation of the artificial bifurcation points of that image leads to populations of artificial bifurcation points that are obviously nowhere near any of the blood vessels. In this case 31 derelict artificial bifurcation points can be found. Thus, the image used for this comparison can definitely be flagged as a false match as opposed to the image used for the previous comparison.
4. Results

As mentioned in Chapter 3.5, the matching process involves two basic steps, one being the bifurcation point analysis and the other being the examination of the segmentation results. Whereas the bifurcation point analysis is performed on all images stored in the database the second method is only used when the match list results yielded by the bifurcation point analysis procedure are inconclusive or incorrect.

As mentioned earlier, the sample used for this project is a pool of twenty mice whereas six pictures of all twenty mice prepared for this project had been taken. Thus, five pictures are used for enrollment and one for identification. The diagram in Figure 39 shows the breakdown on how many of the twenty mice were identified correctly and on the details regarding whether or not the segmentation results had to be examined.

![Pie chart showing the breakdown of matching results]

**Figure 39: Breakdown of the overall matching results**

As the diagram shows, two out of twenty mice were not identified. Thus, the recognition rate is 90%. Whereas 50% were matched, simply by performing bifurcation point analysis 40% required further examination of the corresponding segmentation results. The following chapters contain details on why certain images were not recognized and on why others could not be identified correctly by simply applying the bifurcation point analysis procedure.

4.1 Images that cannot be matched correctly

As previously mentioned two out of twenty mice examined by the algorithms used in this project can not be identified. One of the two is mouse number ten. Figures 40a and 40b show the corresponding images whereas the left image shows the database image and the right image is the unidentified image.
The illustrations clearly show that the right image is severely distorted. On the one hand it is horizontally compressed; on the other the angles of the vessel branches are different which is presumed to be a result of the horizontal compression. Additionally, both images comprise gaps in the blood vessel structure, both in different regions. Consequently, the vessel roots look completely different in both images.

Figures 41a and 42b visualize the information the database contains regarding mouse number ten whereas the left image represents the original image and the enrolled bifurcation points and the right image represents the segmented blood vessel structure.

The segmented image fully resembles the visible vessel structure without any noise. As can be seen in the center-right part of the images there are gaps which disrupt the blood vessels. Obviously, the gaps hide two bifurcation points as well. Although the segmentation algorithm partially compensates for the gaps, the two bifurcation points that are invisible are not found. Similar failures occur in the process of segmenting the unidentified image which can be seen
in Figure 42a. The segmented blood vessel map which can be seen in Figure 42b refers to the image in Figure 42a.

![Figure 42a: Original image of mouse #10 (unidentified image)](image1)

![Figure 42b: Segmented image of mouse #10 (unidentified image)](image2)

This segmentation result as well resembles the vessel structure of the input image. The difference between both segmentation results can be seen in Figures 43a and 43b.

![Figure 43a: Segmented image of mouse #10 (database image)](image3)

![Figure 43b: Segmented image of mouse #10 (unidentified image)](image4)

As can be seen very clearly, the bifurcation points located in the center of the image are closer together in the right image. Additionally, the branch environments have different angles in both images. As previously mentioned, the root structure is completely different in both images. Finally, the upper most bifurcation point is located left of the root in the left image and right of the root in the right image. The conclusion drawn from these findings is that the image used for comparison with the database is not acceptable for use in this project. The image is distorted and thus we assume that the ear had been folded while being taken. Due to the fact that the algorithms used in this project are based on angle environments and lengths of distances between points, distorted images cause difficulties.
The second mouse that could not be identified remains troublesome due to a lack of information provided by the animal’s ear. As can be seen in Figure 44b the image shown in Figure 44a contains solely one bifurcation point that is identified by the segmentation algorithm.

The fact that only one bifurcation point is found causes trouble since at least two bifurcation points are required in order to compute rotation- and translation parameters for transforming points from one image to the other. As a result of the fact that the rotation- and translation parameters are required for the examination of segmentation results neither bifurcation-point-analysis nor segmentation result examination can be performed on this image. The conclusion drawn from this finding is that the unidentified image does not comply with the requirement regarding the minimum amount of bifurcation points per image.

4.2 Images that can be identified solely by performing bifurcation point analysis

As mentioned previously, two mice, namely #9 and #19, yield matching results without any false entries in the match list which means that all potential matches yielded by the bifurcation point analysis algorithm are the correct ones. Whereas the image which turns out to belong to mouse #9 has fourteen pairs of bifurcation points in common with the corresponding database image, the one belonging to mouse #19 has eight pairs in common with its database image.

Whereas the vessel structures of some mice may be absolutely unique, others can show similarities to the vessel structures of other mice. In some cases statistical analysis of the match list yielded by the bifurcation point analysis algorithm can lead to unique identification. Using this sample, eight of twenty mice are identified solely by considering the frequencies of occurrence regarding the amounts of potential matches found in the match list. The number of entries for each mouse is noted and the mouse which has the maximum amount of potential matches is the winner.

The most apparent matching result among the ones which are also matched with false candidates is yielded from the matching operation regarding mouse #16. The matching result can be seen in Figure 45.
As can be seen in the diagram, mouse #1, #2 and #10 appear in the matching table with one entry respectively. In contrast, #16 appears 27 times in the table.

For other mice the matching result is less obvious though still conclusive. As an example, one unidentified image of mouse #3 is compared to the database images and as a result, seven matches are found for #3, four for #16 and one for #12 and #18 respectively. This occurs whenever pairs of bifurcation points found in the unidentified image contain similar distances and angle environments as pairs of bifurcation points found in database images. Following figures demonstrate how similar pairs of bifurcation points yield matches in multiple images. In order to retain the conciseness of the illustrations only the most obvious similarities are shown.
The images on the left hand side are the unidentified images whereas the images on the right hand side show the database images. The colored lines found in the unidentified images symbolize virtual connections between pairs of bifurcation points used for matching. Each color corresponds to a different mouse. Red belongs to mouse #16, orange to #18, magenta to
#12 and green to #3. As an example, Figure 46b shows the blood vessel structure of mouse #16. The matching algorithm finds two pairs of bifurcation points in that image which are similar to pair located in the unidentified image connected by the red line. For visualization the mentioned pairs found in Figure 46b are as well interconnected by red lines. The same applies to Figures 46c through 46f. The correct matches are symbolized by the green lines found in Figure 46g and 46h. Due to the fact that Figure 46h shows the database image of mouse #3 the formerly unidentified image is recognized as an image of mouse #3.

### 4.3 Images that cannot be identified solely by performing bifurcation point analysis

As some unidentified images which are compared to the database yield match lists that allow examination of the frequencies of occurrence to determine a correct match, others require more versatile means of examination in order to find a definite match. In some cases the match lists yielded by the bifurcation point analysis algorithm contain a few correct entries among numerous false entries and in some cases the match list solely contains false entries. As the statistical breakdown, stated at the beginning of this chapter shows, the matching procedure based on bifurcation point analysis yields match lists containing solely false entries for two of twenty mice. In both cases the match list consists of one entry only. Regardless of whether or not the entries contained by single entry match lists are correct, no statistical examinations on the frequency of occurrence can be performed. Thus, images yielding single entry match lists are automatically subject to segmentation result examination.

Whether or not match lists obtained by performing bifurcation point analysis are acceptable as sole evidence leading to decisions on whether or not images match with counterparts stored in the database are decisions that were made based on observations. While the match list containing 30 entries with 27 entries corresponding to the same mouse obviously indicates a match, there are three match lists which contain four entries that involve three different mice. In two of those three cases the double entry indicates the match. The fact that for one of those three tables the match is not indicated by the double entry enforces a policy that requires the match list to comprise a minimum amount of entries and a dedicated match to be the absolute majority among all entries. As a result of our observations we decided to set the minimum amount of entries to eight. Thus, eight of twenty mice require further examination in order to be identified.

As mentioned in Chapter 3.5 various approaches are used to perform statistical examination of the segmentation results in this project. In the following example the unidentified image belongs to mouse #2, a mouse that has a blood vessel structure which is sparsely populated with bifurcation points. Thus, the bifurcation point analysis procedure yields a match list with rather few entries. Each entry corresponds to a different mouse whereas one of those entries refers to mouse #2. In practice, it would be sufficient to pass that table on to the functions that perform the segmentation result examination as discussed in Chapter 3.5.2. In order to point out the fact that the unidentified image matches mouse #2 more clearly, the sample is enlarged. This is done by loosening the thresholds used by the bifurcation point analysis algorithm and by rerunning the bifurcation point analysis. The sample enlargement step has only been taken for illustrative reasons.
The diagram, shown in Figure 47, displays the statistical breakdown of the frequencies of occurrence of each mouse in the match list that results from loosening the thresholds and rerunning the bifurcation point analysis. Where the old match list used to have four entries the match list that is involved in the diagram that can be seen in Figure 47 contains 17 entries. Although the highest frequency of occurrence refers to mouse #2 the result of the bifurcation point analysis is not conclusive since the match list entries belonging to mouse #2 do not make up the majority.

![Figure 47: Frequencies of occurrence in the match list using the enlarged sample](image)

Although the diagram shows that #2 is found most often in the match list, further tests are run on the unidentified image. Next, statistics regarding points abandoned by false rotation and translation parameters are computed. This method had been discussed in Chapter 3.5.2.2. The statistical results can be seen in Figure 48. In contrast to the diagram which can be seen in Figure 47, the bars indicate maximum values of a ratio. For each entry contained by the match list, one individual ratio is computed. The maximum ratio of each mouse which appears in the match list is shown by the bars in Figure 47.

![Figure 48: Statistics regarding points abandoned by false rotation and translation parameters](image)
Again, mouse #2 appears to have the highest ratio among the other mice. The next test is similar to the previous one as it involves statistics regarding the amount of image points versus the amount of transformed points. This method had been discussed in Chapter 3.5.2.4. Figure 49 shows the result.

![Graph showing ratios for different mouse numbers](image1)

**Figure 49: Statistics regarding amounts of image points vs. amounts of transformed points**

Although hardly visible in the diagram above, mouse #2 is slightly ahead of #16. Next, statistics regarding common distances between vessel points are computed and displayed in the graph which can be seen in Figure 50. This method had been discussed in Chapter 3.5.2.3.

![Graph showing ratios for different mouse numbers](image2)

**Figure 50: Statistics regarding common distances between vessel points**

In the case of this example, the result of this test is most representative. Mouse #2 clearly comprises the highest ratio of common distances. The final test involves statistics regarding transformed points that cannot be aligned. This technique is discussed in Chapter 3.5.2.5. The result can be seen in Figure 51.
In this final test #2 is the winner as well. Thus, the overall result of all five tests run on the match list using the enlarged sample is that the unidentified image belongs to mouse #2.

Whereas identification of images which refer to mouse #2 requires additional treatment by the segmentation result examination algorithm due to its sparse population of bifurcation points, other images require the same amount of attention for other reasons. As an example, the unidentified image of mouse #8 contains only a part of the enrolled blood vessel structure as can be seen in Figures 52a and 52b.

Examination of the segmentation results yields a unique and correct match whereas two of the five proposed methods are used. As the method regarding the common distances is strongly related to the method regarding the transformed points that cannot be aligned, only the method regarding the common distances is required for correct recognition in this example. Additionally, statistics regarding the points abandoned by false rotation and translation parameters are considered. All other methods of segmentation result examination are not used in this case.
Generally speaking, during the process of matching, all eight mice which require segmentation result examination, at most three of the five methods were required simultaneously. Whereas all eight mice need the results obtained by the method that computes the statistics regarding the points abandoned by false rotation and translation parameters, six of eight mice need the function that considers the ratio of common distances, and the other two mice need the algorithm that counts the absolute frequency of occurrence of each mouse in the match list yielded by the bifurcation point analysis.
5. Conclusion

As pointed out in Chapter 4, not all of the images destined for identification were successfully identified. Whereas one of the images was severely distorted, the other was under-populated by bifurcation points. On the other hand, images that merely showed fractions of the blood vessel structures they actually had were identified. This proves that while attempting to develop high end algorithms that can handle multiple cases of faulty images it is rather challenging to consider all cases of abnormalities. The fact that all algorithms used in this project are based on the use of bifurcation points and on the features they provide, falsified bifurcation point environments lead to false or to non existent results. However, observations have proven that other sources of noise that deteriorate the quality of the images do not affect the ability of the algorithms presented in this thesis to perform reliable identification.
6. Future work and remaining issues

As pointed out in [17] blood vessel enhancement and detection algorithms involve high computational loads in many cases. Many image processing tools such as image filtering and matching operations require numerous cycles for execution. Algorithms that involve high computational loads require high end and thus expensive machines for execution if they are supposed to run in real time. The alternative is the use of less powerful machines for execution which would lead to longer execution times. Both situations are disadvantageous. In this project the focus was set on accuracy and high recognition rates rather than saving computational power and execution times. Therefore one future remaining issue in this project would be the question, how the algorithms used in this project could be optimized in terms of execution speed.

Chapter 3.4.2 describes various methods used to examine the segmentation results and to use those results for identification. However, this project does not contain an algorithm that regulates when the particular methods introduced in Chapter 3.4.2 are used and to what extent they are considered when computing an overall matching result. Therefore, future work on this project could involve regulating the use of the tools introduced in Chapter 3.4.2 for identification. As an example, fuzzy sets could be defined in order to control the level of contribution provided by each technique.

In addition to the six grey scale images of each mouse that were provided for this project, six corresponding RGB images per mouse were provided as well. However, the color images were never used. Reasonable future work on this project could involve separating falsely detected blood vessels or blood vessel features from background noise, as blood tends to be red while background noise can contain any colour.
7. References

[1] The Jackson Laboratory, Frequently Asked Questions
http://jaxmicejax.org/library/faq/idsystem.html

http://www.ketchum.on.ca/aramis1.html#Importance


[10] Glossary of Meteorology
http://amsglossary.allenpress.com/glossary


References


Appendix A

`launcher.m`

```
#!/usr/bin/env python

% Start of the enhancement procedure

% Input parameters for the function filterImage:
% 1) Original image
% 2) Filter size for the 1st filter
% 3) Standard deviation for the 1st filter in x-direction (perpendicular to the blood vessel)
% 4) -“- in y-direction (in the direction of the blood vessel)
% 5) Filter size for the 2nd filter
% 6) Standard deviation for the 2nd filter in x-direction
% 7) -“- in y-direction
% 8) Amount of angles (directions)

filterImage(i1_1, 70, 5, 15, 30, 3, 10, 16);

% Start of the segmentation procedure

load('enhancedImage1.mat');
load('filteredImage1.mat');

% Input parameters for the function MainTracking:
% 1) Enhanced image
% 2) 16 filtered images (one for each direction)
% 3) Tracked image (if already exists)
% 4) Altered enhanced image (for finding starting points)
%    only important for the extraction of vessels with the smaller filter settings
% 5) 1st threshold for finding starting points
% 6) 2nd threshold (exit condition for the tracking of a vessel portion)
% 7) Decider: Binary value which decides if a post-processing step gets applied to the tracked image or not

MainTracking(enhancedImage1, filteredImage1, zeros(300), enhancedImage1, 5, 20, 0);
load ('tracked_image.mat');

% Applying the post-processing algorithm to the first tracked image

% Input parameters for the function PostProcessing
% 1) Tracked image
% 2) 1st threshold: Minimum length of blood vessel portions in the tracked image, applied before connecting all lost vessels
% 3) 2nd threshold: Minimum length of blood vessel portions in the tracked image, applied after connecting all lost vessels

PostProcessing(tracked_image, 20, 130);
load ('postprocessed_image.mat');

% Loading the enhanced images created with the 2nd matched filter

load('enhancedImage2.mat');
load('filteredImage2.mat');

% Creating an image where all the tracked vessels can not serve as starting points anymore

% Input parameters
% 1) Tracked image before applying the post-processing algorithm to it
% 2) Enhanced image obtained with the 2nd (smaller) matched filter

NewEnhancedImage = CreateNewEnhancedImage(trackedImage, enhancedImage2);

% New tracking procedure with starting points from the new enhanced image

MainTracking(enhancedImage2, filteredImage2, trackedImage, NewEnhancedImage, 0.5, 20, 1);
```
Appendix A

load ('tracked_image.mat');

%Postprocessing the tracked image
PostProcessing(tracked_image,20,130);

%Postprocessing the tracked image

filterImage.m

function filterImage(originalImage, filterSize1, std_x1, std_y1, filterSize2, std_x2, std_y2, amountOfAngles)

%Settings first filter
FILTERSIZE1 = filterSize1;
STD_X1 = std_x1;
STD_Y1 = std_y1;

%Settings second filter
FILTERSIZE2 = filterSize2;
STD_X2 = std_x2;
STD_Y2 = std_y2;

ANGLES = (pi/amountOfAngles) * [0 : amountOfAngles - 1];
image = myScale(originalImage);
dimensions = size(image);

for c1=1:length(ANGLES)

%First filter
filter1 = kernelFilter(FILTERSIZE1, STD_X1, FILTERSIZE1, STD_Y1, ANGLES(c1), 1, -1);
filter2 = -kernelFilter(FILTERSIZE1, STD_X1, FILTERSIZE1, STD_Y1, ANGLES(c1), -1, -1);
filter4 = -kernelFilter(FILTERSIZE1, STD_X1, FILTERSIZE1, STD_Y1, ANGLES(c1), 1, 1);
filter5 = kernelFilter(FILTERSIZE1, STD_X1, FILTERSIZE1, STD_Y1, ANGLES(c1), -1, 1);
filter6 = -kernelFilter(FILTERSIZE1, STD_X1, FILTERSIZE1, STD_Y1, ANGLES(c1), -1, 0);
filter = filter1 + filter2 + filter3 + filter4 + filter5 + filter6;
filteredImage1(c1,:,:) = conv2(image,filter,'same');

%Second filter
filter1 = kernelFilter(FILTERSIZE2, STD_X2, FILTERSIZE2, STD_Y2, ANGLES(c1), 1, -1);
filter2 = -kernelFilter(FILTERSIZE2, STD_X2, FILTERSIZE2, STD_Y2, ANGLES(c1), -1, -1);
filter4 = -kernelFilter(FILTERSIZE2, STD_X2, FILTERSIZE2, STD_Y2, ANGLES(c1), 1, 1);
filter5 = kernelFilter(FILTERSIZE2, STD_X2, FILTERSIZE2, STD_Y2, ANGLES(c1), -1, 1);
filter6 = -kernelFilter(FILTERSIZE2, STD_X2, FILTERSIZE2, STD_Y2, ANGLES(c1), -1, 0);
filter = filter1 + filter2 + filter3 + filter4 + filter5 + filter6;
filteredImage2(c1,:,:) = conv2(image,filter,'same');

end

dimensions_filtered = size(filteredImage1);
for c2=1:dimensions_filtered(2)
    for c3=1:dimensions_filtered(3)
        for c4=1:length(ANGLES)
            pixelValues1(c4) = filteredImage1(c4,c2,c3);
            pixelValues2(c4) = filteredImage2(c4,c2,c3);
        end
        enhancedImage1(c2,c3)=min(pixelValues1);
        enhancedImage2(c2,c3)=min(pixelValues2);
    end
end
Appendix A

```
myScale.m

function retVal = myScale(inputVal)

AMOUNT_OF_GRAYVALUES = 255;
maxVal = max(max(inputVal));
minVal = min(min(inputVal));
oldRange = maxVal - minVal;
oldSteps = oldRange / AMOUNT_OF_GRAYVALUES;
oldStepVector = minVal:oldSteps:maxVal;
dimensions = size(inputVal);
for c1 = 1:AMOUNT_OF_GRAYVALUES
    for c2 = 1:dimensions(1)
        for c3 = 1:dimensions(2)
            if inputVal(c2,c3) < oldStepVector(c1+1)
                if inputVal(c2,c3) >= oldStepVector(c1)
                    retVal(c2,c3) = c1;
                end
            end
        end
    end
end

kernelFilter.m

function retVal = kernelFilter(x, std_x, y, std_y, angle, w, shift)

r = [cos(angle) -sin(angle); sin(angle) cos(angle)];
for c1 = 1:y
    for c2 = 1:x
        u = r * 
        retVal(c1, c2) = gaussDerivated(u(1), std_x) * gauss(u(2), std_y);
    end
end
retVal = retVal / sqrt(sum(sum(retVal.*retVal)));

gaussDerivated.m

function retVal = gaussDerivated(x, std)

for c1 = 1:length(x)
    retVal(c1) = -((x(c1)/std)^2)*exp(-((x(c1)^2)/(2*std^2)))/(std*sqrt(2*pi));
end
```
**gauss.m**

```matlab
function retVal = gauss(x, std)
    retVal = exp(- (x.*x)/(2*std^2))/(std*sqrt(2*pi));
```

**MainTracking.m**

```matlab
function MainTracking(enhancedImage, filteredImages, tracked_image,
    NewEnhancedImage, threshold1, threshold2, decider)
    
    % Display the enhanced image
    figure(1);
    imagesc(enhancedImage); colormap 'gray';
    
    % Finding the starting points for the tracking procedure
    % script startingPoints.m
    position = startingPoints(NewEnhancedImage, threshold1);
    
    % Start of the actual tracking procedure
    % Definition of the threshold for which the tracking should stop
    PERCENT = threshold2;
    enhancedImage_reshaped = reshape(enhancedImage, 1, size(enhancedImage, 1) * size(enhancedImage, 2));
    enhancedImage_reshaped_sorted = sort(enhancedImage_reshaped);
    point = round((size(enhancedImage, 1) * size(enhancedImage, 2) / 100) * PERCENT);
    treshold = enhancedImage_reshaped_sorted(point);
    copy_enhancedImage = enhancedImage;
    for a = 1:size(position, 3)
        original_position(:,:,a) = position(:,:,a);
        b = 0;
        c = 0;
        tolerance = 0;
        exit = 0;
        tracking_counter = 0;
        temp_tracked_image = zeros(300);
        if tracked_image(position(1, 1, a), position(1, 2, a)) == 1
            exit = 1;
            tracking_counter = 40;
        end
        
        % Get initial neighbours
        [neighbour_code, angle1, angle2] = GetInitialNeighbours(position(:,:,a), enhancedImage, filteredImages, 1);
        angle_history1 = [0 0 0 0 angle1];
        angle_history1 = angle_history1';
        angle_history2 = [0 0 0 0 angle2];
        angle_history2 = angle_history2';
        while exit == 0
            % Exit if tracking reached the boarder of the image or if the point
            % already tracked
```
if position(1,1,a)<=2 || position(1,1,a)>=298 || position(1,2,a)<=2 || position(1,2,a)>=298
    break;
end

if tracked_image(position(1,1,a),position(1,2,a))==1 && b==1
    break;
end

temp_tracked_image(position(1,1,a),position(1,2,a))=1;
tracking_counter=tracking_counter+1;

if tracked_image(position(1,1,a),position(1,2,a))==1 && b==0
    temp_tracked_image(position(1,1,a),position(1,2,a))=0;
b=1;
end

copy_enhancedImage(position(1,1,a),position(1,2,a))=1000;
tracked_image(position(1,1,a),position(1,2,a))=1;

%__________________________________________________________________
%Start the tracking in the "upper" direction and continue in the opposite direction
if b==0
    [angle_history1,angle_history2,neighbour_code]=GetNeighbourCode(position(:,:,a),enhancedImage,filteredImages,angle_history1,angle_history2,neighbour_code);
    neighbours=GetNeighbours(neighbour_code,position(:,:,a));
elseif b==1
    if c==0
        c=1;
        position(:,:,a)=original_position(:,:,a);
        tolerance=0;
        [neighbour_code,angle1,angle2]=GetInitialNeighbours(position(:,:,a),enhancedImage,filteredImages,-1);
        angle_history1=[0 0 0 0 angle1];
        angle_history1=angle_history1';
        angle_history2=[0 0 0 0 angle2];
        angle_history2=angle_history2';
    end

    [angle_history1,angle_history2,neighbour_code]=GetNeighbourCode(position(:,:,a),enhancedImage,filteredImages,angle_history1,angle_history2,neighbour_code);
    neighbours=GetNeighbours(neighbour_code,position(:,:,a));
end

maximum=1000;
for d=1:size(neighbours,3)
    if enhancedImage(neighbours(1,1,d),neighbours(1,2,d))<maximum
        maximum=enhancedImage(neighbours(1,1,d),neighbours(1,2,d));
        position(:,:,a)=neighbours(:,:,d);
    end
end

%__________________________________________________________________
%Exit the loop if the angle changes rapidly
%
%First angle history
counter=0;
for d=1:size(angle_history1,1)
    if angle_history1(d)==0
        counter=counter+1;
    end
end

angle_change1=0;
if counter>1;
angle_change1=abs(angle_history1(size(angle_history1,1))-angle_history1(size(angle_history1,1)-1));

%Second angle history
counter=0;
for d=1:size(angle_history2,1)
    if angle_history2(d)~=0
        counter=counter+1;
    end
end

angle_change2=0;
if counter>1
    angle_change2=abs(angle_history2(size(angle_history2,1))-angle_history2(size(angle_history2,1)-1));
end

angle_change=min(angle_change1,angle_change2);

% Exit the loop if the new position does not exceed a predefined threshold or the angle changes to rapidly
if maximum<treshold && tolerance==1
    tolerance=0;
end
if (maximum>treshold && b==1 && tolerance==1) || (angle_change>45 && b==1)
    tracked_image(position(1,1,a),position(1,2,a))=0;
    temp_tracked_image(position(1,1,a),position(1,2,a))=0;
    break;
end
if (maximum>treshold && b==0 && tolerance==1) || (angle_change>45 && b==0)
    tracked_image(position(1,1,a),position(1,2,a))=0;
    temp_tracked_image(position(1,1,a),position(1,2,a))=0;
    b=1;
end
if maximum>treshold && tolerance==0
    tolerance=1;
end

% Deleting the tracked points from the binary image if the amount of tracked points under-runs a certain threshold
if tracking_counter < 40 && decider==0
    for i=1:size(tracked_image,1)
        for j=1:size(tracked_image,2)
            if temp_tracked_image(i,j)==1
                tracked_image(i,j)=0;
            end
        end
    end
end

figure(6); imagesc(copy_enhancedImage); colormap 'gray';
figure(7); imagesc(tracked_image); colormap 'gray';

% Save the tracked image
save('tracked_image.mat', 'tracked_image');
startingPoints.m

function position=startingPoints(enhancedImage,treshold1)

SUB_IMAGES=144

%Finding the (SUB_IMAGES) darkest points in the enhanced image
[y,x]=size(enhancedImage);
size_y=floor(y/sqrt(SUB_IMAGES));
size_x=floor(x/sqrt(SUB_IMAGES));

counter=0;
for n=1:SUB_IMAGES
    counter=counter+1;
    darkest_points(counter)=max(max(enhancedImage))+1;
end

counter=1;
for n=1:sqrt(SUB_IMAGES)
    for m=1:sqrt(SUB_IMAGES)
        for i=(n-1)*size_y+1:n*size_y
            for j=(m-1)*size_x+1:m*size_x
                if enhancedImage(i,j)<darkest_points(counter)
                    darkest_points(counter)=enhancedImage(i,j);
p
            end
        end
    end
    counter=counter+1;
end

%Visualization of the found starting points
copy_enhancedImage=enhancedImage;
for n=1:SUB_IMAGES
    if (position(1,1,n)>1 && position(1,1,n)< 300) && (position(1,2,n)>1 && position(1,2,n)< 300)
        copy_enhancedImage(position(1,1,n)-1:position(1,1,n)+1, position(1,2,n)-1:position(1,2,n)+1)=2000;
    end
end
figure(3); imagesc(copy_enhancedImage); colormap 'gray';

%Finding of the different intensity values for which a point is defined not to be on the vessel
PERCENT=treshold1;

enhancedImage_reshaped=reshape(enhancedImage,1,size(enhancedImage,1)*size(enhancedImage,2));
enhancedImage_reshaped_sorted=sort(enhancedImage_reshaped);

point=round((size(enhancedImage,1)*size(enhancedImage,2)/100)*PERCENT);
treshold=enhancedImage_reshaped_sorted(point);

%The points which are not on a blood vessel get deleted
counter=0
for n=1:size(position,3)
    if darkest_points(n)<treshold
        counter=counter+1;
p
    end
end

position=position_new;

copy_enhancedImage=enhancedImage;
copy_copy_enhancedImage=copy_enhancedImage;

%Visualization of the remaining starting points
for n=1:size(position,3)
    if (position(1,1,n)>1 && position(1,1,n)< 300) && (position(1,2,n)>1 && position(1,2,n)< 300)
        copy_enhancedImage(position(1,1,n)-1:position(1,1,n)+1, position(1,2,n)-1:position(1,2,n)+1)=2000;
    end
end

figure(5); imagesc(copy_enhancedImage); colormap 'gray';

%__________________________________________________________________________
%Deleting those points which lie close to each other

global_counter=0;
for a=1:291
    for b=1:291
        counter=0;
        bright=0;
        for y=a:a+9
            for x=b:b+9
                if copy_enhancedImage(y,x)==2000
                    counter=counter+1;
                    pos(:,1,counter)=[y,x];
                    bright(counter)=enhancedImage(y,x);
                end
            end
        end
        if counter>0
            min_bright=min(bright);
            for c=1:counter
                if bright(c)~=min_bright
                    copy_enhancedImage(pos(1,1,c),pos(1,2,c))=enhancedImage(pos(1,1,c),pos(1,2,c));
                end
            end
        end
    end
end
counter=0;
for y=1:300
    for x=1:300
        if copy_enhancedImage(y,x)==2000
            counter=counter+1;
            temp(:,1,counter)=[y,x];
        end
    end
end
position=temp;

%Visualization of the final starting points

figure(8); imagesc(copy_enhancedImage); colormap 'gray';
% Sorting the starting points according to their brightness
for a=1:size(position,3)
    brightness(a)=enhancedImage(position(1,1,a),position(1,2,a));
end
[brightness,index]=sort(brightness);
for a=1:size(position,3)
    temp(:,:,a)=[position(1,1,index(a)),position(1,2,index(a))];
end
position=temp;

GetInitialNeighbours.m

function [neighbour_code, angle1, angle2]=GetInitialNeighbours(position, enhancedImage, filteredImages, direction)
% Get the angle at the point "position"
[angle1, angle2]=GetAngle(position, filteredImages, enhancedImage);
% Mapping a neighbour code to the obtained angle
if angle1 >= 157.5 && angle1 < 202.5
    neighbour_code=1;
elseif angle1 >= 202.5 && angle1 < 247.5
    neighbour_code=2;
elseif (angle1 >= 90 && angle1 < 112.5) || (angle1 >= 247.5 && angle1 <= 270)
    neighbour_code=3;
elseif angle1 >= 112.5 && angle1 < 157.5
    neighbour_code=4;
end
% If the tracking goes in the opposite ("lower") direction, invert the neighbour code
if direction == -1
    neighbour_code = neighbour_code+4;
end

GetNeighbourCode.m

function [angle_history1, angle_history2, neighbour_code]=GetNeighbourCode(position, enhancedImage, filteredImages, angle_history1, angle_history2, neighbour_code)
% Get the old reference angles from the angle history
% First angle
counter=0;
for i=1:size(angle_history1,1)
    if angle_history1(i)~=-0
        counter=counter+1;
    end
end
angle_old1=sum(angle_history1)/counter;
% Second angle
counter=0;
for i=1:size(angle_history2,1)
    if angle_history2(i)~=0
        counter=counter+1;
    end
end

angle_old2=sum(angle_history2)/counter;
[old_reference_angle1,old_reference_angle2]=GetReferenceAngle(angle_old1,angle_old2);
%__________________________________________________________________________
%Get the new reference angle
%First angle
angle_history1=circshift(angle_history1,-1);
angle_history1(size(angle_history1,1))=angle1;

counter=0;
for i=1:size(angle_history1,1)
    if angle_history1(i)~=0
        counter=counter+1;
    end
end

angle_new1=sum(angle_history1)/counter;
%Second angle
angle_history2=circshift(angle_history2,-1);
angle_history2(size(angle_history2,1))=angle2;

counter=0;
for i=1:size(angle_history2,1)
    if angle_history2(i)~=0
        counter=counter+1;
    end
end

angle_new2=sum(angle_history2)/counter;
[new_reference_angle1,new_reference_angle2]=GetReferenceAngle(angle_new1,angle_new2);
%__________________________________________________________________________
%Calculating the angular differences and the neighbour code
angle_difference1=new_reference_angle1-old_reference_angle1;
angle_difference2=new_reference_angle2-old_reference_angle2;
diff1=abs(new_reference_angle1-old_reference_angle1);
diff2=abs(new_reference_angle2-old_reference_angle2);
if diff1 <= diff2
    angle_difference=angle_difference1;
else
    angle_difference=angle_difference2;
end
if angle_difference > 0
    neighbour_code=neighbour_code+1;
elseif angle_difference < 0
    neighbour_code=neighbour_code-1;
end
%__________________________________________________________________________
%Correction of the neighbour code
if neighbour_code==0
    neighbour_code=8;
end
if neighbour_code==9
    neighbour_code=1;
end
GetNeighbours.m

```matlab
function neighbours = GetNeighbours(neighbour_code, position)
% Mapping the neighbour code to 3 neighbourhood pixels at the point
% "position"

direction = 1;

if neighbour_code > 4
    direction = -1;
end

if neighbour_code == 1 || neighbour_code == 5
    neighbours(:,:,1) = [position(1) + direction, position(2) + direction];
    neighbours(:,:,2) = [position(1), position(2) + direction];
    neighbours(:,:,3) = [position(1) - direction, position(2) + direction];
elseif neighbour_code == 2 || neighbour_code == 6
    neighbours(:,:,1) = [position(1), position(2) + direction];
    neighbours(:,:,2) = [position(1) - direction, position(2) + direction];
    neighbours(:,:,3) = [position(1) - direction, position(2)];
elseif neighbour_code == 3 || neighbour_code == 7
    neighbours(:,:,1) = [position(1) - direction, position(2) + direction];
    neighbours(:,:,2) = [position(1) - direction, position(2)];
    neighbours(:,:,3) = [position(1) - direction, position(2) - direction];
elseif neighbour_code == 4 || neighbour_code == 8
    neighbours(:,:,1) = [position(1) - direction, position(2)];
    neighbours(:,:,2) = [position(1) - direction, position(2) - direction];
    neighbours(:,:,3) = [position(1), position(2) - direction];
end
```

GetAngle.m

```matlab
function [angle1, angle2] = GetAngle(position, filteredImages, enhancedImage)
% In charge of returning the two most significant directions at the point
% "position"
% The second angle is a copy of the first one with the difference that the
% angular difference between them is 90 degrees

for a = 1:size(filteredImages, 1)
    if filteredImages(a, position(1), position(2)) == enhancedImage(position(1), position(2))
        picture = a;
    end
end
angle1 = (180 / size(filteredImages, 1)) * picture - 180 / size(filteredImages, 1) + 90;
if picture > size(filteredImages, 1)/2+1
    angle2 = (180 / size(filteredImages, 1)) * picture - 180 / size(filteredImages, 1) - 90;
else
    angle2 = angle1;
end
```

GetReferenceAngle.m

```matlab
function [reference_angle1, reference_angle2] = GetReferenceAngle(angle1, angle2)
% Calculating reference angle 1
```
if angle1 >= 157.5 && angle1 < 202.5
    reference_angle1=180;
elseif angle1 >=202.5 && angle1 < 247.5
    reference_angle1=225;
elseif (angle1 >=90 && angle1 < 112.5) || (angle1 >= 247.5 && angle1 <=270)
    reference_angle1=90;
elseif angle1 >= 112.5 && angle1 < 157.5
    reference_angle1=135;
end
%Calculating reference angle 2
if angle2 >= 157.5 || angle2 < 22.5
    reference_angle2=180;
elseif angle2 >=22.5 && angle2 < 67.5
    reference_angle2=45;
elseif angle2 >= 67.5 && angle2 < 112.5
    reference_angle2=90;
elseif angle2 >= 112.5 && angle2 < 157.5
    reference_angle2=135;
end

function enhancedImage=CreateNewEnhancedImage(trackedImage,enhancedImage)
%Morphologic operations to expand the tracked region
SE=strel('square',13);
trackedImage=imdilate(trackedImage,SE);
figure(1); imagesc(trackedImage); colormap 'gray';
maximum=max(max(enhancedImage));
for y=1:size(enhancedImage,1)
    for x=1:size(enhancedImage,2)
        if trackedImage(y,x)==1
            enhancedImage(y,x)=maximum;
        end
    end
end
figure(2); imagesc(enhancedImage); colormap 'gray';

function PostProcessing(trackedImage,treshold1,treshold2)
figure(1); imagesc(trackedImage); colormap 'gray';
%Removing small objects
TRESHOLD=treshold1;
trackedImage = bwareaopen(trackedImage,TRESHOLD);
%Applying morphologic operations to connect the branches together
sel = strel('square',5);
trackedImage=imclose(trackedImage,sel);
Appendix A

% Skeletonizing the binary image
trackedImage=bwmorph(trackedImage,'skel',Inf);
figure(5); imagesc(trackedImage); colormap 'gray';

% Removing big, unconnected objects
TRESHOLD=treshold2;
trackedImage = bwareaopen(trackedImage,TRESHOLD);
figure(3); imagesc(trackedImage); colormap 'gray';

% Removing branches below a certain pixel size
trackedImage=MainRegistration(trackedImage);
figure(4); imagesc(trackedImage); colormap 'gray';
save('postprocessed_image.mat', 'trackedImage');

MainRegistration.m

function segmented_image = MainRegistration(segmented_image)

% Finding the bifurcation points in the image
[y,x]=size(segmented_image);
temp=zeros(x);
object(1)=strel([0 0 0; 1 0 1; 0 1 0]);
object(2)=strel([0 1 0; 0 0 1; 0 1 0]);
object(3)=strel([0 1 0; 1 0 1; 0 0 0]);
object(4)=strel([0 1 0; 1 0 0; 0 1 0]);
object(5)=strel([0 1 0; 0 0 1; 1 0 0]);
object(6)=strel([0 0 1; 1 1 0; 0 1 0]);
object(7)=strel([1 0 0; 1 0 1; 0 0 0]);
object(8)=strel([0 1 0; 1 1 0; 0 0 1]);
object(9)=strel([0 0 0; 1 1 1; 0 1 0]);
object(10)=strel([0 1 0; 0 1 1; 0 1 0]);
object(11)=strel([0 1 0; 1 1 1; 0 0 0]);
object(12)=strel([0 1 0; 1 1 0; 0 1 0]);
bifurcation_points = 0;
final = 0;
for i=1:12
    bifurcation_points=imopen(segmented_image,object(i));
    final = max(final, bifurcation_points);
end
bifurcation_points_clean=0;
bifurcation_points_real=zeros(x);
for i=1:12
    bifurcation_points_clean=imopen(final,object(i));
    temp(:,:,i)=bifurcation_points_clean(:,:);
end
pcounter = 1;
for m=1:y
    for n=1:x
        if temp(m,n,1)==1
            bifurcation_points_clean_real(m,n+1)=1;
            temp(m+1,n:n+2,1)=0;
            neighbours_test(:,:,1,pcounter)=[m,n];
            neighbours_test(:,:,2,pcounter)=[m,n+2];
            neighbours_test(:,:,3,pcounter)=[m+1,n+1];
        end
    end
end
Appendix A

\[
\text{points}(i,:,pcounter)=[m,n+1];
pcounter = pcounter + 1;
\]

\[
\text{if } temp(m,n,2)==1
\]
\[
\quad '2'
\quad \text{bifurcation_points_real}(m+1,n)=1;
\quad \text{temp}(m:m+2,n:n+1,2)=0;
\quad \text{neighbours_test}(i,:,1,pcounter)=[m,n];
\quad \text{neighbours_test}(i,:,2,pcounter)=[m+1,n+1];
\quad \text{neighbours_test}(i,:,3,pcounter)=[m+2,n];
\quad \text{points}(i,:,pcounter)=[m+1,n];
pcounter = pcounter + 1;
\]

\[
\text{end}
\]

\[
\text{if } temp(m,n,3)==1
\]
\[
\quad '3'
\quad \text{bifurcation_points_real}(m+1,n)=1;
\quad \text{temp}(m:m+1,n-1:n+1,3)=0;
\quad \text{neighbours_test}(i,:,1,pcounter)=[m,n];
\quad \text{neighbours_test}(i,:,3,pcounter)=[m+1,n-1];
\quad \text{neighbours_test}(i,:,2,pcounter)=[m+2,n];
\quad \text{points}(i,:,pcounter)=[m+1,n];
pcounter = pcounter + 1;
\]

\[
\text{end}
\]

\[
\text{if } temp(m,n,4)==1
\]
\[
\quad '4'
\quad \text{bifurcation_points_real}(m+1,n)=1;
\quad \text{temp}(m:m+2,n-1:n,4)=0;
\quad \text{neighbours_test}(i,:,2,pcounter)=[m,n];
\quad \text{neighbours_test}(i,:,1,pcounter)=[m+1,n-1];
\quad \text{neighbours_test}(i,:,3,pcounter)=[m+2,n];
\quad \text{points}(i,:,pcounter)=[m+1,n];
pcounter = pcounter + 1;
\]

\[
\text{end}
\]

\[
\text{if } temp(m,n,5)==1
\]
\[
\quad '5'
\quad \text{bifurcation_points_real}(m+1,n)=1;
\quad \text{temp}(m:m+2,n-1:n+1,5)=0;
\quad \text{neighbours_test}(i,:,3,pcounter)=[m,n];
\quad \text{neighbours_test}(i,:,2,pcounter)=[m+1,n+1];
\quad \text{neighbours_test}(i,:,1,pcounter)=[m+2,n];
\quad \text{points}(i,:,pcounter)=[m+1,n];
pcounter = pcounter + 1;
\]

\[
\text{end}
\]

\[
\text{if } temp(m,n,6)==1
\]
\[
\quad '6'
\quad \text{bifurcation_points_real}(m+1,n-1)=1;
\quad \text{temp}(m:m+2,n-2:n,6)=0;
\quad \text{neighbours_test}(i,:,2,pcounter)=[m,n];
\quad \text{neighbours_test}(i,:,1,pcounter)=[m+1,n-2];
\quad \text{neighbours_test}(i,:,3,pcounter)=[m+2,n-1];
\quad \text{points}(i,:,pcounter)=[m+1,n-1];
pcounter = pcounter + 1;
\]

\[
\text{end}
\]

\[
\text{if } temp(m,n,7)==1
\]
\[
\quad '7'
\quad \text{bifurcation_points_real}(m+1,n+1)=1;
\quad \text{temp}(m:m+2,n:n+2,7)=0;
\quad \text{neighbours_test}(i,:,1,pcounter)=[m,n];
\quad \text{neighbours_test}(i,:,2,pcounter)=[m+1,n+2];
\quad \text{neighbours_test}(i,:,3,pcounter)=[m+2,n+1];
\quad \text{points}(i,:,pcounter)=[m+1,n+1];
pcounter = pcounter + 1;
\]

\[
\text{end}
\]

\[
\text{if } temp(m,n,8)==1
\]
\[
\quad '8'
\quad \text{bifurcation_points_real}(m+1,n)=1;
\quad \text{temp}(m:m+2,n-1:n+1,8)=0;
\quad \text{neighbours_test}(i,:,3,pcounter)=[m,n];
\quad \text{neighbours_test}(i,:,1,pcounter)=[m+1,n-1];
\quad \text{neighbours_test}(i,:,2,pcounter)=[m+2,n+1];
\quad \text{points}(i,:,pcounter)=[m+1,n];
\]

\[
\quad \text{72}
\]

\[
\text{72}
\]
pcounter = pcounter + 1;
end
end
end

if pcounter>1

%Skipping those points which do not indicate a blood vessel

copy_neighbours=neighbours_test;
copy_segmented_image=segmented_image;

for i=1:size(neighbours_test,4)
    copy_segmented_image=segmented_image;
    copy_segmented_image(points(1,1,i)-1:points(1,1,i)+1,points(1,2,i)-
    1:points(1,2,i)+1)=0;
    for j=1:size(neighbours_test,3)
        temp=zeros(300);
        counter=0;
        for n=1:30
            if (copy_neighbours(1,1,j,i)<2 || copy_neighbours(1,1,j,i)>298) ||
                (copy_neighbours(1,2,j,i)<2 || copy_neighbours(1,2,j,i)>298)
                break;
            end
            temp(copy_neighbours(1,1,j,i),copy_neighbours(1,2,j,i))=1;
        end
        neighbourhood_y=[copy_neighbours(1,1,j,i)-1:copy_neighbours(1,1,j,i)+1];
        neighbourhood_x=[copy_neighbours(1,2,j,i)-1:copy_neighbours(1,2,j,i)+1];
        copy_segmented_image(copy_neighbours(1,1,j,i),copy_neighbours(1,2,j,i))=0;
        for a=1:3
            for b=1:3
                if copy_segmented_image(neighbourhood_y(a),neighbourhood_x(b))==1
                    copy_neighbours(:,:,j,i)=[neighbourhood_y(a),neighbourhood_x(b)];
                    counter=counter+1;
                end
            end
        end
        if counter < 15
            points(:,:,i)=0;
            neighbours_test(:,:,,:,i)=0;
            for a=1:size(temp,1)
                for b=1:size(temp,2)
                    if temp(a,b)==1
                        segmented_image(a,b)=0;
                    end
                end
            end
        end
    end
end
end
launchEnrollment.m

for IMAGE_NR = 1 : 20
    'initializing'
%
% Defining and initializing constants
MATCHING_DISTANCE_THRESHOLD = 20;
AMOUNT_OF_IMAGES_ENROLLMENT = 5;
AMOUNT_OF_FILTERSETS = 2;
REGISTRATION_IMAGES = {'1', '2', '3', '4', 'm2'};
FILTERSET_NR = 1;
%
% Assembling the file name string and opening the file containing the
% segmented images
filename = '\0';
filename(1:10) = 'segmented_';
filename(11 : (length(num2str(IMAGE_NR)) + 10)) = num2str(IMAGE_NR);
filename((length(filename) + 1) : (length(filename) + 13)) = '_1_m2_new.mat';
load(filename);
'initializing.........done'
'fetching points'
%
% Performing feature extraction on each available image for each filterset
% in use
for i = 1 : AMOUNT_OF_IMAGES_ENROLLMENT
    for j = FILTERSET_NR : FILTERSET_NR
        result(:,:,i,j) = result(i,j,:,i,j);
        enhancedImage(:,:,i,j) = enhancedImage_a(i,j,:,i,j);
        filteredImage(:,:,i,j,:) = filteredImage_a(i,j,:,i,j,:);
        featureExtraction(result, enhancedImage, filteredImage);
        load('bifurcation_points.mat');
        load('difference.mat');
        load('angles.mat');
        load('anglePairs.mat');
        load('anglesAverage.mat');
        angles_difference_array(i,j,1:size(angles_difference,2),1:size(angles_difference,3)) =
        angles_difference(i,:,i,j,:);
        difference_real_array(i,j,1:size(difference_real,2),1:size(difference_real,3)) =
        difference_real(i,:,i,j,:);
        angles_average_array(i,j,1:size(angles_average,2),1:size(angles_average,3)) =
        angles_average(i,:,i,j,:);
        bifurcationPoint_array(i,j,1:size(bifurcation_point,2),1:size(bifurcation_point,3)) =
        bifurcation_point(i,:,i,j,:);
    end
end
'fetching points ........... done'
%
% Rearranging feature data
for i = 1 : AMOUNT_OF_IMAGES_ENROLLMENT
    allDistances(:,:,i) = difference_real_array(i,FILTERSET_NR,:,i);
    anglesAverage(:,:,i) = angles_average_array(i,FILTERSET_NR,:,i);
    bifurcationPoints(:,:,i) = bifurcation_point_array(i,FILTERSET_NR,:,i);
end
'computing candidates'
%
% Comparison between all images used for enrollment
% Setting up the candidate list based on branch orientation angles
allCandidates = zeros(1,5);
for i = 1 : AMOUNT_OF_IMAGES_ENROLLMENT - 1
    angles_difference_1(1,:,:,:) = angles_difference_array(i,FILTERSET_NR,:,:);
    distances_1(1,:,:,:) = difference_real_array(i,FILTERSET_NR,:,:);
    for j = i + 1 : AMOUNT_OF_IMAGES_ENROLLMENT
        distanceCandidates = zeros(1,9);
        angles_difference_2(1,:,:,:) = angles_difference_array(j,FILTERSET_NR,:,:);
        distances_2(1,:,:,:) = difference_real_array(j,FILTERSET_NR,:,:);
        [errorTable, diffMatrix] = compareAngleDeviations(angles_difference_1,
        angles_difference_2);
        candidates_temp = getCandidatePairs(errorTable, bifrucationPoints, i, j, 20);
        if size(candidates_temp, 1) > 1
            aan = rotateAngleTrios(anglesAverage, bifrucationPoints, candidates_temp, i, j);
            candidates = getCandidatePairs_advanced(aan, bifrucationPoints, i, j, 20, 0);
        elseif size(candidates_temp, 1) == 1
            candidates = candidates_temp;
        end
        tmp = getPointMappings(candidates);
        allCandidates(size(allCandidates,1) + 1 : size(allCandidates,1) + size(tmp,1),:) =
        tmp;
    end
end

'computing candidates............done'
'computing matchlist'

%seperate combinations of images yielding two or more common matching
%points from those which have less than two
[sa, pa] = getDistancePairs(allCandidates);
newList = zeros(1,9);
counter2 = 1;

%Calculate distances between bifurcation points which emerge from the
%candidate list based on branch orientation angles.
%Set up a new candidate list based on distances between bifurcation points
for a = 1 : AMOUNT_OF_IMAGES_ENROLLMENT
    for b = 1 : AMOUNT_OF_IMAGES_ENROLLMENT
        counter1 = 0;
        pa_sublist = [0 0 0 0 0];
        for c = 1 : size(pa, 1);
            if pa(c, 1) == a && pa(c, 3) == b
                counter1 = counter1 + 1;
                pa_sublist(counter1,:) = pa(c,:);
            end
        end
        if counter1 > 0
            for i = 1 : (size(pa_sublist, 1) - 1)
                for j = (i + 1) : size(pa_sublist, 1)
                    newList(counter2, 1) = pa_sublist(1, 1);
                    newList(counter2, 2) = pa_sublist(i, 2);
                    newList(counter2, 3) = pa_sublist(j, 2);
                    newList(counter2, 4) = getDistance(pa_sublist(1, 1), allDistances,
                    pa_sublist(i, 2), pa_sublist(j, 2));
                    newList(counter2, 5) = pa_sublist(1, 3);
                    newList(counter2, 6) = pa_sublist(i, 4);
                    newList(counter2, 7) = pa_sublist(j, 4);
                    counter2 = counter2 + 1;
                end
            end
        end
    end
end
newList(counter2, 8) = getDistance(pa_sublist(1, 3), allDistances, pa_sublist(i, 4), pa_sublist(j, 4));
newList(counter2, 9) = abs(newList(counter2, 4) - newList(counter2, 8));
counter2 = counter2 + 1;
end
end
end
end
counter = 1;
matchList = zeros(1,9);

%__________________________________________________________________________
%Apply threshold to inter-bifurcation point candidate list and name it
%"matchList"
for i = 1 : size(newList, 1)
if newList(i,9) < MATCHING_DISTANCE_THRESHOLD
matchList(counter,:) = newList(i,:);
counter = counter + 1;
end
end
'computing matchlist............done'
'computing max point image'
pointArray = ones(5,20);
pointArray = pointArray .* -1;

%__________________________________________________________________________
%Find out which image contains most recognized bifurcation points by
%analyzing the "matchList" array
for i = 1 : AMOUNT_OF_IMAGES_ENROLLMENT
pointCounter = 1;
for j = 1 : size(matchList, 1)
addFlag = 0;
if matchList(j,1) == i
for k = 1 : pointCounter
if pointArray(i,k) == matchList(j,2)
addFlag = -1;
end
end
if addFlag == 0
pointArray(i,pointCounter) = matchList(j,2);
pointCounter = pointCounter + 1;
end
addFlag = 0;
for k = 1 : pointCounter
if pointArray(i,k) == matchList(j,3)
addFlag = -1;
end
end
if addFlag == 0
pointArray(i,pointCounter) = matchList(j,3);
pointCounter = pointCounter + 1;
end
end
addFlag = 0;
if matchList(j,5) == i
for k = 1 : pointCounter
if pointArray(i,k) == matchList(j,6)
addFlag = -1;
end
end
end
addFlag = 0;
if matchList(j,5) == i
for k = 1 : pointCounter
if pointArray(i,k) == matchList(j,6)
addFlag = -1;
end
end
end
end
addFlag = 0;
if matchList(j,5) == i
for k = 1 : pointCounter
if pointArray(i,k) == matchList(j,6)
addFlag = -1;
end
end
end
end
addFlag = 0;
if matchList(j,5) == i
for k = 1 : pointCounter
if pointArray(i,k) == matchList(j,6)
addFlag = -1;
end
end
end
end
addFlag = 0;
if matchList(j,5) == i
for k = 1 : pointCounter
if pointArray(i,k) == matchList(j,6)
addFlag = -1;
end
end
end
end
addFlag = 0;
if matchList(j,5) == i
for k = 1 : pointCounter
if pointArray(i,k) == matchList(j,6)
addFlag = -1;
end
end
end
end
addFlag = 0;
if matchList(j,5) == i
for k = 1 : pointCounter
if pointArray(i,k) == matchList(j,6)
addFlag = -1;
end
end
end
end
if addFlag == 0
    pointArray(i,pointCounter) = matchList(j,6);
    pointCounter = pointCounter + 1;
end
addFlag = 0;
for k = 1 : pointCounter
    if pointArray(i,k) == matchList(j,7)
        addFlag = -1;
    end
end
if addFlag == 0
    pointArray(i,pointCounter) = matchList(j,7);
    pointCounter = pointCounter + 1;
end
end
pointCounter_(i) = pointCounter;
end
maxPointImage = myIndexOf(pointCounter_, max(pointCounter_));
'computing max point image............done'
'attaching single assigned points'

%__________________________________________________________________________
%Find all points contained by at least two images other than the image defined by "maxPointImage
[missingPoints, missingBPointAngles, missingBPointAngles_invariant] =
getMissingPoints(matchList, anglesAverage, anglesDifference, bifrucationPoints, maxPointImage, sa, allDistances);

%__________________________________________________________________________
%Combine all bifurcation points found in "maxPointImage" with all points found by the function "getMissingPoints"
registeredImage_points = bifrucationPoints(maxPointImage,:,:);
registeredBPointAngles = anglesAverage(maxPointImage,:,:);
registeredBPointAngles_invariant = anglesDifference(maxPointImage,:,:);
for i = size(registeredImage_points, 3) + 1 : size(registeredImage_points, 3) +
    size(missingPoints, 1)
    if missingPoints(1,1) ~= 0
        registeredImage_points(1,:,i) = missingPoints(i - size(registeredImage_points, 3),:);
    end
end
for i = size(registeredBPointAngles, 3) + 1 : size(registeredBPointAngles, 3) +
    size(missingBPointAngles, 1)
    if missingBPointAngles(1,1) ~= 0 || missingBPointAngles(1,2) ~= 0
        registeredBPointAngles(1,:,i) = missingBPointAngles(i - size(registeredBPointAngles, 3),:);
    end
end
for i = size(registeredBPointAngles_invariant, 3) + 1 : size(registeredBPointAngles_invariant, 3) +
    size(missingBPointAngles_invariant, 1)
    if missingBPointAngles_invariant(1,1) ~= 0 || missingBPointAngles_invariant(1,2) ~= 0
        registeredBPointAngles_invariant(1,:,i) = missingBPointAngles_invariant(i - size(registeredBPointAngles_invariant, 3),:);
    end
end
'attaching single assigned points..........done'
counter1 = 1;
registeredPoints = 0;
for i = 1:size(matchList,1)
    if matchList(i,1) == maxPointImage
        registeredPoints(counter1) = matchList(i,2);
        counter1 = counter1 + 1;
        registeredPoints(counter1) = matchList(i,3);
        counter1 = counter1 + 1;
    end
    if matchList(i,5) == maxPointImage
        registeredPoints(counter1) = matchList(i,6);
        counter1 = counter1 + 1;
        registeredPoints(counter1) = matchList(i,7);
        counter1 = counter1 + 1;
    end
end

%__________________________________________________________________________
%Open the original blood vessel image defined by "maxPointImage". Note: the
%original image is solely used for illustrative reasons
load ('db.mat');
tImage(IMAGE_NR, :, :) = result(maxPointImage, FILTERSET_NR, :, :);
for i = 1 : length(REGISTRATION_IMAGES)
    ri_ = REGISTRATION_IMAGES(i);
    ri = cell2mat(ri_);
    filename = '\0';
    filename(1) = 'i';
    filename(2 : (length(num2str(IMAGE_NR)) + 1)) = num2str(IMAGE_NR);
    filename(length(filename) + 1) = '_';
    filename((length(filename) + 1) : (length(filename) + length(ri))) = ri;
    filename((length(filename) + 1) : (length(filename) + 4)) = '.mat';
    image = cell2mat(struct2cell(load(filename)));
    if i == maxPointImage
        displayImage = image;
        tOriginalImage(IMAGE_NR, :, :) = image;
    end
end

%__________________________________________________________________________
%Mark all bifurcation points in the image that is to be displayed with
%white crosses
if registeredPoints(1) ~= 0
    registeredPoints = removeDuplicateEntries(registeredPoints')';
    if size(registeredImage_points, 3) > size(bifurcationPoints, 3)
        for i = (size(bifurcationPoints, 3) + 1) : size(registeredImage_points, 3)
            registeredPoints(length(registeredPoints) + 1) = i;
        end
    end
    for i = 1:length(registeredPoints)
        x = registeredImage_points(1,2,registeredPoints(i));
        y = registeredImage_points(1,1,registeredPoints(i));
        if x >= 2 && y >= 2
            displayImage(y - 1, x - 1) = 100;
        end
    end
end
displayImage(y - 1, x + 1) = 100;
end
if x >= 2 && y >= 0
    displayImage(y + 1, x - 1) = 100;
end
if x >= 0 && y >= 0
    displayImage(y + 1, x + 1) = 100;
end
if x >= 1 && y >= 1
    displayImage(y, x) = 100;
end

'enrolling data'
%__________________________________________________________________________
%Writing data to variables that are to be enrolled
for i = 1 : length(registeredPoints)
    x = registeredImage_points(1,2,registeredPoints(i));
    y = registeredImage_points(1,1,registeredPoints(i));
    tBifurcationPoints(IMAGE_NR, 2, i) = x;
    tBifurcationPoints(IMAGE_NR, 1, i) = y;
    tBranchAngles(IMAGE_NR, :, i) = registeredBPointAngles(1, :, registeredPoints(i));
    tBranchAngles_invariant(IMAGE_NR, :, i) = registeredBPointAngles_invariant(1, :, registeredPoints(i));
end
for i = 1 : length(registeredPoints) - 1
    x1 = registeredImage_points(1,2,registeredPoints(i));
    y1 = registeredImage_points(1,1,registeredPoints(i));
    for j = i + 1 : length(registeredPoints)
        x2 = registeredImage_points(1,2,registeredPoints(j));
        y2 = registeredImage_points(1,1,registeredPoints(j));
        tDistances(IMAGE_NR, i, j) = sqrt(((x1-x2)*(x1-x2)) + ((y1-y2)*(y1-y2)));
        [tConnectionAngles(IMAGE_NR, i, j, 1), tConnectionAngles(IMAGE_NR, i, j, 2)] =
        getComparisonAngles(registeredBPointAngles(1, :, :), registeredImage_points(1, :, :),
        registeredPoints(i), registeredPoints(j));
    end
end
save('db.mat', 'tBifurcationPoints', 'tBranchAngles_invariant', 'tBranchAngles',
'tConnectionAngles', 'tDistances', 'tImage', 'tOriginalImage');
'enrolling data.............done'
'preparing displayimage..........done'
figure(1);
imagesc(displayImage);
colormap(gray);
end
clear;
end
'end'

featureExtraction.m

function featureExtraction(segmented_image, enhanced_image, filtered_image)
[y,x]=size(segmented_image);
%__________________________________________________________________________
% defining the structure elements used by the morphological operations

object(1)=strel([0 0 0; 1 0 1; 0 1 0]);
object(2)=strel([0 1 0; 0 0 1; 0 1 0]);
object(3)=strel([0 1 0; 1 0 1; 0 0 0]);
object(4)=strel([0 1 0; 1 0 0; 0 1 0]);
object(5)=strel([0 1 0; 0 1 1; 1 0 0]);
object(6)=strel([0 0 1; 1 1 0; 0 1 0]);
object(7)=strel([1 0 0; 0 1 1; 0 1 0]);
object(8)=strel([0 1 0; 1 1 0; 0 0 1]);

% finding bifurcation points in the image

bifurcation_points = 0;
final = 0;

for i=1:8
    bifurcation_points = imopen(segmented_image, object(i));
    final = max(final, bifurcation_points);
end

bifurcation_points_clean=0;
bifurcation_points_real=zeros(x);

for i=1:8
    bifurcation_points_clean=imopen(final,object(i));
    temp(:,:,i)=bifurcation_points_clean(:,:,i);
end

pcounter = 1;

% finding bifurcation points’ neighbors

for m=1:y
    for n=1:x
        if temp(m,n,1)==1
            bifurcation_points_real(m,n+1)=1;
            neighbours_test(:,:,1,pcounter)=[m,n];
            neighbours_test(:,:,2,pcounter)=[m+1,n+1];
            neighbours_test(:,:,3,pcounter)=[m+1,n+1];
            points(:,:,pcounter)=[m,n+1];
            pcounter = pcounter + 1;
        end
        if temp(m,n,2)==1
            bifurcation_points_real(m+1,n)=1;
            neighbours_test(:,:,1,pcounter)=[m,n];
            neighbours_test(:,:,2,pcounter)=[m+1,n+1];
            neighbours_test(:,:,3,pcounter)=[m+1,n+1];
            points(:,:,pcounter)=[m+1,n];
            pcounter = pcounter + 1;
        end
        if temp(m,n,3)==1
            bifurcation_points_real(m+1,n)=1;
            neighbours_test(:,:,1,pcounter)=[m,n];
            neighbours_test(:,:,2,pcounter)=[m+1,n-1];
            neighbours_test(:,:,3,pcounter)=[m+1,n+1];
            points(:,:,pcounter)=[m+1,n];
            pcounter = pcounter + 1;
        end
        if temp(m,n,4)==1
            bifurcation_points_real(m+1,n)=1;
            neighbours_test(:,:,1,pcounter)=[m,n];
            neighbours_test(:,:,2,pcounter)=[m+1,n-1];
            neighbours_test(:,:,3,pcounter)=[m+1,n+1];
            points(:,:,pcounter)=[m+1,n];
            pcounter = pcounter + 1;
        end
        if temp(m,n,5)==1
            bifurcation_points_real(m+1,n)=1;
            neighbours_test(:,:,1,pcounter)=[m,n];
            neighbours_test(:,:,2,pcounter)=[m+1,n+1];
            neighbours_test(:,:,3,pcounter)=[m+2,n];
            points(:,:,pcounter)=[m+1,n];
            pcounter = pcounter + 1;
        end
    end
end
pcounter = pcounter + 1;
end
if temp(m,n,6)==1
     bifurcation_points_real(m+1,n-1)=1;
     neighbours_test(:,:,1,pcounter)=[m,n];
     neighbours_test(:,:,1,pcounter)=[m+1,n-2];
     neighbours_test(:,:,3,pcounter)=[m+2,n-1];
     points(:,:,pcounter)=[m+1,n-1];
pcounter = pcounter + 1;
end
if temp(m,n,7)==1
     bifurcation_points_real(m+1,n+1)=1;
     neighbours_test(:,:,1,pcounter)=[m,n];
     neighbours_test(:,:,1,pcounter)=[m+1,n+2];
     neighbours_test(:,:,3,pcounter)=[m+2,n+1];
     points(:,:,pcounter)=[m+1,n+1];
pcounter = pcounter + 1;
end
if temp(m,n,8)==1
     bifurcation_points_real(m+1,n)=1;
     neighbours_test(:,:,3,pcounter)=[m,n];
     neighbours_test(:,:,1,pcounter)=[m+1,n-1];
     neighbours_test(:,:,2,pcounter)=[m+2,n+1];
     points(:,:,pcounter)=[m+1,n];
pcounter = pcounter + 1;
end
%__________________________________________________________________________%
% Skipping those points which do not indicate a blood vessel

% copy_segmented_image=segmented_image;
for i=1:size(points,3)
    neighbourhood_y=[points(1,1,i)-1:points(1,1,i)+1];
    neighbourhood_x=[points(1,2,i)-1:points(1,2,i)+1];
    counter=0;
    copy_segmented_image(points(1,1,i),points(1,2,i))=0;
    for a=1:3
        for b=1:3
            if copy_segmented_image(neighbourhood_y(a),neighbourhood_x(b))==1 && counter < 3
                counter=counter+1;
                neighbours(:,:,counter,i)=[neighbourhood_y(a),neighbourhood_x(b)];
            end
        end
    end
    copy_segmented_image(points(1,1,i),points(1,2,i))=1;
end

copy_neighbours=neighbours_test;
exit=0;
copy_segmented_image=segmented_image;
temp=zeros(300);
for i=1:size(neighbours_test,4)
    copy_segmented_image=segmented_image;
    for j=1:size(neighbours_test,3)
        counter=0;
        for n=1:20
            neighbourhood_y=[copy_neighbours(1,1,j,i)-1:copy_neighbours(1,1,j,i)+1];
            neighbourhood_x=[copy_neighbours(1,2,j,i)-1:copy_neighbours(1,2,j,i)+1];
            copy_segmented_image(copy_neighbours(1,1,j,i),copy_neighbours(1,2,j,i))=0;
        end
    end
end
Appendix A

verification=0;
for a=1:3
    for b=1:3
        if copy_segmented_image(neighbourhood_y(a), neighbourhood_x(b))==1
            copy_neighbours(:,:,j,i)=[neighbourhood_y(a), neighbourhood_x(b)];
            temp(neighbourhood_y(a), neighbourhood_x(b))=1;
            counter=counter+1;
        end
    end
end
if counter<6
    points(:,:,i)=0;
    neighbours_test(:,:,i)=0;
end
end

% Extracting inter-bifurcation point distances
counter=0;
 bifurcation_point = zeros(1,2,1);
for i=1:size(points,3)
    if points(1,1,i)~=0
        counter=counter+1;
        bifurcation_point(:,:,counter)=[points(1,1,i), points(1,2,i)];
    end
end
for i=1:size(bifurcation_point,3)
    for n=1:size(bifurcation_point,3)
        diff_y=abs(bifurcation_point(1,1,i)-bifurcation_point(1,1,n));
        diff_x=abs(bifurcation_point(1,2,i)-bifurcation_point(1,2,n));
        distance(i,n)=sqrt(diff_y^2+diff_x^2);
    end
end

% computing branch orientation angles
angles_average = zeros(1,3,1);
if bifurcation_point(1,1,1) ~= 0
    counter=0;
    for i=1:size(neighbours_test,4)
        if neighbours_test(:,:,i)~=0
            counter=counter+1;
            bifurcation_neighbours(:,:,counter)=neighbours_test(:,:,i);
        end
    end
    filtered_image_real(:,:,i)=filtered_image(:,:,i);
end
for i=1:size(bifurcation_neighbours,4)
    copy_segmented_image=segmented_image;
    copy_segmented_image(bifurcation_neighbours(1,1,i)-1:bifurcation_neighbours(1,1,i)+1, bifurcation_neighbours(1,2,i)-1:bifurcation_neighbours(1,2,i)+1)=0;
    for j=1:size(bifurcation_neighbours,3)
        counter=0;
        verification=0;
        angles_sum=0;
        for n=1:30
            % your code here
        end
    end
end
Appendix A

neighbourhood_y = [copy_neighbours(1,1,j,i)-1:copy_neighbours(1,1,j,i)+1];
neighbourhood_x = [copy_neighbours(1,2,j,i)-1:copy_neighbours(1,2,j,i)+1];

for c=0:15
    if enhanced_image(copy_neighbours(1,1,j,i),copy_neighbours(1,2,j,i))==filtered_image_real(c+1,copy_neighbours(1,1,j,i),copy_neighbours(1,2,j,i))
        counter=counter+1;
        angles(:,counter,j,i)=(180/16)*c;
    end
end

for a=1:3
    for b=1:3
        if copy_segmented_image(neighbourhood_y(a),neighbourhood_x(b))==1
            verification=1;
        end
    end
    if verification==0
        break;
    end
end

angles_average(:,j,i)=angles_sum/counter;
end
end

% reshaping the orientation information angle array
for i=1:size(angles_average,3)
    max_angle=max(angles_average(:,i,:));
    for j=1:size(angles_average,2)
        angles_difference(1,j,i)=abs(max_angle-angles_average(1,j,i));
    end
end

temp=1;
counter=1;
difference_real = 0;
for i=1:size(distance,2)-1
    temp=temp+1;
    for n=temp:size(distance,2)
        difference_real(:,i,n)=distance(i,n);
    end
end

% writing the feature data to files
save ('bifurcation_points.mat', 'bifurcation_point');
save ('difference.mat', 'difference_real');
save ('angles.mat', 'angles_difference');
save ('anglesAverage.mat', 'angles_average');
**compareAngleDeviations.m**

```matlab
function [totalError, deviations] = compareAngleDeviations(list1, list2)
% This file is in charge of computing differences between sets of branch orientation angles. The variables list1 and list2 contain lists of branch orientation angle triplets. Each triplet contained by list1 is compared to each triplet contained by list2.

for k = 1 : 3
    for i = 1 : size(list1,3)
        for j = 1 : size(list2,3)
            deviations(j,i,k) = abs(list2(1,k,j) - list1(1,k,i));
        end
    end
end

for i = 1 : size(list1,3)
    for j = 1 : size(list2,3)
        totalError(j,i) = ((deviations(j,i,1) + deviations(j,i,2) + deviations(j,i,3)) / 3);
    end
end
```

**getCandidatePairs.m**

```matlab
function retVal = getCandidatePairs(errorMatrix, bifrucationPoints, columnTable, rowTable, THRESHOLD)
% This file compares all possible combinations of orientation angle triplets between two images. The list is filtered by applying a threshold to the column containing the differences between the angle triplets.

[y, x] = size(errorMatrix);
c = 1;
retVal = zeros(1,5);

for i = 1 : x
    for k = 1 : y
        if (errorMatrix(k,i) < THRESHOLD)
            if bifrucationPoints(columnTable, 1, i) ~= 0 && bifrucationPoints(rowTable, 1, k) ~= 0
                retVal(c,1) = columnTable;
                retVal(c,2) = i;
                retVal(c,3) = rowTable;
                retVal(c,4) = k;
                retVal(c,5) = errorMatrix(k,i);
                c = c + 1;
            end
        end
    end
end
```

**compareDistances_.m**

```matlab
function finalList = compareDistances_(image_1, image_2, distances_1, distances_2, THRESHOLD)
% In Analogy to the file compareAngleDeviations.m this file returns a list that displays the distances between all bifurcation points contained by two images. Additionally this function filters the list by applying a threshold to the column which contains the differences between the distances of two pairs of bifurcation points.

counter1 = 1;
counter2 = 1;
finalList = zeros(1,9);

for i = 2 : size(distances_1, 3)
    if distances_1(1,1,i) ~= 0
        counter1 = i;
    end
end
```
for i = 2 : size(distances_2, 3)
    if distances_2(1,1,i) ~= 0
        counter2 = i;
    end
end

counter3 = 1;
counter4 = 1;

distanceList1 = zeros(1, 4);
distanceList2 = zeros(1, 4);

for i = 1 : counter1 - 1
    for j = i + 1 : counter1
        distanceList1(counter3, 1) = image_1;
        distanceList1(counter3, 2) = i;
        distanceList1(counter3, 3) = j;
        distanceList1(counter3, 4) = distances_1(1,i,j);
        counter3 = counter3 + 1;
    end
end

for i = 1 : counter2 - 1
    for j = i + 1 : counter2
        distanceList2(counter4, 1) = image_2;
        distanceList2(counter4, 2) = i;
        distanceList2(counter4, 3) = j;
        distanceList2(counter4, 4) = distances_2(1,i,j);
        counter4 = counter4 + 1;
    end
end

counter5 = 1;

for i = 1 : size(distanceList1, 1)
    for j = 1 : size(distanceList2, 1)
        diff = abs(distanceList1(i, 4) - distanceList2(j, 4));
        if diff < THRESHOLD
            finalList(counter5, 1) = distanceList1(i, 1);
            finalList(counter5, 2) = distanceList1(i, 2);
            finalList(counter5, 3) = distanceList1(i, 3);
            finalList(counter5, 4) = distanceList1(i, 4);
            finalList(counter5, 5) = distanceList2(j, 1);
            finalList(counter5, 6) = distanceList2(j, 2);
            finalList(counter5, 7) = distanceList2(j, 3);
            finalList(counter5, 8) = distanceList2(j, 4);
            finalList(counter5, 9) = diff;
            counter5 = counter5 + 1;
        end
    end
end

rotateAngleTrios.m

function anglesAverage_new = rotateAngleTrios(anglesAverage, bifrucationPoints, candidates, image1, image2)
%__________________________________________________________________________
%This file aligns image1 and image2 and returns the modified branch
%orientation angle array
%while this file deals with rotation variant branch orientation angles the
%candidate list "candidates" refers to rotation invariant branch
%orientation angles
%__________________________________________________________________________
%
%compensation for precision errors : among all candidate list entries the
%entry with the lowest angle deviation is found and used for calculating
%reference rotation angle.
cs = sortrows(candidates, 5);
refCandidate = cs(1,:);
refRotation = refCandidate(5)/2;
When computing the candidate list for the rotation variant branch orientation angles all angles are rotated by the reference rotation angle in order to compensate for precision errors. In order to compensate for the rotation the tolerance in angle deviation is set to 40° average deviation.

cps = getCandidatePairs_advanced(anglesAverage, bifurcationPoints, image1, image2, 40, refRotation);
cps_minus = getCandidatePairs_advanced(anglesAverage, bifurcationPoints, image1, image2, 40, -refRotation);

Based on cps the resulting bifurcation points in each image are grouped to form all possible combinations of inter-bifurcation point connections.

newList = zeros(1,9);
counter2 = 1;
for a = 1 : 21
    for b = 1 : 21
        counter1 = 0;
        pa_sublist = [0 0 0 0 0];
        for c = 1 : size(pa, 1);
            if pa(c, 1) == a && pa(c, 3) == b
                counter1 = counter1 + 1;
                pa_sublist(counter1,:) = pa(c,:);
            end
        end
        if counter1 > 0
            for i = 1 : (size(pa_sublist, 1) - 1)
                for j = (i + 1) : size(pa_sublist, 1)
                    newList(counter2, 1) = pa_sublist(1, 1); %1st image
                    newList(counter2, 2) = pa_sublist(i, 2); %1st bifurcation point in 1st image
                    newList(counter2, 3) = pa_sublist(j, 2); %2nd bifurcation point in 1st image
                    newList(counter2, 4) = pa_sublist(i, 5); %distance between 1st and 2nd point in 1st image
                    newList(counter2, 5) = pa_sublist(1, 3); %2nd image
                    newList(counter2, 6) = pa_sublist(i, 4); %1st bifurcation point in 2nd image
                    newList(counter2, 7) = pa_sublist(j, 4); %2nd bifurcation point in 2nd image
                    newList(counter2, 8) = pa_sublist(j, 5); %distance between 1st and 2nd point in 2nd image
                    newList(counter2, 9) = newList(counter2, 4) + newList(counter2, 8);
                    counter2 = counter2 + 1;
                end
            end
        end
    end
end

Based on cps_minus the resulting bifurcation points in each image are grouped to form all possible combinations of inter-bifurcation point connections.

newList_minus = zeros(1,9);
counter2 = 1;

for a = 1 : 21
    for b = 1 : 21
        counter1 = 0;
        pa_sublist = [0 0 0 0 0];
        for c = 1 : size(pa, 1);
            if pa(c, 1) == a && pa(c, 3) == b
                counter1 = counter1 + 1;
                pa_sublist(counter1,:) = pa(c,:);
            end
        end
        if counter1 > 0
            for i = 1 : (size(pa_sublist, 1) - 1)
                for j = (i + 1) : size(pa_sublist, 1)
                    newList_minus(counter2,1) = pa_sublist(1,1); %1st image
                    newList_minus(counter2,2) = pa_sublist(i,2); %1st bifurcation point in
                    newList_minus(counter2,3) = pa_sublist(j,2); %2nd bifurcation point in
                    newList_minus(counter2,4) = pa_sublist(i,5); %distance between 1st and
                    newList_minus(counter2,5) = pa_sublist(1,3); %2nd image
                    newList_minus(counter2,6) = pa_sublist(i,4); %1st bifurcation point in
                    newList_minus(counter2,7) = pa_sublist(j,4); %2nd bifurcation point in
                    newList_minus(counter2,8) = pa_sublist(j,5); %distance between 1st and
                    newList_minus(counter2,9) = newList_minus(counter2,4) +
                    newList_minus(counter2,8);
                end
                counter2 = counter2 + 1;
            end
        end
    end
end

%__________________________________________________________________________
%Based on newList and newList_minus find the pair of bifurcation points
%which has the best matching inter-bifurcation point connection angles in
%each image.
cps_ = verifyInterPointConnectionOrientation(newList, anglesAverage, bifrucationPoints);
cps_ = sortrows(cps_, 9);
cps_minus_ = verifyInterPointConnectionOrientation(newList_minus, anglesAverage, bifrucationPoints);
cps_minus_ = sortrows(cps_minus_, 9);
if cps_(1, 9) <= cps_minus_(1, 9)
    newRefCandidate1 = cps_(1,:);
else
    newRefCandidate1 = cps_minus_(1,:);
end

%__________________________________________________________________________
%Based on the best matching pairs of bifurcation points compute rotation
%and translation parameters between 2 images
if newRefCandidate1(1) == 0
    x_fP_1 = bifrucationPoints(newRefCandidate1(1), 2, newRefCandidate1(2));
    y_fP_1 = bifrucationPoints(newRefCandidate1(1), 1, newRefCandidate1(2));
    x_dP_1 = bifrucationPoints(newRefCandidate1(1), 2, newRefCandidate1(3));
Appendix A

\[ y_{dP_1} = bifrucationPoints(newRefCandidate1(1), 1, newRefCandidate1(3)) \]
\[ x_{fP_2} = bifrucationPoints(newRefCandidate1(5), 2, newRefCandidate1(6)) \]
\[ y_{fP_2} = bifrucationPoints(newRefCandidate1(5), 1, newRefCandidate1(6)) \]
\[ x_{dP_2} = bifrucationPoints(newRefCandidate1(5), 2, newRefCandidate1(7)) \]
\[ y_{dP_2} = bifrucationPoints(newRefCandidate1(5), 1, newRefCandidate1(7)) \]

\[ [\text{translationMatrix}, \text{refPoint}] = \text{getTranslationMatrix}(x_{fP_1}, y_{fP_1}, x_{dP_1}, y_{dP_1}, \]
\[ x_{fP_2}, y_{fP_2}, x_{dP_2}, y_{dP_2}) \]

\text{else}

\text{translationMatrix} = [1 0 0; 0 1 0; 0 0 1];
\text{end}

%__________________________________________________________________________
%Compute the new angle triplets

\text{angle} = \text{translationMatrix}(1,1);
\text{angleOffset} = -\text{rad2deg}(\text{acos}(\text{angle}));

\text{anglesAverage_new} = \text{anglesAverage};
\text{direction} = \text{getRotationDirection_angleTranslation}(\text{anglesAverage}, \text{bifrucationPoints}, \text{image1},
\text{image2}, \text{angleOffset});

\text{for } i = 1 : \text{size}(\text{anglesAverage}, 3)

\text{if } \text{anglesAverage}(\text{image2}, 1, i) == 0
\text{break};
\text{end}

\text{for } j = 1 : 3
\text{anglesAverage_new}(\text{image2}, j, i) = \text{anglesAverage}(\text{image2}, j, i) + (\text{angleOffset} \ast \text{direction});
\text{end}
\text{end}

\text{rearrangeAngleTrios.m}

\text{function } \text{returnAngleTrio = rearrangeAngleTrios(\text{angleTrio}, \text{angleOffset})}

%__________________________________________________________________________
%This file rearanges angle triplets in a way that the smallest angle value
%becomes the first. Additionally, an offset can be applied

\text{initAngle} = -180;
\text{counter} = 1;

\text{for } i = \text{initAngle} : 180 - \text{initAngle}
\text{for } j = 1 : 3
\text{if } \text{angleTrio}(j) > i && \text{angleTrio}(j) <= i + 1
\text{returnAngleTrio}(\text{counter}) = \text{angleTrio}(j) + \text{angleOffset};
\text{counter} = \text{counter} + 1;
\text{end}
\text{end}
\text{end}

\text{getCandidatePairs_advanced.m}

\text{function } \text{cps = getCandidatePairs_advanced(\text{anglesAverage}, \text{bifrucationPoints}, \text{image1}, \text{image2},
\text{tolerance}, \text{angleOffset})}

%__________________________________________________________________________
%This file provides the functionality of getCandidatePairs extended by the
%possibility to add an angle offset

\text{for } i = 1 : \text{size}(\text{anglesAverage}, 3)
\text{if } \text{anglesAverage}(\text{image1}, 1, i) == 0
\text{break};
\text{end}

88
Appendix A

\[
\begin{align*}
\text{atl}(1,:) &= \text{rearrangeAngleTrios}(\text{anglesAverage}(\text{image1},:,:)\text{,}0); \\
\text{end} \\
\text{for } i = 1 : \text{size}(\text{anglesAverage},3) \\
& \quad \text{if } \text{anglesAverage}(\text{image2},1,i) == 0 \\
& \quad \quad \text{break}; \\
& \quad \text{end} \\
\text{at2}(1,:) &= \text{rearrangeAngleTrios}(\text{anglesAverage}(\text{image2},:,:)\text{,} \text{angleOffset}); \\
\text{end} \\
[\text{errorTable, diffMatrix}] &= \text{compareAngleDeviations}(\text{at1}, \text{at2}); \\
\text{candidates} &= \text{getCandidatePairs}(\text{errorTable}, \text{bifurcationPoints}, \text{image1}, \text{image2}, \text{tolerance}); \\
\text{cps} &= \text{candidates}; \\
\end{align*}
\]

\textbf{getDistancePairs.m}

\begin{verbatim}
function \[\text{singleArray, probeArray}\] = getDistancePairs(\text{allCandidates})

%__________________________________________________________________________
%In order to align two images, both images that are to be aligned must have
%two points each in common. This file seperates the combinations of images
%which have one point only in common from those which have two or more in
%common. Latterly mentioned are stored in the variable "probeArray".

\text{probeArrayCounter} = 1;
\text{singleArrayCounter} = 1;
\text{singleArray} = zeros(1,5);
\text{probeArray} = zeros(1,5);

\text{for } i = 1 : 21
\text{for } j = 1 : 21
\text{localCounter} = 0;
\text{for } k = 1 : \text{size}(\text{allCandidates},1)
\text{if } (\text{allCandidates}(k,1) == i && \text{allCandidates}(k,3) == j) \text{ || } (\text{allCandidates}(k,1) == j && \text{allCandidates}(k,3) == i)
\text{localCounter} = \text{localCounter} + 1;
\text{end}
\text{end}
\text{if } \text{localCounter} == 1
\text{for } k = 1 : \text{size}(\text{allCandidates},1)
\text{if } \text{allCandidates}(k,1) == i && \text{allCandidates}(k,3) == j
\text{singleArray(\text{singleArrayCounter},:) = allCandidates(k,:)};
\text{singleArrayCounter} = \text{singleArrayCounter} + 1;
\text{end}
\text{end}
\text{elseif } \text{localCounter} == 2
\text{for } k = 1 : \text{size}(\text{allCandidates},1)
\text{if } \text{allCandidates}(k,1) == i && \text{allCandidates}(k,3) == j
\text{probeArray(\text{probeArrayCounter},:) = allCandidates(k,:)};
\text{probeArrayCounter} = \text{probeArrayCounter} + 1;
\text{end}
\text{end}
\end{verbatim}

\textbf{verifyInterPointConnectionOrientation.m}

\begin{verbatim}
function matchList_va = verifyInterPointConnectionOrientation(matchList, anglesAverage, 
bifurcationPoints)

%__________________________________________________________________________
%This file compares each pair of inter-bifurcation point connection angles
%with all other pairs of inter-bifurcation point connection angles

counter = 1;
matchList_va = zeros(1,9);
\end{verbatim}
for i = 1 : size(matchList, 1)
    [alfa_1_1, alfa_2_1] = getComparisonAngles(anglesAverage(matchList(i,1), :, :),
        bifrucationPoints(matchList(i,1), :, :), matchList(i,2), matchList(i,3));
    [alfa_1_2, alfa_2_2] = getComparisonAngles(anglesAverage(matchList(i,5), :, :),
        bifrucationPoints(matchList(i,5), :, :), matchList(i,6), matchList(i,7));
    score1 = computeScore(alfa_1_2, alfa_1_1, 20, 120);
    score2 = computeScore(alfa_2_2, alfa_2_1, 20, 120);
    if score1 >= 0.8 && score2 >= 0.8
        matchList_va(counter,:) = matchList(i,:);
        counter = counter + 1;
    end
end

getComparisonAngles.m

function [alfa_1, alfa_2] = getComparisonAngles(angles_average, bifrucation_points, point1,
    point2)
    %__________________________________________________________________________
    %This file computes the inter-bifurcation point connection angles
    x_diff = bifrucation_points(1,2,point2) - bifrucation_points(1,2,point1);
    y_diff = bifrucation_points(1,1,point2) - bifrucation_points(1,1,point1);
    if y_diff == 0
        y_diff = 0.0001;
    end
    if x_diff == 0
        x_diff = 0.0001;
    end
    alfa_1 = angles_average(1,2,point1) - 90 + (90 - ((atan(x_diff / y_diff) / pi) * 180));
    alfa_2 = 180 - [(angles_average(1,2,point2) - 90) + ((atan(y_diff / x_diff) / pi) * 180)];
    if alfa_1 > 180
        alfa_1 = (alfa_1 - 180) * 1;
    end
    if alfa_2 > 180
        alfa_2 = (alfa_2 - 180) * 1;
    end

computeScore.m

function score = computeScore(value, mue, sigma, amountOfXvals)
    %__________________________________________________________________________
    %This file sets up a Gauss curve and computes a score based on the
    %location of the input value on the Gauss curve
    range = (mue - ceil(amountOfXvals / 2)) : (mue + ceil(amountOfXvals / 2));
    gaussDist = gaussWithOffset(range, sigma, mue);
    gaussDist = gaussDist / max(gaussDist);
    score = 0;
    for k = 1 : length(gaussDist) - 1
        if value > range(k) && value <= range(k + 1)
            score = gaussDist(k);
        end
    end

getTranslationMatrix.m

function [translationMatrix, refPoint] = getTranslationMatrix(x_fp_1, y_fp_1, x_dP_1, y_dP_1,
    x_fp_2, y_fp_2, x_dP_2, y_dP_2)

90
%This file computes translation matrixes for registration between two images. 4 points, 2 per image must be given. The output is a 3x3 translation matrix and a reference point

\[
\begin{align*}
\text{x\_trans} &= \text{x\_fP\_1} - \text{x\_fP\_2}; \\
\text{y\_trans} &= \text{y\_fP\_1} - \text{y\_fP\_2}; \\
\text{x\_fP\_2\_new} &= \text{x\_fP\_1}; \\
\text{y\_fP\_2\_new} &= \text{y\_fP\_1}; \\
\text{x\_dP\_2\_new} &= \text{x\_dP\_2} + \text{x\_trans}; \\
\text{y\_dP\_2\_new} &= \text{y\_dP\_2} + \text{y\_trans}; \\
\text{distance} &= \text{complex}(\text{x\_dP\_1} - \text{x\_fP\_1}, \text{y\_dP\_1} - \text{y\_fP\_1}); \\
\text{distance\_} &= \text{complex}(\text{x\_dP\_2\_new} - \text{x\_fP\_2\_new}, \text{y\_dP\_2\_new} - \text{y\_fP\_2\_new}); \\
\text{rotationAngle} &= \text{angle}(\text{distance}) - \text{angle}(\text{distance\_}); \\
\text{translationMatrix} &= \begin{bmatrix}
\cos(\text{rotationAngle}) & \sin(\text{rotationAngle}) & 0 \\
-\sin(\text{rotationAngle}) & \cos(\text{rotationAngle}) & 0 \\
\text{x\_trans} & \text{y\_trans} & 1
\end{bmatrix}; \\
\text{refPoint} &= [\text{x\_fP\_2}, \text{y\_fP\_2}, 1];
\end{align*}
\]

\textbf{getRotationDirection\_angleTranslation.m}

\begin{verbatim}
function direction = getRotationDirection\_angleTranslation(anglesAverage, bifrucationPoints, image1, image2, angleOffset)
%This file returns the direction in which the image has to be rotated for proper alignment

anglesAverage\_minus = anglesAverage; 
anglesAverage\_plus = anglesAverage; 
for i = 1 : size(anglesAverage, 3) 
    if anglesAverage(image2, 1, i) == 0
        break;
    end 
    for j = 1 : 3
        anglesAverage\_minus(image2,j,i) = anglesAverage(image2,j,i) - angleOffset;
    end 
end 
for i = 1 : size(anglesAverage, 3) 
    if anglesAverage(image2, 1, i) == 0
        break;
    end 
    for j = 1 : 3
        anglesAverage\_plus(image2,j,i) = anglesAverage(image2,j,i) + angleOffset;
    end 
end 
cl\_minus = getCandidatePairs\_advanced(anglesAverage\_minus, bifrucationPoints, image1, image2, 20, 0); 
cl\_plus = getCandidatePairs\_advanced(anglesAverage\_plus, bifrucationPoints, image1, image2, 20, 0); 
sum\_minus = 0; 
sum\_plus = 0; 
for i = 1 : size(cl\_minus, 1)
    sum\_minus(i) = cl\_minus(i, 5);
end 
for i = 1 : size(cl\_plus, 1)
    sum\_plus(i) = cl\_plus(i, 5);
end 
\end{verbatim}
if min(sum_plus) >= min(sum_minus)
    direction = -1;
else
    direction = 1;
end

getPointMappings.m

function retVal = getPointMappings(candidates)
%__________________________________________________________________________
%In the candidate list which is passed to this function by the variable
"candidates" contains multiple point mapping assignments for each point.
%This file is in charge of finding the most suitable point mapping
%assignments and thereby eliminating dual assignments. The algorithm is
%explained in Chapter 3.4.4.1
quitFlag = -1;
pointCounter1 = -1;
pointCounter2 = -1;
pointCounter3 = -1;
initPoint = -1;
marker = 1;
while marker < size(candidates, 1)
    if initPoint == candidates(marker,2)
        marker = marker + 1;
    end
pointCounter1 = marker;
pointCounter2 = marker;
initPoint = candidates(pointCounter1,2);
while candidates(pointCounter1,2) ~= initPoint
    pointCounter1 = pointCounter1 + 1;
end
pointCounter2 = pointCounter1;
while candidates(pointCounter2,2) == initPoint && pointCounter2 <= size(candidates, 1)
    if pointCounter2 > size(candidates, 1)
        break;
    end
end
%______________________________________________________________________
%The list "newCandidates" solely contains entries regarding the point
%with the number "initpoint". The list is sorted by the 5th column
%which contains data regarding the average angle differences.
newCandidates = candidates(pointCounter1:pointCounter2-1,:);
newCandidates_sorted = sortrows(newCandidates,5);
%______________________________________________________________________
%The temporary assignment for the point defined by "initpoint"
winner1 = newCandidates_sorted(1,:);
[height,len] = size(newCandidates_sorted);
%______________________________________________________________________
%Besides "winner1" all other entries regarding the point defined by
 "%initpoint" are deleted from the list defined by "candidates"
if height > 1
    for i = 2 : height
        candidates = deleteEntry(candidates,newCandidates_sorted(i,:));
    end
else
    quitFlag = 0;
end
%______________________________________________________________________
%Sort candidate list by the 4th column
candidates_ = sortrows(candidates,4);
pointCounter1 = 1;
\% "winner1" assigns a one particular point to "initpoint". Now all other
\% entries are found regarding the point assigned to "initpoint"
while candidates_(pointCounter1,4) ~= winner1(4)
  pointCounter1 = pointCounter1 + 1;
end

pointCounter2 = pointCounter1;
while candidates_(pointCounter2,4) == winner1(4) && pointCounter2 <= size(candidates_, 1)
  if pointCounter2 > size(candidates_, 1)
    break;
  end
end
\%
\% Again "newCandidates" is used to collect all entries regarding one
\% particular point. This time it is the point assigned to "initpoint"
\% rather than "initpoint"
newCandidates = candidates_(pointCounter1:pointCounter2-1,:);
newCandidates_sorted = sortrows(newCandidates,5);
winner1 = newCandidates_sorted(1,:);
[height,len] = size(newCandidates_sorted);
\%
\% Analogous to the previous proceeding the entry containing the lowes
\% average angle difference is kept and all others regarding the same
\% point are deleted
if height > 1
  for i = 2 : height
    candidates = deleteEntry(candidates,newCandidates_sorted(i,:));
  end
else
  quitFlag = 0;
end
\%
\% The algorithm starts over again without the entries that had been
\% deleted
end
retVal = candidates;

deleteEntry.m

function table = deleteEntry(table_i, record)
\%
\% This file is in charge of deleting an entry defined by "record" from an
\% array defined by "table_i"
[y,x] = size(table_i);
entry = -1;
for i = 1 : y
  equalityFlag = 1;
  for k = 1 : x
    if table_i(i,k) == record(k)
      equalityFlag = equalityFlag * 1;
    else
      equalityFlag = equalityFlag * 0;
    end
  end
  if equalityFlag == 1
    entry = i;
  end
end
Appendix A

getDistance.m

function distance = getDistance(image, allDistances, point1, point2)
%__________________________________________________________________________
%This file returns the distance between two bifurcation points regardless
%of whether the order of the two points entered as parameters is correct
[x,y,z] = size(allDistances);
if point1 ~= point2
    if point1 > y
distance = allDistances(image, point2, point1);
    elseif point2 > y
distance = allDistances(image, point1, point2);
    elseif point1 <= y && point2 <= y
        if allDistances(image, point1, point2) == 0
            distance = allDistances(image, point2, point1);
        else
            distance = allDistances(image, point1, point2);
        end
    end
else
    distance = 0;
end

myIndexOf.m

function retVal = myIndexOf(array, value)
%__________________________________________________________________________
%This function searches for the value "value" in the array "array" and
%returns the index
retVal = -1;
for i = 1 : length(array)
    if array(i) == value
        retVal = i;
    end
end

getMissingPoints.m

function [missingPoints, missingBPointAngles, missingBPointAngles_invariant] =
getMissingPoints(matchList, anglesAverage, anglesDifference, bifrucationPoints, maxPointImage,
sa, allDistances)
%__________________________________________________________________________
%This file is in charge of transforming all bifurcation points that are at
%least contained by two images though not by "maxPointImage" to
"maxPointImage". This file returns the cartesian coordinates, the rotation
%variant branch orientation angles and the rotation invariant branch
%orientation angles
pointMap = getMaxPointImageMissingPoints(matchList, anglesAverage, bifrucationPoints, 
maxPointImage, sa, allDistances);
if pointMap(1,1,1) ~= 0
counter = 1;
for i = 1 : size(pointMap, 1)
    for j = 1 : size(pointMap, 2)
        if pointMap(i,1,j) ~= 0
            imageArray(counter) = pointMap(i,1,j);
            counter = counter + 1;
        end
    end
    matchList_va = verifyInterPointConnectionOrientation(matchList, anglesAverage, bifrucationPoints);
    info = findBestMatchingPairScore(matchList_va, maxPointImage, imageArray);
    if info(1,1) ~= 0
        if info(1) == maxPointImage
            bestImage = info(5);
        else
            bestImage = info(1);
        end
        for j = 1 : size(pointMap, 2)
            if pointMap(i,1,j) == bestImage
                bestImagePoint = pointMap(i,2,j);
            end
        end
        if info(1) == maxPointImage
            x_fP_1 = bifrucationPoints(info(1), 2, info(2));
            y_fP_1 = bifrucationPoints(info(1), 1, info(2));
            x_dP_1 = bifrucationPoints(info(1), 2, info(3));
            y_dP_1 = bifrucationPoints(info(1), 1, info(3));
            x_fP_2 = bifrucationPoints(info(5), 2, info(6));
            y_fP_2 = bifrucationPoints(info(5), 1, info(6));
            x_dP_2 = bifrucationPoints(info(5), 2, info(7));
            y_dP_2 = bifrucationPoints(info(5), 1, info(7));
            otherImage = info(5);
        elseif info(5) == maxPointImage
            x_fP_2 = bifrucationPoints(info(1), 2, info(2));
            y_fP_2 = bifrucationPoints(info(1), 1, info(2));
            x_dP_2 = bifrucationPoints(info(1), 2, info(3));
            y_dP_2 = bifrucationPoints(info(1), 1, info(3));
            x_fP_1 = bifrucationPoints(info(5), 2, info(6));
            y_fP_1 = bifrucationPoints(info(5), 1, info(6));
            x_dP_1 = bifrucationPoints(info(5), 2, info(7));
            y_dP_1 = bifrucationPoints(info(5), 1, info(7));
            otherImage = info(1);
        end
        [translationMatrix, refPoint] = getTranslationMatrix(x_fP_1, y_fP_1, x_dP_1, y_dP_1, x_fP_2, y_fP_2, x_dP_2, y_dP_2);
        [x_newPt(i), y_newPt(i)] = transformPoint(translationMatrix, bifrucationPoints(otherImage), bifrucationPoints(otherImage,1,bestImagePoint), refPoint(1), refPoint(2));
    else
        x_newPt(i) = 0;
        y_newPt(i) = 0;
        otherImage = 1;
        bestImagePoint = 1;
    end
Appendix A

missingPoints = zeros(i, 2);
missingBPointAngles = zeros(i, 3);
for j = 1 : i
missingPoints(j,2) = round(x_newPt(j));
missingPoints(j,1) = round(y_newPt(j));
missingBPointAngles(j,:) = anglesAverage(otherImage, :, bestImagePoint);
missingBPointAngles_invariant(j,:) = anglesDifference(otherImage, :, 
bestImagePoint);
end
end
else
missingPoints = [0 0];
missingBPointAngles = [0 0 0];
missingBPointAngles_invariant = [0 0 0];
end

findBestMatchingPairScore.m

function info = findBestMatchingPairScore(matchList, maxPointImage, imageArray)
%__________________________________________________________________________
%This file finds among the pairs of bifurcation points contained by
"matchList" the most appropriate pair for calculation of translation
%parameters. The length of the distances and the differences between the
%distances are considered

counter = 1;
subMatchList = zeros(1,9);
for j = 1 : length(imageArray)
    for i = 1 : size(matchList, 1)
        if ((matchList(i, 1) == maxPointImage) && (matchList(i, 5) == imageArray(j))) ||
        ((matchList(i, 1) == imageArray(j)) && (matchList(i, 5) == maxPointImage))
            subMatchList(counter, :) = matchList(i, :);
            counter = counter + 1;
        end
    end
end
if subMatchList(1,1) ~= 0
    for i = 1 : size(subMatchList, 1)
        distanceSum(i) = subMatchList(i,4) + subMatchList(i,8);
    end
    mue_distanceSum = max(distanceSum);
sigma_distanceSum = ceil(mue_distanceSum * 0.75);
amountOfXVals_distanceSum = abs(ceil((mue_distanceSum - sigma_distanceSum) * 6));
mue_deviations = 0;
sigma_deviations = 4;
amountOfXVals_deviations = abs(ceil((mue_deviations - sigma_deviations) * 6));
    for i = 1 : size(subMatchList, 1)
        score_distanceSum(i) = computeScore(distanceSum(i), mue_distanceSum, 
sigma_distanceSum, amountOfXVals_distanceSum);
        score_deviations(i) = computeScore(subMatchList(i, 9), mue_deviations, 
sigma_deviations, amountOfXVals_deviations);
        score_total(i) = score_distanceSum(i) * score_deviations(i);
    end
    maxScore = myIndexOf(score_total, max(score_total));
    info = subMatchList(maxScore, :);
else
    info = subMatchList;
end
**transformPoint.m**

```matlab
function [xVal yVal] = transformPoint(translationMatrix_, x, y, x_ref, y_ref)
% Given an arbitrary translation matrix and arbitrary cartesian coordinates
% this file computes the translated cartesian coordinates

x_offset = x - x_ref;
y_offset = y - y_ref;

translationMatrix = translationMatrix_;

x_trans = translationMatrix(3,1);
y_trans = translationMatrix(3,2);

translationMatrix(3,1) = 0;
translationMatrix(3,2) = 0;

point_offset_new = [x_offset y_offset 1] * translationMatrix;

x_offset_new = point_offset_new(1);
y_offset_new = point_offset_new(2);

xVal = x_offset_new + x_ref + x_trans;
yVal = y_offset_new + y_ref + y_trans;
```

**getMaxPointImageMissingPoints.m**

```matlab
function missingPointMap = getMaxPointImageMissingPoints(matchList, anglesAverage, bifrucationPoints, maxPointImage, sa, allDistances)
% This file finds points that have been found in two images other than the
% image defined by "maxPointImage"

% Filter "matchList" based on inter-bifurcation point connection angles
matchList va = verifyInterPointConnectionOrientation(matchList, anglesAverage, bifrucationPoints);

% Reformat "matchList va" for further processing

singleMatchList = getSingleMatchList(matchList va);

% Convert "singleMatchList" to a pointmap. The definition of "pointmap" can
% be found as a comment in the file "getPointAssignmentGroupings.m"

pointMap = getPointAssignmentGroupings(singleMatchList);

% If not empty, include "sa" in the pointmap
if sa(1,1) ~= 0
    pointMap_ = attachSingleAssignedPoints(pointMap, sa, allDistances);
else
    pointMap_ = pointMap;
end

% Find points that are not included in "maxPointImage" and isolate them in
% the array "missingPointMap"

counter = 1;
missingFlag = 0;
missingPoints = 0;

for i = 1 : size(pointMap_, 1)
    missingFlag = 0;
    for j = 1 : size(pointMap_, 3)
```
if pointMap_(i, 1, j) == maxPointImage
    missingFlag = -1;
end
end
if missingFlag == 0
    missingPoints(counter) = i;
    counter = counter + 1;
end
end
missingPointMap = zeros(length(missingPoints),2,5);
for i = 1 : length(missingPoints)
    currentIndex = missingPoints(i);
    if currentIndex ~= 0
        missingPointMap(i,:,:) = pointMap_(currentIndex,:,1:5);
    end
end

getSingleMatchList.m

function singleMatchList = getSingleMatchList(matchList)

%___________________________________________________________________________
%This file reformats matchlists and converts them from following shape
%| image1 | point1_1 | point1_2 | x | image2 | point2_1 | point1_2 | x | x |
% to the shape
%| image1 | point1_1 | image2 | point2_1 |
%| image2 | point2_1 | image2 | point2_2 |

counter = 1;
for i = 1 : size(matchList, 1)
    singleMatchList(counter, 1) = matchList(i, 1);
    singleMatchList(counter, 2) = matchList(i, 2);
    singleMatchList(counter, 3) = matchList(i, 5);
    singleMatchList(counter, 4) = matchList(i, 6);
    counter = counter + 1;
    singleMatchList(counter, 1) = matchList(i, 1);
    singleMatchList(counter, 2) = matchList(i, 3);
    singleMatchList(counter, 3) = matchList(i, 5);
    singleMatchList(counter, 4) = matchList(i, 7);
    counter = counter + 1;
end
singleMatchList = removeDuplicateEntries(singleMatchList);

removeDuplicateEntries.m

function retVal = removeDuplicateEntries(array)

%__________________________________________________________________________
%This file removes duplicate entries from arrays

counter = 1;
duplicateFlag = -1;
deleteFlag = ones(1,size(array,1));
deleteFlag = deleteFlag .* -1;
for i=1:size(array,1)-1
    tmp = array(i,:);
    for j=i+1:size(array,1)
        if array(j,:) == tmp
            deleteFlag(j) = 0;
        end
    end
end
for i=1:size(array,1)
    if deleteFlag(i) == -1
        retVal(counter,:) = array(i,:);
        counter = counter + 1;
    end
end

getPointAssignmentGroupings.m

function pointMap_singleColumn_final = getPointAssignmentGroupings(singleMatchList)
    %__________________________________________________________________________
    %This file reformats "singleMatchList" in following matter:
    %Following list
    %| 1 | 2 | 3 | 2 |
    %| 1 | 3 | 3 | 3 |
    %| 1 | 7 | 3 | 5 |
    %| 2 | 7 | 3 | 3 |
    %| 2 | 8 | 3 | 2 |
    %| 2 | 9 | 3 | 5 |
    %| 2 | 11 | 3 | 7 |
    %is turned into following container (pointMap_singleColumn_final is psf)
    %psf(1,:,:)     psf(2,:,:)      psf(3,:,:)      psf(4,:,:)
    %| 1 | 2 |      | 1 | 3 |       | 1 | 7 |       | 2 | 11 |
    %| 3 | 2 |      | 3 | 3 |       | 3 | 5 |       | 3 | 7 |
    %| 2 | 8 |      | 2 | 7 |       | 2 | 9 |
    groupCounter = 1;
    pointCounter = 1;
    pointMap = zeros(1,4,1);
    singleMatchList_ = singleMatchList;
    while size(singleMatchList, 1) > 0
        pointMap(groupCounter, :, pointCounter) = singleMatchList(1,:);
        singleMatchList = deleteEntry(singleMatchList, singleMatchList(1,:));
        pointCounter = pointCounter + 1;
        singleMatchList_ = singleMatchList;
        for i = 1 : size(singleMatchList, 1)
            for j = 1 : pointCounter - 1
                if min(singleMatchList(i,1:2) == pointMap(groupCounter, 1:2, j)) ||
                    min(singleMatchList(i,3:4) == pointMap(groupCounter, 1:2, j)) ||
                    min(singleMatchList(i,1:2) == pointMap(groupCounter, 3:4, j)) ||
                    min(singleMatchList(i,3:4) == pointMap(groupCounter, 3:4, j))
                    pointMap(groupCounter, :, pointCounter) = singleMatchList(i,:);
                    singleMatchList_ = deleteEntry(singleMatchList_, singleMatchList(i,:));
                    pointCounter = pointCounter + 1;
                end
            end
        end
        singleMatchList = singleMatchList_;
        groupCounter = groupCounter + 1;
        pointCounter = 1;
    end
    pointMap_final = zeros(1,4,size(pointMap,3));
    for i = 1 : size(pointMap, 1)
        pm_(1,:, :) = pointMap(i,:, :);
        pm_1(:, :) = pm_(1,:, :);
        pm_2 = pm_1';
end
pointMap_ = removeDuplicateEntries(pm_2);
pointMap_final(i,1:size(pointMap_ _, 2),1:size(pointMap_ _, 1)) = pointMap_(;,:,')';

end

counter = 1;
for i = 1 : size(pointMap_final, 1)
  for j = 1 : size(pointMap_final, 3)
    if pointMap_final(i,1,j) ~= 0
      counter = j;
    end
  end
  len(i) = counter;
end

pointMap_shortened = zeros(size(pointMap_final,1),4,max(len));
for i = 1 : size(pointMap_final, 1)
  pointMap_shortened(i,:,:) = pointMap_final(i,:,1:max(len));
end

counter = 1;
pointMap_singleColumn = zeros(1,2,1);
for i = 1 : size(pointMap_shortened, 1)
  for j = 1 : size(pointMap_shortened, 3)
    if pointMap_shortened(i,1,j) ~= 0
      pointMap_singleColumn(i,1,counter) = pointMap_shortened(i,1,j);
      pointMap_singleColumn(i,2,counter) = pointMap_shortened(i,2,j);
      counter = counter + 1;
      pointMap_singleColumn(i,1,counter) = pointMap_shortened(i,3,j);
      pointMap_singleColumn(i,2,counter) = pointMap_shortened(i,4,j);
      counter = counter + 1;
    end
  end
  counter = 1;
end

pointMap_singleColumn_final = zeros(1,2,5);
for i = 1 : size(pointMap_singleColumn, 1)
  pms_(_,1,:) = pointMap_singleColumn(i,:,);  %may need to be adjusted for 3D input
  pms_1(:,:) = pms_(_,1,:);
  pms_2 = pms_1';
  pointMap_singleColumn_ = removeDuplicateEntries(pms_2);
  pointMap_singleColumn_final(i,1:size(pointMap_singleColumn_, 2),1:size(pointMap_singleColumn_, 1)) = pointMap_singleColumn_(;,:');
end

function pointMap_ = attachSingleAssignedPoints(pointMap, sa, allDistances)
%__________________________________________________________________________
%This file uses all points that had not been included in the computation of
%the inter-bifurcation point candidate list as some points were
%discarded by the line "[sa, pa] = getDistancePairs(allCandidates);" Only
%"pa" was used. This file includes the "sa" entries as far as possible and
%adds their points to the "pointMap" structure

MATCHING_DISTANCE_THRESHOLD = 7;

marker = 1;

%__________________________________________________________________________
%Rearrange "sa" to "tempPointMap"

tempPointMap = getPointAssignmentGroupings(sa(:,:,4));
addFlag = 1;

%__________________________________________________________________________
%Verify whether or not points in both pointmaps are equal. If not equal,
%non-equal points are added to "tempPointMap"

attachSingleAssignedPoints.m
for i = 1 : size(tempPointMap, 1)
    for j = 1 : size(tempPointMap, 3)
        for m = 1 : size(pointMap, 1)
            for n = 1 : size(pointMap, 3)
                if tempPointMap(i,:,j) == pointMap(m,:,n)
                    if tempPointMap(i,1,j) ~= 0
                        for p = 1 : size(pointMap, 3)
                            if pointMap(i,:,p) ~= [0 0]
                                for q = 1 : size(tempPointMap, 3)
                                    if tempPointMap(i,:,q) == pointMap(m,:,p)
                                        addFlag = -1;
                                    end
                                end
                            else
                                addFlag = -1;
                            end
                            if addFlag == 0
                                for r = 1 : size(tempPointMap, 3)
                                    if tempPointMap(i,1,r) ~= 0
                                        marker = r;
                                    end
                                end
                                marker = marker + 1;
                                tempPointMap(i,:,marker) = pointMap(m,:,p);
                            end
                        end
                    end
                end
            end
        end
    end
end
counter = 1;
matchList_n = zeros(1,9);

"tempPointMap" is converted to a matchlist in order to compare
inter-bifurcation point distances

for h = 1 : size(tempPointMap, 1)
    for i = 1 : size(tempPointMap, 3) - 1
        for j = i + 1 : size(tempPointMap, 3)
            image1 = tempPointMap(h,1,i);
            image2 = tempPointMap(h,1,j);

            for k = 1 : size(pointMap, 1)
                for m = 1 : size(pointMap, 3) - 1
                    for n = m : size(pointMap, 3)
                        if pointMap(k,1,m) == tempPointMap(h,1,i) && pointMap(k,1,n) == tempPointMap(h,1,j) && tempPointMap(h,2,i) == pointMap(k,2,m) && pointMap(k,1,m) ~= 0 && pointMap(k,1,n) ~= 0
                            matchList_n(counter, 1) = image1;
                            matchList_n(counter, 2) = tempPointMap(h,2,i);
                            matchList_n(counter, 3) = pointMap(k,2,m);
                            matchList_n(counter, 4) = getDistance(image1, allDistances, tempPointMap(h,2,i), pointMap(k,2,m));
                            matchList_n(counter, 5) = image2;
                            matchList_n(counter, 6) = tempPointMap(h,2,j);
                            matchList_n(counter, 7) = pointMap(k,2,n);
                            matchList_n(counter, 8) = getDistance(image2, allDistances, tempPointMap(h,2,j), pointMap(k,2,n));
                            matchList_n(counter, 9) = abs(matchList_n(counter, 4) - matchList_n(counter, 8));
                            counter = counter + 1;
                        end
                    end
                end
            end
        end
    end
end

Appendix A
counter = 1;

% A threshold is applied to the match list
matchList_n_final = zeros(1,9);
for i = 1 : size(matchList_n, 1)
    if  matchList_n(i,9) < MATCHING_DISTANCE_THRESHOLD
        matchList_n_final(counter,:) = matchList_n(i,:);
        counter = counter + 1;
    end
end

counter1 = 1;
counter2 = 1;
aditionalPointMap = zeros(size(tempPointMap, 1),2,1);

% Compare thresholded match list to "tempPointMap" and check which entries have not been abolished by thresholding
for i = 1 : size(tempPointMap, 1)
    for j = 1 : size(tempPointMap, 3)
        for k = 1 : size(matchList_n_final, 1)
            addFlag1 = -1;
            if tempPointMap(i,1,j) == matchList_n_final(k,1) && tempPointMap(i,2,j) == matchList_n_final(k,2)
                addFlag1 = 0;
            elseif tempPointMap(i,1,j) == matchList_n_final(k,1) && tempPointMap(i,2,j) == matchList_n_final(k,3)
                addFlag1 = 0;
            elseif tempPointMap(i,1,j) == matchList_n_final(k,5) && tempPointMap(i,2,j) == matchList_n_final(k,6)
                addFlag1 = 0;
            elseif tempPointMap(i,1,j) == matchList_n_final(k,5) && tempPointMap(i,2,j) == matchList_n_final(k,7)
                addFlag1 = 0;
            end
            if addFlag1 == 0
                addFlag2 = 0;
                for m = 1:size(additionalPointMap, 3)
                    if additionalPointMap(counter1, 1, m) == tempPointMap(i,1,j) && additionalPointMap(counter1, 2, m) == tempPointMap(i,2,j)
                        addFlag2 = -1;
                    end
                end
                if addFlag2 == 0
                    additionalPointMap(counter1,counter2) = tempPointMap(i,1,j);
                    additionalPointMap(counter1,counter2) = tempPointMap(i,2,j);
                    counter2 = counter2 + 1;
                end
                if additionalPointMap(counter1,1,1) ~= 0
                    counter1 = counter1 + 1;
                end
            end
        end
    end
end
counter2 = 1;
end

% integrate left over points in "pointMap" and delete duplicate "pointMap" entries
pointMap_final_ = embedAdditionalPoints(additionalPointMap, pointMap);
pointMap_final_f = removeDuplicatePointMapEntries(pointMap_final_);
pointMap_ = pointMap_final_f;

embedAdditionalPoints.m

function pointMap_final_ = embedAdditionalPoints(additionalPoints, pointMap)

% This file merges two point maps

pointMap_final = pointMap;
for i = 1 : size(additionalPoints, 1)
    for j = 1 : size(additionalPoints, 3)
        for m = 1 : size(pointMap, 1)
            for n = 1 : size(pointMap, 3)
                if additionalPoints(i,:,j) == pointMap(m,:,n)
                    if additionalPoints(i,1,j) ~= 0
                        for p = 1 : size(additionalPoints, 3)
                            addFlag = 0;
                            marker = 1;
                            for q = 1 : size(pointMap, 3)
                                if additionalPoints(i,:,p) == pointMap_final(m,:,q)
                                    addFlag = -1;
                                    end
                                end
                                if pointMap_final(m,1,q) ~= 0
                                    marker = q;
                                end
                            end
                            marker = marker + 1;
                            if addFlag == 0
                                pointMap_final(m,:,marker) = additionalPoints(i,:,p);
                            end
                        end
                    end
                end
            end
        end
    end
end

for i = 1 : size(pointMap_final, 1)
    tempArray(:,1) = pointMap_final(i,:,1);
    tempArray_ = tempArray';
    tempArray_ = sortrows(tempArray_, 1);
    tempArray__ = tempArray_';
    if size(pointMap_final, 1) > 1
        pointMap_final_(i,:,1) = moveNonZeroEntriesUp(tempArray__(:,1));
    else
        pointMap_final_ = zeros(1,2,5);
    end
end
moveNonZeroEntriesUp.m

function array_ = moveNonZeroEntriesUp(array)
%This file moves all zero entries to the end of the array
array_ = zeros(size(array, 1), size(array, 2))
counter1 = 1;
while array(1,counter1) == 0
    counter1 = counter1 + 1;
end
counter2 = counter1;
counter3 = 1;
while array(1,counter2) ~= 0 && counter2 <= size(array, 2);
    array_(:,counter3) = array(:,counter2);
    if counter2 == size(array, 2)
        break;
    end
    counter2 = counter2 + 1;
    counter3 = counter3 + 1;
end

removeDuplicatePointMapEntries.m

function array_ = removeDuplicatePointMapEntries(array)
%This file removes duplicate entries in point map data structures
counter = 1;
deletepos = 0;
for i = 1 : size(array, 1) - 1
    a1(:,:, :) = array(i,:,:);
    for j = i + 1 : size(array, 1)
        a2(:,:, :) = array(j,:,:);
        if isEqual_2D(a1, a2) == 0
            deletepos(counter) = j;
            counter = counter + 1;
        end
    end
end
if deletepos(1) ~= 0
    counter = 1;
    for j = 1 : size(array, 1)
        deleteFlag = -1;
        for k = 1 : length(deletepos)
            if j == deletepos(k)
                deleteFlag = 0;
            end
        end
        if deleteFlag == -1
            array_(counter,:) = array{j,:, :};
            counter = counter + 1;
        end
    end
else
    array_ = array;
end
```matlab
clear;
MATCHING_DISTANCE_THRESHOLD = 7;

mouseNr = 9;
MATCHING_IMAGE = '5';
IMAGE_NR = mouseNr;
AMOUNT_OF_MICE = 20;
FILTER_BIG_STDDEV_1 = 5;
FILTER_BIG_STDDEV_2 = 15;
FILTER_SMALL_STDDEV_1 = 3;
FILTER_SMALL_STDDEV_2 = 10;
ri = MATCHING_IMAGE;

% Load unidentified image
filename = '\0';
filename(1) = 'i';
filename(2 : (length(num2str(IMAGE_NR)) + 1)) = num2str(IMAGE_NR);
filename(length(filename) + 1) = '_';
filename((length(filename) + 1) : (length(filename) + length(ri))) = ri;
filename((length(filename) + 1) : (length(filename) + 4)) = '.mat';

image = cell2mat(struct2cell(load(filename)));

% Enhance and segment unidentified image
[filteredImage, enhancedImage] = launchSegmentation3(image);
load('I:\fromAndi2603\FINAL\Image Postprocessing\postprocessed_image.mat');
result = trackedImage;
cd('I:\');

% Extract features from unidentified image
featureExtraction(result, enhancedImage, filteredImage);
load('bifurcation_points.mat');
load('difference.mat');
load('angles.mat');
load('anglePairs.mat');
load('anglesAverage.mat');

% Load the database
load('db.mat');
counter = 1;

% 1st filtering step : view all inter-bifurcation point distances and filter
% out the ones that are more than 7 pixels longer or shorter than the ones
% found in the unidentified image. The ones that are shorter than 50 pixels
% are filtered out as well
for i = 1 : size(difference_real, 2)
    for j = 1 : size(difference_real, 3)
        difference_real(1,j,i) = difference_real(1,i,j);
    end
end

for i = 1 : size(difference_real, 2)
    for j = 1 : size(difference_real, 3)
        ref = difference_real(1,i,j);
        for k = 1 : AMOUNT_OF_MICE
            for l = 1 : size(tDistances, 2)
                for m = 1 : size(tDistances, 3)
```
if abs(tDistances(k,l,m) - ref) < 7 && tDistances(k,l,m) ~= 0 &&
  tDistances(k,l,m) > 50
  candidate(counter, 1) = k;
candidate(counter, 2) = l;
candidate(counter, 3) = m;
candidate(counter, 4) = i;
candidate(counter, 5) = j;
counter = counter + 1;
end
end
end
end
counter1 = 1;
counter2 = 1;
%__________________________________________________________________________
%2nd filtering step : view rotation invariant branch orientation angles and
%filter out all sets of angles that have higher average angle differences to
%the angle sets in the unidentified image than 7.
for i = 1 : size(candidate, 1)
imNr = candidate(i, 1);
point11 = candidate(i, 2);
point12 = candidate(i, 3);
point21 = candidate(i, 5);
point22 = candidate(i, 6);
%_______________________________________________________________________
%sort the branch orientation angles by the angle sizes
angles_difference_11_2(1,:,1) = rearangeAngleTrios(tBranchAngles_invariant(imNr, :, point11), 0);
angles_difference_21_2(1,:,1) = rearangeAngleTrios(angles_difference(1, :, point21), 0);
angles_difference_12_2(1,:,1) = rearangeAngleTrios(tBranchAngles_invariant(imNr, :, point12), 0);
angles_difference_22_2(1,:,1) = rearangeAngleTrios(angles_difference(1, :, point22), 0);
%_______________________________________________________________________
%compute the average angle differences
error_1a_2 = abs(angles_difference_11_2(1,1,1) - angles_difference_21_2(1,1,1));
error_1b_2 = abs(angles_difference_11_2(1,2,1) - angles_difference_21_2(1,2,1));
error_1c_2 = abs(angles_difference_11_2(1,3,1) - angles_difference_21_2(1,3,1));
error_2a_2 = abs(angles_difference_12_2(1,1,1) - angles_difference_22_2(1,1,1));
error_2b_2 = abs(angles_difference_12_2(1,2,1) - angles_difference_22_2(1,2,1));
error_2c_2 = abs(angles_difference_12_2(1,3,1) - angles_difference_22_2(1,3,1));
avg1_2 = ((error_1a_2 + error_1b_2 + error_1c_2) / 3);
avg2_2 = ((error_2a_2 + error_2b_2 + error_2c_2) / 3);
%_______________________________________________________________________
%filter out all comparisons that exceed the value of 7
if (avg1_2 < 7) && (avg2_2 < 7)
candidate_2nd_2(counter2, :) = candidate(i, :);
counter2 = counter2 + 1;
end
end
for i = 1 : size(bifurcation_point, 3)
tBifurcationPoints(21,:,i) = bifurcation_point(1,:,i);
tBranchAngles(21,:,i) = angles_average(1,:,i);
end
counter = 1;
%__________________________________________________________________________
%3rd filtering step : view the inter-bifurcation point connection angles,
%compare them to the ones found in the unidentified image, score the differences and discard all scores that are below 85%

for i = 1 : size(candidate_2nd_2, 1)
    %______________________________________________________________________
    %"alfa_11" and "alfa_21" are the inter-bifurcation point connection angles of the database images
    alfa_11 = tConnectionAngles(candidate_2nd_2(i,1), candidate_2nd_2(i,2), candidate_2nd_2(i,3), 1);
    alfa_21 = tConnectionAngles(candidate_2nd_2(i,1), candidate_2nd_2(i,2), candidate_2nd_2(i,3), 2);
    %______________________________________________________________________
    %"alfa_12" and "alfa_22" are the inter-bifurcation point connection angles of the unidentified image
    [alfa_12, alfa_22] = getComparisonAngles(angles_average(1,:,:), bifurcation_point(1,:,:), candidate_2nd_2(i,5), candidate_2nd_2(i,6));
    score1 = computeScore(alfa_12, alfa_11, 20, 120);
    score2 = computeScore(alfa_22, alfa_21, 20, 120);
    scoretable(i, 1) = score1;
    scoretable(i, 2) = score2;
    %______________________________________________________________________
    %Filter out all entries below 85%
    if score1 >= 0.85 && score2 >= 0.85
        candidate_3rd_2b(counter,:) = candidate_2nd_2(i,:);
        counter = counter + 1;
    end
end
for i = 1 : size(candidate_3rd_2b, 1)
    regImage = candidate_3rd_2b(i, 1);
    score(i, 1) = regImage;
    %______________________________________________________________________
    %Perform segmentation result examination
    [score(i, 2), score(i, 3), score(i, 4), st, score(i, 5), score(i, 6)] = compareImages(tImage(regImage,:,:), result, tOriginalImage(regImage,:,:), image, tBifurcationPoints, candidate_3rd_2b(i,:));
end
score = sortrows(score, 2);

launchSegmentation3.m

function [filteredImage, enhancedImage] = launchSegmentation3(im_)

%Start of the enhancement process

cd ('I:\fromAndi2603\FINAL\Image Enhancement');
filterImage(im_, 'I1_1_enhanced_3_10_16.mat', 50, 5, 15, 16);
load('enhancedImage.mat');
load('filteredImage.mat');

%__________________________________________________________________________
%Start the segmentation procedure

cd ('I:\fromAndi2603\FINAL\Image Segmentation');
MainTracking(enhancedImage,filteredImage,zeros(300),enhancedImage,5,20,0);
load ('tracked_image.mat');
cd('I:\fromAndi2603\FINAL\Image Postprocessing');
PostProcessing(tracked_image,15,130);
load ('postprocessed_image.mat');

% pause;

%Creation of an enhanced image with smaller filter settings

cd ('I:\fromAndi2603\FINAL\Image Enhancement');
filterImage(im_, 'i1_1_enhanced_3_10_16.mat', 50, 3, 10, 16);
load('enhancedImage.mat');
load('filteredImage.mat');

%Creating an image where all the tracked vessels can not serve as starting
%points anymore

cd('I:\fromAndi2603\FINAL\Image Segmentation');
NewEnhancedImage=CreateNewEnhancedImage(trackedImage,enhancedImage);

% New tracking procedure with starting points from the new enhanced Image

cd('I:\fromAndi2603\FINAL\Image Segmentation');
MainTracking(enhancedImage,filteredImage,trackedImage,NewEnhancedImage,0.3,20,1);
load ('tracked_image.mat');

%Postprocessing the tracked image

cd('I:\fromAndi2603\FINAL\Image Postprocessing');
PostProcessing(tracked_image,20,130);

compareImages.m

function [score, score2, score3, st, score4, dpcs] = compareImages(image1_, image2,
origImage1_, origImage2, bifrucationPoints, matchList_entry)
%This file implements all proposed methods for statistical examination of
%the segmentation results
score = 0;
score2 = 0;
score3 = 0;
st = [0 0];
score4 = 0;
derelictPointCounter = 20000;
dpcs = 0;
image1(:, :) = image1_(1, :, :);
origImage1(:, :) = origImage1_(1, :, :);
%First both images are aligned, regardless of whether or not the
%transformation parameters are correct
fp1 = matchList_entry(2);
fp2 = matchList_entry(5);

dp1 = matchList_entry(3);

regImage = matchList_entry(1);
inputImage = 21;

x_fp1 = bifrucationPoints(regImage, 2, fp1);
y_fp1 = bifrucationPoints(regImage, 1, fp1);

x_fp2 = bifrucationPoints(inputImage, 2, fp2);
y_fp2 = bifrucationPoints(inputImage, 1, fp2);

x_dp1 = bifrucationPoints(regImage, 2, dp1);
y_dp1 = bifrucationPoints(regImage, 1, dp1);

x_dp2 = bifrucationPoints(inputImage, 2, dp2);
y_dp2 = bifrucationPoints(inputImage, 1, dp2);

[translationMatrix, refPoint] = getTranslationMatrix(x_fp2, y_fp2, x_dp2, y_dp2, x_fp1, y_fp1, 
x_dp1, y_dp1);

rotAngle = acos(translationMatrix(1,1));

vesselPoints = zeros(3,1);
counter = 1;
counter2 = 1;

%create artificial bifurcation points in "image1" and store them in the
%variable "vesselPoints". "image1" is the database image
for i = 20 : 20 : 280
    for j = 1 : 300
        if image1(j, i) > 0
            vesselPoints(1, counter) = j;
            vesselPoints(2, counter) = i;
            vesselPoints(3, counter) = counter2;
            counter = counter + 1;
        end
    end
    counter2 = counter2 + 1;
end

amountOfVesselPoints_reg = counter;
counter = 0;

%Compute statistics regarding points abandoned by false rotation and
%translation parameters
for i = 1 : size(vesselPoints, 2)
    x = vesselPoints(2,i);
    y = vesselPoints(1,i);
    num = vesselPoints(3,i);

    %mark all artificial bifurcation points in the original image
    %corresponding to the database image
    if x >= 2 && y >= 2
        origImage1(y - 1, x - 1) = 100;
    end
    if x >= 0 && y >= 2
        origImage1(y - 1, x + 1) = 100;
    end
    if x >= 2 && y >= 0
        origImage1(y + 1, x - 1) = 100;
    end
    if x >= 0 && y >= 0
        origImage1(y + 1, x + 1) = 100;
    end
    if x >= 1 && y >= 1
        origImage1(y, x) = 100;
    end

%transform all artificial bifurcation points created in the database
%image to the unidentified image
```
[x_rotated_ y_rotated_] = transformPoint(translationMatrix, x, y, refPoint(2), refPoint(1));

x_rotated = round(x_rotated_);
y_rotated = round(y_rotated_);

% verify whether the transformed points have valid coordinates and discard the ones that have invalid coordinates. mark the valid ones in the original image of the unidentified ear
if x_rotated >= 2 && x_rotated <= 298 && y_rotated >= 2 && y_rotated <= 298
    counter = counter + 1;
    vesselPoints_inpImage(counter, 1) = y_rotated;
    vesselPoints_inpImage(counter, 2) = x_rotated;
    vesselPoints_inpImage(counter, 3) = num;
    if x_rotated >= 2 && y_rotated >= 2
        origImage2(y_rotated - 1, x_rotated - 1) = 100;
    end
    if x_rotated >= 0 && y_rotated >= 2
        origImage2(y_rotated - 1, x_rotated + 1) = 100;
    end
    if x_rotated >= 2 && y_rotated >= 0
        origImage2(y_rotated + 1, x_rotated - 1) = 100;
    end
    if x_rotated >= 0 && y_rotated >= 0
        origImage2(y_rotated + 1, x_rotated + 1) = 100;
    end
    if x_rotated >= 1 && y_rotated >= 1
        origImage2(y_rotated, x_rotated) = 100;
    end
end

end

amountOfVesselPoints_img = counter;

figure(1);
imagesc(image1);
colormap(gray);
figure(2)
imagesc(image2);
colormap(gray);
figure(3);
imagesc(origImage1);
colormap(gray);
figure(4);
imagesc(origImage2);
colormap(gray);

% Compute the ratio of successfully transformed points over the total amount of points
score = amountOfVesselPoints_img / amountOfVesselPoints_reg;

% Do not perform the other tests if none of the points were successfully transformed to the unidentified image because that would mean that the transformation parameters are completely wrong
if score > 0
    trackedPoints = allignPoints(vesselPoints_inpImage, image2);
    offset = 0;
    rotationAngle_ = acos(translationMatrix{1, 1});
    angle_ = pi/2 - rotationAngle_;
    newIm = zeros(630);
```
translationMatrix2 = [cos(angle_) -sin(angle_) 0; sin(angle_) cos(angle_) 0; 0 0 1];

% %According to the rotation parameters, create artificial lines in the %database analogous to the vertical lines created in the database image %used to create artificial bifurcation points

for i = 103 + offset : 20 : 528 + offset
    for j = 103 : 528
        p_1 = [j i 1];
        [p_2(1) p_2(2)] = transformPoint(translationMatrix2, i, j, 315, 315);
        p_2(1) = round(p_2(1));
        p_2(2) = round(p_2(2));
        if p_2(1) > 0 && p_2(2) > 0
            newIm(p_2(1), p_2(2)) = 1;
        end
    end
end

subIm = newIm(166:465, 166: 465);

% %Shift the lines so that they run through the vessel system at the same %spots the vertical lines in the database image do

figure(5);
imagesc(subIm);
colormap(gray);
x_shift = trackedPoints(2);
counter = 0;
while (1)
    x_shift_minus = x_shift - counter;
    x_shift_plus = x_shift + counter;
    if x_shift_minus == 0
        return;
        break;
    end
    if x_shift_plus == 301
        return;
        break;
    end
    if subIm(trackedPoints(1), x_shift_minus) == 1
        direction = -1;
        break;
    end
    if subIm(trackedPoints(1), x_shift_plus) == 1
        direction = 1;
        break;
    end
    counter = counter + 1;
end

shift = direction;
while (1)
    newIm = zeros(630);
    counter = 1;
    for i = 103 + shift : 20 : 528 + shift
        for j = 103 : 528
            p_1 = [j i 1];
            [p_2(1) p_2(2)] = transformPoint(translationMatrix2, i, j, 315, 315);
            p_2(1) = round(p_2(1));
            p_2(2) = round(p_2(2));
            if p_2(1) > 0 && p_2(2) > 0
newIm(p_2(1), p_2(2)) = counter;
end

counter = counter + 1;
end

subIm = newIm(166:465, 166:465);

if subIm(trackedPoints(1), trackedPoints(2)) > 0 || subIm(trackedPoints(1) - 1, trackedPoints(2)) > 0
break;
end

if direction == -1
    shift = shift - 1;
elseif direction == 1
    shift = shift + 1;
end

end

figure(6);
imagesc(subIm);
colormap(gray);

vesselPoints_inpImage_2 = zeros(3,1);
vPcounter = 1;

% %__________________________________________________________________________
% %create artificial bifurcation points in the unidentified image
% for i = 1 : 1 : size(subIm, 1) - 0
% for j = 1 : 1 : size(subIm, 2) - 0
%     x = j;
%     y = i;
%     noGoFlag = -1;
%     if y > 1 && y < 299 && x > 1 && x < 299
%         surrounding = origImage2(y - 1 : y + 1, x - 1 : x + 1);
%         for m = 1 : 3
%             for n = 1 : 3
%                 if surrounding(m, n) == 201
%                     noGoFlag = 0;
%                 end
%             end
%         end
%     end
% end

if subIm(i, j) > 0 && image2(i, j) == 1 && noGoFlag == -1
% %__________________________________________________________________________
% %create the points
% vesselPoints_inpImage_2(1, vPcounter) = y;
% vesselPoints_inpImage_2(2, vPcounter) = x;
% vesselPoints_inpImage_2(3, vPcounter) = subIm(i, j);
% vPcounter = vPcounter + 1;
% %__________________________________________________________________________
% %and mark them in the original unidentified image
% if y >= 2 && x >= 2
%     orgImage2(y - 1, x - 1) = 200;
% end
% if y >= 0 && x >= 2
%     orgImage2(y - 1, x + 1) = 200;
% end
% if y >= 2 && x >= 0
%     orgImage2(y + 1, x - 1) = 200;
% end
% if y >= 0 && x >= 0
%     orgImage2(y + 1, x + 1) = 200;
% end
% if y >= 1 && x >= 1
%     orgImage2(y, x) = 200;
% end
%
% %__________________________________________________________________________
% %and mark them in the original unidentified image
% if y >= 2 && x >= 2
%     orgImage2(y, x - 1) = 200;
% end
% if y >= 0 && x >= 2
%     orgImage2(y, x + 1) = 200;
% end
% if y >= 2 && x >= 0
%     orgImage2(y + 1, x - 1) = 200;
% end
% if y >= 0 && x >= 0
%     orgImage2(y + 1, x + 1) = 200;
% end
% if y >= 1 && x >= 1
%     orgImage2(y, x) = 200;
% end
%
% %__________________________________________________________________________
% %and mark them in the original unidentified image
% if y >= 2 && x >= 2
%     orgImage2(y, x - 1) = 200;
% end
% if y >= 0 && x >= 2
%     orgImage2(y, x + 1) = 200;
% end
% if y >= 2 && x >= 0
%     orgImage2(y + 1, x - 1) = 200;
% end
% if y >= 0 && x >= 0
%     orgImage2(y + 1, x + 1) = 200;
% end
% if y >= 1 && x >= 1
%     orgImage2(y, x) = 200;
% end
%
% %__________________________________________________________________________
% %and mark them in the original unidentified image
% if y >= 2 && x >= 2
%     orgImage2(y, x - 1) = 200;
% end
% if y >= 0 && x >= 2
%     orgImage2(y, x + 1) = 200;
% end
% if y >= 2 && x >= 0
%     orgImage2(y + 1, x - 1) = 200;
% end
% if y >= 0 && x >= 0
%     orgImage2(y + 1, x + 1) = 200;
% end
% if y >= 1 && x >= 1
%     orgImage2(y, x) = 200;
% end
%
% %__________________________________________________________________________
% %and mark them in the original unidentified image
% if y >= 2 && x >= 2
%     orgImage2(y, x - 1) = 200;
% end
% if y >= 0 && x >= 2
%     orgImage2(y, x + 1) = 200;
% end
% if y >= 2 && x >= 0
%     orgImage2(y + 1, x - 1) = 200;
% end
% if y >= 0 && x >= 0
%     orgImage2(y + 1, x + 1) = 200;
% end
% if y >= 1 && x >= 1
%     orgImage2(y, x) = 200;
% end
%
Appendix A

\texttt{origImage2(y, x) = 201;}
\texttt{end}
\texttt{end}

\texttt{figure(7);}
\texttt{imagesc\(\texttt{origImage2}\);}
\texttt{colormap\(\texttt{gray}\);}
\texttt{for i = 1 : 1 : size\(\texttt{subIm}, 1\) - 1}
\texttt{for j = 1 : 1 : size\(\texttt{subIm}, 2\) - 1}
\texttt{subset\_subIm = subIm\(i : i + 1, j : j + 1\);}
\texttt{subset\_image2 = image2\(i : i + 1, j : j + 1\);}
\texttt{if subset\_subIm\(1, 1\) \> 0 && subset\_subIm\(2, 2\) \> 0 && subset\_image2\(1, 2\) == 1 && subset\_image2\(2, 1\) == 1}
\texttt{x = j;}
\texttt{y = i + 1;}
\texttt{vesselPoints\(_\texttt{inpImage}_2\)(1, vPcounter) = y;}
\texttt{vesselPoints\(_\texttt{inpImage}_2\)(2, vPcounter) = x;}
\texttt{vesselPoints\(_\texttt{inpImage}_2\)(3, vPcounter) = subIm\(i, j\);}
\texttt{vPcounter = vPcounter + 1;}
\texttt{if y \>= 2 && x \>= 2}
\texttt{\texttt{origImage2(y - 1, x - 1) = 200;}}
\texttt{end}
\texttt{if y \>= 0 && x \>= 2}
\texttt{\texttt{origImage2(y - 1, x + 1) = 200;}}
\texttt{end}
\texttt{if y \>= 2 && x \>= 0}
\texttt{\texttt{origImage2(y + 1, x - 1) = 200;}}
\texttt{end}
\texttt{if y \>= 0 && x \>= 0}
\texttt{\texttt{origImage2(y + 1, x + 1) = 200;}}
\texttt{end}
\texttt{if y \>= 1 && x \>= 1}
\texttt{\texttt{origImage2(y, x) = 201;}}
\texttt{end}
\texttt{end}
\texttt{if subset\_subIm\(2, 1\) \> 0 && subset\_subIm\(1, 2\) \> 0 && subset\_image2\(1, 1\) == 1 && subset\_image2\(2, 2\) == 1}
\texttt{x = j + 1;}
\texttt{y = i + 1;}
\texttt{vesselPoints\(_\texttt{inpImage}_2\)(1, vPcounter) = y;}
\texttt{vesselPoints\(_\texttt{inpImage}_2\)(2, vPcounter) = x;}
\texttt{vesselPoints\(_\texttt{inpImage}_2\)(3, vPcounter) = subIm\(i + 1, j\);}
\texttt{vPcounter = vPcounter + 1;}
\texttt{if y \>= 2 && x \>= 2}
\texttt{\texttt{origImage2(y - 1, x - 1) = 200;}}
\texttt{end}
\texttt{if y \>= 0 && x \>= 2}
\texttt{\texttt{origImage2(y - 1, x + 1) = 200;}}
\texttt{end}
\texttt{if y \>= 2 && x \>= 0}
\texttt{\texttt{origImage2(y + 1, x - 1) = 200;}}
\texttt{end}
\texttt{if y \>= 0 && x \>= 0}
\texttt{\texttt{origImage2(y + 1, x + 1) = 200;}}
\texttt{end}
\texttt{if y \>= 1 && x \>= 1}
\texttt{\texttt{origImage2(y, x) = 201;}}
\texttt{end}
end

end

figure(8);
imagesc(OrigImage2);
colormap(gray);

vesselPoints_inpImage_2 = sortrows(vesselPoints_inpImage_2', 3)';

% Find the distances between adjacent artificial bifurcation points lying on
% the same artificial lines in both, the unidentified image and the database
% image

[vPDistances_ii vPDistances_ri vPCoordinates_ii vPCoordinates_ri vPOrientations_ii
vPOrientations_ri] = getVesselPointDistances(vesselPoints_inpImage_2, vesselPoints_inpImage');

% Among all distances between the artificial bifurcation points, find the
% ones that are similar. Thereby, compare the database image with the
% unidentified image
% "score2" is the amount of similar distances and "score4" is the total
% amount of distances

[score2, score4] = findSetOfEqualVesselPointDistances(vPDistances_ri, vPDistances_ii,
vPCoordinates_ii, vPCoordinates_ri, vPOrientations_ii, vPOrientations_ri);
score2 = score2 / score4;

% Compute statistics regarding the amount of artificial bifurcation points
% created in the unidentified image versus the amount of points successfully
% transformed from the database image

score3_1 = size(vesselPoints_inpImage_2, 2) / amountOfVesselPoints_img;
score3_2 = amountOfVesselPoints_img / size(vesselPoints_inpImage_2, 2);
if score3_1 > 1
    score3 = score3_2;
else
    score3 = score3_1;
end

% Compute the surroundings of the artificial bifurcation points created in
% the unidentified image (40x40 pixel squares with the point in the center)
for i = 1 : size(vesselPoints_inpImage_2, 2)
    vesselSurrounding(i, 1) = vesselPoints_inpImage_2(1, i) - 20;
    vesselSurrounding(i, 2) = vesselPoints_inpImage_2(1, i) + 20;
    vesselSurrounding(i, 3) = vesselPoints_inpImage_2(2, i) - 20;
    vesselSurrounding(i, 4) = vesselPoints_inpImage_2(2, i) + 20;
end
derelictPointCounter = 1;

% For each point that had been transformed to the unidentified image, verify
% whether or not a created point can be found within the surroundings
for i = 1 : size(vesselPoints_inpImage, 1)
    containedFlag = -1;
    for j = 1 : size(vesselSurrounding, 1)
        if vesselPoints_inpImage(i, 1) >= vesselSurrounding(j, 1) &&
           vesselPoints_inpImage(i, 1) <= vesselSurrounding(j, 2)
            if vesselPoints_inpImage(i, 2) >= vesselSurrounding(j, 3) &&
               vesselPoints_inpImage(i, 2) <= vesselSurrounding(j, 4)
                containedFlag = 0;
            end
        end
    end
end
Appendix A

if containedFlag == -1
    derelictPointCounter = derelictPointCounter + 1;
end
end

%Score the amount of derelict (transformed points which do not have a %counterpart in their vicinity) points versus all points created in the %unidentified image

dpcs = computeScore(derelictPointCounter, 0, (size(vesselPoints_inpImage, 1) / 8), 100);
end

allignPoints.m

function trackedPoints = allignPoints(trackedPoints_, inputImage)
%This algorithm originates from the center of an image and searches for the %next artificial bifurcation point. As soon as that point is found, the %shortest distance to the next vessel is found. Finally, a point lying on a %vessel and having the shortest distance to the found artificial %bifurcation point is returned

initPoint = [150 150];
refPoint_2 = initPoint
breakflag = -1;

for i = 1 : 149
    x_left = initPoint(2) - i;
    x_right = initPoint(2) + i;
    y_top = initPoint(1) + i;
    y_bottom = initPoint(1) - i;

    for k = 1 : size(trackedPoints_, 1)
        tPy = trackedPoints_(k, 1);
        tPx = trackedPoints_(k, 2);

        if trackedPoints_(k, 1) <= y_top && trackedPoints_(k, 1) >= y_bottom &&
        trackedPoints_(k, 2) <= x_right && trackedPoints_(k, 2) >= x_left
            refPoint_1 = trackedPoints_(k, :);
            breakflag = 0;
            break;
        end
    end

    if breakflag == 0
        break;
    end

end

breakflag = -1;
counter = 1;

while counter < 149
    y_plus = refPoint_1(1) + counter;
    y_minus = refPoint_1(2) - counter;

    if y_plus <= 300
        if inputImage(y_plus, refPoint_1(2)) == 1
            refPoint_2 = [y_plus refPoint_1(2)];
            break;
        end
    end

    if y_minus >= 1
        if inputImage(y_minus, refPoint_1(2)) == 1 && y_plus >= 1
            break;
        end
    end
end

if y_minus >= 1
    if inputImage(y_minus, refPoint_1(2)) == 1 && y_plus >= 1
        break;
    end
end

break;
Appendix A

refPoint_2 = [y_minus refPoint_1(2)];
break;
end

counter = counter + 1;
end

trackedPoints = refPoint_2;

**getVesselPointDistances.m**

function [vPDistances_ii vPDistances_ri vPCoordinates_ii vPCoordinates_ri vPOrientations_ii vPOrientations_ri] = getVesselPointDistances(vP_ii, vP_ri)
%
%This file is in charge of calculating all distances between artificial
%bifurcation points generated in one database image and one unidentified
%image respectively
%_ii stands for the unidentified image and _ri for the database image
%the input parameters refer to the arrays holding the artificial
%bifurcation points

vP_ii_min = vP_ii(3, 1);
vP_ii_max = vP_ii(3, size(vP_ii, 2));

vP_ri_min = vP_ri(3, 1);
vP_ri_max = vP_ri(3, size(vP_ri, 2));

vP_ii_rearanged = zeros(vP_ii_max, 2, 1);
vP_ri_rearanged = zeros(vP_ri_max, 2, 1);

for i = vP_ii_min : vP_ii_max
    counter = 1;
    for j = 1 : size(vP_ii, 2)
        if vP_ii(3, j) == i
            vP_ii_rearanged(i, 1, counter) = vP_ii(1, j);
            vP_ii_rearanged(i, 2, counter) = vP_ii(2, j);
            counter = counter + 1;
        end
    end
end

for i = vP_ri_min : vP_ri_max
    counter = 1;
    for j = 1 : size(vP_ri, 2)
        if vP_ri(3, j) == i
            vP_ri_rearanged(i, 1, counter) = vP_ri(1, j);
            vP_ri_rearanged(i, 2, counter) = vP_ri(2, j);
            counter = counter + 1;
        end
    end
end

vPDistances_ii = zeros(size(vP_ii_rearanged, 1), 1, size(vP_ii_rearanged, 3) - 1);
vPDistances_ri = zeros(size(vP_ri_rearanged, 1), 1, size(vP_ri_rearanged, 3) - 1);

vPCoordinates_ii = zeros(size(vP_ii_rearanged, 1), 4, size(vP_ii_rearanged, 3) - 1);
vPCoordinates_ri = zeros(size(vP_ri_rearanged, 1), 4, size(vP_ri_rearanged, 3) - 1);

vPOrientations_ii = zeros(size(vP_ii_rearanged, 1), 1, size(vP_ii_rearanged, 3) - 1);
vPOrientations_ri = zeros(size(vP_ri_rearanged, 1), 1, size(vP_ri_rearanged, 3) - 1);

for i = 1 : size(vP_ii_rearanged, 1)
    if vP_ii_rearanged(i, 1, 1) ~= 0
        counter = 1;
        while vP_ii_rearanged(i, 1, counter) ~= 0 && counter < size(vP_ii_rearanged, 3)
            counter = counter + 1;
        end
    end
end
amountOfDistances = counter - 1;
vP_ii_tmp(:,:,,:) = vP_ii_rearanged(i, :, :);
vP_ii_tmp = moveNonZeroEntriesUp(sortrows(vP_ii_tmp', 1)');
for j = 1 : amountOfDistances
y1 = vP_ii_tmp(1, j);
y2 = vP_ii_tmp(1, j + 1);
x1 = vP_ii_tmp(2, j);
x2 = vP_ii_tmp(2, j + 1);

vPDistances_ii(i, 1, j) = sqrt(((x1 - x2)^2) + ((y1 - y2)^2));
vPCoordinates_ii(i, 1, j) = y1;
vPCoordinates_ii(i, 2, j) = x1;
vPCoordinates_ii(i, 3, j) = y2;
vPCoordinates_ii(i, 4, j) = x2;

vPOrientations_ii(i, 1, j) = atan((y2 - y1) / (x2 - x1));
end
end

for i = 1 : size(vP_ri_rearanged, 1)
if vP_ri_rearanged(i, 1, 1) ~= 0
    counter = 1;
    while vP_ri_rearanged(i, 1, counter) ~= 0 && counter < size(vP_ri_rearanged, 3)
        counter = counter + 1;
    end

    amountOfDistances = counter - 1;
vP_ri_tmp(:,:,,:) = vP_ri_rearanged(i, :, :);
for j = 1 : amountOfDistances
    y1 = vP_ri_tmp(1, j);
    y2 = vP_ri_tmp(1, j + 1);
    x1 = vP_ri_tmp(2, j);
    x2 = vP_ri_tmp(2, j + 1);

    vPDistances_ri(i, 1, j) = sqrt(((x1 - x2)^2) + ((y1 - y2)^2));
    vPCoordinates_ri(i, 1, j) = y1;
    vPCoordinates_ri(i, 2, j) = x1;
    vPCoordinates_ri(i, 3, j) = y2;
    vPCoordinates_ri(i, 4, j) = x2;
    vPOrientations_ri(i, 1, j) = atan((y2 - y1) / (x2 - x1));
end
end
end

findSetOfEqualVesselPointDistances.m

function [retVal, totalCounter] = findSetOfEqualVesselPointDistances(vPDistances_ri,
    vPDistances_ii, vPCoordinates_ii, vPCoordinates_ri, vPOrientations_ii, vPOrientations_ri)

  % This file is in charge of finding similar distances between artificial bifurcation points whereas distances found in the database image are compared to those found in the unidentified image. Thereby, the length of the distances is considered as well as the orientation. "retVal" refers to the amount of matches found and "totalCounter" to the total amount of distances found.

  TOLERANCE_IN_PIXELS = 5;
  TOLERANCE_IN_ORIENTATION = pi/8;
  WINDOWSIZE = 15;
  matchCounter = 0;
  comparisonCounter = 0;
  matchTable = [0 0];
d1 = 0;
d2 = 0;
d1_ = 0;
d2_ = 0;
mtCounter = 1;
cmpCounter = 0;

for i = 1 : size(vPDistances_ii, 1)
    for j = 1 : size(vPDistances_ii, 3)
        d1(j) = vPDistances_ii(i, 1, j);
    end
for k = 1 : size(vPDistances_ri, 1)
    for l = 1 : size(vPDistances_ri, 3)
        if vPDistances_ri(k, 1, l) == 0
            break;
        end
        d2(l) = vPDistances_ri(k, 1, l);
    end
end
d1counter = 1;
d2counter = 1;

for s = 1 : length(d1)
    if d1(s) > 5 && d1(s) < 150
        d1_(d1counter) = d1(s);
        d1counter = d1counter + 1;
    end
end

for s = 1 : length(d2)
    if d2(s) > 5 && d2(s) < 150
        d2_(d2counter) = d2(s);
        d2counter = d2counter + 1;
    end
end

if d1_(1) == 0 || d2_(1) == 0
    matchFlag = -1;
else
    matchFlag = 0;
end

if matchFlag == 0
    comparisonCounter = comparisonCounter + 1;
    if length(d1_) == length(d2_)
        for l = 1 : length(d1_)
            if d1_(l) == 0 && d2_(l) == 0
                if abs(d1_(l) - d2_(l)) > TOLERANCE_IN_PIXELS
                    matchFlag = -1;
                end
            end
        end
        if matchFlag == 0
            matchCounter = matchCounter + 1;
            matchTable(mtCounter, 1) = i;
            matchTable(mtCounter, 2) = k;
            mtCounter = mtCounter + 1;
        end
    elseif length(d1_) > length(d2_)
        for l = 0 : ((length(d1_) - length(d2_)) - 1)
            matchFlag = 0;
        end
end
for m = 1 : length(d2_)
  if d1_(1) == 0 && d2_(1) == 0
    if abs(d1_(m + l) - d2_(m)) > TOLERANCE_IN_PIXELS
      matchFlag = -1;
    end
  end
  if matchFlag == 0
    matchCounter = matchCounter + 1;
    matchTable(mtCounter, 1) = i;
    matchTable(mtCounter, 2) = k;
    mtCounter = mtCounter + 1;
  end
end

elseif length(d1_) < length(d2_)
  for l = 0 : ((length(d2_) - length(d1_)) - 1)
    matchFlag = 0;
    for m = 1 : length(d1_)
      if d1_(1) == 0 && d2_(1) == 0
        if abs(d2_(m + l) - d1_(m)) > TOLERANCE_IN_PIXELS
          matchFlag = -1;
        end
      end
      if matchFlag == 0
        matchCounter = matchCounter + 1;
        matchTable(mtCounter, 1) = i;
        matchTable(mtCounter, 2) = k;
        mtCounter = mtCounter + 1;
      end
    end
  end
end

counter = 0;
totalCounter = 0;
for i = 1 : size(vPDistances_ii, 1)
  for j = 1 : size(vPDistances_ii, 3)
    d1 = vPDistances_ii(i, 1, j);
    if d1 > 5
      totalCounter = totalCounter + 1;
    end
  end
  window_top_1 = vPCoordinates_ii(i, 1, j) - WINDOWSIZE;
  window_bottom_1 = vPCoordinates_ii(i, 1, j) + WINDOWSIZE;
  window_left_1 = vPCoordinates_ii(i, 2, j) - WINDOWSIZE;
  window_right_1 = vPCoordinates_ii(i, 2, j) + WINDOWSIZE;
  window_top_2 = vPCoordinates_ii(i, 3, j) - WINDOWSIZE;
  window_bottom_2 = vPCoordinates_ii(i, 3, j) + WINDOWSIZE;
  window_left_2 = vPCoordinates_ii(i, 4, j) - WINDOWSIZE;
  window_right_2 = vPCoordinates_ii(i, 4, j) + WINDOWSIZE;
  orientation_ii = vPOrientations_ii(i, 1, j);
  for k = 1 : size(vPDistances_ri, 1)
    for l = 1 : size(vPDistances_ri, 3)
d2 = vPDistances_ri(k, 1, l);
orientation_ri = vPOrientations_ri(k, 1, l);
if d1 > 5 && d2 > 5
  if abs(d2 - d1) <= TOLERANCE_IN_PIXELS && abs(orientation_ri - orientation_ii) <= TOLERANCE_IN_ORIENTATION
    x1 = vPCoordinates_ri(k, 2, l);
y1 = vPCoordinates_ri(k, 1, l);
x2 = vPCoordinates_ri(k, 4, l);
y2 = vPCoordinates_ri(k, 3, l);
    if vPCoordinates_ri(k, 1, l) >= window_top_1 && vPCoordinates_ri(k, 1, l) <= window_bottom_1 && vPCoordinates_ri(k, 2, l) <= window_right_1 && vPCoordinates_ri(k, 2, l) >= window_left_1
      if vPCoordinates_ri(k, 3, l) >= window_top_2 && vPCoordinates_ri(k, 3, l) <= window_bottom_2 && vPCoordinates_ri(k, 4, l) <= window_right_2 && vPCoordinates_ri(k, 4, l) >= window_left_2
        counter = counter + 1;
      end
    end
  end
end
end
end
end
end
retVal = counter;